# Ontological Support to address the Multi-Dimensionality of Complex Systems Engineering Challenges

Andreas Tolk, Matthew D. Haase SimIS Inc. Portsmouth, VA <u>andreas.tolk@simisinc.com</u> <u>mdhaase@gmail.com</u> Joe Bricio Engineer Virginia Beach, VA joebricio@gmail.com

# ABSTRACT

Every systems engineer knows about the six interrogatives: who, what, where, when, how, why. Engineers and technical managers also know what knowledge about each system has to for each phase of its life cycle, and that in each phase multiple team members and stakeholders are involved. The likelihood for communication break-downs and misunderstandings is already great for just one system and increases manifold in complex systems that are part of a portfolio. Semantic technologies allow for a more consistent approach to capture the expert knowledge associated with systems across their life cycles and provide an economical approach to enforce consistency and enable flexibility in domain specific engineering efforts. The key elements and enablers that ensure expert knowledge is captured and consistently composed and represented are ontologies, and rule sets, and reasoning logic.

Ontologies have been identified as successful support to ensure semantic consistency. This paper evaluates what ontologies can do to successfully address the multi-dimensionality of complex systems engineering challenges and makes some recommendations how a general solution may look like.

## **ABOUT THE AUTHORS**

**Andreas Tolk** is Chief Scientist at SimIS Inc. in Portsmouth, VA, and adjunct Professor of Engineering Management and Systems Engineering with a joint appointment to Modeling, Simulation, and Visualization Engineering at Old Dominion University in Norfolk, VA. He is author/editor of six textbooks and published more than 250 contributions in journals and peer reviewed proceedings. He holds a M.S. and Ph.D. in Computer Science from the University of the Federal Armed Forces in Munich, Germany.

**Joe Bricio** is an Engineer at the Maritime and Joint Systems Department NSWC Dahlgren Division/NSWC Dam Neck. He recently returned from Australia as an Exchange Engineer and Scientist Program assignment building ISO/IEC compliant life cycle engineering models for naval systems and now works with Navy International Program Office as the Cooperative Architecture, Technology and Resourcing Planning Manager. Joe is integrating semantic technology to the International Multi-Dimensional Decision Analysis Tool that will help program managers conduct cooperative opportunity assessment prior to major program milestone decisions. He holds a M.S. in Modeling and Simulation from Old Dominion University and is pursuing his PhD in Systems Engineering/ Engineering Management from Old Dominion University.

**Matthew D. Haase** is a Junior Systems Engineer at SimIS Inc. in Portsmouth, VA. He is completing his M.S. in Modeling and Simulation from Old Dominion University. He was previously employed as a Computer Scientist at Space and Naval Warfare Systems Center Atlantic in Charleston, SC. Mr. Haase holds a B.S. in Computer Science from Virginia Tech.

# Ontological Support to address the Multi-Dimensionality of Complex Systems Engineering Challenges

Andreas Tolk, Matthew D. Haase SimIS Inc. Portsmouth, VA <u>andreas.tolk@simisinc.com</u> <u>mdhaase@gmail.com</u> Joe Bricio Engineer Virginia Beach, VA joebricio@gmail.com

# INTRODUCTION

The globalization of product oriented engineering activities relies on geographically dispersed technical knowledge to coalesce under very competitive cost, schedule constraints is driving the global engineering and technical management communities to better capture and share domain specific yet context dependent knowledge. In an interconnected fabric of user, regulatory, statutory requirements directly related to system performance, interoperability and utility the classical document based approach to both the engineering but also the compliance efforts is causing potentially unnecessary miscommunication and undue risks among all participants and stakeholder efforts across the system and software life cycles. The answer to these challenges has been to move away from the document based approach and introduce model based systems engineering (MBSE) as the supporting method, such as envisioned by Wymore (1993). MBSE is driven by the same rigorous systems engineering processes as have been used before. The main difference is that the common means of communication is a set of orchestrated models that allow various viewpoints on a consistent representation of the system under development.

