OWL for Space Mission Systems development at JPL with semantic architecture styles

Nicolas F. Rouquette[1], Gary M. Wasserman[123], and Vanessa D. Carson[14]

[1] Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109, USA
{nicolas.f.rouquette,gary.m.wasserman,vanessa.d.carson}@jpl.nasa.gov
[2] Department of Computer Science
University of California, Davis
[3] Kestrel Institute
3260 Hilview Avenue
Palo Alto, California 94304
[4] Department of Mathematics
University of Southern California
Los Angeles, California 90089-2532

Abstract. The Jet Propulsion Laboratory has over 40 years of experience in engineering complex space mission projects for NASA. This paper presents a perspective on the challenges involved in using OWL for representing the diverse heterogeneous viewpoints involved in the engineering and operation of space missions.

1 Introduction

Over more than 50 years, the Jet Propulsion Laboratory has had a solid track record of successes across a wide range of robotic space missions for NASA and other agencies. These accomplishments have enabled us to stretch the frontiers of scientific knowledge about space by leaps and bounds. Each leap in science raises expectations for corresponding leaps in engineering capability and know-how to baseline current capabilities as a stepping stone to deliver the additional capabilities necessary to tackle the next round of scientific inquiry. What makes NASA and space exploration so unique are the severe constraints under which this spiral of scientific and engineering knowledge growth takes place: inherently high-risk one-of-a-kind engineering artifacts designed with increasingly tight budgets and whose return-on-investment is measured on a single multi-year product launch and where product families arise from opportunistic success instead of architectural intent. This complex environment creates unique short-term and long-term application opportunities for ambitious vision of the semantic web. Currently, the success of space missions depends critically on intangible factors such as experience, wisdom, craft, and particularly a spirit of out-of-the-box thinking to anticipate where the limits of our scientific knowledge are such that we can

engineer space systems that, while operating safely within these limits, will give us precious window seats on the scientific discoveries ahead. Limited budget and schedule resources mandate improvements in engineering practices for space mission design, development and operations.

The overarching challenge is thus a complex optimization problem. The controlled variable is the engineering methodology itself. Convincing engineers to switch methodology would require a strong incentive that outweighs the practical inconveniences and setbacks for doing so. Ultimately, the effectiveness of an engineering methodology depends on the degree to which it helps engineers evaluate, compare non-functional attributes such as reliability and assurance in terms of the characteristics of the design itself and of the engineering work that produced it. The major hurdle in this area stems from the intrinsic heterogeneity of the various kinds of models involved in a typical space mission project life-cycle. The semantic web provides the only perspective in which we can unify the multitude of scientific, engineering and organizational objectives involved in a space mission. The optimization analogy from control systems makes sense only in the context of an overall architecture for describing space missions that accounts for enough viewpoints from science and engineering such that the optimization problem can be precisely stated. Without a single unifying perspective, it make sense, therefore, to view this problem from the semantic web perspective where plurality is the norm. Within a narrow field of engineering, the heterogeneity inherent in specialized engineering model representations can be resolved by mapping the semantics of each specialized model representation to a common upper ontology that provides a coherent, semantics-preserving abstract representation of all specialized models [1]. On a broader scale, it is unlikely that a single upper ontology can accommodate the multitude of concerns and objectives found across the many disciplines involved in a space mission project. In [1], the authors contrast their Product Specification Representation Language (PSRL) ontology with the closely related Process Specification Language (PSL) ontology[2] as a distinction between specifying the manufactured product itself and the manufacturing of that product. The concerns and objectives that reflect the different viewpoints used for a particular concept (e.g., "product" in PSRL & PSL) induce yet another level of semantic heterogeneity across the multitude of viewpoint-specific ontologies. Handling semantic heterogeneity involves a pervasive concept identification problem: how to distinguish when two ontological representations from different viewpoints refer to the same individual or to distinct individuals.

Concept identification criteria is one of the important issues in ontology development discussed in the OntoClean methodology[4] for analyzing relationships among taxonomic concepts in a practical manner with tool support for concept annotation and reasoning. This methodology represents a step in the right direction for tackling semantic heterogeneity across viewpoints. To capture subtle differences among viewpoints such as distinguishing a product as an engineered artifact in PSRL's viewpoint vs. the outcome of a process in PSL's viewpoint, we need ontological knowledge about how particular descriptions of a concept in a

given viewpoint relate to the concept itself, i.e., the criteria that give its identity as a recognizable trait across all descriptions. These issues have been extensively researched within the WonderWeb project's development of the DOLCE library of foundational ontologies[6]. DOLCE makes an important distinction between *descriptions* of generic state of affairs involving a set of inter-related individuals and a particular *situation* of this state of affairs occurring at a specific time. The methodology for applying DOLCE's Description&Situation ontology is more complex than OntoClean since it forces the ontology developer to think carefully about the ontological nature of each domain-specific concept separately from other semantic representations of that concept for purposes of description and situation across multiple viewpoints.

