

Domain Knowledge-Based Reconciliation of Model Fragments

Darijus Strasunskas¹, Yun Lin² and Sari Hakkarainen³

¹ Norwegian University of Science and Technology, Norway and Vilnius University, Lithuania. Darijus.Strasunskas@idi.ntnu.no

^{2,3} Norwegian University of Science and Technology, Norway. (Yun.Lin, Sari.Hakkarainen)@idi.ntnu.no

Introduction

Modelling is the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication [5] that often is applied in the early phases of systems development: analysis and design. However, different people usually present different models even when given the same domain and the same problem. Same information about a system can be modelled on various levels of abstraction, from different viewpoints, and consider different aspects. Model heterogeneity generally arises due to the creative nature of the modelling activity. Other factors such as the richness of the modelling language [4], and the ambiguities of modelling grammars typically strengthen model heterogeneity.

Modelling is usually cooperative activity among several developers/analysts, where a final model must be composed from different intermediate model fragments. The challenge in modelling is to arrive at a coherent, complete and consistent description of the problem in a particular domain. In this paper, we seek to answer the questions: “How can we relate different views and aspects in modelling?” and: “How can we manage the changes in a distributed modelling environment?”

Ontologies have been used in various roles for different types of consolidation purposes, e.g. [2, 3, 12] for data integration, or schema and ontology integration through upper level ontology. Similarly to conceptual models, ontologies are built with the aim of sharing knowledge, or definitions, with other people. In addition, they are created to support automatic reasoning. Here, we elaborate on how ontologies could be used as an intermediate medium for model consolidation. The focus is on end-user support for the individual developers. Their views and perceptions of the Uni-

verse of Discourse (UoD) are integrated based on some explicit basic knowledge about the domain. Ontology is used as a reference point and built with the sole intention to share knowledge with others. We use ontology to define and formalize basic knowledge and the main objects in the domain. Below settings and an illustration of the problem are discussed, followed by a section presenting our approach in detail. Before concluding the paper, our approach is compared to the current state of the art.

The Complex Activity of Distributed Modelling

In general, modelling is a complex and difficult task. It is usually a creative and collaborative process, during which different stakeholders are expressing them at different levels of abstraction, focusing on different aspects, and producing different variants of each model. Thus, the problem here is how to support the management of logically and/or geographically distributed modelling tasks. The problem can be illustrated by the following scenario which will be used throughout the paper.

Consider a process of designing an offshore platform at an oil company. Different (groups of) engineers are responsible for modelling different parts and aspects of a new oil platform. One group is responsible for the pipeline system design, i.e. the technological equipment; another is modelling the platform on which all equipment will stand; and yet another is dealing with the capacity of oil extraction and production, i.e. the drilling and pumping devices.

Developers (groups) work separately having only weekly meetings to align and reconcile their models. During the meetings every developer goes through the changes and decisions made after the last meeting. Other developers know, based on previous experience and common knowledge, what impact those changes have on their own models. For instance, if the production engineer decides to increase the oil pumping production by a certain amount, the engineer responsible for pipe lines knows that some of the pipes should be changed to wider and thicker ones to support the increased pressure. Meanwhile, the engineer designing a platform can see an impact to her part of the work as the platform will need to carry heavier constructions built on it. The problem here is how to support this kind of collaboration activity by at least partially computerizing and automating this troublesome task of model reconciliation.

Such rough impact assessments are based on ad-hoc expertise shared by all developers engaged in this project. They are aware of the dependencies that hold between model elements, even if the parts are not explicitly or di-

rectly related. The dependency type considered here is the impact relationship where, if one element is modified, then the other is impacted by this modification, yet the elements are not otherwise physically or logically related. Other types of dependency are (but not restricted to) derived-from, composed-of, and based-on relationships.

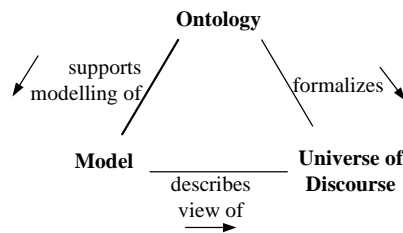


Fig. 1. Abstract view of our approach

Ontology as Intermediate Model

In order to relate the variety of model fragments we need to have common reference point. Our basic assumptions consider ontology, model and UoD as depicted in Fig.1. The approach is inspired by one of the linguistics' methods for describing the meaning of objects – the so called semiotic triangle [6].

