Semantic Support for DSL Demonstrator in an Additive Manufacturing Environment

Fathia Bettahar
Capgemini Engineering
Toulouse, France
e-mail : fathia.bettahar@capgemini.com

Abstract— **In this work, we describe a Domain-Specific Language (DSL) demonstrator that controls the interaction with the user in an additive manufacturing environment. Its objective is to give appropriate answers to queries asked by a user. Several modules of the environment are used to bring theses answers: the DSL grammar, which lets users query the components of a complicated system without having to learn an unfamiliar query language; the user interface, which can be adaptable according to user needs; and the graph database management system developed to parse the knowledge base. All the modules share a central ontology describing the additive manufacturing domain. Our ontology is used both for building a formal grammar and as knowledge base, providing concept definitions.**

Keywords-Ontology; Domain-Specific Language; Additive Manufacturing.

I. Introduction

The development of complex software systems within multidisciplinary sub-teams raises new software engineering challenges. It is important to understand complex problems and to give appropriate solutions. Model-Driven Engineering (MDE) [1] is the approach proposed to solve these challenges. It raises the level of abstraction in traditional programming languages by using models. MDE focuses on the use of models to understand, analyze, and develop complex software system. MDE advocates the description of a system by a set of multiple Domain-Specific Languages (DSLs) [2].

DSLs, as opposed to General Programing Languages (GPLs), are designed to describe a program at the adequate level of abstraction. DSLs narrow the gap between a problem domain and its implementation. Tools and collaboration of DSLs are among important research challenges [3] in model-driving engineering. Particular challenges include tools for defining and composing domain-specific languages. Usually, different domain specific languages are used simultaneously to define the system from several viewpoints, hence the necessity of semantic interoperability and collaboration between them.

In this work, we propose an approach based on ontology able to define DSL and which aims at a semantic interoperability between DSLs in a complex software system. The objective of this approach is to foster communication and collaboration among team members (developers, domain experts and end-users). This objective can be achieved since ontologies give a common semantic representation of the domain modeled, which can be shared by DSLs in Model-Driven Software Engineering (MDSE).

In this paper, we propose a method to build a DSL in the context of the additive manufacturing domain. We define a Domain Specific Language describing the user Request (DSLReq) from Additive Manufacturing Ontology (AMO). *The focus of this paper is to present a semantic support DSL demonstrator in Additive Manufacturing (AM) domain.* Thus, our approach produces a grammar useful to parse an additive manufacturing knowledge base represented in a semantic graph.

The rest of the paper is organized as follows: In Section II, we present the research work related to our approach. In Section III, we present our grammar, its functionality and introduce its rules in part B. We also discuss the role of ontology in our approach in part A. In part C, we give a scenario example to explain our developed system. The paper ends with some concluding remarks and some guidelines for future work.

II. Related Work

To understand the ontology role, we can refer to the ontology definition of Gruber [4] "explicit specification of conceptualization", where conceptualization is "a set of objects, which an observer thinks exist in the world of interest and relations between them". So, ontology O related to a domain D is a set of the concepts ® and relations ® among them as in (1).

$$O_D = \underset{C \in D}{\{} C, R\} \qquad (1)$$

Each ontology concept is defined by some properties and axioms. This definition makes ontologies very expressive and powerful means for domain modeling, and they are proposed as one of the approaches to create Domain Specific Languages (DSLs).

Many research works use ontologies to support the development of DSLs. Ontologies help in the initial phase of DSL development called Domain Analysis Phase [5]. In this phase, the ontology defines the domain, its terminology, the concepts, and their dependencies.

In [6], the authors proposed the OntoDSL tool that allows to use ontologies during the design phase and to guide the DSL programmer to develop an expressive language. OntoDSL provides automated reasoning services that can be practically used by DSL designers and DSL users. In other works, researchers emphasize the advantages of using ontologies to support DSLs development. For that reason, [7] proposes an Onto2Gra framework to automatically generate the DSL grammar from an ontology. This framework provides the mapping of concepts into grammar symbols based on Context Free Grammars (CFG), and the mapping of relations into grammar productions. These approaches used ontologies to solve DSL challenges, in particular interoperability with other languages, because more than one language must be combined in the modeling of systems.

Ontologies have proved themselves in various domains as a tool for adding semantic to data and for achieving a desirable level of interoperability [6]. For that reason, we believe that ontology is the solution of both the interaction problem between heterogenous components within a complex computer system and the interaction problem between DSLs.

