

Semantically-Rigorous Systems Engineering Modeling Using SysML and OWL

J. Steven Jenkins⁽¹⁾, Nicolas F. Rouquette⁽²⁾

⁽¹⁾Systems and Software Division
Jet Propulsion Laboratory
California Institute of Technology MS 301-225
Pasadena CA 91109
USA
steven.jenkins@jpl.nasa.gov

⁽²⁾Systems Engineering Section
Jet Propulsion Laboratory
California Institute of Technology
MS 301-490
Pasadena CA 91109
USA
nicolas.f.rouquette@jpl.nasa.gov

Copyright © 2012 California Institute of Technology. U.S. Government sponsorship acknowledged. This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Introduction

The Systems Modeling Language (SysML)[18] has found wide acceptance as a standard graphical notation for the domain of systems engineering. SysML subsets and extends the Unified Modeling Language (UML)[16, 17] to define conventions for expressing structural, behavioral, and analytical elements, and relationships among them. SysML-enabled modeling tools are available from multiple providers, and have been used for diverse projects in military aerospace, scientific exploration, and civil engineering.

The Web Ontology Language (OWL)[7, 8] has found wide acceptance as a standard notation for knowledge representation. OWL-enabled modeling tools are available from multiple providers, as well as auxiliary assets such as reasoners and application programming interface libraries, etc. OWL has been applied to diverse projects in a wide array of fields.

While the emphasis in SysML is on notation, SysML inherits (from UML) a semantic foundation that provides for limited reasoning and analysis. UML's partial formalization (FUML)[13], however, does not cover the full semantics of SysML, which is a substantial impediment to developing high confidence in the soundness of any conclusions drawn therefrom. OWL, by contrast, was developed from the beginning on formal logical principles, and consequently provides strong support for verification of consistency and satisfiability, extraction of entailments, conjunctive query answering, etc. This emphasis on formal logic is counterbalanced by the absence of any graphical notation conventions in the OWL standards. Consequently, OWL has had only limited adoption in systems engineering.

The complementary strengths and weaknesses of SysML and OWL motivate an interest in combining them in such a way that we can benefit from the attractive graphical notation of SysML and the formal reasoning of OWL. This paper describes an approach to achieving that combination.

The Role of Logical Reasoning in Systems Engineering

It is universally accepted that quantitative analysis is an essential element of the practice of systems engineering. Systems engineers routinely employ mathematical models to calculate measures of effectiveness, performance metrics, failure probabilities, costs, etc. In contrast, systems engineers do not generally feature a role for logical analysis, although most competent systems engineers would recognize the importance of the following example applications:

Requirements Tracing In a properly-formed forest of requirements, every requirement levied upon a subcomponent traces up to at least one requirement levied upon its parent component. Moreover, no requirement levied upon a subcomponent traces to a requirement levied upon a subcomponent at the same level or lower. Such properties can be straightforwardly verified using simple (and fast) algorithms applied to graph data structures. Graph structures, in turn, are fundamental to knowledge representation in OWL.

Interface Consistency Interfaces joined in a well-formed system design must be of compatible type. A rigorous system model, therefore, will not only assign a type to each interface, but will also explicitly declare which interface types are mutually compatible. Testing interfaces for compatibility, however, is more subtle than merely checking their types for asserted compatibility. The actual predicate to be tested is whether the interface types are compatible types *or* are subtypes of compatible types; evaluating this predicate requires finding the transitive closure of the *has supertype* property. Knowledge representation systems commonly employ reasoners that, among other things, find transitive closures.

Viewpoint Consistency The design of complex systems invariably involves constructing views that correspond to multiple viewpoints. For example, in spacecraft design we might describe a single avionics card from conceptual viewpoints that address power consumption, heat dissipation, lifetime, radiation susceptibility, mass properties, volume, computational performance, fault containment, etc. The end of integrating these views is to produce a specification for the card that will lead to acquisition of a physical realization that satisfies all constraints in the union of all views. For a complex system consisting of multiple components, each of which is described from multiple viewpoints, it is a challenge to ensure that every conceptual viewpoint is accounted for in some realization. Finding counterexamples is an application of a common technique in knowledge representation called *conjunctive query answering*.

It is important to note to distinguish these examples from similar but more pedestrian activities encountered in tracking policy and procedure compliance and assessing progress against a schedule. There are notorious failures in systems engineering that could have been prevented by more careful attention to this type of analysis[1].