As it has always been the case, requirements are driving the system engineering phases: every major complex system development, from medical and healthcare systems, to personal communications and complex military realtime safety critical software intensive systems need to deal with establishing balanced requirements that take into consideration costs over the system life, regulatory compliance with various domain specific governance bodies, and the competitive edge by providing the most utility value to the end users with enough flexibility and modularity to potentially unforeseen future modes of system employment. Furthermore it needs to be taken into account that changes in the global science, technology, research, development and engineering environment across many domains coupled with the need to achieve efficiencies in product conceptualization, development, manufacturing, sustainment and retirement become main drivers for achieving efficiencies in the way we capture and share the multi-faceted nature of system knowledge. In other words, our systems will be integrated into a broader set of portfolios then we ever envisioned before. How the systems will be used and with whom they have to interoperate with is often hardly perceivable when the systems engineering processes begin.

This paper outlines key elements that address the ability of a large global engineering effort to achieve efficiencies and staying competitive by leveraging semantic technologies, augmenting the engineering effort with knowledge representation, and integrating regulatory compliance validation. Within the context of addressing system design robustness, Ross and Rhodes (2008) partially addresses the motivation to better characterize the multi-stakeholder knowledge. They recommend taxonomy based approaches to enabling system dialogue stakeholder on the system requirements and introduces the need to better structure multi-domain data and information to have the ability to have a more dynamic analysis of system robustness based on subject matter expert envisioned and anticipated future characterization. Their ideas can be generalized by using the whole ontological spectrum of semantic technologies to capture the various viewpoints in an orchestrated set of models.

This paper builds upon this concept and discusses an approach to multi-stakeholder knowledge capture and representation that is semantically tied to multi-stakeholder analytical infrastructures mediated and integrated by semantic technologies and defined ontological structures.

# UNDERSTANDING THE PROBLEM DOMAIN

The Interim DoD Instruction 5000.02 (US DoD 2015) defines the Operation of the Defense Acquisition System. It states that acquisition, requirements, and budgeting, are closely related and must operate simultaneously with full cooperation and in close coordination. Validated "Capability Requirements" provide the basis for defining the products that will be acquired through the acquisition system and the budgeting process determines Department priorities and resource allocations and provides the funds necessary to execute planned programs. Throughout a product's life cycle, adjustments may have to be made to keep the three processes aligned. Capability Requirements may have to be adjusted to conform to technical and fiscal reality. Acquisition programs may have to adjust to changing requirements and funding availability. Budgeted funds may have to be adjusted to make programs executable or to adapt to evolving validated Capability Requirements and priorities. Stable Capability Requirements and funding are important to successful program execution. Those responsible for the three processes at the DoD level and within the DoD Components must work closely together to adapt to changing circumstances as needed, and to identify and resolve issues as early as possible. This observation clearly points towards a continuous assessment of cost and technology to ensure that the warfighters need are met. Therefore, all activities defined and documented in the instruction shall ensure that solutions are provided that improve the capabilities of the warfighter within the given budget constraints. Tracing the effects of decisions (a) on costs and technology for each system over the entire life cycle, as well as (b) for the portfolio the system contributes to, is pivotal when trying to reach an optimal solution. The capability requirements process ensures that validate requirements that are reflecting the current needs of the warfighter are building the foundation for all decisions.

The capability requirements process starts with the material development decision based on the initial capabilities document (or an equivalent document). In the material solution phase, a draft capability development document is produced that leads into milestone A. The technology maturation and risk reduction phase produces the capability development document, which is the basis for the authorities to release requests for proposals and reaches milestone B. In the following engineering and manufacturing development phase the capability production document leads into milestone C, which marks the beginning of the production and deployment phase, which is followed by operations and support phases and ultimately the disposal of the system at the end of its life cycle. As explicitly captured in the Interim DoD Instruction 5000.02, capability requirements are not expected to be static during the product life cycle. As knowledge and circumstances change, consideration of adjustments or changes may be requested by acquisition, budgeting, or requirements officials. Changing of requirements needs to be accompanied by assessing cost and technology in the light of these changes.

