# Ontology-driven Development of the Metamodels for Modelling Distributed Parallel Software Systems

CONFERENCE PAPER · DECEMBER 2015

DOI: 10.1109/AIMS.2015.67

Vitaliy Mezhuyev
Universiti Malaysia Pahang

# Ontology-driven Development of the Metamodels for Modelling Distributed Parallel Software Systems

Vitaliy Mezhuyev
*Faculty of Computer Systems and Software Engineering*
University Malaysia Pahang
Gambang, Malaysia
mejuev@ukr.net

Eugene Malakhov
*Department of Mathematical Support of Computer Systems*
Odessa I.I. Mechnikov National University
Odessa, Ukraine
opmev@mail.ru

***Abstract* — The paper discusses a new technique for development of metamodels for the modelling distributed parallel software systems. The approach is an important stage of Domain-Specific Mathematical Modelling (DSMM), developed to enhance the methodology of Domain-Specific Modelling. The advantage of DSMM is a possibility of constructing metamodels for modelling domains, having different mathematical properties and structures. The paper analyses applicability of OWL-DL ontologies for expressing properties of software systems. Identification of the metatypes as OWL classes and use of OWL restrictions as rules of metamodels' grammars give us an effective way for the design and verification of software systems. The proposed approach have advantages of the model-driven software development and allows verification of software systems at earlier design stage.**

***Keywords* — *ontology, metamodel, domain specific mathematical modelling, parallel software system, logical analysis.***

## I. INTRODUCTION

The methodology of Domain-Specific Modelling (DSM) becomes more and more popular in the modern software engineering [1-4]. The essence of DSM is a construction of metamodels and their application for modelling domains with the purpose of software systems development.

Despite of the power of DSM, its theoretical base and practical implementation have several drawbacks. To overcome these limitations we have proposed the methodology for Domain Specific Mathematical Modelling (DSMM) [5]. Its advantage is a possibility of development of metamodels for modelling domains, having different mathematical properties and structure [6]. An architecture of implementing DSMM software tools was described in [7].

Metamodel gives a description of an abstract syntax of a Domain Specific Language (DSL) and defines a set of rules for combining DSL' constructs in order to create valid models of a domain. While metamodel' rules define a *syntax* of a DSL, they also have to be driven by *semantic* considerations of a domain. Otherwise, it can lead to situation that a conceptual primitive or a rule of a domain is not represented in the syntax of a DSL. This why consideration of an ontological approach for analysis of semantics of a domain with the purpose of metamodels development is an actual scientific problem.

In our previous works, we consider the method for the development of metamodels as logical and algebraic systems [8]. An algebraic layer was used for the modelling mathematical structure of a domain, while a logical system expresses semantics of domain. For example, we use vector algebra and logic of syllogisms to produce the metamodel, called "vector logic" [9]. Here, the model of a domain consists of the syllogisms, which are instances of the metamodel' types "logical vectors". Vector algebra was used to define algebraic metatypes, the rules of the metamodel' grammar and corresponding mathematical methods. In particular, the new method for the reasoning was proposed, where the inference is the sum of the logical vectors that represent given as syllogisms assumptions.

In this paper, we analyse applicability of OWL-DL [10] for the development of metamodels and further modelling software systems. The advantage of this approach that OWL-DL can be used not only for linguistic, but sematic modelling of a domain. Feasibility of using ontologies for the development of metamodels we prove on examples of modelling and validation of properties of distributed parallel software systems. Application of Web Ontology Language (OWL) is also illustrated on example of modelling a network of distributed computing nodes. In both cases, ontological metamodelling we consider as an important stage of the DSMM.

This paper is organized as follows. First, we discuss the proposed technique for the ontology-driven development of the metamodel for modelling distributed parallel software systems on the base of OWL-DL. Next, we consider an example of the metamodel development for the modelling topology of computing nodes of a distributed software system. Chapter 3 discusses improvements achieved. Conclusions and prospects for the future research finalize the paper.

