WHY SEMANTICS MATTER: A DEMONSTRATION ON KNOWLEDGE-BASED CONTROL SYSTEM DESIGN

W. Pessemier, G. Raskin, and H. Van Winckel, Institute of Astronomy, KU Leuven, Leuven, Belgium
P. Saey and G. Deconinck, ESAT, KU Leuven, Leuven, Belgium

Abstract

Knowledge representation and reasoning are hot topics in academics and industry today, as they are enabling technologies for building more complex and intelligent future systems. At the Mercator Telescope, we've built a software framework based on these technologies to support the design of our control systems. At the heart of the framework is a metamodel: a set of ontologies based on the formal semantics of the Web Ontology Language (OWL), to provide meaningful reusable building blocks. Those building blocks are instantiated in the models of our control systems, via a Domain Specific Language (DSL). The metamodels and models jointly form a knowledge base, i.e. an integrated model that can be viewed from different perspectives, or processed by an inference engine for model verification purposes. In this paper we present a tool called OntoManager, which demonstrates the added value of semantic modeling to the engineering process. By querying the integrated model, our web-based tool is able to generate systems engineering views, verification test reports, graphical software models, PLCopen compliant software code, Python client-side code, and much more, in a user-friendly way.

INTRODUCTION

Semantic models consist of pieces of information, and the relationships between those pieces. The ability to link any piece of information with another, thereby conveying the meaning (semantics) of the information, is what sets them apart from more "rigid" models such as those found in relational databases or object-oriented software. Semantic models are therefore well suited to represent all sorts of knowledge about the real world. At the Belgian Mercator Telescope (La Palma, Spain) we use the expressive power of semantic models to capture engineering knowledge of the telescope control system, which is being ported to a Programmable Logic Controller (PLC). As will be demonstrated in this paper, we have developed systems, electrical and software engineering models of several subsystems of the telescope, and successfully used these models for documentation, verification and implementation purposes.

FRAMEWORK ARCHITECTURE

As shown in Fig. 1, we have built a software framework centered around a Knowledge Base (KB) that combines (integrates) metamodels and models [1]. This KB can be queried by a tool which we developed (OntoManager), and the results of those queries can be fed into a template system to produce documents such as web pages and source code files. The metamodels provide the building blocks to construct models.

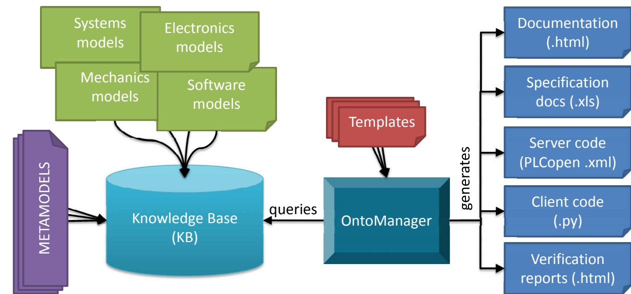

Figure 1: Framework architecture.

They are true *ontologies* as they define a vocabulary in such a way that the meaning of this vocabulary is well defined. Fig. 2 shows small excerpts of our systems metamodel (sys), mechanics metamodel (mech) and electronics metamodel (elec). They define concepts ("classes") such as `sys:Feature` and `mech:Assembly`, and relationships ("properties") such as `sys:hasFeature` and `elec:isConnectedTo`. Unlike a simple vocabulary, the meaning of these terms is further constrained whenever possible. For instance, the definition of a `mech:Assembly` says that "something" is a mechanical assembly if and only if it has at least two mechanical parts. Models based on this vocabulary will have to adhere to these constraints in order to be valid. For the underlying formal semantics (SubClassOf, EquivalentTo, Domain, ...) we depend on the Semantic Web standards RDFS (Resource Description Framework Schema) and OWL (Web Ontology Language) [2].

Figure 2: Small excerpts of some metamodels.

Pre-Press Release 23-Oct-2015 11:00

Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-148-9

Models that use the metamodel vocabulary consist of **explicit** statements such as "x rdf:type elec:Contact" (x is an electric contact) and "x elec:isConnectedTo y" (x is electrically connected to y). However, according to the semantics of the metamodel (see Fig. 2), many more **implicit** statements may be inferred. For example, the formerly mentioned statement implicitely leads to "x rdf:type elec:Conductor" (because elec:Contact is a subclass of elec:Conductor) while the latter statement implicitely leads to "y elec:isConnectedTo x" (because the elec:isConnectedTo relationship is symmetric according to the metamodel). A so-called *reasoner* or *inference engine* (SPIN API[1] in our case) can automatically produce these inferred statements and add them to the KB. Similarly, supposing two contacts are connected to each other, one of them carrying a 24V signal and the other one a 4...20mA signal, then the constraint pattern of the elec:Conductor will match, and a constraint violation will be generated by the reasoner and added to the KB. The metamodels, models and inferred knowledge can thus all be represented as lists of statements, which can be stored in text files. In our framework we parse those files using the open source RDFLib[2] Python package. While systems can be modeled "directly" as a list of statements, real-world systems will unavoidably result in many thousands of explicit statements (and a multitude of implicit ones). We therefore need a more efficient way to populate the models, as explained in the next section.