UoD is basic knowledge about a particular domain. Ontology is used to represent (a portion of) UoD and to transform it into a man/machine understandable format. Ontology captures main concepts from a domain and represents relationships among the concepts in a machine readable and reasoning-able way. The goal is to capture common knowledge about which entities and what kind of dependency relationships they have in particular UoD. A model is instantiation of the ontology, where the basic theory supports representation of a particular problem. Consequently, our framework has two layers: an ontology layer for representing and formalizing a given UoD; and a model layer for modelling a problem (solution to the problem) within that UoD.

The advantage of this framework is that it separates the basic knowledge as the most reusable knowledge and places this in an ontology layer, keeping the layer of models separately. Ontology layer is composed of a set of concepts representing abstract entities in the real world, and relations among them that are normally based on the external and functional properties of the concepts. Model layer is instantiated abstract entities. To return to our scenario, each modelling of a variant of the oil platform is based on

knowledge already captured in the ontology. Thus, ontology layer is used to reason about dependencies among modelled objects based on their properties and model layer is used for reconciliation of the situated model fragments. Relationships between the properties set a foundation for reasoning about the behaviour of objects in a domain.

Functional View

The model layer provides an environment where all model fragments are stored and managed. In order to achieve this there is one prerequisite for the distributed modelling activity and three iterative execution steps. Our approach consists of the following steps.

step 0 – Ontology development. This step is run only the first time when entering into new domain, when the knowledge about that domain is not yet formally described. The abstract objects, their properties and relationships are defined.

step 1 – Properties mapping. Mappings between function and external properties based on particular domain are produced.

step 2 – Collaborative modelling. During this step, models to solve the problem in question are developed, distributed and assessed. While modelling, the developers instantiate (decompose) the abstract concepts from the ontology layer.

step 3 – Model reconciliation. This step deals with model integration, change notification, i.e., the information captured in previous steps is used to reason about the dependencies among model fragments.

Model fragments are at different abstraction levels within the same domain. Therefore, certain concepts in the model fragments may be referred as being sub-class concepts of the concepts in ontology layer. These relationships, depicted as ‘kind_of’, and other formalized relationships among the concepts in the ontology layer are used to reason about the dependency between model fragments.

Functional and External Properties

In order to formalize the relationships, functional and external properties need to be defined for any ontology, in our scenario the oil drilling domain. A functional property of a thing is significant only when the function is used in a relationship with another thing. For example, the load limit of a platform needs to be mentioned only when the platform is expected to support things (constructions) put on it. Usually, external properties constrain the value of a thing’s functional property. An external property of a

thing is present visibly even when the thing stands alone. The length, width, height and weight are external properties of a platform. Thus, a functional property is a property of an entity that denotes the main function of object in a particular UoD. An external property is a property of entity that denotes physically distinguishing features. Both properties are intrinsic properties in the sense of [13].

A contract holds between two things if they have a relationship, where some functional property of one thing and some external property of the other thing are mapped. Functional properties and external properties are mapped under certain conditions, which define a rule in an ontology as follows.

$$\text{Rule : } Func(x) \xrightarrow{map} Ext(y) \quad (1)$$

Recall the oil platform scenario. We demonstrate implementation of our approach in the next section. The external and functional¹ properties and the mappings are explained. We discuss a limited set of concepts, namely, Platform, Oil, Pipe and PipeSystem. The ontology is built in OWL using Protégé 3.0.

Ontology Building and Rules Definition

The ontology layer is used to capture the functional and external properties of entities in a UoD, and to represent the relationship between them. In the ontology fragment depicted in Fig. 2 in UML. Platform and PipeSystem are both defined as subclass of Facility. PipeSystem is composed of Pipe and Oil.

OWL distinguishes two types of properties, datatype property and object property. The former is an attribute of an object. The latter is a relationship between two objects. A datatype property can be regarded both as an external property and a functional property in our approach. For example, a Platform has a support relationship with PipeSystem, a functional load property with Platform and an external weight property of Platform.

An ontology model does not explicitly distinguish between external and functional properties. OWL is used to annotate them. A vocabulary **semAnn** is used to distinguish between them. The following is an OWL representation of the functional load property and the external weight property.