## II. PROPOSED METAMODELLING TECHNIQUE

Modern market of software (SW) and hardware (HW) systems needs reducing cost and increasing functionality of computer systems. In addition, modern HW and SW systems have reached the level of complexity, for which the requirements of safety and fault tolerance are crucial. That is why the development of technique for modelling computer systems, which allows their verification at earlier design stage, is an actual scientific and technical problem.

Let us consider an applicability of DSMM for the development of metamodels for modelling software and hardware systems. We consider conceptual modelling, including formulation of requirements and specifications, as a first phase of a computer system development.

In [11] we proposed a conceptual metamodel for the specification of software systems with natural language. Using this conceptual metamodel allows us to structure initially disjoint requirements and specifications, make their unification, and highlight different aspects of a system design. However, structuring of users' statements about future system properties within a certain conceptual scheme does not limit the feasibility of the metamodelling. To use advantages of the metamodelling approach we need express metamodel as a formal specification of conceptualisation, i.e. an ontology [12]. In this paper, we learn applicability of Web Ontology Language, OWL.

OWL is actually a family of logics that have significantly different properties. The OWL languages are characterized by formal semantics. In particular, OWL-DL (DL is a Description Logic [13]) allows us to verify properties of ontologies by means of special tools for logical analysis (so called reasoners, e.g. FaCT++ [14], Pellet [15], RacerPro [16], HermiT [17]). Specification of a software system with OWL-DL provides a combination of two most important stages of software development: conceptual modelling (e.g. specification of requirements and quality attributes) and formal verification of software properties.

Flowing DSM approach, development of a model of a software or a hardware system to be done inside its metamodel. The metamodeling is a task that identifies and specifies the world structures that are of an interest to solve a given problem [18]. Suitability of a metamodel to create models in a given domain depends on how "close" the structure of the models, constructed using that language, resemble the structure of the domain abstractions they are supposed to represent [19].

Guizzardi assumed, that a model M, produced in a language L, should have, at least, a homomorphism of the abstraction A that M represents. This evaluation can be performed based on the analysis of the relation between the structure of a modelling language and the structure of a domain conceptualization [19], i.e. a metamodel and a model.

In proposed approach, we use formalized conceptualisation of a domain (i.e., an ontology) as an input for a metamodel development. Note, much of the recent work are focused on the use of metamodeling as a language definition technique. As Atkinson and Kühne point, a traditional "language definition" interpretation of metamodeling is not a sufficient foundation. They propose to distinguish linguistic metamodelling and ontological metamodelling [20].

While linguistic metamodeling is used for a language definition, ontological metamodeling is needed for modelling semantics, e.g. for categorization and interpretation of model elements. Each of the elements of ontological metamodel gives semantic interpretation of the model' elements in the context of a specific domain.

Laarman and Kurtev emphasize on difficulties to build ontological models and metamodels, due to the current metalanguages support mainly linguistic metamodeling and do not provide constructs for ontological metamodeling [18]. Typically, for description of the meaning of metamodels and models a natural language is used.

In this work, we express semantics of metamodels in the concepts of OWL-DL: class, individual, and property. As result, we produce a software ontology, consisting of a set of axioms, which put constraints on typical sets of individuals (called "classes") and on relationships between them. Exactly these axioms provide *semantics* of models of software systems and allow for reasoning tools to infer additional information based on the explicitly provided data.

## *A. Ontology-driven Development of the Metamodel for Modelling Paralell Software Application*

Let us consider development of the metamodel for software modelling on the base of OWL-DL ontology. The most common way to build an ontology is to organize their objects into taxonomy subclass-superclass. Figure 1 shows a simplified ontology of a parallel software system (Application), having hasSubclass relationships between their classes (for example, Task hasSubclass UserTask) and hasInstance relationships between a class and its instances (e.g., UserTask hasInstance $Task_1$). In this taxonomy we use class Application as a root (traditionally, all OWL classes are subclasses of the class Thing [10]).