POPULATING THE KB

To develop models based on the metamodel vocabulary in a convenient way, we need a modeling language. Graphical languages such as UML and SysML are an option (as they can be extended with stereotypes), but in our experience, complex models of real-world systems require more syntax than these languages typically offer. We have therefore developed a Domain Specific Language (DSL) called Ontoscript. Ontoscript is an "internal" DSL as it is valid coffeescript[3], only it is used in a particular way. We have adopted this idea from the Giant Magellan Telescope project [3]. An example of an Ontoscript model is displayed in Fig. 3. It shows how an instance of an I/O module of type "EL1088" is added to a project, and its terminals are connected to the pins of a connector. When this script is executed, then the IO_MODULE_INSTANCE function is called, producing hundreds of statements. For instance, for each channel and each terminal of the "EL1088" I/O module **type**, a corresponding channel and terminal will be added automatically to the I/O module **instance**. These newly created terminal instances can then be further described and connected (via the elec:isConnectedTo relationship) to the pins of some previously modeled connector. This idea is very similar to object-oriented software, because when a class is instantiated, then also all variables (and sub-variables) of the class definition must be added to the instance. In our framework this functionality is programmed by Ontoscript functions such as IO_MODULE_INSTANCE, CLASS_INSTANCE, etc. When the Ontoscript models are executed, they produce text files containing thousands of statements. When parsing these text files using RDFLib, we have a KB which we can start to query.

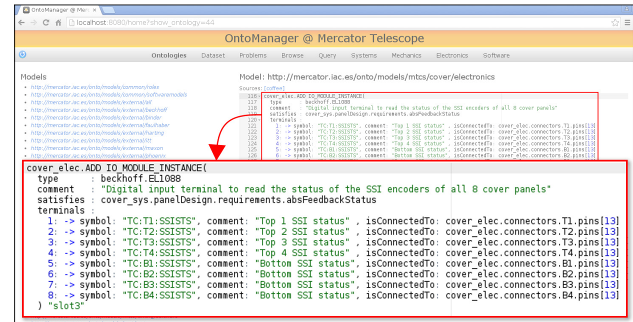

Figure 3: I/O module instance modeled in Ontoscript.

QUERYING THE KB

Using the "Query" tab of our OntoManager tool, we can submit arbitrary queries to the KB (which is essentially an RDFLib instance). The de-facto query language of the Semantic Web (called SPARQL[4]) uses pattern matching and filters to select the requested information. Such a SPARQL query executed by the OntoManager tool is shown in Fig. 4. The query selects all I/O module types in the KB, their manufacturer and their description, and counts their instances. By looking at the query, one can see that this kind of information can be retrieved with only knowing the metamodel and some simple SPARQL syntax. SPARQL also supports more advanced queries (e.g. including numerical calculations or comparison), which are often used for verification purposes.

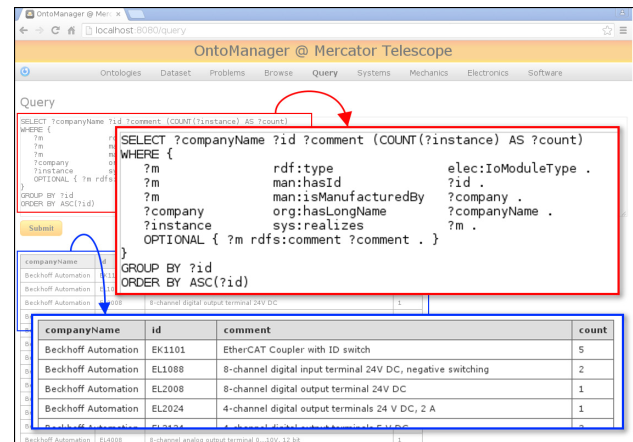

Figure 4: Arbitrary query executed in OntoManager (red) and its results (blue).