¹ Our definition of functional property is different from the OWL functional property.

```

<owl:DatatypeProperty rdf:ID="load">
  <rdfs:domain rdf:resource="#Platform"/>
  <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  <SemAnn:functional_property rdf:ID="load"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="weight">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Pipe"/>
        <owl:Class rdf:about="#Facility"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  <SemAnn:external_property rdf:ID="weight"/>
</owl:DatatypeProperty>

```

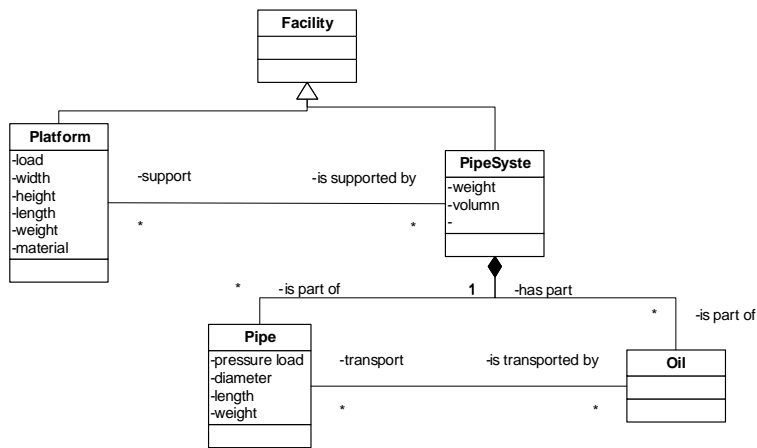


Fig. 2. Ontology fragment

In this domain ontology, we define the relationship between Platform and PipeSystem and assign them the related functional and external properties, followed by an example. Let r be a relationship (r' is a reverse relationship of r), f be a function, EP be a set of external properties, and v be value of a property. Then, fp^f is a functional property of a thing related to the function, ep^f is an external property of a thing constraining the function of another thing and ep_i is an external property. Finally, ct is a contract between two things and cs is a constraint of properties. Example:

```

r(Platform, PipeSystem) = 'support'
r'(PipeSystem, Platform) = 'is_supported_by'

```

```

f(Platform) = 'supporting'
EP(Platform)={epi(Platform) | (i=1,2,...,v)}={Length, Width, Height,
                                             Material, Weight, ...}
fpsupporting(Platform)=Platform.load
epsupporting(PipeSystem)=PipeSystem.weight
fpsupporting(Platform) → epsupporting(PipeSystem)=ct(Platform, PipeSystem)
v(Platform.load) <= v(PipeSystem.weight)
v(Platform.load) = cs(Platform.length, Platform.width, ...)

```

Collaboration and Model Reconciliation

Model fragments in the model layer refer the knowledge that is stored in the ontology layer in order to 1) consolidate concepts, 2) find connection points, and 3) apply constraints when reconciling and integrating the distributed model fragments. Thus, a method for impact prediction and change propagation is needed.

The vocabulary **semAnn** is used to link the model instances to the abstract concepts that are defined on the ontology level. These concepts, including the relations, external and functional properties annotate the corresponding model fragments in local models as follows.

```

<semAnn:concept/relation/property
  rdf:resource="REFERENCE_ONTO:CONCEPT/
  OBJECT_PROPERTY/DATATYPE_PROPERTY" >
  MODEL_FRAGMENT
</semAnn:concept/relation/property>

```

As two distributed models are connected, the relationships between models are built automatically. If one relationship in the model reflects a relationship in the ontology and a functional property is related to that relation, then there must be a functional property in one model and an external property in another. These two properties build an interface for the two models and they are expressed in constraints. If the functional property already is defined in the model and annotated as a property, then it is denoted a functional property. If the functional property is not defined in the model, it can be added referring to the functional property in the ontology. The related function and value of the functional property are annotated using `Rule:function` and `Rule:value`. The square brackets indicate optional statements in the annotation structure below.

```

<semAnn:property rdf:resource="REFERENCE_ONTO:DATATYPE
PROPERTY" >
  [MODEL_FRAGMENT]
  <semAnn:functional_property/external_property>
  <Rule:function rdf:resource="RULE#FUNCTION" />
  [<Rule:value>VALUE</Rule:value>]
  </semAnn:functional_property/external_property>
</semAnn:property>