Our approach has the same goals as OntoDSL because it is a framework in a model-driven approach. Differently from OntoDSL our approach aims to build a DSL from an ontology able to parse the knowledge base. The result of our approach is a grammar describing a user request to get a response from the ontology and the AM domain without having to learn an unfamiliar query language.

III. Principle

In this work, we present our approach in additive manufacturing context. The development and industrial adoption of Additive Manufacturing (AM) technologies required the development of computer systems able to aid decision making and the data management in a homogenous way. Ontologies can answer these requirements because they have competencies [7] to both exchange data across applications and automatically reason over knowledge for decision making.

Besides, additive manufacturing systems manifest a low degree of interoperability, and this creates an interaction problem between enterprises or different branches of an enterprise. Ontologies are valuable tools for solving such problems [8]. Given that, an ontology builds on classes to represent materials, products components, process's parameters, and process's parts [9].

Our approach is based on two pillars. The first pillar is a domain specific language that allows understanding, validation, and modification of the additive manufacturing domain. The second pillar is an ontology used both for the construction of a formal grammar and knowledge base, providing concept definitions.

For that purpose, we define a framework based on ontology. This framework is structured around three interconnected models, as illustrated in Figure 1. AM ontology defines an additive manufacturing domain, a user interface, which generates a Cypher query [10] from our DSL grammar and a semantic graph database based on AM ontology. We have generated the DSLReq grammar from our ontology.

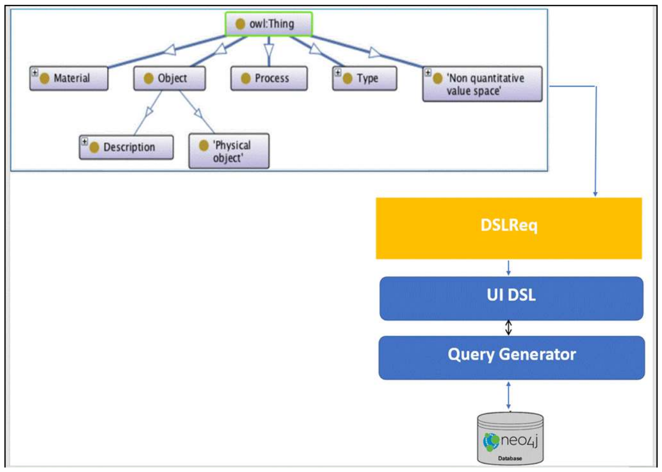


Figure 1. Approach Architecture

In this section, we discuss firstly the ontology building method. Secondly, we explain the process of DSLReq generation and the role of ontology in this process.

A. *Additive Manufacturing Ontology (AMO)*

The methodology of developing an ontology is chosen according to the complexity of the domain and its requirements. For the manufacturing domain, the method MOP (Machine of a Process) [11] is used. MOP aims to reuse existent ontology to limit the development from scratch. Upper-level ontology for manufacturing was used as a reference to evaluate and to improve specific domain ontology. Thus, this method aims to hold interoperability among domain users. In order to achieve a high-level of interoperability, we developed our additive manufacturing ontology according to the research work in [9].

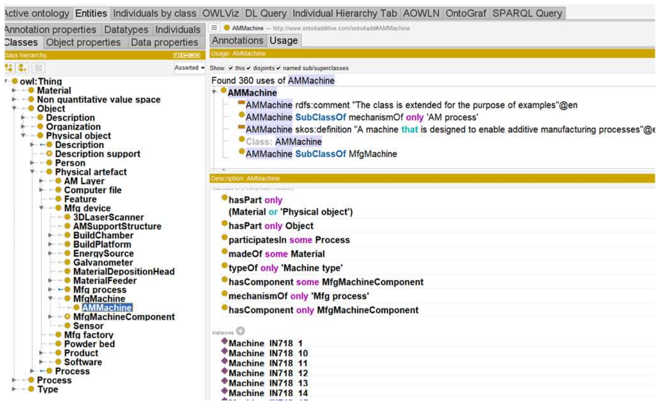
[9] proposes first the reuse of upper-level ontology DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [12] to facilitate the grouping of classes sharing common high-level characteristics. Second, it proposes the extension of the upper-level branch to additive manufacturing.