While the capability requirements process is pretty uniform, the defense acquisition process is defined with the help of four different models and two hybrids. Model 1 deals with hardware intensive programs, model 2 with defense unique software intensive programs, model 3 describes incrementally fielded software intensive programs, and model 4 describes the accelerated acquisition program. The two hybrids focus on hardware dominant program types (hybrid program A) and software dominant program types (hybrid program B). Although the models and hybrids differ in detail, they all support the main phases known from the descriptions above. The material development decision starts the material solution analysis that leads to milestone A. The following phase is the technology maturation and risk reduction which leads to milestone B. If there are concurrent or competing technologies this phase prepares the decision which way to go. Engineering and manufacturing development leads to milestone C, which initiates the production and deployment phase. This phase is normally started with some low-rate initial production to support operational testing and evaluation to ensure the initial operational capability. If this is successful, the full operational capability is produced and the system is sustained and eventually disposed. In particular for the milestones it is pivotal to critically reflect cost and technology in the light of the accomplished results and the updated requirements. The enclosures provide more detail on these and additional decision points.

In this domain of major complex system development, managing the inherent communication challenges that are the subject of typical systems engineering practices is complicated by the structural, spatial, organizational, and temporal separations between teams, team members, and life cycle phases. The potential for miscommunication threatens the understandability and repeatability of the life cycle processes, as differently-tailored processes and terms are harmonized on an ad-hoc basis to the extent they are harmonized at all. Further challenges arise during iterations of the system design in ensuring that stakeholders affected by changes are informed and understand the import of the changes proposed. The international standard ISO/IEC IEEE 15288 addresses the system life cycle processes. On the organization level, project enabling processes are needed for the life cycle model management,

infrastructure management, project portfolio management, human resource management, and quality management. On this level are also agreements made regarding the acquisition of needed elements as well as the supply chain to and from other organization. These organization wide processes are the foundation for project specific processes that coordinate project planning and project assessment and control, supported by decision management, risk management, configuration management, information management, and agreements on measurements and metrics processes. Within this framework, the technical engineering processes can be conducted to successfully design, develop, use, and retire a system. These technical processes define stakeholder requirements definition, requirement analysis, architectural design, implementation, integration, verification, transition, validation, operation, maintenance, and disposal. In each of these 25 life cycle processes, team members of various domains, military commands, and stakeholders are working on their particular tasks. For software intensive systems, the ISO/IEC IEEE 12207 defines the same 25 life cycle processes and adds additional six software implementation processes, eight software support services, and three software reuse service to the mix.



Figure 1: System Life Cycle Processes defined in the IEEE/ISO/IEC 15288

While the software engineering processes clearly articulate the need for the orchestration of all efforts, the traditional document centric approach is insufficient. Even with optimal governance of all processes it is unlikely that all documents, manuals, and technical descriptions always directly represent the current status of the system, its components, or its environment. As an answer to this problem, model based systems engineering (MBSE) was proposed. The International Council on Systems Engineering (INCOSE) defines MBSE as "the formalized application of modeling to support system requirements, design, analysis, verification and validation, beginning in the conceptual design phase and continuing throughout development and later life cycle phases." Driven by requirements, functional and behavioral models, performance models, structural and component models, as well as other engineering and analysis models are orchestrated based on a common system model that provides the common ground of the analysis of alternatives and solutions. For the full benefit, MBSE shall be applied across all phases of the system life cycle. In other words: instead of using documents to ensure a consistent view of the system under development, an orchestrated set of models is used. If to views share the same information element, an update of this element is immediately transparent to all other users of the element. Furthermore, if the change would trace the violation of a constraint in another domain, the author of this change can be informed about this violation, and the change may eventually be forbidden. Also, if requirements are changing, the set of models allow to trace which constraints and solutions are affected by such a change. Overall, experts have the chance to better understand the consequences of their decisions not only in their domains, but also in the many domains that are influenced by the work in other teams, or even in other life cycle phases of the system.

Only in few and trivial projects, the underlying model will be a single structure. As discussed above, we will have many experts from various domains and different background contributing to the development of a system. In addition, the system is likely part of a portfolio that comprises additional systems that by themselves are still under development. If the procurement is conducted by multiple nations, more challenges have to be addressed. It is therefore highly likely that a significant amount information and knowledge is captured in a multitude of models that are not necessarily interoperable with each other. Semantic technologies in general, and in particular ontologies, rules and reasoning logic, offer solutions to these challenges.