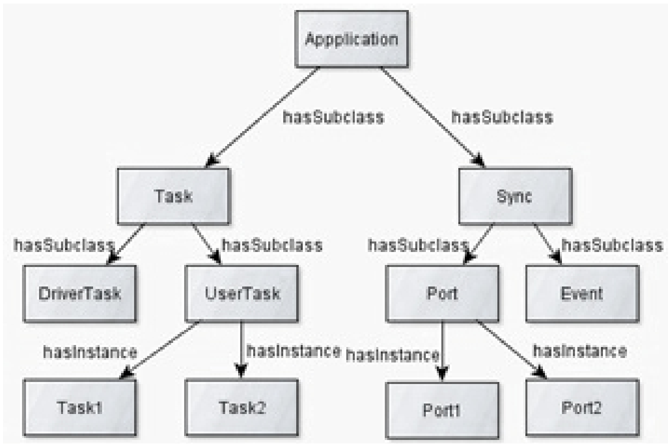

Figure 1. Simple ontology of a parallel software system

Class Task defines an active software entity (process or thread), and Sync (Synchronization object) is used for synchronisation of communicating tasks in a parallel software system [21]. Like in DSMM, in OWL-DL hasSubclass defines the generalization relationship in terms of mathematical sets, e.g. when Port is a subclass of Sync, all elements of the set Port are also belong to the set Sync. The same rule is applicable to other OWL classes. It also means inheritance of the properties: for example, if all Sync have the synchronization predicate and Port is Sync, then it follows that the Port also should have the property of synchronization predicate. This simple rule, expressed in OWL-DL, allows us to deduct additional information based on the explicitly provided data, which is not possible in a classical MDA.

In our approach, Task is a combination of two classes – user and driver tasks. Class Sync includes typical for parallel programming synchronisation objects – mutex, semaphores, ports, events etc. [21] Here the simplified definition of classes Task and Sync with DL (we limit Sync by Port and Event classes)

$$Task \equiv DriverTask \cup UserTask \tag{1}$$

$$Sync \equiv Port \cup Event \tag{2}$$

OWL classes Task and Sync cannot overlap. This means that any concept of the ontology cannot be an instance of more than one of these classes. Next DL formula expresses this relationship

$$Task \cap Sync = \emptyset \tag{3}$$

Not all classes in the ontology may have direct instances. For example, Sync is the OWL superclass for Port, Event and other types of synchronization objects only that can have direct instances. This property can be formalized by the following DL formulas

$$\begin{gathered} \forall has\text{Instance}.Sync = \emptyset \\ \forall has\text{Instance}.\forall hasSubclass.Sync \neq \emptyset \end{gathered} \tag{4}$$

Formulas (1)-(4) are the predicates of OWL classes, which used in DSMM as rules of the metamodel' grammar to define how the types Task and Sync can be instantiated. (1)-(4) are also used as the input to logical reasoners for verification of software models, derived from the metamodel. In this simple example, we can check correctness of superclass-subclass and class-instance relationships of software ontology. Moreover, logical reasoners allows us to verify compliance of concepts, compute derived hierarchies and allocate equivalent classes of ontologies.

OWL-DL ontologies can cover only static structural properties of a domain (in our example, of a distributed parallel software system). To model a dynamic behaviour it is necessary to define logical formulas for functional, safety, liveness and real-time properties of a system. Checking these properties is beyond the capabilities of OWL-DL ontologies and corresponding reasoning tools.

Let us consider the rules of DSMM metamodels' grammar, formulated as OWL properties. Previous predicates define the rules for instantiating metamodels' types. Let us combine instances of the metamodels' types in more complex structures. For it, we express relationships of objects of a software ontology as OWL properties. For example, there is relationships between tasks in a parallel software system - PutData (send data into Sync object) and GetData (pick up data from a Sync object) [21], linking instances of the classes software task Task and synchronization object Sync.

OWL properties of objects combine instances from a particular domain with instances, having particular range. The domain for PutData is the set of instances of the class Task. Range is the set of instances of the synchronization object (of the class Sync). We use these sets in DSMM for checking types of objects, formulated in a specification of a software system. For example, if a user formulate a rule that a particular $object_1$ is associated with an $object_2$ by the property PutData, when it follows, that an $object_1$ is an instance of the class Task, and an $object_2$ is an instance of the class *Sync*.

