

Modeled Object's Property-based Support for Distributed Model Driven Development^a - a Position Paper

Y. Lin and D. Strasunskas^β

Norwegian Univ. of Science and Technology, Dept. of Computer & Information Science,
NO-7491, Trondheim, Norway
{yun.lin; darijus.strasunskas}@idi.ntnu.no

Abstract. This position paper discusses a vision for methodological support in distributed modeling. Framework has two layers: an ontology layer for representation of basic knowledge about a domain; and a model layer, where models concerning a certain problem within domain are stored and interchanged. Ontology serves as knowledge reference point to define various properties of entities and represent dependency between the objects. Ontology and model fragments associated to the concepts from ontology by sub-class relationship allow to interrelate different model fragments and to assess change impact.

1 Introduction

Conceptual modeling is seen as “the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication” [3], is applied in the early phases of information system analysis and design. However, it is known that different people usually present different models given the same domain and the same problem. The same information about system can be modeled at various levels of abstraction and from different viewpoints considering different aspects. Variations among models generally appear due to the creative nature of the modeling activity, as well as other factors such as the richness of the modeling language [2], the ambiguities of modeling grammars, and others. This problem becomes more evident when the process is distributed. Moreover, during the model development process the variability of the model versions increases due to the highly interactive and iterative nature of the development process and to the different, sometimes conflicting, angles to the problem and solution taken by the different stakeholders.

In this paper, we present our vision on how distributed modeling can be supported. The focus is on support for individual developers and allowing them expressing their views and perceptions of the Universe of Discourse (UoD), which are integrated

^α This research is partially supported by Network of Excellence project INTEROP has been granted within the 6th Framework Programme of the EU (IST-2003-508011).

^β Authors' names are listed alphabetically.

based on a basic knowledge about a domain. Ontology is used as a reference point and is built to share knowledge with other people. In our approach we use ontology to define and formalize basic knowledge about a domain and the main object in the domain. The paper is structured as follows. Section 2 discusses settings for our approach, presents and illustrates the approach using a small example. Finally, section 3 concludes the paper and lays down future work.

2 Distributed Modeling

2.1 Complexity of distributed modeling

Modeling is usually a creative and collaborative process, during which different stakeholders are focusing on various aspects, expressing them at different levels of abstraction, and producing several variants of each.

The success of distributed project depends on how well “laissez-faire” rule is obeyed, meaning that developers should be allowed to express what they want in whatever form. More precisely, [1] emphasized a list of requirements for development environments to enable collaboration in geographically distributed developments. Most relevant rules to our matter are listed below.

- *Unrestricted product object types* – a development environment should allow the developers to share any type of object that they might find useful for supporting their cooperation.
- *Unrestricted relation types* – a development environment should allow the developers to create any type of relation between any two objects.
- *Incremental product refinement* – a product development environment should provide the developers with flexible mechanisms for incrementally refining the product. The developers should be allowed to start with vague products, and to refine them into more complete and formal ones.

In summary, the “laissez-faire” rule in modeling adds to the complexity and variability of different model fragments, and impede model and change management.

2.2 An Approach

In order to relate this variety of model fragments we need to have common reference point. Development of the approach is inspired by one of the linguistics’ methods for describing the meaning of objects - semiotic triangle. Version of the semantic triangle in Fig.1 is inspired by [4].

Ontology is used in order to capture the basic knowledge about 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 knowledge about which entities and what kind of dependency relationships they have in particular UoD. As consequence, our framework has two layers: an ontology layer for representation and formalization

of the basic knowledge about a domain; and a model layer for modeling a problem (solution to the problem) within domain (see fig.2).

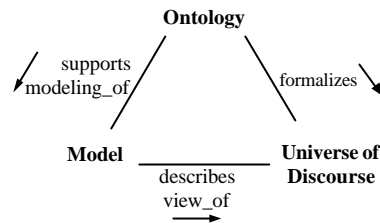


Fig. 1. Functional view of our approach

The advantage of this approach is that it separates the basic (most reusable) knowledge and places this in an ontology layer, whereas a layer of models is positioned below. The ontology layer is composed of a set of concepts representing abstract entities in real world, relations among them based on external and function properties of the concepts. The layer of models provides the environment where all model fragments are stored and managed. 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 formalized relationships among the concepts in the ontology layer are used to reason about the dependency between model fragments.

