Integrating System Design Information Using a Self-Defining Ontology

Stephen Hookway[1,*], William Norsworthy, Jr.[1]

[1]Charles River Analytics, 625 Mt. Auburn St., Cambridge, MA, United States

Abstract

Engineers use a variety of software tools to support system design and development. These tools help engineers encode and reason about complex requirements and designs, but they also create data silos of information related to the components and systems they are designing. We describe a self-defining ontology approach that automatically extracts an ontological representation of system design information from tool-specific design data to integrate information across tools and to validate system designs for faster error-free design and development of new systems.

Keywords

self-defining ontology, model-based systems engineering (MBSE), system design and validation

1. Introduction

Using a Web Ontology Language (OWL) ontology to capture information generated during the design life cycle of parts and systems enables the alignment of concepts between digital engineering tools based on meaning. An off-the-shelf reasoner, such as Pellet [1], can reason across information in the knowledge base to identify any logical inconsistencies, even if they are inferred.

2. Problem Definition

Encoding information using an ontology provides both the means to perform data integration and the ability to use the semantic representation to reason about the information that has been encoded. Unfortunately, a major challenge of most knowledge representation efforts involves identifying an existing ontology or creating a new ontology that suitably captures the semantics of the domain. High-level ontologies like the Basic Formal Ontology (BFO) [2] can be used to abstract domain concepts, making it possible to easily encode information. The trade-off is that high-level ontologies require external logic to process the information encoded by the ontology. This external logic becomes extremely tricky to write, maintain, and troubleshoot as the ontology grows more complex.

Alternatively, low-level domain ontologies encode equivalent information directly as properties on the classes they are describing. This enables an automated reasoner, such as Pellet, to detect and enforce any logical inconsistencies even if they are inferred. The challenge with creating domain-level ontologies is that it is often difficult for knowledge engineers to identify upfront which domain concepts should be defined, at which level, and with what relationships. Creating a new ontology takes a significant amount of effort and the ontology must be continually updated to ensure it can encode all concepts in the domain.

3. Solution

We devised a novel technique for "self-defining" ontologies that uses the system design data that is being encoded to automatically drive the creation of a domain ontology that encodes the right concepts,

Posters, Demos, and Industry Tracks at ISWC 2024, November 13–15, 2024, Baltimore, USA

*Corresponding author.

shookway@cra.com (S. Hookway); wnorsworthy@cra.com (W. N. Jr.)

at the right level, and with the right relationships to reason about the system design and to detect inconsistencies between requirements and design.

An ontology's TBox defines the classes, properties, relationships, and constraints of the domain. A key insight is that the system requirements defined by engineers in tools like IBM's Dynamic Object Oriented Requirements System (DOORS) provide the TBox information for the domain. The requirements define parts and systems, show how they relate, and provide constraints on properties. This naturally fits OWL's definition of classes, properties, and property constraints (via class expressions or Shapes Constraint Language (SHACL) constraints).

Consider the following example requirement: "The backup battery shall have a runtime of greater than 60 minutes." This information can be encoded as a class expression by defining a BackupBattery class with a runtime property that is restricted to values greater than 60 minutes. An OWL reasoner is then able to enforce this constraint and identify even indirect inconsistences where designs of different types of backup batteries do not meet this constraint. Figure 1 shows an example of how this information is encoded as an OWL class expression.