```

Consider again the oil platform scenario in the OWL representation below. The `platformmodel` is built by an expert on platform engineering, and the `pipesystemmodel` is designed by another engineer. When two models are integrated to a `connectedmodel`, local concepts are aligned with common concepts.

```

<semAnn:property rdf:resource="uri://domonto#length">
  <platformmodel:platform_length id="id2">
    ...
  </platformmodel:platform_length>
<semAnn:external_property>
  <Rule:value>580m</Rule:value>
  <Rule:cs rdf:resource="uri://rule#cs">
</semAnn:external_property>
</semAnn:property>

<semAnn:property rdf:resource="uri://domonto#load">
  <platformmodel:carrying_capacity id="id6">
    ...
  </platformmodel:carrying_capacity >
<semAnn:functional_property>
  <Rule:function
    rdf:resource=" uri://rule#supporting"/>
  <Rule:value>500ton</Rule:value>
  <Rule:cs rdf:resource="uri://rule#CS">
</semAnn:functional_property>
</semAnn:property>

<semAnn:property rdf:resource="uri://domonto#weight">
  <pipesystemmodel:weight id="id9">
    ...
  </pipesystemmodel:weight>
<semAnn:external_property>
  <Rule:function rdf:resource="uri://rule#supporting"/>
  <Rule:value>200ton</Rule:value>
</semAnn:external_property>
</semAnn:property>

<semAnn:relation rdf:resource="uri://domonto#support">
  <connectedmodel:hold id="cid5"/>
  <Rule:function rdf:resource="uri://rule#supporting"/>
  <Rule:ct rdf:resource="uri://rule#CT">
</semAnn:relation>

```

Platform can support PipeSystem; thus the properties of PipeSystem in the model should satisfy the limits of load of platform represented in another model. When an external property, e.g. `length` in the platform model is changed, the functional property `load` will be impacted according to the constraints defined in the rules. Since two models are connected, the contract between two models should be checked. Because the weight of PipeSystem is involved in the contract, the model fragments referring to the PipeSystem concept have to make corresponding changes. The changes can be traced using the annotation information in local models.

Two-Layered Approach Revisited – Semantic Reconciliation

In distributed models concepts as they are used may vary in a way they are denoted and represented. Those same-concepts-in-different-models need to be reconciled according to the ontology when models are to be integrated. Lets assume that context similarity has already been considered during the agreement and identification of the same concepts, as described in [9]. Here, distributed local models adjust their semantics referring to the ontology. The local models – `platformmodel` and `pipesystemmodel` – locate corresponding concepts in ontology. The relationships between the objects in ontology provide a clue for integrating the models. In the ontology of the scenario, OWL object support property is related to the functional load property and connects two objects – `Platform` and `PipeSystem`. That indicates how the `platformmodel` is to be integrated with the `pipesystemmodel`.

The rules, which contain the two functional properties, should be applied during the integration. These rules constrain the changes of models and are also used to check change impact on consolidated models. Three possible impacts are: 1) changes on the functional property may impact other models (e.g., changes on the range of load of `Platform` will impact the maximum weight of `PipeSystem`), 2) changes on external property of one object may impact its functional property (e.g., changes on length of `Platform` will impact its load), and 3) changes on external properties may impact other models (e.g., changes on length of `Platform` will impact maximum weight of `PipeSystem`). The procedure of checking for change of property is as follows.

```

if one property in a local model is changed
{ check the property in ontology level;
  if the property is functional property
    if the functional property is related with
      Object_Property
      { if the functional property is involved in con-
        tract-rules
          check changes on the other object which is
          involved in the contract-rules;
        else
          check changes on the other object which is
          involved in this Object_property;}
    else
      if the property is involved in constraint-rules
        { check changes impacting other properties involved
          in the constraint-rules;}
  }
}

```

Our preliminary prototype is implemented in Python. The main interface window consists of 5 panels: 1) a panel listing of model fragments stored

in a repository; 2) a modelling panel for editing instantiated (related or associated) abstract entities; 3) a panel for ontology browsing; 4) a notification panel for listing changes and their impacts; and in addition 5) a chat panel for discussion between team members.

Related Work

Ontology is commonly defined being an explicit (formal) specification of a conceptualization [1] in the recent literature. Therefore, application of ontologies is in resolving semantic heterogeneity. With respect to the integration of data sources, ontologies can be used for the identification and association of semantically corresponding information concepts [12]. Ontologies are previously used to provide semantic interoperability in information sharing e.g., [2, 3, 12]. The semantics of a resource in a particular domain can be explicitly defined by associating concepts, terms and various information resources with concepts in an ontology.

Further, ontologies can be used as means to abstract from different representation formats and to relate various product fragments at different abstraction levels. Ontologies (domain models) are used in [8] to relate various fragments of system specification to establish dependency relationships for change impact prediction. The end product of system development is seen as a collection of loosely coupled product (specification) fragments from various perspectives that focus on different aspects. The co-ordination of the development process and integration of different product fragments utilizes a common reference layer, i.e. ontology.

An on-going research project [7] is looking at supporting requirements elicitation and composing software from re-usable architectures, frameworks, components and software packages. The use of ontology and its reasoning mechanism helps to maintain semantic consistency. Ontology system there has two layers; one for requirements elicitation and the other for re-usable parts. The ontology system bridges gaps between a requirements specification and an architectural design at a semantic level by establishing relationships between the two layers [7].