The separation of description and situation in formal ontology is particularly relevant to the "Recommended Practice for Architectural Description of Software-Intensive Systems description" standard[9]. This standards explains heterogeneity in architecture descriptions and models using concepts of multiple viewpoints, concerns and objectives across stakeholders. On a practical level, heterogeneity leads to an entropy of multiple specialized models and descriptions of systems difficult to map to one another because schedule pressures prevent most engineers to take the necessary steps to seek out common ontological abstractions like PSRL in mechanical engineering. Experience in applying the IEEE-1471 recommendations gives hope that reducing the entropy of descriptions involved [11] can be practical. Little is available to switch from a recommendation to useful methodology guidance. Beyond IEEE-1471[10], the RM-ODP standard[13] offers a conceptual framework that, although incompletely addressing the space domain, provides nonetheless a useful categorization of concepts for ontology development[14]. Work on formalizing RM-ODP in logic [15] argues for defining modeling elements using a combination of basic modeling concepts (e.g., object, action, state) and specification concepts (e.g., type, instance, composition). Their approach shows a step in the right direction towards formalizing heterogeneous models with enough rigor to properly reflect the differences across viewpoints in the RM-ODP, the source of heterogeneity. In fact, it is tempting to consider replacing the basic modeling concept ontology in [15] with DOLCE's richer upper ontology since the former seems to be a subset of the conceptual taxonomy of the latter. A richly axiomatized concept taxonomy might seem more complex to use in practice than an informal taxonomy or even an ad-hoc folksonomy; however, the value the former is in the degree to which all ontological considerations relevant to the application domain have already been factored in a way that provides useful guidance and an authoritative reference for modeling. Earlier work at JPL on the Mission Data System project[23] would suggest extending the specification concept ontology to properly account for the multiple themes[21] as distinct viewpoints on the central notion of "state". DOLCE's top-level distinction between endurants and perdurants provides a semantic basis for making the distinction between "state" and "state type" where the former refers to an endurant (i.e., it is an characteristic of a system at a particular time) and the latter refers to a perdurant concept that describes the class of endurant states

whose characterization of a system is specified in terms of semantically equivalent domain-specific information criteria (e.g., temperature, voltage, current, etc..).

MDS includes concepts of architecture like estimation, control, determination that, when combined with domain-specific typing combine to form an architecture style rich in constraints[23]. At JPL and elsewhere[16,17,18], significant efforts were made to capture the MDS architecture style using a variety of formalisms, ADLs[25,28]and code generation techniques to maintain the consistency between an architecture specification and its implementation. The JPL architecture framework closely mimicked UCI's xADL framework using advanced generic programming techniques in C++ and a code generation techniques[20] most of which are currently found in UML 2.0's notion of structured classes with ports and in OMG's Model-Driven Architecture initiative for model-based transformation. SRI's approach[18] based on Maude's Equational and Rewriting Logic[30] makes a strong case for using formal Architecture Description Languages instead of informal ones. Their conclusions combine synergistically with those reached in [19]. Perhaps ACME and its constraint language, Armani[27], can be seen as early precursors[28] of a trend towards formalizing ADLs in a practical tool-supported methodology. This trend has gained significant momentum in recent years with assume-guarantee reasoning techniques[31] whose applicability to space mission systems has already been demonstrated[32]. Perhaps the outcome of this trend may be an emphasis on specifying architecture components in formal logic[33] to better relate the specification and operational aspects of components as modules as suggested in [29] for the Maude system.

A highly rigorous methodology like [33] seems very attractive in light of earlier experiences [19] suggesting that, for ADLs, formal methodologies can yield practical advantages over informal methodologies. Yet, even [19] would require extensions to provide adequate methodology support for MDS-like specifications. At JPL, the C++ framework for state analysis was designed to facilitate design-time and runtime verification based on ideas borrowed from xADL such as the distinction between the *prescribed* and *described* architecture. The former is equivalent to a DOLCE perdurant description concept. The latter is equivalent to a DOLCE endurant situation concept that reflects the architecture of the system at a particular time. Runtime architecture description in xADL requires introspection mechanism to ensure that the architecture description reflects the system's current architecture configuration across runtime operations that modify the architecture – e.g., creating and deleting components/connectors, binding component/connector ports and a partial implementation of OMG's Corba3 concept of interceptors with the capability to add/remove interceptors on component/connector ports for wrapping method calls (before, after, around, modify) and for handling/rerouting exceptions thrown. Interception is a powerful mechanism that is somewhat unique to JPL's C++ implementation of the MDS framework for State Analysis compared to other implementations in C++[16], Java[20,17,19] and Maude[18]. At the time this feature was designed generic programming techniques in C++[34] based on static metaprogramming[36] were

notorious for their excessive template stress factor on C++ compilers. The initial motivation for adding interception to MDS turned out to be a particularly important feature in subsequent investigations of dynamic processor scheduling techniques in MDS[37]. Here, interception provides a useful mechanism that decouples the design of a software architecture (without interception) from architecture aspects woven into the architecture to add extra functionality (e.g., dynamic scheduling) compared to the original architecture (e.g., static scheduling). In particular, weaving a dynamic scheduler requires intercepting the activity of major components to include usage measurements as well as extracting from the architecture prescription metadata about dynamic scheduling properties such as a-priori utility ranking in our experiments.[37]. While powerful, the C++ runtime system for MDS and the metaprogramming tool support were unwieldy and difficult to scale. In part, these issues reflect the antiquated nature of C++'s runtime system that has, since then, been significantly extended in a recent revision of the standard. On one hand, the new C++ CLI would make a number of architecture issues like interception support for dynamic scheduling dramatically simpler. On the other hand, there would still be fundamental issues of insufficient formalism in State Analysis' architecture.

2 Architecture style squabbles that amount to things that matter

Motivated to find ways to increase both rigor and pragmatism, later work switched to term-rewriting[39] techniques using a high-performance infrastructure[40] that makes generative programming positively rewarding and extremely agile despite being harder to advocate with project managers. DMS is a language/domain neutral infrastructure for transformation. Initial experiments focused on reverse engineering the xADL architecture prescription and description models used for JPL's Rocky7 experimental rover into ACME models where JPL and CMU collaborated to develop an architecture style for State Analysis. Master students from CMU developed a simplified version of State Analysis[17] whose main conceptual difference with JPL's State Analysis database and its C++ runtime system in MDS is in the strategies for handling type information associated to the State Analysis specification concepts: state, estimator, controller, measurement, command, etc... In [17], the state analysis concepts are implicitly typed in the sense that the domain-specific information that distinguishes a temperature state from a position state is in the values stored in the state object, e.g. instead of being part of the signature of the state object. Specifically, the architecture style for State Analysis in [17] uses a set of component types that is domain-agnostic, i.e.,: *StateVariableT, EstimatorT, ControllerT, ...* In contrast, the initial C++ implementation in MDS used a set of generic component types parameterized by the domain, i.e.,: *template<type T> class StateVariable<T>;, template<type T> class Estimator<T>;, ...* Initial efforts at JPL to develop an architecture style for State Analysis in ACME attempted to follow the C++ generic approach. Unfortunately, this quickly led to unwieldy and com-

