Notes on Development of Standards for Product Development and Systems Engineering

Product development systems engineering standards should include a specification for a language with syntax and semantics, and an interface specification for external data sources and tools, and a knowledge base (ontology) that represents standard systems engineering concept terminology. The term ontology will be used to refer to the knowledge base units that people and tools share. Product Modeling and systems engineering standards include language syntax, semantics, and the product development domain ontologies. Requirements and constraints are discussed below. SysMl/UML and OWL 2 each satisfy an intersecting, but somewhat complementary set of the requirements outlined below. Neither language satisfies all of the requirements.

1. For the language part it is highly preferable to use, or extend, an existing language that best meets the product modeling requirements. A possible solution is to figure out how to combine features of SysML and OWL into a coherent package. This could involve bidirectional translations between the languages and extensions of each language to accommodate the other (for example, UML profiles extend UML, and there is a UML profile for OWL, see the Ontology Definition Metamodel, http://doc.omg.org/ptc/2007-09-09).

2. Product Modeling languages require good syntax, and possibly multiple syntaxes. Specifically a good graphical syntax is needed, and UML/SysML is a good candidate. Textual and data interchange formats are needed also.

3. The language must contain syntax and semantics for classes, relations, composition, and individuals. These constructions can be used to represent product models including both requirements and design. Requirements and designs can be represented as product model subclasses. This enables product models to form taxonomies, for example, subclass hierarchies for the various categories of vehicle. In general, this requires constructions for union, intersection, and restrictions such as are found in OWL 1.1. Composition enables classes to to be reused independently, for example, to have engines be used in both cars and boats without requiring cars to have propellers. Both UML and OWL support classes and relations, with more precise semantics in OWL. Composition is supported more fully in UML/SysML, but has less precise semantics than the capabilities of OWL 1.1 role inclusion. UML and OWL both support individuals, but OWL is more expressive in this regard. In particular, UML/SysML has union and restriction, but not intersection. There is an OWL profile of UML that has intersection. UML can restrict the types and multiplicity of properties (called “redefinition”). It does not support qualified cardinality. OWL constructions for both universal and extensional restrictions extend the expressiveness of UML.

4. The language must have the expressiveness needed to represent classes of cars that have VIN numbers and belonging to individual people. Individual cars have data on their usage, maintenance, owner, etc, that are useful to manufacturers in getting information about how their cars are faring the real world, as well as classes of specifications that have authors, approvers, whether it's ready for production, information about what materials the product is

made of, what its parts are, how it's assembled, etc. A class of cars categorizes individual cars and has properties, like VIN number, that have values on individual cars. A class of specifications has properties such as author and material codes, which have values on individual specifications.

5. Product Modeling requires expressing behavior. UML/SysML provide a good syntax for behavior expressed in terms of state transition, activity, and interaction diagrams, as well as an operational semantics (virtual machines) for each of these to facilitate consistent interpretationi. UML/SysML do do not provide a formal semantics for behavior, but this is planned for a portion of UML Activities in the work on Executable UML (see http://www.omg.org/techprocess/meetings/schedule/Executable_UML_Foundation_RFP.html). OWL does not have a behavior model. OWL lacks special syntax to express behavior, such as state charts, as well as a model of time. The semantics of behavior is not decidable general, which violates an OWL design principle, though restricted behaviors and behavior languages might be decidable. Some behavior modeling can be done in OWL, but a fair amount of behavior modeling is well beyond OWL or even rule languages in expressiveness (see the Process Specification Language, ISO 18629, http://www.nist.gov/psl).

6. One objective for the product modeling language is to provide meaning for the terms from a subject matter expert viewpoint, rather than from an information technology viewpoint. This requires capturing common sense categorization and relations, as in a logic-based semantics. For example, axioms such as the assertion that two classes are equivalent, or that a relation is transitive, are needed to express meaning of concepts. OWL 1.1 and OWL 2 provide a logic-based semantics. UML/SysML provide some aspects of a logic-based semantics, and the UML profiles for OWL provides additional logic-based semantics. For example, the definition of generalization in UML is the same as in OWL.

7. Many engineering tasks involve reasoning. The problem of determining whether a product satisfies a design or requirements specification, or the problem of class containment of say a design model class in a requirements model class. These problems require verification. Verification typically involves both observation (measurement) and reasoning. Semantics that is sufficient to support the different kinds of reasoning is a basic requirement on the language. Reasoning at the level of classes includes the capability to show the intersection of requirement and design classes is non empty. In addition to reasoning at the level of classes, reasoning is needed to verify that an instance in a member of a class. Reasoning about individuals is needed to establish that any individual (e.g., lawn mower) that is built according to the design and operated per the requirements is an instance that satisfies the product requirements. Examples of both kinds of reasoning are in [Graves and Horrocks OWLDL 2008]. Reasoning at the level of classes is not provided by tools that use UML, SysML, or Express, as their semantics is not formal enough. It is possible the OWL profile could provide the reasoning foundation needed.

8. The standards include not only language with syntax and semantics, but a standard domain ontology. Most likely there will be multiple ontologies needed, for example there should be

domain ontologies for measurement units. The domain ontology expresses the terminology of the product modeling domain. For this one looks to standards such as STEP AP233. Such a domain ontology is often called a foundation ontology. Specific application ontologies would be extension of the PM Foundation Ontology. Again it is preferable to state requirements for the PM Foundation Ontology and select or extend an existing foundation ontology that most closely approximates the requirements. For example, Ian and I have found DOLCE works well as a starting point. There are specific requirements on a foundation ontology to represent concepts (classes) for physical objects, observable characteristics (qualities), and data value spaces. These requirements for terminology should be separated from the language itself. Specific applications will extend the ontology. For example, product verification and test requires concepts for processes, methods, as well as, reasoning about verification evidence. Experience using DOLCE reaffirms my belief in the need for a predefined terminology (foundation ontology) that can be used in for product development.

Systems Engineering standards (e.g., AP233, etc.) need to be based on a language with a formal semantics. After developing the requirements then do a comparison with OWL 1.1 and where there are differences propose a solution.

Capability requirements for product development systems engineering	Notes	selection criteria
Scalability • Evidence that implementations can handle large volume of data		
Concurrent Design • Support for collaborative engineering • Ability to manage knowledge bases as units	languages and tools need the ability to use multiple knowledge bases and combine them	Meta level features • Knowledge base (ontology) management
Interoperability • Import/export info • Both data exchange and API	Information from multiple language and tools need to be combined in a single framework. This does not mean that any single tool is practical or desirable – not only do you need to share data, but services	Data Interchange Formats • XML data interchange formats • API specification
Modeling Expressiveness • Product model structure and behavior • External environment • Observational qualities and methods for measuring them	Modeling expressiveness requires both a syntax and a formal semantics.	Features • Individuals, Classes , and Relations • Data types • Class constructors for • Behavior
Human Factors • Must have syntax that is readable • Make use of well developed syntax for special sublanguage such as state charts for behaviors	Multiple syntaxes are useful. In particular graphical syntax and syntax for sublanguages such as state transition diagrams are needed.	Syntax • Readable text syntax • Graphical syntax
Operations • Syntax checking • Checking consistency • Complete classification	The ultimate engineering tasks are to show that the final requirement and design classes have a non empty intersection and includes all properly operated instances of the product. Class-level reasoning is needed which is not currently provided by languages and tools which are UML, SysML, or Express based.	Semantics • Formal logic-based semantics – provides basis for sound reasoning • Suitably efficient theorem proving algorithms