## WHY SEMANTIC TECHNOLOGIES?

Semantics is part of the semiotic triad made up of syntax, semantics, and pragmatics. Syntax is focusing on the structure of information, semantics deals with the meaning, and pragmatics with its application and use. Semantic technologies are therefore concerned with ensuring that information is consistently interpreted among different and distributed applications. Semantic technologies became well known by the development activities on the Internet to establish a semantic web (Berners-Lee et al. 2001, Shadbolt et al. 2006). The semantic web stack identifies several layers of technologies and protocols that support the implementation of this vison.

The set of standards allowed moving towards a new level of machine support and automatic reasoning in the recent years was never been possible before. In addition, the technologies are continuously further developed and exchanged within the semantic web community to foster agreement and improvement.





The basis for general communication in the web is the use of a machine and vendor independent codification allowing the interpretation of signals as symbols. This is supported by the use of UNICODE as the lowest building block. The second foundational leg is the use of Uniform Resource Identifiers (URI). Everything on the web becomes a resource and can be addressed as such in a standardized way. The Extensible Markup Language (XML) provides the elementary syntax for resources, which means allows communicating structure within a resource that can be search for, extracted, etc. It is possible to provide and restrict the structure and content of elements through the use of an XML Schema. Another approach that is seemingly gaining more and more support is Turtle, which is XML independent, providing an alternative. The Resource Description Framework (RDF) allows to structure resources and their relations similar to using a data model. Most of these frameworks use XML, but that is no longer necessary, as alternative exist. However, whenever data need to be exchanged in a consistent way, which includes

the idea of transactions on one or several resources, support of a common RDF is needed. RDF organizes resources as triples, which provide the structure for higher operations. While RDF is more a data model, the use of an RDF Schema (RDFS) compares to the definition of a taxonomy of terms describing properties and classes of RDF resources. It allows building hierarchies and complex relationship using well-defining types of associations connecting well-defined structured tagged with common terms. While RDF works with general triples, RDFS standardizes the most important of these triples as recognized by the community. The Web Ontology Language (OWL) is an extension of RDFS. It adds more standardized terms describing resources, properties, and relations, such as cardinality, equality, typing, enumerations, and more. OWL allows reasoning over these extensions, if all supporting resources follow the standard accordingly.

Queries about resources do not require that the resources are documented in RDFS and OWL. The SPARQL Protocol and RDF Query Language (SPARQL) allows searching for triples, conjunctions of triples, disjunctions of triples, and optional patterns. As such, SPARQL works with RDF, RDFS, and OWL. The Rule Interchange Format (RIF) is not yet standardized, but only recommended. It allows the communication of constraints within the semantic web stack to ensure consistency between the layers. An alternative for OWL based resources is the Semantic Web Rule Language (SWRL) that utilizes description logic subsets of OWL.

The use of OWL should not be decided without proper consideration of the issues of decidability and computational complexity. Not all dialects of OWL are decidable, and even if they are decidable, their application can lead to non-polynomial computing time. Within our projects, we were focusing on OWL-DL, which is decidable, with particular interest in the profiles defined for OWL 2: OWL 2 EL, OWL 2 QL, and OWL 2 RL. A good review of available semantic web tools and standards and their computational constraints is given by Hitzler et al. (2009). The reason why OWL is our recommendation for MBSE becomes obvious when we evaluate the ontological spectrum, as it was published by Daconta et al. (2003): OWL is more expressive than most alternative methods that are used to design and develop models while it remains decidable by a machine. As such, it has the potential to become the *Lingua Franca* for the description of models and allows to apply computational logic to reason over such models. The following figure shows the ontological spectrum using examples of currently applied web methods.