plex architecture styles; paradoxically because these efforts made naive use of Acme's Armani language for modeling architecture style constraints in logic[27], a specification-level variant of the infamous architecture mismatch pitfall[26]. At the time, this mismatch was viewed in the context of language wars at JPL: C++ detractors jumping on assertions that template metaprogramming amounts to madness[5] and Java detractors claiming rampant object-oriented polymorphism pay little attention to important architecture matters that induce high risk for spaghetti-like re-entrant callbacks and fragile base classes that jeopardize overall reliability[35]. These skirmishes of implementation-level programming arguments hide an important issue in formal architecture specifications: how to decide whether domain-specific type information *must* be part of the architecture style or *separated* from the architecture style.

In the MDS project at JPL, C++ detractors reached enough critical mass to make a significant revision of the C++ implementation framework. Inspired from the GoldenGate collaboration project with Sun to develop and demonstrate a framework for State Analysis in Real-Time Java, the C++ framework was refactored using a polymorphic base class for domain-specific state values. This *simplified* the State Analysis concepts from static metaprogramming using C++ templates to dynamic polymorphism using C++ RunTimeTypeInference (RTTI) mechanism: i.e., from: *template<type T> class StateVariable<T>;* to simply: *class StateVariable;*. However, this shifted the problem to the implementation of achievers, i.e., controllers and estimators. An estimator typically processes input measurements from sensors and queries state variables to compute updates for the state variables it is estimating whereas a controller typically queries state variables to evaluate its control law and compute output commands to actuators.[23]. In a polymorphic architecture style for State Analysis, whether it is implemented in Java, C++ or any other language, it is architecturally important to ensure that the prescribed architecture is properly wired to ensure that an achiever that queries a state variable of a given domain-specific type (e.g., temperature) will in fact be connected to query a variable of that kind. With the C++ framework, this requirement was enforced as early as possible to avoid checking it at every computation. Thus, the C++ compiler could enforce that static architecture prescriptions were type consistent and the runtime architecture framework could enforce that any component/connector binding operation is type consistent, whether such operations occurred at the initial construction of the runtime architecture from the prescription model or from subsequent changes due to runtime modifications via editing bindings or interception rerouting. With the simplified state analysis design, the compiler and the runtime architecture framework have a polymorphic view of state values disconnected of any domain-specific type information. In this case, the responsibility for domain-specific type consistency shifts to the implementation of the

[5] Stanley Lippman briefly consulted at JPL and upon review of the author's C++ code had only one friendly adjective to describe the programming style behind it: *mad*. This is considerably more gentle than other descriptions associated with Alexandrescu's style[34] from which the MDS code was largely inspired.

achievers (i.e., controllers and estimators). Therefore, a practical consequence is that implementers have to include runtime type checking code to fulfill type consistency requirements.

Conceptually, type checking code is an incidental detail that a particular implementation strategy like runtime polymorphism turns into an essential implementation matter. However, the underlying issue is beyond static vs. runtime polymorphism. Hopefully, a convincing argument for this view on typing can be made using the Graph State Variable framework from State Analysis[22]. The GSV framework is intended to support modeling domain-specific notions of state where the domain includes notions of functional relationships among states in that domain. The most common application of this framework pertains to modeling the position of uncoupled physical objects using a 6-degree of freedom (6-dof) representation for state position and orientation where relative motion constraints among physical objects are modeled using functional relationships on state position & orientation. In this domain, the GSV framework provides a unified state interface to update knowledge about the 6-dof state of a set of physical objects and provides mechanisms to query their relationships. In the robotics domain, the Denavit-Hartenberg convention[24] is commonly used to design articulated mechanisms such as robotic arms. In this context, an application of the GSV framework would have as state information the 4 transformation parameters that the D-H convention uses for describing each the characteristics of each link (length, twist and angle) and of its articulation (joint angles). Clearly, the 6-dof and D-H domains have very different type information associated to each state. More importantly, the arithmetic properties of functions of state types like associativity, commutativity and invertability depend on the properties of the state domain. In the 6-dof domain, the choice of a frame of reference for inferring position and orientation relationships among distant objects depends on several considerations including an analysis of orders of magnitude in the numerical computations involved and avoiding non-essential state variables since that would otherwise add superfluous statistical error terms. This is a well-trodden territory for interplanetary spacecraft navigation where reference frames are adjusted to the nearest planet to the spacecraft such that maneuvers near that planet are computed with adequate precision and certainty. For example, navigation maneuvers shortly after launch are computed using earth-relative position information while orbit insertion maneuvers for the Cassini spacecraft in 2004 were computed relative to Saturn and the entry-descent-landing maneuvers for MER in January 2004 were computed relative to Mars. In an application of the GSV framework for the D-H domain, similar considerations for adjusting frame perspective apply; additionally, D-H induces specific architecture differences. In robotics, transformation matrices that describe linkages are often non-invertible. This means that to position a robotic arm like MER's such that the science instrument at the end of the arm is located within a specific distance from a target observation site, it is not realistic to use the GSV framework to compute the inverse kinematics of the robotic arm to infer the joint angles required to achieve that positioning. Instead, it is necessary to have a closed feedback loop between

forward kinematic computations that update knowledge about the location of the end of the arm and the robotic arm and control decisions that command the arm actuators to eventually reach the desired positioning. In other words, domain specificity amounts to much more than the type information with which state knowledge is represented (e.g., 6-dof vs. D-H parameters): domain specificity has a strong influence on decisions governing what constitutes sensible architecture patterns in that domain. It is unlikely that type systems alone can adequately capture this kind of domain-specific information in a parametric manner. In highly formalized notions of architecture like [33] were domain genericity stems from parametric specification capabilities, it seems that handling complex issues of architecture heterogeneity across domains requires additional specification modeling capabilities beyond the flexibility that type parameters provide to a specification language.