An Aside on the Word *Ontology*

A formal ontology is a set of statements (commonly called *axioms*). In practice, “creating an ontology” or words to that effect generally means defining a set of concepts and properties applicable to a domain of discourse. Such definitions are known as a *terminological component* or *TBox* in knowledge representation[2]. TBox axioms for systems engineering might define concepts like *component*, *function*, *requirement*, and *work package*, data properties like *mass* and *cost*, and object properties (relationships) like *performs*, *specifies*, and *supplies*. In contrast, a set of axioms describing specific individuals and their properties (using terminology from a TBox) is known as an *assertion component* or *ABox*. For example, “the Curiosity rover is a component” and “its mass is 899 kg” are ABox axioms. Strictly speaking, an ontology may contain both TBox and ABox axioms. An ontology containing both is commonly called an *information base*.

Developing OWL Ontologies for Systems Engineering

The purpose of developing ontologies for systems engineering is to have a common controlled vocabulary for a very broad range of assertions about complex systems under design and development throughout their life cycles. Using a controlled vocabulary and enforcing rules for well-formedness permits, among other things, durable information storage, lossless information interchange, interdisciplinary information integration, and automated analysis and product generation.

One might ask whether SysML itself provides such an ontology (or the kernel of one); while it is true that SysML was developed for systems engineering, it is equally true that a number of foundation concepts from systems engineering (e.g., work package, objective, environment, etc.) do not explicitly appear in it. More fundamentally, however, we view SysML as one domain in which systems engineering concepts apply, but not the only domain. Our existing engineering analysis tools, for example, embody (usually implicitly) fundamental concepts like component, function, interface, message, etc. Our systems engineering ontologies reflect our usage conventions; we intend for them to provide the formal unifying framework for all systems engineering information in any language, in any repository.

Because our systems engineering ontologies are expressed in OWL, they are amenable to formal validation (syntactic and semantic). We use formal reasoning techniques to ensure that they are consistent and satisfiable, and constrained within the bounds of Description Logic, which ensures that certain reasoning operations remain tractable.

Ontology Categories

We have partitioned our ontologies into three broad categories. The boundaries between categories are not sharp, and the assignment of an ontology to one category or another has no effect on its expressivity or applicability. Instead the categories are reminders of differing foci and objectives.

Within each category are multiple individual ontologies. Again, the boundaries between ontologies are not sharp; division into multiple ontologies is simply for the purpose of improving understanding and ease of management.

Foundation Foundation ontologies define broad, general concepts and properties that establish an overall framework for systems engineering. There are four major ontologies in the foundation category:

Base The base ontology defines a small number of general concepts (e.g., *container*) and properties (e.g., *contains*) that are refined in other ontologies.

Mission The mission ontology defines concepts and properties used to describe the execution of a mission and its context: objectives, performing elements, functions, interfaces, requirements, etc.

Analysis The analysis ontology defines concepts and properties used for qualitative and quantitative characterization of individuals of any time.

Project The project ontology defines concepts and properties used to describe the entities and endeavors involved in designing, analyzing, acquiring, integrating, and testing the elements of a mission: projects, programs, work packages, deliverables, etc.

Discipline Discipline ontologies define (mostly by specialization) concepts and properties pertinent to a particular discipline. The primary objective of discipline ontologies is information interchange; if all systems engineering models, regardless of topic or organization of origin, use common vocabulary for, say, mass properties, it becomes a simple matter to extract the mass of any modeled component. Some example discipline ontologies include:

Electrical Defines concepts and properties for current sources and loads, signal types, conditioning and distribution equipment, etc.

Mechanical Defines mass properties, mechanical interface types, etc.

Verification and Validation Defines process and analysis specializations to capture V&V activities and results.

Application Application ontologies define concepts and properties pertinent to a particular class of engineered system. A propulsion subsystem ontology, for example, would draw from multiple discipline and foundation ontologies to characterize components like thrusters as they are typically employed in a propulsion application.

Each ontology is identified by a unique Internationalized Resource Identifier (IRI)[10], which also establishes a unique XML[6] namespace for all definitions therein. To simplify notation, we use XML prefixes to denote namespaces; the class Component in the mission ontology is denoted *mission:Component*.

Examples

The following examples illustrate (and simplify) some of the concepts and properties in the Foundation ontologies.