**Figure 3: The Ontological Spectrum** 

Practically, OWL allows to describe every model defined in every modeling language and method that is on the same or on a lower level within the ontological spectrum. This includes technical manuals and specifications, artefacts captured within the DoD Architecture Framework, or model description based on the Unified Modeling Language (UML). It is also applicable to data models, interface specifications, and so forth. In other words: OWL becomes the common language that allows to describe and express all models that are part of the orchestrated sets of models needed for MBSE. Once all models are captured in OWL, and once all concepts in all participating models are expressed using a common controlled vocabulary, the full power of an ontological description can be applied.

The definition of the controlled vocabulary is a problem on its own and deserves a complete research project to cope with this in detail, but it can easily be recommended to start with the IEEE/ISO/IEC 24765, which defines the vocabulary to be used for systems and software engineering standards, as well as terms defined within the National Information Exchange Model (NIEM), that addresses multiple application domains of interest.

One possible use case is to identify any inconsistencies between the models, or even within one model, simply based on logical descriptions that are machine readable: OWL can identify logical inconsistencies by using logical reasoners checking the knowledge based provided by the ontological representation of the set of tools. If one tools contains the logical fact that a system specification is within a desired cost frame why another tools contains the fact that this is not the case, the reasoner presents this inconsistency to the users to come up with a solution. Another use case is to compare to what degree two processes are isomorph, or at least if they can be mapped to each other. An example is to let a reasoner find out which life cycle processes conducted following the Interim DoD Instruction 5000.02 are supported by processes defined in the ISO/IEC IEEE 15288. A simpler use case is to check if all required data for a certain process has either been provided by another subject matter experts in another team or domain, or if it will be produced by another process, or if the data are not yet defined.

The following section gives some examples derived from an ontology project conducted for our customer that proved the feasibility of such conceptual ideas.

# **APPLICATION EXAMPLES**

As part of an effort supporting our customer, we ontologized a procurement support and analysis tool under development to support systems acquisitions. The goals of the effort consisted of first, establishing a formal, structured understanding of the system in its current state, including the necessary inputs, outputs, and resources in order for it to be employed; and second, identifying gaps and opportunities where the system could be integrated with other model based system engineering applications in the acquisition portfolio.

As our customer was from the defense domain, we chose to employ the Department of Defense Architectural Framework (DoDAF) Meta Model (DM2) as the upper ontology for the domain ontology, which consisted of three parts: a data ontology, a functional flow ontology, and a software architecture ontology. The data ontology described the format and part-whole relations of the explicitly defined input data structures. It also captured data structures that were either not mentioned or not fully described in the system documentation, but whose existence and necessity could be inferred from the operation of a whole. The functional flow ontology captured the activities performed by the system from an operator-level perspective. It leveraged the constructs of the data ontology to describe inputs and outputs of the activities. It also linked activity performers (namely, personnel) with the activities they support. The system architecture ontology described the topology of the system from a system- and service-level perspective. It mapped resource flows (including the data structures described in the data ontology) between systems, services, and users and documented the standards to which the system must conform. Figure 4 illustrates the relationships between the four ontologies.



Figure 4: Upper and Domain Ontology Relationships

Uschold & Gruninger (1996) suggested that ontologies could be applied in Systems Engineering in support of specification, reliability, and reusability. Regarding specification, they argued that ontologies could be used

informally for requirements gathering and system conception followed by formally for capturing a software system specification. They listed the applications supporting reliability as consistency checking, i.e. comparing a system design to an ontologized specification, and capturing and tracing assumptions on which various components rely. Similarly, by explicitly capturing assumptions inherent to previously encountered problem domains, they argued that ontologies can support the decision on when an existing tool or solution can be applied to a new domain, and what modifications would be necessary for a successful integration. We share their views and extend this list to include employing ontologies to allow users to understand the pre-conditions that are necessary to employ an existing system. In the following sections we discuss the areas where we applied the developed ontologies, along with areas where we see potential for future applications. We conclude with lessons learned during the project.

# **Ontological Applications**

## Verification of System Consistency

The first application of the ontologies was verifying that the system design, as captured in the ontologies, was internally consistent. This was accomplished automatically upon enabling a reasoner on the completed ontological structure, as any inconsistencies would have been immediately flagged as errors.