To model hierarchical structure of a software system, we use OWL subproperties. Subproperties specialize own superproperties, for example, the property hasName specializes more general property hasAttribute. In particular, in software ontology [21], we divide OWL properties on task management services (e.g., StartTask and StopTask) and synchronization services (e.g., PutData and GetData).

Each property of a class in OWL ontology can have an appropriate inverse property. For example, a property isPartOf is an inverse property for consistsOf, e.g. from "Topology consistsOf Nodes" it follows that "Nodes isPartOf Topology". Inverse properties allows us formally define semantically opposite pairs of OWL properties, for example, StartTask and StopTask, PutData and GetData etc., which is caused by the symmetry of communication of tasks in a parallel software system. The principle of symmetry we use for the verification of runtime properties at a design stage.

Note, that formulated as inverse pairs OWL properties are not expressive for the casual relationships, for example, from the statement, that $Task_1$ StartTask $Task_2$ does not follow that $Task_2$ StopTask $Task_1$. Formulation of such properties (causal relationship between $Task_1$ and $Task_2$) needs using logical implication.

OWL-restrictions allows us to define the rules of metamodels' grammar that serve to build valid specifications of a software application. Mathematically, such the model is the set, which contains instances of classes of synchronization objects (Sync) and tasks (Task):

$$Application \equiv Task \cup Sync \tag{5}$$

Port is one of the objects to synchronize communication of software tasks in a parallel system. A valid application assumes that if a $Task_1$ put the data into port (i.e. call the PutData routine), then there must be another task $Task_2$, which takes the data from the same port (i.e. calls the GetData procedure).

Every OWL-DL restriction defines an anonymous class (set of individuals), containing instances that satisfy a given logical formula. For example, OWL restriction for the class Task: PutData.Port defines an anonymous class, instances of which are the objects of the type Task and participate in the PutData interaction. Other OWL restriction for the class Task: GetData.Port defines an anonymous class, instances of which are the members of the class Task and participate in the GetData interaction. Intersection of these anonymous classes defines the superclass ValidTask, elements of which are involved both in PutData and GetData software interactions

$$\begin{gathered} ValidTask \equiv \forall PutData.Port \cap \\ \forall GetData.Port \end{gathered} \tag{6}$$

Thus, the rule for a valid interaction of a task with a port we define as an intersection of sets that include instances of tasks involved in GetData and PutData interactions. Such the OWL restriction is a specification of valid class of software application and defines the set of its valid instances. Note that the definition (6) results from the property of the symmetry of interaction of software tasks in a parallel system.

This approach illustrates applicability of OWL-DL for the development of metamodels for modelling software systems and verification of their properties. We can also consider other properties to demonstrate the proposed approach. For example, let's use OWL-DL to specify a main task in a software application, as including definition of the main function

$$MainTask \equiv Task \cap \exists hasFunction.main \tag{7}$$

MainTask belongs to the class Task and includes (at least one) main function. Note that an application task should have only one main function. To express it, use of OWL restrictions on the number of elements in a class is needed (which determines the exact number of relationships in which may participate an instance of the class).

Another way is a concretization of the property hasFunction by the definition of the interaction hasMainFunction as a functional property. This means that for a given instance of the Application class, there is only one instance of the class MainFunction, associated with an instance of an Application through the property isMainFunctionOf. Given that a valid software application always has a main function, property (7) is a necessary part of the specification.

## B. *Ontology-driven Development of the Metamodel for Modelling Topology of Computing Nodes*

Let us consider an example of ontology-driven development of the metamodel for modelling topology of computing nodes of a distributed software system. In order to model heterogeneous and distributed computing nodes we will express them as instances of OWL-DL class Node having unified representation of the properties in a certain configuration (including attributes - Parameters, Paths, Compiler Options, etc.).

In our approach, an ontology of computing network consists of hardware nodes (Node) and relationships (Link), connecting these nodes. Links can be Unidirectional or Bidirectional. To be reachable, each node in the topology must have at least one input and one output. Thus, between any two nodes of the computer network should be at least one way. Since unidirectional and bidirectional links are relations between two nodes, they can be considered as a binary OWL properties. For example hasUniLink (a one-way communication) and hasBiLink (a two-directional communication).