3 From graphs to formal logic

The background for handling heterogeneity via architecture styles stems from earlier work on analyzing the Entity-Relational schema that defines MDS' State Analysis Database[23] as if it were a formal specification. Using the DMS toolkit as a platform for agile language transformation techniques, initial experiments focused on reverse engineering the xADL-like architecture prescription and description models into Acme architecture styles. Although similar to techniques used in [19], scaling up to handle all of the implementation details of the actual C++ implementation showed two things. First, at the implementation level, the architecture has an overwhelming abundance of incidental information that hides the essential aspects of the architecture in ways that make understanding challenging. Second, this experiment confirmed other reports that DMS is truly a scalable infrastructure[40] for large-scale transformations operating on thousands of complex objects. Subsequently, the experimental strategy focused on analyzing the ER schema from the conceptual level instead of enforcing consistency directly at the implementation level. Initially, the ER schema was analyzed as a graph. To make this practical and efficient, the ER schema was first transformed into an architecture style in Acme according to various interpretation rules and the Acme architecture style was subsequently transformed into a graph description fed into Mathematica's Combinatorica package. Other transformations produced a visualization specification as input to Tom Sawyer's Graph Layout Toolkit library whose Graph editor was integrated into Mathematica's interactive environment. This transformation machinery made it possible to explore several strategies for mapping the ER schema into and architecture style. On a practical level, visualization transformation and constrained layout with GLT made it possible to incorporate the notation guidelines from State Analysis into the automatic layout. Despite this technological wizardry, the overall methodology was insufficiently simple relative to our intuitions about the conceptual complexity of state analysis. The ER schema involves well over 30 concepts while conceptually less

than a dozen concepts are necessary to capture the conceptual essence of state analysis.

The goal of these experiments was to find a criteria that would conceptually factor out of the ER schema a list of essential concepts that would constitute the core of state analysis. Intuitively, the idea was to provide an rigorous basis to explain how ER-level distinctions like *STATE_VARIABLE_TYPE* and *STATE_VARIABLE* can be described as variations of the same concept. This problem can be approached as a kind of bipartite graph matching. This approach made sense in the context of analyzing graphs of equations modeling the steady-state behavior of a physical plant for designing high-performance numerical simulation algorithms whose computations faithfully reflect the causality of the model. In the context of ER schemata, there is no single preference criteria like causality that would make bipartite graph matching a useful hammer for breaking to the schema down into its core constituent concepts.

Finally, a more productive track appeared in experimenting with yet another transformation from the Acme architecture style to theorem provers and constraint languages. The inspiration for this approach arose from discussions at an HDCP meeting at CMU with Daniel Jackson and Greg Morris on modeling State Analysis in Alloy. Experiments with the Simplify theorem prover and SAT-based specification techniques like Alloy and SAL led to mixed results: the separability of the ER schema depends greatly on the degree to which domain specific knowledge is available as axioms and constraints. Earlier, CMU had modeled many engineering constraints of state analysis using Armani constraints. However, the Armani rules have to be applied on a case-by-case basis according to the domains used. In a similar manner, the formalization of State Analysis' ER schema would require some form of second-order reasoning to associate axioms and constraints to first-order formalizations of state analysis' concepts.

4 A semantic architecture style for heterogeneous view specifications

These observations lead us to postulate that formal ontology development for space missions would require integrating in a practical, tool-supported methodology a number of important ideas from the literature.

1. Architecture Style & Modeling Paradigm: the limits of type parametricity[6] Originally, State Analysis was investigated as if an architecture style could precisely define what it is. While it is possible to have a lightweight, semi-formal architecture style that reflects the main concepts[17,19], refining these architecture styles to large-scale implementations leads to a lot of practical problems due to the conflicts between downward type restrictions due to architecture refinement and implementation-level type variability due to domain specificity that breaks type hierarchies. Methodologies like State Analysis[23], IEEE-1471[10] and RM-ODP yield architecture styles once they

[6] Cite Vanderbilt stuff on metamodeling?

have been tailored to the specific concerns of a particular domain. Here, we use the term *modeling paradigm* to distinguish a generic methodology where domain specialization is beyond simple notions of generic typing with type parameters.

2. Heterogeneity & Compositionality: the rationale for category theory
   Heterogeneity makes modeling significantly harder because inherent to heterogeneity is the notion of multiple viewpoints[9]. A viewpoint forces modellers to follow modeling practices similar to denotational semantics due to the necessity of distinguishing between a specific interpretation of a concept in a given viewpoint and the essence of that concept independent of any viewpoint. This is a similar approach to a formalization of the RM-ODP framework[15] that proposes a matrix organization for categorizing the relationships between the concepts in the universe of discourse (essential concepts) and models of that universe (constructed by specifying viewpoint-specific concepts). This approach is very similar to algebraic specification techniques based on category theory applied to modeling[44].
3. Viewpoint integration & semantics: is there enough fibration in the ontology?
   Perhaps the most surprising terminology from category theory is the concept of fibration[7]. Hints of fibration can be seen in the context of the RM-ODP standard[14] to precisely establish correspondences among viewpoint concepts and between viewpoint concepts and those in the universe of discourse[15]. The precise explanation of fibration as it pertains to modeling is formalized in [41] as requirements on an abstraction mechanism for entity-relation information models[42] whose extension for behavior modeling[43] closely resembles the issues of viewpoint integration in RM-ODP. More importantly, this methodology clearly demonstrates the utility of richly axiomatized upper-ontologies like SUMO and DOLCE in the construction of an ontology where the organization of the ontology is structurally consistent with the relations among the concepts of the ontology.
4. Viewpoint locality & Architecture modularity: semantic compatibility & compositionality
   Clean humor aside, how does category theoretic fibration help in ontology development for heterogeneous engineering models? Even proponents in category theory recognize the practical limitations of the approach[45]: if viewpoints are interchangeable in the sense of information preserving correspondences between them, then the mathematical approach provides a good strategy for modularizing these viewpoints; however, if correspondences loose information in one direction, then it is unclear what methodological guidance this approach can offer.
5. Category theory & Formal specifications: SpecWare to the rescue!
   With mounting evidence of the benefits of category theory, the issue of rigorous ontological development merits a review of current practices and tool