Concepts

mission:Component A *mission:Component* is a designed entity that performs a function or presents an interface. Examples include Launch Vehicle, Star Tracker, or Mission Operations Team.

mission:Function A *mission:Function* is an activity performed by a Component. Examples include transport instruments to Martian surface, conduct geological investigation, or transmit science data to earth.

mission:Interface A *mission:Interface* identifies a set of mechanical, electrical, signal, or other properties that describe some aspect of a component's connection to or interaction with another component. Examples include Launch Vehicle to Spacecraft Interface and Telemetry Downlink Interface.

mission:Junction A *mission:Junction* represents the mating or connection of two *mission:Interfaces*.

mission:Requirement A *mission:Requirement* is an assertion that must be true for every acceptable realization of the system design.

project:WorkPackage A *project:WorkPackage* is an authority that supplies one or more components and authorizes one or more components.

Object Properties

base:contains A *mission:Component* *base:contains* zero or more *mission:Components*. A *mission:Interface* *base:contains* zero or more *mission:Interfaces*.

mission:performs A *mission:Component* *mission:performs* zero or more *mission:Functions*. (We might consider a component that performs no functions to be underspecified, but a model with such a component is not necessarily ill-formed.)

mission:presents A *mission:Component* *mission:presents* zero or more *mission:Interfaces*. (Similarly, a component that presents zero interfaces may be underspecified but is not necessarily ill-formed.)

mission:joins A *mission:Junction* *mission:joins* at most two *mission:Interfaces*.

mission:specifies A *mission:Requirement* specifies zero or one specified elements, which may be a *mission:Component*, an occurrence of *mission:presents*, or an occurrence of *mission:performs*. When the specified element is *mission:presents*, we say the requirement is an *interface requirement*. When the specified element is *mission:performs*, we say the requirement is a *functional requirement*.

mission:refines A *mission:Requirement* *mission:refines* zero or more *mission:Requirements*. Refinement connects requirements at a lower implementation level with those at a higher level.

project:authorizes A *project:WorkPackage* *project:authorizes* zero or more authorized elements. Any concept or object property can be an authorized element; the meaning of such a declaration is that that occurrence of the concept or property exists (or is true) because the *project:WorkPackage* has declared it to be so. Authorizing a *mission:Component* means being the customer for it.

project:supplies A *project:WorkPackage* *project:supplies* zero or more supplied elements. A *mission:Component* is an example of a supplied element.

The *project:authorizes* and *project:supplies* relationships are key to integrating technical and programmatic concerns. They provide the vocabulary to establish delegation of authority, without which no large project can succeed, and make explicit the customer/supplier relationships and deliverables that may be the subjects of acquisition contracts.

Embedding Systems Engineering Ontologies in SysML/UML

Building profiles from ontologies necessitates establishing formal relationships between the elements of those ontologies and their counterparts in SysML/UML. In ontological terms, one could say we are stating axioms about elements in our ontologies. Relating those elements to SysML/UML, however, requires the ability to specify (and reason about) SysML/UML concepts and properties. A relatively straightforward way to provide this ability is to produce (by automated transformation) ontologies for SysML, UML, and other related specifications. We implemented such transformations in the Operational Query/View/Transformation Language (QVTo)[14, 11]. The primary benefit of this approach (in addition to being able to reason about SysML itself) is the ability to express relationships between our ontologies and SysML using OWL axioms. That is, we can declare that some class (or property) defined in a JPL ontology is a subclass (or subproperty) of some corresponding element in SysML, or vice versa. For example, we declare that *mission:Component* is a specialization of SysML Block, and *mission:Requirement* is a specialization of SysML Requirement. These embeddings are made on a concept-by-concept basis, and require matching of the ontological commitments in each domain.

Unfortunately, the embedding of OWL relationships (object properties) in SysML/UML relationships is not so straightforward. This stems in part from the fact that occurrences of object properties in OWL are not (in general) reified. That is, the statement that says, in essence, “This spacecraft contains a propulsion subsystem” does not have an identity, and cannot therefore be the subject or object of another statement about the containment relationship. Relationships in UML modeling tools are reified (although the reification may be obscured by its representation as an anonymous line on a diagram). This mismatch must be addressed before integrating the two.