[1] http://topbraid.org/spin/api
[2] http://github.com/RDFLib
[3] http://coffeescript.org
[4] http://www.w3.org/TR/sparql11-overview

Pre-Press Release 23-Oct-2015 11:00
Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-148-9

VIEWING THE KB

While the Query tab of OntoManager can be used to extract some specific information from the KB, we can also predefine some queries and a suitable template as a so-called "view". Several of these views have been integrated in OntoManager so far, as can be seen in Fig. 5. The main part of the figure shows a web-based view of the I/O module instance "slot3" of the "Cover" electrical configuration. The latter is part of the dust cover subsystem of the Mercator Telescope, which seals the telescope tube during daytime via eight pivoting aluminum panels (similar to the petals of a flower). Dozens of queries have been executed in the background to produce this "electronics" view, e.g. to retrieve the system properties, the channel and terminal properties, the electric connections to the connectors of the cabinet, and so on. The results of these queries are fed into a web-based template (written in mako[5] syntax) hosted by a web server (based on the Pyramid[6] framework). We have developed very similar views for other frequently used electrical components such as connectors and motor drives. All views are linked to each other via the semantic model: for instance, we can see that terminal 1 of the shown I/O module instance is connected to pin 13 of the connector instance `T1`. By clicking the `T1` hyperlink, the connector view will be shown, offering the connector's pin layout and other details. Thanks to a predefined template for viewing I/O modules, the Ontoscript model `slot3` of Fig. 3 can thus be displayed as a nicely formatted and "clickable" web page without any additional effort. The resulting web pages document the electrical system and may be used as specification documents (for drawing the electrical schemas) or for troubleshooting purposes. Due to the many visible hyperlinks, it is very easy to browse the documentation, thereby navigating trough the electrical system from one component to another.

In a similar way, also information of other engineering disciplines may be linked to the electrical information via the underlying semantic model. For instance, in Fig. 5 the shown I/O module instance satisfies a requirement, which can be inspected by clicking the corresponding hyperlink (arrow 1). The resulting web page shows the properties of the `absFeedbackStatus` requirement, from which the `panelDesign` systems engineering view can be opened (arrow 2). The I/O module is also linked to software models via its interface: see arrow 3. The `SM_CoverPanel` PLC function block has been fully modeled using the Ontoscript DSL and can therefore be illustrated in the web browser. It should be noted that a large fraction of the `SM_CoverPanel` model is the result of a model transformation implemented in the Ontoscript DSL. For instance, in this model we only specified some essential information about the `startOpening` process. When running the DSL script, the model is expanded automatically with additional software features (methods, variables, struct definitions) and even an implementation. As can be seen on the figure, every variable of this implementation can be clicked. In other words: the model is aware of each individual variable and each individual operation of this implementation. Running the ontoscript models thus generated a **model** of the implementation, not just some **text**. One of the advantages is that we can use the same model to generate source code instead of web pages. Using the library view (arrow 4) we can convert the model into source code that can be loaded in a commercial PLC programming environment (arrow 5) or into a Python project (arrow 6). The generated Python source code contains an OPC UA (OPC Unified Architecture) information model based on our in-house developed Unified Architecture Framework (UAF[7]). One can see that with just three Python instructions, we are able to read any variable exposed by the main PLC of the Mercator Telescope.

[5] `http://www.makotemplates.org`

[6] `http://www.pylonsproject.org`

CONCLUSIONS

Several subsystems of the Mercator Telescope have been developed using the presented tools, and are in operation. The models (and the systems they represent) currently consist of 55 I/O module instances, 159 PLC function block definitions, and 626 PLC function block instances. Based on these experiences, we attempt to answer the initial question: why semantics matter?

1. *Because any piece of information is just one query "away"*. Semantic models expose the relationships between all kinds of (electrical, software, systems, ...) information and thereby facilitate organizing, integrating, browsing and finding (querying) information.

2. *Because well defined semantics allow verification.* Semantic models allow "implicit" information to be inferred from the explicit information captured by the models, thereby allowing constraints verification.

3. *Because the same semantics may be used for building future "smart" systems.* Semantic modeling languages are a key enabling technology for future intelligent systems. One can imagine a newly developed astronomical instrument being able to expose its capabilities and to "learn" about the capabilities and services of its environment (e.g. observatory or lab), in order to automate integration or even collaboration with "fellow" instruments.

REFERENCES

[1] W. Pessemier et al., "A practical approach to ontology-enabled control systems for astronomical instrumentation", Proc. ICALEPCS 2013, San Francisco, October 2013, TUCOCB03 (2013).

[2] D. Allemang and J. Hendler, *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*, 2nd Ed., (Waltham, MA: Morgan Kaufmann Publishers, 2011).

[3] J. M. Filgueira, "GMT software and controls overview", Proc. SPIE 8451, Amsterdam, July 2012, 845111 (2012).

[7] `http://github.com/uaf/uaf`

ISBN 978-3-95450-148-9

Pre-Press Release 23-Oct-2015 11:00

Copyright © 2015 CC-BY-3.0 and by the respective authors

Pre-Press Release 23-Oct-2015 11:00

Copyright © 2015 CC-BY-3.0 and by the respective authors

Figure 5: Example of some views, all linked to the I/O module instance “slot3”.

ISBN 978-3-95450-148-9