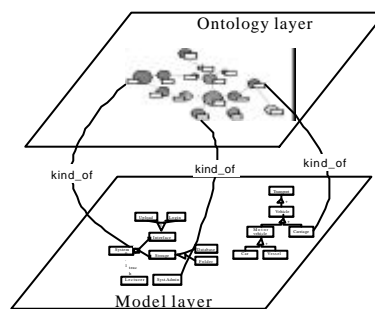


Fig. 2. Functional view of our approach

We capture two types of properties of objects in ontology, i.e., function property and external property. Relationship between those properties is a foundation for reasoning about the behavior of objects in a domain.

2.3 An Example

Our goal is to capture function and external properties of entities in UoD, represent the relationship between them using ontology. Here is the example with ontology built in OWL with Protégé 2.1 beta and constraints containing external properties and function properties.

From the ontology fragment depicted in Fig.3, we can know three kinds of substance are solid, liquid and gas which are disjoint with each other. Substance can be held by container, but open container can not be used to hold gas. Sealed container is disjoint with open container so it can hold any kinds of substances. The container with hole which can not hold liquid is not considered here. Platform can support container and another platform as well.

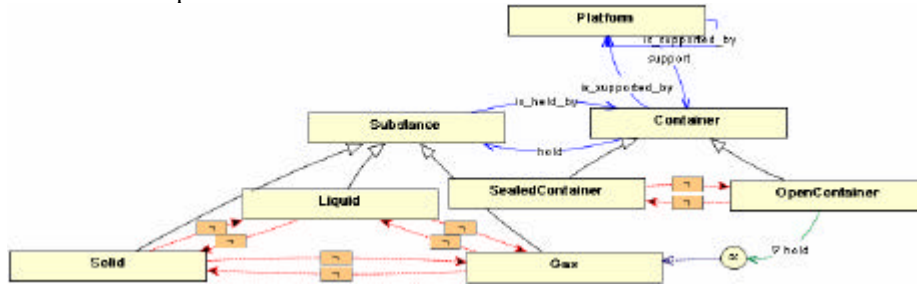


Fig. 3. Ontology fragment built by Protégé 2.1

Two types of properties are distinguished in OWL, namely, datatype property and object property. The latter is actually the relationships between two objects. We do not mean it as our definition of property here. The datatype properties can be external property or function property. The function properties should relate to the object property in OWL as showed in Fig.4 by relationships between classes 'Substance', 'Container' and 'Platform'. For example, holding substance is a kind of function of the container. So, property 'support_capacity' is a function property of a platform. The external properties are datatype properties which are independent with other objects. The values of these properties determine what objects they are. Properties of class platform, substance and container are showed in Fig. 4.

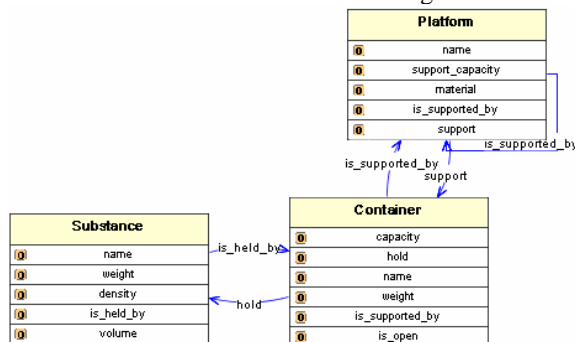


Fig. 4 Relationships and function properties

Fig.4 does not show explicitly which properties are external properties and function properties. We distinguish them by annotations in OWL. The follows are OWL examples of function property 'support_capacity' and external property 'weight'.

```
<owl:DatatypeProperty rdf:ID="support_capacity">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
  <rdfs:domain rdf:resource="#Platform" />
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty" />
```

```

    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      function property</rdfs:comment>
  </owl:DatatypeProperty>

  <owl:FunctionalProperty rdf:ID="weight">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >externalized property
    </rdfs:comment>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Container"/>
          <owl:Class rdf:about="#Substance"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdf:type
    rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  </owl:FunctionalProperty>