The AM ontology includes broad classes and relations, which can be easily specialized to meet specific modeling scenarios and requirements: MATERIAL, NON-QUANTITATIVE VALUE SPACE, OBJECT, PROCESS and TYPE. OBJECT covers three classes: PYSICAL OBJECT, ORGANIZATION, and DESCRIPTION. PYSICAL OBJECT covers various classes, among which MFGDEVICE, PRODUCT, and FEATURE. MFGDEVICE is composed of various classes, the most important one being MFGMACHINE for the conceptualization of manufacturing machines. AMMACHINE defines an explicit model for additive manufacturing machines. AMMACHINE has a relationship with the AMPROCESS class called MecanismOf. This relationship gives all processes in which a machine can participate.

Axioms are added to explain that a machine may never be employed in any manufacturing process, for example, as defined in (2):

$$\text{MFGMACHINE} \sqsubseteq \text{MFGDEVICE} \sqcap \forall \text{mecanismOf.MFGPROCESS}$$
$$\text{AMMACHINE} \sqsubseteq \text{MFGMACHINE} \sqcap \forall \text{mecanismOf.AMPROCESS} \tag{2}$$

Figure 2 illustrates the AMMACHINE concept and its properties in our AM ontology created with Protégé [13].



```
owl:Thing
  Material
  Non quantitative value space
  Object
    Description
    Organization
    Physical object
      Description
      Description support
      Person
      Physical artefact
        AM Layer
        Computer file
        Feature
        Mfg device
          3DLaserScanner
          AMSupportStructure
          BuildChamber
          BuildPlatform
          EnergySource
          Galvanometer
          MaterialDepositionHead
          MaterialFeeder
          Mfg process
          MfgMachine
            AMMachine
          MfgMachineComponent
          Sensor
        Mfg factory
        Powder bed
        Product
        Software
      Process
  Process
  Type
```

```
AMMachine — http://www.onto4additive.com/onto4add#AMMachine

Found 360 uses of AMMachine
AMMachine
    AMMachine rdfs:comment "The class is extended for the purpose of examples"@en
    AMMachine SubClassOf mechanismOf only 'AM process'
    AMMachine skos:definition "A machine that is designed to enable additive manufacturing processes"@e
    Class: AMMachine
    AMMachine SubClassOf MfgMachine

hasPart only
(Material or 'Physical object')
hasPart only Object
participatesIn some Process
madeOf some Material
typeOf only 'Machine type'
hasComponent some MfgMachineComponent
mechanismOf only 'Mfg process'
hasComponent only MfgMachineComponent

Machine IN718 1
Machine IN718 10
Machine IN718 11
Machine IN718 12
Machine IN718 13
Machine IN718 14
```

Figure 2. AMMachine Class in AMO ontology

The ontology supports the instances of concepts and relations between them. We use the Protégé plug-in to populate our ontology.

B. *From ontology to DSL grammar*

Within our system, grammar advocates the description of user queries by a set of domain-specific language rules. Each rule consists of an ontology element: concept, relation, or attribute. Our approach is to build a DSL grammar able to generate a Cypher request to parse AMO. For that purpose, we define the request structure according to the ontology structure.

To convert the ontology into a grammar, we define a set of translation rules. The rules are defined in a hierarchical structure. *Get* is the super rule that connects all rules. It is composed of three sub-rules: *Type, Where* and *Return*. Firstly, the *Type* rule is the result of rule translation to map concepts and relations into the grammar. Secondly, the *Where* rule defines constraints of cardinality like RDFS range and RDFS domain within the grammar. Finally, the *Return* rule translates ontology instances corresponding to the request result (Figure 3).

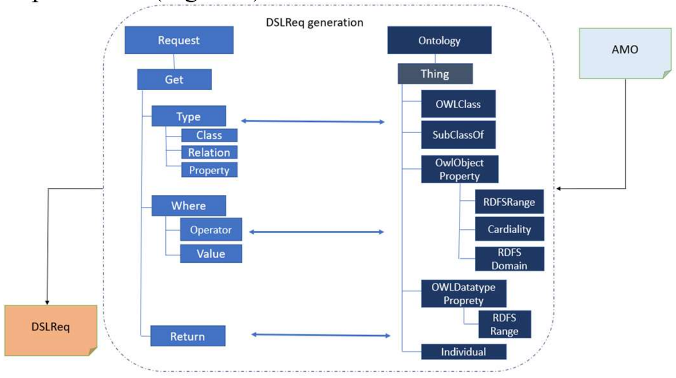


Figure 3. From ontology to DSL

Each rule is related to an element in the ontology. So, the grammar is composed of three elements related to the ontology: **Node**, **Relation** and **Property**. The grammar always starts by *Type* rule. The request takes as its input the elements in the *Type* rule, operates on them as specified in the *Where* and *Return* rules, and then produces the RDF triple researched by the user. The grammar structure is presented in rule (3):

$$\begin{gathered} Get\text{:} \\ \{\text{Get}\}'\text{Get}'\big(\text{type} = (\text{Type})\big)('\text{Where}'(\text{clauses}{+}{=} \\ \text{Clause})\ast)?\,(\text{retours} = (\text{Return})) \end{gathered} \tag{3}$$