We did not expect to find any inconsistencies, as the model referent was an already existing, functional system – presumably, any inconsistencies would have prevented its operation. However, the consistent ontological representation provides formal evidence that the design is appropriate in theory as well as practice, as well as establishing a consistent foundation for later system extensions. However, we could use the ontological representations of DoDI 5000.02 and ISO/IEC IEEE 15288 to evaluate which phases are supported.

# **Identification of Task Connections**

The second application was to derive the preconditions that enable the system to be used, especially those related to the performers that are required for an activity, with the goal of allowing operators and prospective operators to identify the data and support team they would need in order to employ the system for certain activities. We found in course of ontologizing the system that the system documentation often contained non-explicitly declared assumptions about the state of the system prior to use. Using the ontologies enabled these implications to be made explicit. For example, certain system activities require custom models to be created in order to perform analysis. These models must then be converted into executable models using a scripting language and then loaded into the system. The user documentation assumes the existence of these models and doesn't mention the need for an experienced modeling support team to create them. Without these models, only parts of the functionality can be provided. The ontologies were able to document this requirement, and include it in the requirements for not only the activity in question, but for all the activities that follow the initial activity.

# **Future Applications**

The following use cases are planned applications identified in expert discussions in preparation of the project, where we believe the ontologies will provide significant value when employed under the given conditions.

#### **Identification of Applicable Standards and Tools**

This application involves using the ontologies to derive standards, systems, and structures that are appropriate for particular system activities and tasks. As the system referent evolves and its applications change, new desired capabilities are captured and added to the ontologies. This allows system engineers to determine whether existing structures within the system can satisfy the input and output requirements and conform to constraining standards, saving time and money.

#### **Ontology-Driven Engineering**

It is likely that new capabilities introduce new functionality that cannot be provided by the existing system tools. This application is the complement of the above application – instead of identifying the right tool for a process, we use the ontology to identify steps in the process that currently are not supported by a tool. The ontology allows a team to derive what an appropriate tool or system would need to accomplish to fill the gap – the inputs it must consume, the outputs it produces, and the applicable controlling standards. In other words, we use the ontology to identify gaps as well as enumerate the already known requirements for the tool. Lin & Harding (2007) already demonstrated that this idea is feasible for systems engineering projects.

## **Model Interoperability**

Finally, the ontologies will be leveraged to support system interoperability with other modeling tools. Each tool contains its own modeling conceptualization. In order to be a valid integration, these conceptualizations must be shown to be compatible; or, if they are incompatible, the conceptualizations must be harmonized. Ontologies can capture these conceptualizations in a formal way, and as previously described, they can also be reasoned over to provide strong evidence that conceptualizations are indeed compatible, as envisioned by Tolk and Miller (2011).

## Lessons Learned

Despite their capabilities, semantic technologies are not panaceas. In order to be effective, they must be employed carefully. In this subsection we discuss some lessons learned from previous experiences employing semantic technologies.

## **Expectation Management and User Education**

Because of the relative novelty of using semantic technologies in systems engineering, a key task of the engineering team must be educating clients about both the potential and limitations of these technologies. Many potential users are unaware of ontologies and how they can be employed. The engineering team must balance the explanatory power provided by the ontology with that required to describe it to customers and enable its utilization by clients.

Despite a vibrant and growing online community of practitioners, ontology authoring tools, such as Stanford's Protégé (Noy et al. 2001), are still maturing and require a significant amount of technical sophistication to employ properly. Authors and users must "get into the guts" of the ontology in order to formulate non-trivial queries, and the terminology and functionality is often presented in a way that can cause confusion for users not already immersed in ontological concepts. As ontology authoring tools mature, some of these usability issues will disappear, but today they remain a significant barrier to entry.

Beyond the issue of usability of tools, the act of authoring ontologies is a sophisticated engineering task. An ontology is not merely a dictionary of a controlled vocabulary (though it contains a controlled vocabulary) or a database that can be queried for information (though it will answer queries.) Rather, ontologies are powerful and complex modeling instruments. Depending on the audience and intended usage, a directly-accessible ontology may not be the best tool to provide. In such cases, it may be better to leverage the ontology to generate the information, and then present it in a more appropriate form.