OWL provides mechanisms to define arbitrary classes and properties, so there is no difficulty in principle with creating, for every object property *p* in some ontology, a corresponding class *P* to represent occurrences of that

property, as well as (unreified) source and target properties that connect the reified occurrence to the model elements that it relates. Less obviously, perhaps, OWL (version 2) also provides a property chain mechanism that can be used to declare that the existence of a reified object property occurrence of class P with source A and target B implies $A p B$. A further preliminary step, then, is to supplement the system engineering ontologies with axioms that implement this reification pattern for every object property

Reified relationships are the key to a semantics-preserving mapping between UML and OWL. Without reification, there are many possible combinations for mapping OWL classes and object properties to UML classes, associations, association classes, properties and other relationships (e.g., dependencies). The Object Management Group's Ontology Definition Metamodel (ODM)[15] specification explains some of these possibilities but does not recommend a particular one. More importantly, the ODM lacks a unifying pattern for handling the various ways in which conceptual relationships are modeled as associations, dependencies, generalizations, ports, etc. A generic reification pattern simplifies the UML/OWL mapping because it separates the problem of modeling a conceptual relationship in OWL in terms of classes, object properties and property chain axioms from the problem of choosing an adequate embedding of this conceptual relationship in UML or in a profile extension of UML.

Analyzing and Validating Ontologies

OWL ontologies can be serialized in several forms, all of which are analogous to source code, and fit reasonably well into the software development and code management paradigm. Processing of ontologies, however, including reasoning, requires that they be loaded into a repository that exposes them in the form of software objects and services. We use the Sesame[3] repository. Sesame provides a Java application program interface and a REST-style[12] web interface. Both interfaces support direct manipulation of axioms, as well as query processing using the SPARQL[9] language.

An ontology contains explicitly-stated, or *asserted*, axioms. These asserted axioms may, in turn, imply *entailed* axioms. For example, if we assert that every spacecraft is a component, and Curiosity is a spacecraft, then we can determine by entailment that Curiosity is a component. Entailed axioms are useful for reasoning about ontologies; we use the Pellet reasoner to extract entailments and load them into the repository.

Any well-formed ontology will satisfy certain criteria. For example, every concept in a well-formed ontology should be *satisfiable*. That is, it should be logically possible for an instance of the concept to exist. A somewhat more subtle criterion is that the domain and range of a subproperty should be subclasses of the domain and range, respectively, of the specialized superproperty. Other criteria might be developed reflecting local conventions, allocation of responsibility, etc.

Ontologies, of course, lend themselves to formal verification of criteria. Our ontology processing workflow implements three distinct validation steps, all run under a continuous integration system that invokes validation upon any change to ontology code:

XML Syntax A simple test to ensure that the ontology source file is syntactically-valid XML.

Consistency The Pellet[4] OWL reasoner checks that no axioms in any ontology contradict any other axioms in that ontology or its imported ontologies.

Satisfiability The Pellet reasoner checks that every defined class can (in principle) contain at least one member. (It is possible through mistakes in disjointness and subclass axioms to define classes that can be necessarily empty. Such classes are described as *unsatisfiable*.)

Well-Formedness A battery of SPARQL-based tests ensure that the ontologies comply with local rules for ontology development. For example, the object property reification pattern described above requires 18 axioms for each declared object property. The test for that pattern ensures that all required axioms are present. The current test battery includes 25 such tests.

Transforming OWL Ontologies into UML Profiles

The complete ontology set, including embedding axioms, is in essence a formal specification for a UML profile expressed in OWL. This formal specification is input to another QVT transformation that produces the profile itself. In principle, the transformation could produce a tool-neutral XMI file that could be ingested by any SysML tool. In practice, however, usability concerns argue for producing tool-specific profiles. These profiles can exploit extensions that, for example, provide tool tips or custom user interface widgets that assist modelers in learning and applying the profile.