[7] This terminology alone can raise skepticism in the reader in no doubt influenced from commercials touting the benefits of fiber-rich cerals about the real purpose of this paper, whether it is to kill trees or to convince that *this* paper is a good source of intellectual fiber.

support. OWL-centric ontology development yields a substantial value if concepts are organized as a classification taxonomy (Rector), especially if the ontology is OWL-DL where efficient implementations of classifiers are available and where complete OWL-DL is theoretically possible (Handbook) and practically available (Pellet). However, OWL does little to support a category theoretic style of ontological modeling with notions of products, co-products and fibration[46] despite the many theoretical benefits these approaches would provide. There is great promise in formal specification tools that make category theory a practical reality like Kestrel's SpecWare system[47] used to demonstrate how specifications can be parametrized in terms of domain theories in a compositional way[49].

6. Formal specifications & Architecture Description Languages: Acme resembling SpecWare with Armani[27] disguised as SNARK in a poor man's clothing

   JPL's experience in modeling State Analysis with Acme might have had a mixed success in terms of practical adoption; however, from a research standpoint, it provides a convincing case that formal architecture description languages are necessary to tackle the intrinsic complexity of heterogeneous engineering models as an issue of formal specification, composition and refinement. Turns out these are the very same characteristics that the SpecWare system provides. However, while there have been promising demonstrations like [49], it is clear there is still room for practical improvements to make the methodology even more pragmatic[48]. Additional mechanisms would also be required to provide support for fibration analysis in the style of the requirements described in [41].

7. Viewpoint correspondences, modeling purpose and workflow: Architecture Specifications tied to Semantic Web services

   Modeling methodologies are unanimous on at least one point: specifications make sense only if there is a well-defined purpose behind them. IEEE-1471 and RM-ODP describe this on the basis of the stakeholders' objectives and concerns that provide the rationale for architecture descriptions and view-based modeling. State Analysis describes this with the concept of *goal* as the rationale for the behavior of controllers and estimators. In requirements engineering, the KAOS methodology (reference...) also places a great emphasis on goals as the starting point for requirements elicitation and development. For pragmatic reasons, it is important for whatever engineering methodology to provide an abstraction mechanism by which users can get a bird's eye view of complex heterogeneous viewpoints and understand their relationships in terms of modeling workflows that are related to the workflows relevant to the real-world systems being modeled and the objectives stakeholders have about such systems as functional issues of system state determination, knowledge, control and planning. So far, the cleanest architecture methodology for tieing architecture, workflows and functionality together seems to be the OWL-S specification for web services, particularly recent efforts to define a solid foundation for them in SWSL with the FLOWS methodology.

5 From methodology organization to ontological content: many vexing issues ahead

The previous section presented cogent arguments towards a rigorous methodology for heterogeneous model-based engineering. Most of the cogent arguments presented were about organizing the methodology in a sufficiently rigorous and formal manner to enable rich semantic analyses across heterogeneous engineering models. This section focuses on the ontological concepts of the methodology itself: i.e., what kinds of abstractions apply across heterogeneous engineering domains and applications.

5.1 Identity and referential ambiguity

The OntoClean methodology[4] emphasizes the notion of identity criteria as an essential property of ontologies to ensure that the interpretation of statements about an ontology are unambiguous in terms of which individuals or sets of individuals they refer to. A great deal of complexity in heterogeneous modeling stems from the referential ambiguity that occurs when interpreting statements across different views. Much of this ambiguity stems from unidentified view-specific contexts that are necessary to resolve ambiguities. Instead of a traditional modeling approach based on hierarchical containment, State Analysis advocates a heterarchical[8] organization where items can be grouped arbitrarily according to each modeling viewpoint. The basic modeling concepts involved are: *item* and *group*. Following the OntoClean methodology, it makes sense to require that these concepts carry a global identity criteria to avoid referential ambiguity due to context. For pragmatic purposes, it makes sense to require that both items and groups provide an identity criteria in a way that ensures each hierarchical reference to an item is unambiguously interpretable.

To illustrate these points, we use a simplified example of a vehicle. One possible approach for a globally unique identity criteria would use a composition of two criteria: functional role and location placement. Functional roles could be further decomposed in terms of steering (yes/no) and driving (yes/no). Location roles could be further decomposed in terms of lateral (left/right) and longitudinal (front/back) sides. For a typical 4-wheel automobile, these criteria suffice: the front wheels are typically steerable and driving wheels. The driving role provides no useful identity criteria for an all-wheel drive vehicle. While location role might suffice for a particular viewpoint, it may be impractical in another viewpoint. For example, front-driving vs rear-driving vehicle designs make a substantial differences to the analysis of acceleration, braking, and stability; pragmatically, it makes sense to handle these differences in separate viewpoints so that the analysis techniques can be specialized accordingly without the unnecessary complexity of handling issues beyond the scope of each viewpoint's area of concern. In such circumstances, it is therefore essential to ensure that a particular vehicle is analyzed according to the functional viewpoint applicable

[8] See: http://en.wikipedia.org/wiki/Heterarchy

to that vehicle. Without the ability to relate location and functional roles for a vehicle's wheels, ambiguities arise where a vehicle might be analyzed according to a different functional viewpoint than the vehicle's functional characteristics.