#### Need for Multidimensional Ontological Representation

Most people already know about the need for upper ontologies and a domain ontologies to express general terms and concepts, including the controlled vocabulary, as well as the details and specific of the application domain. Hofmann (2013) introduced an additional line of thought of interest, namely methodological ontologies (HOW we express something) and referential ontologies (WHAT we express). The leads to a matrix of ontologies needed to address the diversity of domains as well as methods used to describe them. For the practice this means that only because two groups use ontologies in OWL doesn't mean that we can easily exchange information that makes sense.

#### Logical Constraints

The nature of logic introduces certain constraints on how ontologies may be employed. We found that the logical characteristics of classes impacted how we utilized the domain ontologies to identify task connections and had implications for how the ontologies must be constructed. Specifically, inverse relationships can not be automatically inferred between classes. Figure 5 illustrates an example scenario and explains why this is the case.

This limitation naturally impacts the usefulness of a class-only ontology for discovering non-obvious task dependency chains and single points of failure. One possible solution would be to explicitly declare each inverse relationship when the primary relationship is added. However, this negates some of the labor-saving advantage of using a reasoner. It also introduces modeling errors in many (perhaps most) cases. To continue with the example given in Figure 5, it makes sense that all **LoadBalancingSystemFunctions** must be performed by some system component, else the function would not be a function. On the other hand, performing a **LoadBalancingSystemFunction** is not inherent to all **TaskProcessors**; perhaps some **TaskProcessors** perform other tasks or none at all. Declaring that all **TaskProcessors** perform some **LoadBalancingSystemFunction** would be inaccurate.

Our proposed solution was to instantiate the ontology by creating individual members in each class, and then relating those instances to each other to capture individual characteristics. Among other things, this allows the reasoner to infer inverse relationships between individuals. However, in many cases instantiating an ontology entails essentially duplicating all the work done in capturing the class structure of the system.



LoadBalancingSystemFunction and TaskProcessor are classes taken from the domain ontology and are represented here by circles. The object properties (relationships) 'activityPerformedByPerformer' and its inverse, 'performerPerformsActivity,' are represented by solid and dotted lines, respectively. Instantiations (individuals) of the classes are represented by diamonds. In the domain ontology, all LoadBalancingSystemFunctions are performed by some TaskProcessor. This is captured by making LoadBalancingSystemFunction a subclass of the anonymous class 'activityPerformedByPerformer some TaskProcessor'. Note that the inverse function 'performerPerformsActivity' is not inferred between TaskProcessor and the anonymous super-class, as this would imply that all individuals in TaskProcessor must perform the given activity, rather than only some of them. Once the individuals are related, however, the inverse relationship can be inferred.



#### **Limits of Representation**

There exists a trade-off in ontological engineering between ontological expressiveness and computational decidability. In order to maintain decidability in polynomial time, ontological languages may impose limitations on the range of relationships that can be described between entities. This can result in perfectly reasonable and intuitive ontological constructs which nevertheless are not representable in a decidable form. For example, DM2 classifies things as either individuals, types (sets of individuals; classes), or tuples (ordered relationships.) This corresponds well with the OWL concepts of individuals, sets of individuals, and property assertions describing relationships between individuals and sets of individuals. However, DM2 allows tuples to contain other tuples within their tuple space (i.e., a relationship can relate two relationships, or a relationship to a class or individual.) This is disallowed in OWL; property assertions can only be made between individuals and classes. Certain other types of relationships are also disallowed in OWL – for example, a property that contains a transitive sub-property it not legal.

These limitations have ramifications on the choice of upper ontologies. If an upper ontology is selected that cannot be represented in a decidable form, the modeler will need to either use a subset of the upper ontology when constructing the domain ontology, or will need to need to re-implement the upper ontology in a decidable way. Obviously, the latter choice is undesirable when attempting to maintain consistency across multiple ontological structures.