The *Type* rule defines the input type of the request. An input may be a concept, a relation, or a property of the AM ontology. For that reason, the *Type* rule contains a *Node* element corresponding to an ontology concept. It also contains a *Relation* rule. The *Relation* rule is composed of the ontology relation and a range of relation called *destNode*. The *Type* rule is presented in (4), where *C, C'* are ontology concepts and *Rel* is the relation between them:

$$\begin{aligned} &C\ rel\ C' \ \Longrightarrow Type: \\ &\{\text{type}\}(\text{firstNode} = \text{Node})(\text{relations} + \ = \text{RelationTo})) \ast \\ &\text{RelationTo:} \\ &(\text{relation} = \text{Rel})(\text{destNode} = \text{Node}) \\ &\text{Rel:} \\ &\{\text{Rel}\}(\text{name} = \text{ID})?'\,:'\,(\text{relationName} = \text{ID}|'\text{All}') \\ &\text{Node:} \\ &\{\text{Node}\}(\text{name} = \text{ID})?'\text{:}'(\text{className} = \text{ID}|'\text{All}') \end{aligned} \tag{4}$$

The *Where* rule defines constraints about an ontology element. The constraint is defined by an operator and its value. A constraint about an attribute is called data property in OWL [14]. We use a second rule in our grammar to define this constraint. Considering *C*, a concept of AM ontology and p its data type property. We define a restriction on p as follows (5):

$$\begin{aligned} &p\ Domain\ C\ and\ p\ has\ value \ \Longrightarrow Clause\text{:} \\ &(\text{property} = \text{ID})'\text{of}'(\text{node} = [\text{Node}]) \\ &(\text{operator} = \text{Operator})(\text{value} = \text{Value}) \end{aligned} \tag{5}$$

The *Return* rule lists the result of the query. The query result is defined in the *Return* rule, as seen in (6). A function is a facultative element, which presents any logic and mathematic functions that we can apply to a property. Our rule returns instances from the ontology according to type and constraints defined in the mentioned grammar rules in (4) and (5).

$$\begin{gathered} Return\text{:} \\ \{\text{Return}\}'\text{Return}'(\text{properties}{+}{=}\ \text{Property}) + \\ \text{Property:} \\ (\text{function} = \text{Function})?\,\text{propertyNam} = \text{ID}'\text{of}'\text{node} \\ = [\text{Node}] \end{gathered} \tag{6}$$

The elements defined in our DSL grammar are instances of EObject of Eclipse Modeling Framework (EMF) [15] Ecore models. To implement our DSL grammar, we use the Xtext [16] tool of the EMF ecosystem.

C. Scenario example

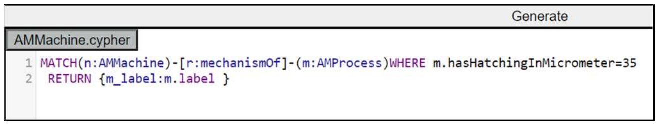
In this scenario example, we explain our system by considering the following user story: we seek the process list where the hatching in micrometer equals to 35. This list aims to classify processes in categories.

To answer this user story, there are four steps:

1. The user can write a simple query without having to learn a Cypher language like in (7). In our interface, we define queries templates with our DSLReq, which can help the user to autocomplete the request.

$$\begin{gathered}\boldsymbol{Get}\ n\colon AMMachine\ r\colon mechanismOf\ m\colon AMProcess \\ \boldsymbol{Where}\ hasHatcingInMicromer\ of\ m = 35 \\ \boldsymbol{Return}\ label\ of\ m \end{gathered} \tag{7}$$

2. The query must have a syntactic and semantic validation. Our DSL validator generates an error if the term given by the user is not in the AM ontology.
3. The system generates a Cypher query by using the Query generator (Figure 4).



```
MATCH(n:AMMachine)-[r:mechanismOf]-(m:AMProcess)WHERE m.hasHatchingInMicrometer=35
 RETURN {m_label:m.label }
```

Figure 4. Cypher query

4. The system connects to the knowledge base and parses the ontology by using this request. Finally, it gives the AMProcess list appropriate to answer the user query.