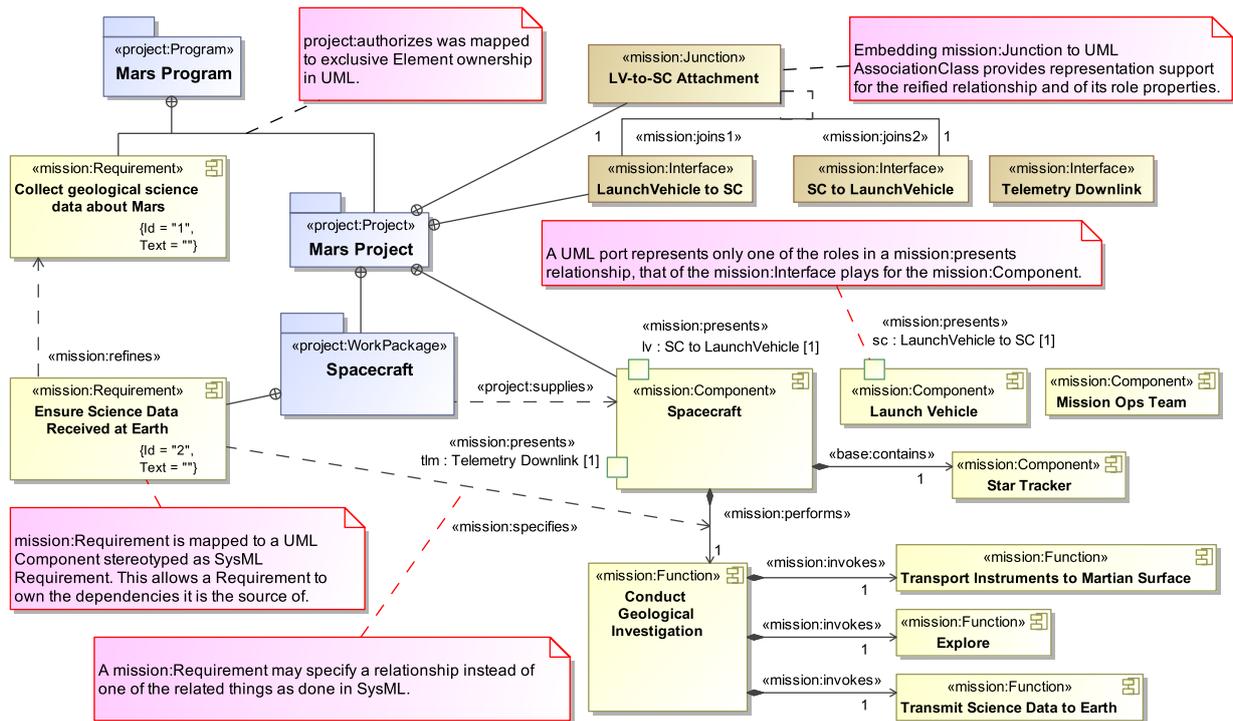


Figure 1: Example of profile application.

Building Profiled System Models

Loading the generated profiles allows, but does not mandate, building models that apply the profiles properly. New stereotypes for classes and properties from the ontologies appear as options, but building a well-formed model that obeys semantic constraints is a methodological consideration outside the scope of the profile itself. We have active projects under way building mission models using a candidate methodology closely tied to the profiles.

Figure 1 is a (greatly simplified) SysML Block Definition Diagram that illustrates application of the profiles to a Mars mission.

Transforming Profiled SysML Models into OWL Ontologies

A final step in the integration process is a QVTo transform that translates such a profiled SysML model in the form of ABox axioms using the TBox vocabulary in the original systems engineering ontologies, amenable to the full repertoire of semantic processing (formal reasoning, conjunctive query answering, model transformation, etc.)

In principle, the ABox transformation is bi-directional, allowing for the possibility of creating SysML models from other sources (e.g., CAD data) using OWL as an intermediate representation.

Processing and Validating Profiled System Models

Because profiled system models have rigorous semantics, it is possible to test assertions about ABox ontologies derived from profiled models in much the same way that we tested assertions about TBox ontologies. Some simplified examples of assertions to test about a profiled model include

- every *mission:Component* *mission:performs* at least one *mission:Function*,
- every *mission:Function* *mission:isPerformedBy* (inverse of *mission:performs*) exactly *one* *mission:Component*,
- every *mission:Requirement* except an identified root set *mission:refines* at least one *mission:Requirement*.

Other more subtle tests involving, for example, consistency of component and requirement hierarchies are possible with more complex SPARQL queries.

Beyond model validation, there are two major use cases for processing OWL ontologies created by transforming profiled SysML models:

Long-Term Repositories OWL repositories have excellent scaling properties[5]. It is entirely feasible to build long-term archives of facts asserted about multiple missions over many years, ranging into hundred of millions of statements or more. Such repositories would be extremely valuable for observing long-term trends in system performance, cost, complexity, etc. Moreover, because the foundation theory and standards of the semantic web are based on open, tool-neutral standards, this approach is robust against obsolescence.