Bidirectional relationships are symmetrical, i.e. if a $Node_1$ connected with a $Node_2$, it follows that $Node_2$ has a connection with $Node_1$ (see Figure 2). From the mathematical point of view, the property hasBiLink is its own inverse property.

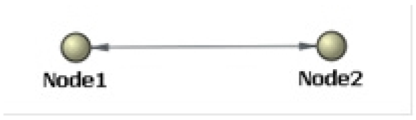

Figure 2. Symmetry of bidirectional communications

Bidirectional communications are transitive. If a $Node_1$ has a connection with a $Node_2$, and a $Node_2$ with a $Node_3$, then $Node_1$ is reachable from the $Node_3$, i.e. there is a path between them (see Figure 3). Note, that we do not distinguish in these considerations the concept of a physical connection and a logical path.

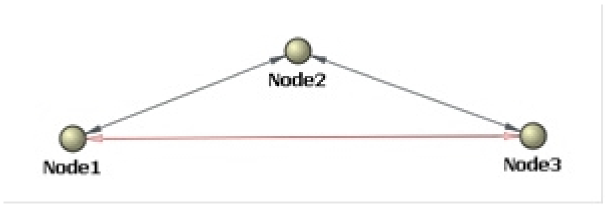

Figure 3. Transitivity of bidirectional communications

Given specifications allow us to define the rules for the metamodel grammar. For example, by imposing restrictions on OWL class Node we model the next property of computing topology

$$\exists hasBiLinkNode \tag{8}$$

Expression (8) defines an anonymous OWL class of nodes, having at least one bidirectional link with another node. This OWL class gives a simple rule of the metamodel for modelling some real topology.

To model the reachability property (i.e. to be accessible, each node of the topology must have both incoming and outgoing links), let us define two subproperties of the hasUniLink: hasInUniLink (has a unidirectional incoming link) and hasOutUniLink (has a unidirectional outcoming link).

Intersection of such OWL classes, having input and output unidirectional links, gives us an anonymous OWL class that meets the needed property:

$$\exists hasInUniLink.Node \cap \exists hasOutUniLinkNode \tag{9}$$

Given that nodes can also have bidirectional links, we formulate the metamodel' rule, specifying that there is at least one path between any two nodes

$$(\exists hasInUniLink.Node \cap \exists hasOutUniLinkNode) \cup \exists hasBiLinkNode \tag{10}$$

Based on the OWL-DL specifications and with the help of logical reasoners the validity of computing topology can be verified.

## III. Analyses of improvements achieved

Development of metamodels on the base of OWL-DL gives us an effective approach for the automation of software systems development and validation of their properties. Let us summarize the improvements achieved:

- OWL-DL allows us to define not only an abstract syntax of a metamodel, but also to capture semantics of a considered domain. Moreover, the proposed ontology-driven approach guarantees the important principle of predominant role of semantics over syntax;
- formal semantics of ontology-driven metamodels allows us using logical reasoners to check properties of derived models at the early stages of software systems development;
- expressing metamodel with help of OWL-DL allows us to infer additional information, not given by users directly in models, which is not possible in a classical MDA approach;
- use of an ontology as an input for a metamodel development guarantees homomorphism of derived models to domain conceptualisation;

The proposed approach opens a space for discussions:

- instantiation semantics for the linguistic and the ontological layers of the ontology-driven metamodel need to be elaborated;
- advantages and disadvantages of proposed approach for the generation of source code in a target programming language to be analysed;
- possibility of expressing temporal semantics of a software system to be learned (ontological metamodel layers are usually frozen, i.e., they cannot be changed at run-time [22]).

## IV. Conclusion

Traditionally, the metamodeling considered as a definition of an abstract syntax for a domain specific language. Semantic aspects of a metamodel are typically left outside of consideration. In this paper, we discussed a natural applicability of ontologies for expressing semantics of a metamodel. Definition of the metamodels' types as OWL classes and use of OWL-restrictions as rules of metamodels' grammar allows development of the valid, i.e. corresponding to system specifications, models of software systems. In our future works, we will enhance the set of software properties, which can be modelled and checked with ontology-driven metamodels. One of the priorities is development of the metamodel on the base on Temporal Logic of Actions [23] to study real-time and liveness properties of software systems. Another important direction is linking linguistic aspects of metamodelling with mathematical theory of modelling, which will allow us to express mathematical structures of systems.