This example illustrates the point that State Analysis' notions of items and groups are an incomplete conceptual basis to support heterogeneous modeling with multiple, inter-related viewpoints. The item-group relations involve a combination of two modeling concerns. Although modeling part-whole relations is reasonably well understood from a philosophical perspective[51,4,6], the ontological patterns documented so far[50] support simple hierarchies whereas heterarchies would require additional support for modeling the semantic constraints across hierarchies. Additional semantics are necessary to make consistent distinctions between parthood as a hierarchical organizational criteria and kinding as a semantic clustering criteria. This idea is briefly mentioned in [50, see Pattern 3] where the editors suggest scripting tools as a practical solution. This suggestion in fact points to a larger issue of ontology development within OWL-DL where part of the semantics of an ontology is not expressible within the ontology itself due to the limitations of OWL-DL and must consequently be defined outside the ontology itself.

5.2 A calculus for ontological development

How often would an ontology developer need various scripting tools to apply various ontology construction patterns like [50]? To appreciate the importance of this issue, it is important to take a broader perspective on heterogeneous modeling. Since the set of viewpoints is inherently open to future additions, the basic development methodology for defining a new viewpoint will inevitably involve a combination of composition and refinement from existing ontologies. The underlying issue inherent in viewpoint modeling is the thorny notion of context[8].

In the WonderWeb project, a great deal of attention went into formalizing a reasonably pragmatic approach for modeling context with the ontology of descriptions and situations[5]. This approach avoids the devastating problems that would derail a naive approach based on simple ontology import mechanisms where semantic incompatibilities among similar concepts across viewpoints are inherently unavoidable in open heterogeneous modeling. According to its authors, DOLCE is a pragmatic compromise between overly simplistic approaches like ontological imports and theoretically complex approaches based on formal contexts and bridging axioms. While a step in the right direction, DOLCE, in its current form[6], presents a new set of practical challenges for ontology development.

Recent work in DOLCE[7, see 3.3] points to the need for a powerful calculus with operations for *reifying* concepts and for *augmenting* and *embedding* ontologies like the Description & Situation (D&S) ontology. D&S is particularly significant because it could significantly facilitate efforts towards formalizing ex-

isting concept terminologies within an organization[9]. Without such a calculus, it will be difficult for practitioners to replicate the compelling examples from DOLCE. For example, [6, sec. 12.5] discusses how D&S is repeatedly applied to a core ontology of services. A similar augmentation would be sensible to specialize the ontology for an Architecture Description Language (e.g., [27]) according to the architecture modeling requirements of a specific viewpoint. This capability would provide architects with a potentially rigorous approach for detecting and resolving architecture mismatches[26] across heterogeneous viewpoints. For pragmatic purposes, if the ontologies are kept within OWL-DL, the calculus itself would clearly have to operate within OWL-Full. What rationale would warrant the effort of developing this kind of calculus?

There are, at least, two important reasons for having an ontology calculus. First, describing the reifications, augmentations and embeddings is important for verification and validation purposes: the embedding itself defines the logical context for asserting the truth value of statements about an augmented ontology[8]. This makes the validity of such statements logically dependent on the validity of the embedding context itself. This argument motivates for an explicit, declarative description of an embedding context. Second, for configuration management purposes, it makes sense to automate the embedding process to simplify the maintenance and evolution of ground ontologies prior to augmentation and embedding. The combination of these two reasons points to a perspective on ontology development in a broader context yet.

5.3 Modularity, functionality and behavior

A calculus for ontologies makes sense from a configuration management perspective[53] where changes to various ontologies are warranted for various business reasons. If an ontology calculus is used, then it makes sense to "replay" the processes that engineers use to augment, extend and embed various ontologies for diverse viewpoint modeling purposes. An ontology calculus that supports both specification and execution of operations performed would be an useful application target for ontologies like PSL[2] that have rigorous semantics for process-related concepts. FLOWS[3] would be especially attractive since it subsumes the concepts of existing standards (e.g., OWL-S) and formalizes notions like *message* and *channel* that are particularly important for Architecture Description Languages like [25,27].

6 Conclusion

Originally, the goal of this paper was to present a first cut for an ontology to convey the concepts necessary to account for heterogeneous viewpoint modeling according to [9,13]. Substantial progress[14,15] has been made in this area since

[9] In NASA's case, see for example: http://sweet.jpl.nasa.gov/ontology and http://nasataxonomy.jpl.nasa.gov

the availability of these standards; however, the modeling issues have yet to be investigated within the broader scope of formal ontologies[6,7], architecture description languages[27,25], algebraic methods for relating viewpoints in terms of refinements[42], multiple views[45], criteria for modularity[51], relationships among views, modularity and context[45,15,8]. Clearly, there are many complex issues.

Recent advances, especially from open-source tools, give pragmatic reasons to remain optimistic and hopeful that significant progress could be made in the near term in ways that leverage established tools and languages like UML and OWL to improve methodologies according to recommended practices in a mathematically sound manner as suggested in [12]. However, this optimism depends on the availability of an ontology calculus that can be automated to make any experimental progress independently reviewable, repeatable and analyzable everytime foundational ontologies evolve and viewpoint libraries change. Such a calculus would significantly help strengthen existing synergies among experimental research, academic research in formal logic (e.g., common logic), mathematics (e.g., category theory) and theoretical computer science (e.g., co-algebraic specification techniques) in a way that facilitates the maturation of proven methodologies into commercially-supported ontology development tools.

7 Acknowledgements

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration, a specific, joint collaboration agreement with the Kestrel Institute with additional support from external collaborations with Carnegie Mellon University and NASA Ames under NASA's Office of Space Exploration Code T program. We wish to thank fellow engineers across several projects at JPL, CMU, Kestrel Institute, Mindswap, NASA, Mitre and Semantic Designs who have contributed with valuable insights and comments pertaining to this work: Jonathan Aldrich, Matthias Anlauff, John Anton, Ira Baxter, Patrick Cassidy, Ken Clark, Robert Clark, Alessandro Coglio, David Cyrluk, Daniel Dvorak, David Garlan, Cordell Green, Klaus Havelund, Doug Jensen, Rajeev Joshi, Gerard Holzmann, Cheng Hu, Mark Kordon, Meemong Lee, Michael Mehlich, Kenny Meyer, Bijan Parsia, Robert Rasmussen, Kirk Reinholtz, Andrew Schain, Marcel Schoppers, Bradley Schmerl, Peter Shames, Joseph Skipper and Doug Smith.