## **CONCLUDING REMARKS**

Ontologies have been proven to be a powerful tool to capture knowledge in machine readable form. They are expressive enough to describe the content of documentations and manuals, the structure of databases, and model descriptions using the Unified Modeling Language UML or the System Modeling Language SysML. They can utilize and express XML-based information, including the semantic technologies used on the Internet. As such, they are a strong candidate to become the common language to express knowledge between experts from all different team members and stake holders from all life cycle phases. This is true for the system under consideration as well as for all other systems that are part of the portfolio and all systems with which the system will interact.

In order to apply ontologies, a highly educated workforce is needed. In ontology expressed in OWL is of little use to the user if he is not supported by a variety of tools that support the work with it. In particular good visualization tools are needed that display the information, in particular dependencies and other relations. The power of ontologies is furthermore to apply logical reasoners that can check the knowledge base for consistency and derive new information from the current descriptions. Again, the user needs guidance provided by suggestive interfaces helping to formulate the questions he is interested in. Despite improvements over the recent years, the user friendliness of OWL needs further improvement to make it acceptable and applicable in the broader context of systems engineering.

In order to support MBSE on the broader level of system of systems engineering, where the individual systems are operational and managerial independent, geographically distributed, and are developed evolutionary (Maier 2015), the application of standards is highly important, but constraint by the lake of a common governance: a common standard for data, process, and constraint modeling and engineering will never be able to be enforced, which will also lead to differences in content and methodology. Ontologies can capture the content, the methodology, and provided a framework to evaluate if these different approaches and systems are compatible enough to work together. They provide the necessary transparency to allow for collaboration without giving the autonomy of the individual systems away, which is pivotal when systems have to work over the organizational or national borderlines.

## BIBLIOGRAPHY

Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The semantic web. Scientific American, 284(5): 28-37.

- Daconta, M.C., Obrst, L.J., & Smith, K.T. (2003). *The semantic web: a guide to the future of XML, web services, and knowledge management.* Hoboken, NJ: John Wiley & Sons.
- Hofmann, M. (2013). Ontologies in Modeling and Simulation: An Epistemological Perspective. In *Ontology, Epistemology, and Teleology for Modeling and Simulation* (pp. 59-87). Springer Berlin Heidelberg.
- Hitzler, P., Krötzsch, M., & Rudolph, S. (2009). Foundations of semantic web technologies. Boca Raton, FL: CRC Taylor & Francis Group.
- INCOSE. (2007). INCOSE Systems engineering vision 2020, Report INCOSE-TP-2004-004-02, September.
- ISO/IEC IEEE 12207 2008. Systems and software engineering: software life cycle processes.
- ISO/IEC IEEE 15288 2008. Systems and software engineering: system life cycle processes.
- ISO/IEC IEEE 24765 2010. Systems and software engineering: vocabulary.
- Lin, H. K., & Harding, J. A. (2007). A manufacturing system engineering ontology model on the semantic web for interenterprise collaboration. *Computers in Industry*, 58(5), 428-437.
- Maier, M.W. (2015). The role of modeling and simulation in system of systems development. In L.B. Rainey & A. Tolk (Eds.), Modeling and simulation support for system of systems engineering applications, (pp. 11-41). Hoboken, NJ: John Wiley & Sons.
- Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., & Musen, M.A. (2001). Creating semantic web contents with protege-2000. *IEEE Intelligent Systems*, 16(2), 60-71.
- Ross, A., & Rhodes, D. (2008). Architecting systems for value robustiness: research motivations and progress. *Second Annual Systems Conference IEEE 2008* (pp. 1-8). Piscataway, NJ: IEEE.
- Shadbolt, N., Hall, W., & Berners-Lee, T. (2006). The semantic web revisited. Intelligent Systems 21(3): 96-101.
- Tolk, A., & Miller, J.A. (2011). Enhancing simulation composability and interoperability using conceptual/ semantic/ontological models. *Journal of Simulation*, 5(3), 133-134.
- US DoD. (2015). (Interim) DoD Instruction 5000.02, Operation of the defense acquisition system. January 7.
- Uschold, M., & Gruninger, M. (1996). Ontologies: principles, methods and applications. *Knowledge Engineering Review*, 11(02), 93-136.
- Wymore, A.W. (1993). Model-based systems engineering. Boca Raton, FL: CRC Taylor & Francis Group.