Software and Systems Modeling (2022) 21:1271–1272
https://doi.org/10.1007/s10270-022-01021-0

EDITORIAL

On the relationship between models and ontologies

Jeff Gray[1] · **Bernhard Rumpe**[2]

Published online: 21 July 2022

The application of ontologies toward various engineering domains has gained much momentum recently, such that it is beneficial to understand the relationship between ontology building and modeling.

Historically (and we quote https://en.wikipedia.org/wiki/Ontology): "Ontology is the branch of philosophy that studies concepts such as existence, being, becoming, and reality" and goes back to Greek Philosophers, such as Parmenides, Plato or Aristoteles. It tries to understand how "entities are grouped into basic categories and which of these entities are fundamental" atoms. Also, "Ontology is sometimes referred to as the science of being." Ontology tries to determine fundamental ontological concepts, like particularity and universality, abstractness and concreteness, or possibility and necessity. When speaking of "an ontology," philosophers refer to a concrete theory within the science of being. There is a large body of approaches and associated publications on ontologies that are deeper than needed for this discussion, but expand on the application of ontologies for many disciplines.

Ontologies are now prominent in several engineering domains, but mainly computer science and especially information science are responsible for building up practically usable approaches and tools for ontology application. In computer science, an ontology encompasses a representation, formal naming, and definition of the categories, properties, and relations between the concepts, data, and entities that substantiate one, many, or all domains of discourse. More simply, an ontology is a way of showing the properties of a subject area and how they are related, by defining a set of concepts and categories that represent the subject (Wikipedia).

Even though the terms and names used are quite different, ontologies can be compared to modeling techniques, such as UML class diagrams, where classes and associations represent entities and relationships. Even hierarchical ontological decomposition seems to be well-represented in class diagrams. Of course, there are subtle differences. Furthermore, some constructs that class diagrams offer, such as multiplicities, are not well-represented in the typical ontology approaches, but also the idea of rule-based derivation of knowledge is not present in class diagrams. A logic-based extension is needed, such as UML's object constraint language (OCL), to represent such rules.

However, a slight difference appears in the form of usage. Although class diagrams originally were used to structure a domain of data for a software system under development, ontologies were often used to organize data into information and knowledge in an academic discipline or field. Ontologies are used for problem solving within such a domain. Interestingly, with a slight difference in their intended meaning, it seems that these approaches largely overlap. It is therefore not so surprising that several comparison papers have been published, sometimes only explaining how to encode UML in XML or RDF, but also tools have been developed that try to unify the more data-oriented viewpoint of class diagrams with the ontological viewpoint.

The UML (which not only contains class diagrams, but also provides a set of behavior modeling techniques, a logic language, and a precisely standardized syntax) is easily able to cover the language constructs that typical ontology approaches provide. Most engineers describe designs that contain entities and decomposition, obviously reflecting the typical structural decomposition of mechanical, biological and other kinds of complex systems, but neither use other relationships or ontology rules to build further knowledge. However, they often are not able to describe behavior or the dynamic changes of the structure.

Thus, an interesting question to ask is, "Why are ontologies sometimes (or increasingly) preferred over a full modeling language?" If the answer is tied to the potential simplicity of the underlying ontology language, would it not be better if we had modeling tools that come with "beginner modes," where the model only consists of typical ontology language

✉ Bernhard Rumpe
bernhard.rumpe@sosym.org

Jeff Gray
jeff.gray@sosym.org

[1] University of Alabama, Tuscaloosa, AL, USA

[2] RWTH Aachen University, Aachen, Germany

constructs? And it seems that there is also the possibility to close the gap between these two approaches, namely ontology building and modeling, so that they do not compete with each other in a largely overlapping purpose, but rather merge their complementary benefits.

As an aside, for all who are deeply interested in understanding how ontologies are used in science, we recommend the paper by Brian Henderson-Sellers on "Bridging metamodels and ontologies in software engineering" in the *Journal of Systems and Software* (2011), where he clearly worked out how the two forms of uses for ontologies, namely describing a concrete (data) structure and describing actually the meta-model of a (data) structure, can be distinguished from each other.

1 Content of this issue

1. **Theme Section on Open Environmental Software Systems Modeling**
   Guest Editors: Tao Yue, Paolo Arcaini, Ji Wu, and Xiaowei Huang
2. **PoEM 2020 Special Section**
   Guest Editors: Janis Grabis and Dominik Bork
3. **Theme Section on Agile Model-Driven Engineering**
   Guest Editors: Kevin Lano, Shekoufeh Kolahdouz-Rahimi, Javier Troya, and Hessa Alfraihi
4. **Regular Papers**
   - "Are models better read on paper or on screen? A comparative study" by Mohamed El-Attar
   - "SOCAM: A service-oriented computing architecture modeling method" by Paola Reyes-Delgado, Hector Duran-Limon, Manuel Mora, and Laura Rodriguez-Martinez
   - "Model-driven development of asynchronous message-driven architectures with AsyncAPI" by Abel Gómez, Markel Iglesias-Urkia, Lorea Belategi, Xabier Mendialdua, and Jordi Cabot
   - "GoRIM: A model-driven method for enhancing regulatory intelligence" by Daniel Amyot, Okhaide Akhigbe, Gregory Richards, and Lysanne Lessard
   - "Using two case studies to explore the applicability of VIATRA for the model-driven engineering of mechatronic production systems" by Gennadiy Koltun and Mathis Pundel
   - "Utilizing multi-level concepts for multi-phase modelling–Context-awareness and process-based constraints to enable model evolution" by Tobias Franz, Christoph Seidl, Philipp M. Fischer, and Andreas Gerndt

Funding Open Access funding enabled and organized by Projekt DEAL.