```

Note that `owl:FunctionalProperty` is different from our definition of function property. The function properties are relationships between objects, based on the function of the objects. The potential constraints of these relationships are determined by the external properties of the related objects. For instance, a platform can support a container and a container can hold a substance. How much substance a container can hold is decided by the capacity of the container. Whether the platform can support such a container with a certain amount of substance will depend on the support capacity of the platform, the weight of substance, and the weight of container. If the substance is a kind of liquid, the weight of the liquid is determined by its density and its volume in the container. In order to express these constraints, we need to formulize these constraints by describing dependencies between those external properties. Next follows a simple example of one constraint and possible rules to define the dependency between function and external properties of the objects.

1. Container's function property 'capacity' defines container's function to hold a liquid, and relates external properties of those two objects, i.e., capacity of the container and volume of a liquid (eq.1).

$$\text{Liquid.volume} \leq \text{container.capacity} \quad (1)$$

2. Platform's function property 'support_capacity' relates container and platform, so that the constraint involves weight of the container and stored liquid, where weight of the liquid is depends on a liquid's density and volume as well.

$$\text{Platform.support_capacity} \geq \text{container.weight} + \text{liquid.weight}, \quad (2)$$

while

$$\text{Liquid.weight} = \text{liquid.density} \times \text{liquid.volume}$$

Above constraints should be put on the same level as ontology. When modeler builds her system model, all the concepts, relationships and constraints should be referred (instantiated in models). However, constraints like quantity restrictions are not supported by those ontology languages. We can not present them in OWL, so the way to insert these constraints into ontology is putting them as comments of those relative classes or properties.

Model fragments in model layer refer the knowledge stored in the ontology layer in

order to consolidate concepts, find connection points and apply constraints when distributed model fragments are integrated. Another goal of referring the common knowledge is impact prediction and change propagation. For example, a platform model is built by an expert on platform engineering, and the model of the container and the liquid held in the container are designed by another engineering working in a chemical plant. When two models are integrated, local concepts are agreed to referring common concepts. Each object in model layer try to relate itself with the concept presented in the ontology layer. Platform can support container, so the model describing that the container is put on the platform with satisfied support capacity constraints is reasonable. When one changes platform model by changing its external property 'material' which will impact its function support capacity, the concept platform in ontology layer linked to the changed object platform in local model will be detected. Then the changes in support capacity are reflected in ontology layer through constraints comments which are annotated in ontology. Again, because weight of container and its content is involved in the constraints, the changes should impact models which refer the container concept in the ontology level.

3 Concluding Remarks and Future Works

This position paper presented the vision of a methodological approach to facilitate management of distributed modeling activities based on distinguishing two main layers: the basic ontology layer; the model layer. The ontology layer contains a set of predefined valid relationships for the creation of different models and reason about relationship between model fragments. We capture two types of properties of objects in ontology, i.e., function property and external property. Relationship between those properties, i.e., how change of one property type influences the change of another, is a foundation for reasoning about the behavior of objects in a domain.

This paper is a first step towards implementing the environment for collaborative modeling. The challenge is creating an algorithm for automatic update of the models based on observed changes in the local ontology fragments and formalized relationships in the ontological layer. For the extension of the approach it is planned to incorporate input/ output dependencies for the function properties, which would guide modelers in modeling of possible combinations and combination of the model fragments.

References

1. Farshchian B.A. (2001) A Framework for Supporting Shared Interaction in Distributed Product Development Projects. PhD thesis, IDI-NTNU, Trondheim, Norway, 2001.
2. Moriarty T. (2000) The Importance of Names, The Data Administration Newsletter 15.
3. Mylopoulos J. (1992) Conceptual Modeling and Telos. Chapter 2 in: Loucopoulos and Zicari (eds). Conceptual Modeling, Databases, and CASE. Wiley, pp. 49-68.
4. Sølvsberg, A., Bergheim, G. and Sandersen, E. A taxonomy of concepts for the science of information systems. In Falkenberg E.D. and Lingren, P. (eds.), Information System Concepts: An in-depth analysis, pages 269–321. North Holland, 1989.