An interesting approach is described in [14], where knowledge is organized in knowledge grid. They separate between epistemology and ontology treating both as inseparable profiles of the unified human cognition process. The epistemology mechanism used as a semantic description tool to reflect human subjective cognition. The mechanism helps humans understand and relate their knowledge to the one captured in ontology. Ontology reflects people's consensus on semantics [14].

The work that has been done so far in the area of development and maintenance of ontologies mainly has focused on one ontology, which is edited by the developer. On another hand, there are some tools which allow collaborative ontology creation. For instance, Hozo [10] environment for distributed ontology development is based on splitting ontology into component ontology and establishing dependency between them. The target ontology is obtained by compiling the component ontologies, based on predefined links between them.

In summary, there are different application areas for ontology usage. We find our approach novel as ontologies are used as supervising guidelines during modelling activity. The approach allows checking models under development whether they are semantically correct within particular domain, i.e., how a model corresponds to basic domain knowledge captured in ontology.

Concluding Remarks and Future Works

A vision of a methodological approach to facilitate management of collaborative logically or geographically distributed modelling activities is presented. The approach is based on distinguishing two main layers: an ontology layer; and a model layer. Ontologies are used as a medium for common knowledge representation and as a guide for models reconciliation. The ontology layer contains a set of predefined valid relationships for the creation of situated models. Further, it provides reasoning about relationships between model fragments. We capture two types of object properties in ontology – functional and external property. Relationship between those properties is the foundation for reasoning about the behaviour of objects in a particular domain, e.g., how change of one property influences the change of another. We provide the motivation for our research, discuss the settings and provide conceptual description of the approach followed by scenario that illustrates the applicability of our approach.

There are some remaining challenges to our approach and to the current version of our prototype, however. One is to create an algorithm for automatic update of the models based on both observed changes in the model fragments and on formalized relationships in the ontological layer. Further, description of the rules in a related web-based syntax would be an advantage for the approach as it will allow usage of the same reasoning mechanism as in the ontology layer. The proposal for *Semantic Web Rule Language* (SWRL) [11], whose syntax is based on a combination of OWL DL and the Datalog sublanguage of RuleML, is a good candidate for the further implementation.

References

- [1] Gruber TR (1993) A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, vol 5, no 2, pp 199-220
- [2] Gruber TR (1991) The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases. In: Allen JA, Fikes R, Sandewall E (eds) *Principles of Knowledge Representation and Reasoning*. Morgan Kaufman
- [3] Kashyap V, Sheth A (1994) Semantics-Based Information Brokering. *Proc of the 3rd Intl. Conf on Information and Knowledge Management*
- [4] Moriarty T (2000) The Importance of Names. *The Data Administration Newsletter* 15
- [5] Mylopoulos J (1992) Conceptual Modeling and Telos. In: Loucopoulos, Zicari (eds) *Conceptual Modeling, Databases, and CASE*. Wiley
- [6] Ogden CK, Richards IA (1923) *The Meaning of Meaning*. 8th ed. Harcourt, Brace & World Inc. New York
- [7] Saeki M (2004) Ontology-Based Software Development Techniques. *ERCIM News*, no 58, p 14
- [8] Strasunskas D, Hakkarainen S (2003) Process of Product Fragments Management in Distributed Development. *Proc of the (CoopIS'2003)*, Springer, LNCS2888
- [9] Strasunskas D, Lin Y (2004) Model and Knowledge Management in Distributed Development: Agreement Based Approach. *Proc of the 13th Intl. Conf. on Information Systems Development (ISD'2004) Vilnius Lithuania*
- [10] Sunagawa E, Kozaki K, Kitamura Y, Mizoguchi R (2003) An Environment for Distributed Ontology Development Based on Dependency Management. In: Fensel D et al (eds) LNCS 2870. Springer
- [11] Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosz B, Dean M (2004) SWRL: A Semantic Web Rule Language Combining OWL and RuleML
- [12] Wache H, Vogele T, Visser U, Stuckenschmidt H, Schuster G, Neumann H, Hubner S (2001) Ontology-Based Integration of Information – A Survey of Existing Approaches. In: Stuckenschmidt H (ed) *IJCAI-01 Workshop: Ontologies and Information Sharing*
- [13] Wand Y, Weber R (1995) On the Deep Structure of Information Systems. *Information Systems Journal*, vol 5, pp 203-223
- [14] Zhuge H (2004) China's e-Science Knowledge Grid Environment. *IEEE Intelligent Systems*, vol 19, no 1, pp 13-17