Integration With Specialized Analysis Tools Over the years we have acquired (and often developed) highly specialized, discipline-specific analysis tools. There is often a substantial setup cost to using these tools for a particular application. Building rich system models with rigorous semantics opens up the possibility of constructing input specifications for these tools by extracting features from system models by transformation. Doing so allows the use of these powerful tools earlier in the life cycle because it essentially eliminates the setup cost. This is especially important in concept development where we may want to examine a number of architecturally-distinct approaches and compare them on the basis of figures of merit calculated by analysis tools.

Conclusions and Future Work

The technique of representing UML and SysML in OWL and relating our ontologies to UML and SysML with embedding axioms has proven to be general and flexible. Pre-processing ontologies using SPARQL queries to produce digests for profile generation enforces separation of concerns in a way that simplifies the end-to-end process. QVTo has proven to be a powerful tool for generating “correct by construction” profiles, although the current Eclipse implementation of OVTo has some serious performance deficiencies that require remediation. SPARQL and Sesame are useful for reasoning about ontologies. We have not encountered any performance issues so far; our current ontology set plus entailments is about 31 000 axioms. During processing the repository grows to around 250 000 axioms, most of which are transient and/or redundant.

Transformation from OWL ontologies to SysML profiles is nearly complete. The last remaining step is to complete embedding rules for OWL datatype properties. Transformation from profiled SysML models to OWL is working in rudimentary form. It is simpler in design than the ontology-to-profile transformations, so we do not expect any difficulties in completing it. Our focus in the future will be developing discipline ontologies and analysis transformations.

References

- [1] National Aeronautics and Space Administration. *Apollo 13: The NASA Mission Reports*. Apogee Books, 2000.
- [2] Franz Baader and Werner Nutt. Basic Description Logics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, pages 47–104. Cambridge University Press, 2003.
- [3] Aduna B.V. User Guide for Sesame. 2011. Available from: <http://www.openrdf.org/doc/sesame2/users/>.
- [4] Clark & Parsia, LLC. Pellet: OWL 2 Reasoner for Java. 2011. Available from: <http://clarkparsia.com/pellet/>.
- [5] World Wide Web Consortium. Large Triple Stores. Available from: <http://www.w3.org/wiki/LargeTripleStores>.
- [6] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Fifth Edition). 2008. Available from: <http://www.w3.org/TR/2008/REC-xml-20081126/>.
- [7] World Wide Web Consortium. OWL 2 Web Ontology Language Document Overview. 2009. Available from: <http://www.w3.org/TR/2009/REC-owl2-overview-20091027/>.
- [8] World Wide Web Consortium. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. 2009. Available from: <http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/>.
- [9] World Wide Web Consortium. SPARQL 1.1 Query Language. 2012. Available from: <http://www.w3.org/TR/2012/WD-sparql11-query-20120724/>.
- [10] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), January 2005. Available from: <http://www.ietf.org/rfc/rfc3987.txt>.

- [11] Radomil Dvorak. Model Transformation with Operational QVT. 2008. Available from: <http://www.eclipse.org/m2m/qvto/doc/M2M-QVTO.pdf>.
- [12] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002. Available from: <http://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf>.
- [13] Object Management Group. Semantics of a Foundational Subset for Executable UML Models (fUML), url = <http://www.omg.org/spec/FUML/1.0/PDF>, note = Version 1, year = 2011.
- [14] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. 2008. Version 1. Available from: <http://www.omg.org/spec/QVT/1.0/PDF/>.
- [15] Object Management Group. Ontology Definition Metamodel. 2009. Version 1.0. Available from: <http://www.omg.org/spec/FUML/1.0/PDF>.
- [16] Object Management Group. OMG Unified Modeling Language™(OMG UML), Infrastructure. 2011. Version 2.4.1. Available from: <http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF>.
- [17] Object Management Group. OMG Unified Modeling Language™(OMG UML), Superstructure. 2011. Version 2.4.1. Available from: <http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF>.
- [18] Object Management Group. OMG Systems Modeling Language (OMG SysML™). 2012. Version 1.3. Available from: <http://www.omg.org/spec/SysML/1.3/PDF>.