References

1. Patil, L., Dutta, D., Sriram, R: Ontology-Based Exchange of Product Data Semantics. IEEE Trans. on Automation Science and Engineering, Vol. 2, No. 3, July 2005 213–225.
2. Gruninger, M., Menzel, C.: Process Specification Language: Principles and Applications. AI Magazine 24(3) (2003) 63–74.

3. Battle, S., Bernstein, A., Boley, H., Grosof, B.., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIllraith, D., McGuinness, D., Su, J, Tabet, S.: Semantic Web Services Ontology (SWSO) Draft version 1.1 (2005) http://www.daml.org/services/swsf/1.1/swso/
4. Welty, C., Guarino, N.: Supporting ontological analysis of taxonomic relationships. Data & Knowledge Engineering 39 (2001) 51–74.
5. Gangemi, A., Mika, P.: Understanding the Semantic Web through Descriptions and Situations. Int. Conf. on Ontologies, Databases and Applications of SEmantics (ODBASE 2003), Catania, Italy, Nov. 3-7, 2003. http://www.loa-cnr.it/Papers/ODBASE-CONTEXT.pdf
6. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: WonderWeb Deliverable D18: Ontology Library (final). IST Project 2001-33052 WonderWeb: Ontology Infrastructure for the Semantic Web. http://wonderweb.semanticweb.org/deliverables/documents/D18.pdf
7. Gangemi, A., Borgo, S., Catenacci, C., Lehmann, J.: Task Taxonomies for Knowledge Content: D07. Deliverable of the EU FP6 project "Metokis". May 4, 2005. http://www.loa-cnr.it/Papers/D07_v20a.pdf
8. Menzel, C.: The Objective Conception of COntext and Its Logic. Minds and Machines, 9 (1999), 29-56. http://philebus.tamu.edu/cmenzel/Papers/mm-paper.pdf
9. IEEE Computer Society: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems description. http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html
10. Hillard, R.: IEEE-1471 and Beyond. Position paper for SEI's First Architecture Representation Workshop (2001) 16-17. http://www.enterprise-architecture.info/Images/Documents/IEEE%201471-%20Beyond.pdf
11. Land, R.: Applying the IEEE-1471-2000 Recommended Practice to a Software Integration Project. Int. Conf. on Software Engineering Research and Practice (SERP'03), CSREA Press, Las Vegas, Nevada, June 2003. http://www.mrtc.mdh.se/index.phtml?choice=publications&id=0529
12. Gnilloud, G., Frank. W.F.: Use Case Concepts from an RM-ODP Perspective. Journal of Object Technology, vol 4, no 6, Special Issue: Use Case Modeling at UML-2004, August 2005, 95–107. http://www.jot.fm/issues/issue_2005_08/article8.pdf
13. ISO/IEC 10746-1,2,3,4, ITU-T Recommendation X.901, X.902, X.903, X.904: Open Distributed Processing - Reference Model. OMG (1995-96).
14. Wegmann, A., Naumenko, A.: Conceptual Modeling of Complex Systems Using an RM-ODP Based Ontology. Proc. of the 5th IEEE Int. Enterprise Distributed Object Computing Conference - EDOC 2001, Seattle, USA (2001) 200-211.
15. Naumenko, A., Wegmann, A.: A Formal Foundation of the RM-ODP Conceptual Framework. EPFL-DSC Technical Report N. DSC/2001/040, http://lcawww.epfl.ch/Publications/Naumenko/TR01_040.pdf, July 2001.
16. Boehm, B.E.W., Bhuta, J., Garlan, G., Gradman, E., Huang, L., Lam, A., Madachy, R., Medvidovic, N., Meyer, K., Meyers, S., Pérez, G., Reinholtz, W.K., Roshandel, R., Rouquette, N.: Using Empirical Testbeds to Accelerate Technology Maturity and Transition: The SCRover Experience, Proceedings of ISESE, 2004, 117–126.
17. Garlan, D., Reinholtz, W.K., Schmerl, B., Sherman, N., Tseng, T.:Bridging the Gap between Systems Design and Space Systems Software. 29th Annual IEEE/NASA Software Engineering Workshop (SEW-29), Greenbelt, MD, USA (2005). http://www.cs.cmu.edu/~able/publications/polaris/