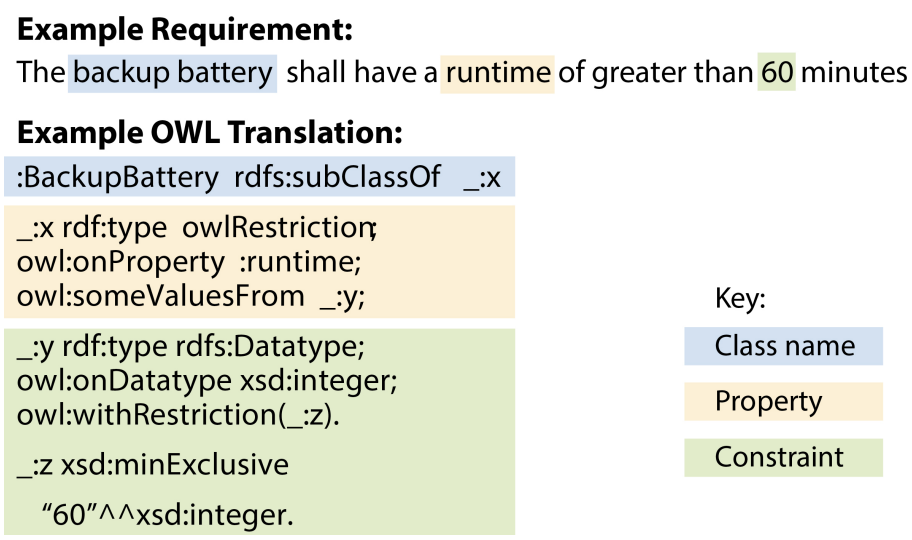

Figure 1: Example translation of a text-based requirement to an OWL class expression.

While the TBox provides the ontology's "schema," the ontology's ABox contains the instance data that is represented using that schema. This instance data naturally maps to designs of parts/systems meant to meet the requirements. For example, the design of a "Backup Battery," which has a default runtime associated with it, is meant to satisfy the runtime originally defined in the Backup Battery requirements in DOORS (and encoded in the TBox).

Systems engineers use tools like Dassault Systemes' Cameo Systems Modeler (CSM) during system design. CSM uses the Systems Modeling Language (SysML) to serialize design data. Figure 2 shows an example translation of Cameo SysML data into an OWL Named Individual.

4. Conclusion

Using domain ontologies offers several advantages over higher-level ontologies including the ability to automatically reason about consistency and perform integration of requirements and design. Our strategy for self-defining ontologies enables domain-level ontologies to automatically be created from the data they describe and enables integrated digital engineering tools to share information and enforce consistency across tools for faster error-free design and development of new systems. We have demonstrated this technique using proof-of-concept translation techniques for natural language requirements (e.g., systemic functional grammars for information extraction) and automated translation techniques based on the SysML standard.

Example XMI (SysML) for block diagram:

```
<packagedElement xmi:type='uml:Class'
xmi:id='_2021x_2_f890376_1667327651971_385076_2809' name='backupBattery'>
      <ownedAttribute xmi:type='uml:Property'
      xmi:id='_2021x_2_f890376_1667327661312_442299_2838' name='runtime'
      aggregation='composite'>
            <type
            href='http://www.omg.org/spec/SysML/20181001/SysML.xmi#SysML_dataType.Integer'>
                  <xmi:Extension extender='MagicDraw UML 2021x'>
                        <referenceExtension
                        referentPath='SysML::Libraries::PrimitiveValueTypes::Integer'
                        referentType='DataType'
                        originalID='_16_5_1_12c903cb_1245415335546_8641_4088'/>
                  </xmi:Extension>
            </type>
            <defaultValue xmi:type='uml:LiteralInteger'
            xmi:id='_2021x_2_f890376_1667328557054_138481_2879' value='90'/>
      </ownedAttribute>
</packagedElement>
```

Example OWL Translation (instance of Backup Battery design with runtime property):

```
:backupBattery_2021X…2809 rdf:type owl:NamedIndividual, :BackupBattery;
    :runtime "90"^^xsd:integer.
```

Figure 2: Example translation of SysML exported from Cameo to an instance of an OWL class. Blue represents the instance name, tan the property, and green the property value

Future work will explore the use of large language models (LLMs) for data translation using the same self-defining strategy for the target representation.

5. Acknowledgments

This material is based upon work supported by the Strategic Systems Programs (SSP) under Contract No. N64267-24-C-0011. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of SSP.

6. References

[1] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical OWL-DL reasoner, Journal of Web Semantics 5 (2007), 51–53. doi:10.1016/j.websem.2007.03.004.

[2] R. Arp, B. Smith, A. D. Spear, Building Ontologies with Basic Formal Ontology, MIT Press, Cambridge, MA, 2015.