IV. CONCLUSION AND FUTURE WORK

In this paper, we proposed a query demonstrator able to give appropriate results for user requests by using a DSL grammar. All modules of our system shared a central ontology describing the additive manufacturing domain.

This ontology is used for building and validation of a DSL request. It is also used to support queries for retrieving that knowledge. However, an additive manufacturing environment requires knowledge contributions from different stakeholders, so it is necessary for software engineering to interact with other engineering disciplines. For that reason, the presented approach needs to be complemented by a set of multiple Domain-Specific Languages. Each DSL will relate to an engineering discipline and will be in interaction with the other DSLs. To achieve these results, the ontology will provide interoperability between the DSLs in our system.

REFERENCES

[1] M. Franzago, D. D. Ruscio, I. Malavolta, and H. Muccini, "Collaborative Model-Driven Software Engineering: A Classification Framework and a Research Map," *IEEE Trans. Software Eng.*, vol. 44, no. 12, pp. 1146–1175, Dec. 2018, doi: 10.1109/TSE.2017.2755039.

[2] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?" in *Model-Driven Engineering Languages and Systems*, vol. 8107, A. Moreira, B. Schätz, J. Gray, A. Vallecillo, and P. Clarke, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–17. doi: 10.1007/978-3-642-41533-3_1.

[3] A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, "Grand challenges in model-driven engineering: an analysis of the state of the research," *Software and Syst Model*, vol. 19, no. 1, pp. 5–13, Jan. 2020, doi: 10.1007/s10270-019-00773-6.

[4] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *International Journal of Human-Computer Studies*, vol. 43, no. 5–6, pp. 907–928, Nov. 1995, doi: 10.1006/ijhc.1995.1081.

[5] R. Tairas, M. Mernik, and J. Gray, "Using Ontologies in the Domain Analysis of Domain-Specific Languages," in *Models in Software Engineering*, vol. 5421, M. R. V. Chaudron, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 332–342. doi: 10.1007/978-3-642-01648-6_35.

[6] K. Michal, Š. Michal, and B. Zdeněk, "Interoperability through ontologies," *IFAC Proceedings Volumes*, vol. 45, no. 7, pp. 196–200, 2012, doi: 10.3182/20120523-3-CZ-3015.00039.

[7] M. von Rosing and J.A. Zachman Sr., "The Need for a Role Ontology," *IJCSSA*, vol. 5, no. 1, pp. 1–24, Jan. 2017, doi: 10.4018/IJCSSA.2017010101.

[8] M. Mohd Ali, R. Rai, N. Otte, and B. Smith, "A product life cycle ontology for additive manufacturing," *Computers in Industry*, vol. 105, pp. 191–203, Dec. 2018, doi: 10.1016/j.compind.2018.12.007.

[9] E. Sanfilippo, F. Belkadi, and A. Bernard, "Ontology-based knowledge representation for additive manufacturing," *Computers in Industry*, vol. 109, pp. 182–194, Aug. 2019, doi: 10.1016/j.compind.2019.03.006.

[10] "Cypher Query Language - Developer Guides." https://neo4j.com/developer/cypher/ (accessed Mar. 10, 2022).

[11] L. Ramos, R. Gil, D. Anastasiou, and M. J. Martin-Bautista, "Towards a Machine of a Process (MOP) ontology to facilitate e-commerce of industrial machinery," *Computers in Industry*, vol. 65, no. 1, pp. 108–115, 2014.

[12] S. Borgo *et al.*, "DOLCE: A descriptive ontology for linguistic and cognitive engineering1," *Applied Ontology*, pp. 1–25, Nov. 2021, doi: 10.3233/AO-210259.

[13] "Protégé" https://protege.stanford.edu/ (accessed Mar. 01, 2022).

[14] "OWL - Semantic Web Standards." https://www.w3.org/OWL/ (accessed Feb. 17, 2022).

[15] R. Gronback, "Eclipse Modeling Project | The Eclipse Foundation." https://www.eclipse.org/modeling/emf/ (accessed Mar. 23, 2022).

[16] "Xtext - Language Engineering Made Easy!" https://www.eclipse.org/Xtext/ (accessed Feb. 21, 2022).