## References

[1] Steven Kelly and Juha-Pekka Tolvanen. Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Pr.; 2008. 427 p.

[2] Richard C. Gronback. Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional; 2009. 736 p.

[3] Steve Cook, Gareth Jones, Stuart Kent, and Alan Cameron Wills. Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley Professional; 576 p.

[4] JetBrains MPS. http://www.jetbrains.com/mps/

[5] Vitaliy Mezhuyev. Methodology of Domain Specific Mathematical Modelling. Proceedings of the 3rd International Congress on Natural Sciences and Engineering ICNSE 2014 (Kyoto, Japan, May 7-9, 2014); Pp. 54-64.

[6] Vitaliy Mezhuyev. Metamodelling Architecture for Modelling Domains with Different Mathematical Structure. Advanced Computer and Communication Engineering Technology. Lecture Notes in Electrical Engineering; 2015. Vol. 315. Pp. 1049-1055.

[7] Vitaliy Mezhuyev. Architecture of software tools for Domain-Specific Mathematical Modelling. Proceedings of 2014 International Conference on Computer, Communication, and Control Technology (Langkawi, Malaysia, September 2-4, 2014). Pp. 166–170.

[8] Vitaliy Mezhuyev. Development of Metamodels as Logical and Algebraic Systems. Proceedings of the 2014 International Conference on Information Science, Electronics and Electrical Engineering ISEEE (Hokkaido, Japan, April 26-28, 2014). Pp. 1850-1855.

[9] Vitaliy Mezhuyev. Vector logic: theoretical principles and practical implementations. Papers of ZNU; 2006. Pp. 91-97.

[10] OWL Web Ontology Language Guide. W3C Recommendation 10 February 2004. Available from: http://www.w3.org/TR/owl-guide

[11] Vitaliy Mezhuyev, Bernhard Sputh, EricVerhulst. Interacting Entities Modelling Methodology for Robust Systems Design. Advances in System Testing and Validation Lifecycle. CPS publishing; 2010. Pp. 75-80.

[12] T. R. Gruber. A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220, 1993.

[13] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider. The Description Logic Hand book: Theory, Implementation, Applications. CambridgeUniversityPress, Cambridge, UK; 2003. 624 p.

[14] Tsarkov, D.; Horrocks, I. FaCT++ Description Logic Reasoner: System Description. Automated Reasoning. Lecture Notes in Computer Science 4130. 2006 Pp. 292–297.

[15] Sirin, E.; Parsia, B.; Grau, B. C.; Kalyanpur, A.; Katz, Y. Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web 5 (2). 2007. Pp. 51–53.

[16] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The RacerPro knowledge representation and reasoning system. Semantic Web Journal, 3(3). 2012. Pp. 267–277.

[17] HermiT OWL Reasoner. http://hermit-reasoner.com/

[18] Alfons Laarman, Ivan Kurtev. Ontological Metamodeling with Explicit Instantiation. Volume 5969 of the series Lecture Notes in Computer Science. 2010. Pp 174-183

[19] Giancarlo Guizzardi. On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models . Proceedings of the 2007 conference on Databases and Information Systems. Pp. 18-39.

[20] Colin Atkinson, Thomas Kühne. Model-Driven Development: A Metamodeling Foundation. Journal IEEE Software Volume 20 Issue 5, September 2003. Pp. 36-41.

[21] Vitaliy Mezhuyev, Eric Verhulst. Modelling OpenComRTOS tasks interaction. Mathematical systems and machines; 2010. №2. Pp. 32-41.

[22] Ingo Ott, Martin Schader. Ontological Metamodel Extension for Generative Architectures (OMEGA). Lehrstuhl für Wirtschaftsinformatik III, Univ. Mannheim, 2004. 38 p.

[23] Lamport L. Specifying systems: the TLA+ language and tools for hardware and software engineers. Addison-Wesley, Boston; 2002. 364 p.