18. Denker, G., Talcott, C.: Formal Checklists for Remote Agent Dependability. In 5th Int. Workshop of Rewriting Logic and Its Applications, Barcelona, Spain, March 27-28, 2004. http://www.csl.sri.com/~denker/publ/DenTal04.pdf
19. Roshandel, R., Schmerl, B., Medvidovic, N., Garlan, D., Zhang, D.: Understanding Tradeoffs among Different Architectural Modeling Approaches. Proc. of the 4th. Working IEEE/IFIP Conf. on Software Architectures, Oslo, Norway, June 11-14, 2004. http://www.cs.cmu.edu/~able/publications/WICSA4-scrover/
20. N. Rouquette, UML/MDA Reality Check:Heterogeneous Architecture Styles, invited presentation at UML 2003.
21. Dvorak, D., Rasmussen, R.D., Reeves, G., Sacks, A.: Software Architecture Themes in JPL's Mission Data System. Proceedings of the AIAA Guidance Navigation and Control Conference, AIAA-99-4553, 1999.
22. Bennett, M., Rasmussen, R.D.: Modeling Relationships Using Graph State Variables. Proceedings of the IEEE Aerospace, 2002.
23. Ingham, M.D., Rasmussen, R.D., Bennett, M.B., Moncada, A.C.: Engineering Complex Embedded Systems with State Analysis and the Mission Data System. Proceedings of the 1st AIAA Intelligent Systems Conference, 2004.
24. Spong, M., Vidyasaga, M.: Forward Kinematics: The Denavit-Hartenberg Convention. Robot Dynamics and Control, ch. 3, Wiley, 1989 http://www4.cs.umanitoba.ca/~jacky/Teaching/Courses/74.795-Humanoid-Robotics/ReadingList/chap3-forward-kinematics.pdf
25. Dashofy, E. van der Hoek, A., Taylor, R.N.: An Infrastructure for the Rapid Development of XML-Based Architecture Description Languages. Proceedings of the ICSE 2002 International Conference on Software Engineering, Orlando, Florida, May 2002.
26. Garlan, D., Allen, R., Ockerbloom, J.: Architectural Mismatch or, Why it's hard to build systems out of existing parts. Proc. of the 17th Int. Conf. on Software Engineering, ICSE-17, April 1995; revised in IEEE Software, Vol 12, Issue 6, Nov 1995 17-26. http://www.cs.cmu.edu/~able/publications/archmismatch-icse17/
27. Allen, R.J., Garlan, D.: A Formal Basis for Architectural Connection. ACM Trans. on Software Engineering and Methodology, July 1997. Revised 1998. http://www.cs.cmu.edu/~able/publications/wright-tosem97-revision/
28. Schmerl, B.: xAcme: CMU Acme Extensions to xArch, ABLE technical report, http://www-2.cs.cmu.edu/~acme/pub/xAcme/guide.pdf, 2001.
29. Meseguer, J., Braga, C.O.: Modular Rewriting Semantics of Programming Languages. Submitted for publication. http://maude.cs.uiuc.edu/papers/abstract/MBModSem_2003.html
30. Clavel, M. Duran, F., Eker, S., Lincoln, P., Mari-Oliet, N., Meseguer, J., Talcott, C.: The Maude 2.0 System. Proc. of the 14th Int. COnf. on Rewriting Techniques and Applications (RTA 2003), Nieuwenhuis (Ed.), Lecture Notes in Computer Science, Vol. 2706, June 2003, 76-87. http://www.csl.sri.com/papers/1558/
31. Henzinger, T.A., Minea, M., Prabhu, V.: Assume-guarantee Reasoning for Hierarchical Hybrid Systems. Proc. of the 4th Int. Workshop on Hybrid Systems: Computation and COntrol (HSCC), Lecture Notes in Computer Science, 2004. 272–290. http://www-cad.eecs.berkeley.edu/~tah/Publications/assume-guarantee_reasoning_for_hierarchical_hybrid_systems.html
32. Giannakopoulou, D., Pasareanu, C., Barringer, H.: Assumption Generation for Software Component Verification. Proc. 17th IEEE Conf. on Automated Software Engineering, ASE-2002.

33. Lau, K.K., Ornaghi, M.: Specifying Compositional Units for Correct Program Development in Computational Logic., In M. Bruynooghe and K.-K. Lau, eds. Program Development in Computational Logic, Lecture Notes in Computer Science, 3049, 2004 1-29. http://www.cs.man.ac.uk/~kung-kiu/pub/lopstr-book.pdf
34. Alexandrescu, A.: Modern C++ Design: Generic Programming and Design Patterns Applied. Addison-Wesley, 2001.
35. Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming. Second Edition, Addison-Wesley, 2002.
36. Czarnecki, K., Eisenecker, U.W.: Generative Programming - Methods, Tools and Applications. Addison-Wesley, 2000.
37. Clark, R., Jensen, E.D., Rouquette, N.: Software Organization to Facilitate Dynamic Processor Scheduling. Proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004.
38. van den Brand, M.G.J, Klint, P., Vinju, J.J.: Term Rewriting with Traversal Functions, CWI Technical Report SEN-R0121, ISSN 1386-369X, 2001 http://db.cwi.nl/rapporten/abstract.php?abstractnr=1079
39. Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, Vol 55, Cambridge University Press, 2003.
40. Baxter, I., Pidgeon, C., Mehlich, M.: DMS: Program Transformations for Practical Scalable Software Evolution. Proceedings of ICSE, p. 625-634, 2004.
41. Colomb, R., Dampney, C.N.G., Johnson, M.: The use of category-theoretic fibration as an abstraction mechanism in information systems. Acta Informatica, 38, 1, (2001) 1–44.
42. Colomb, R., Dampney, C.N.G.: An approach to Ontology for Institutional Facts in the Semantic Web. Technical Report 15/02, ISIB-CNR, Padova, Italy, November 2002.
43. Colomb, R.: Extending Ontology to Behavior in Communities of Interoperating Information Systems Agents. Technical Report 21/02, ISIB-CNT, Padova, Italy, November 2002.
44. Johnson, M., Dampney, C.N.G.: On Category Theory as a (meta) Ontology for Information Systems Research. In Formal Ontology in Information Systems. Welty C., Smith, B. (Eds). ACM Press, (2001) 59-69.
45. Dampney, C.N.G., Johnson, M.: Enterprise Information Systems: Specifying the links among project data models using category theory. Proc. of the Int. Conf. on Enterprise Information Systems, (2001) 619–626.
46. Johnson, M. Rosebrugh, R.: Coproducts in Categorical Information System Specification. Proc. of SCI2001, Vol XIV (2001) 145-150.
47. McDonald, J. Anton, J.: SpecWare - Producing Software Correct by Construction. Kestrel Institute Technical Report KES.U.01.3, March 2001. ftp://ftp.kestrel.edu/pub/papers/specware/specware-jm.pdf
48. Pavlovic, D., Pepper, P., Smith, D.R.: Colimits for Concurrent Collectors
49. Pavlovic, D., Smith, D.R.: Composition and Refinment of Behavioral Specifications. Proc. of 16th Annual Int. Conf. on Automated Software Engineering, Nov 26-29 (2001). ftp://ftp.kestrel.edu/pub/papers/pavlovic/CRBS.ps
50. Rector, A., Welty, C.: Simple part-whole relations in OWL Ontologies. W3C Working Draft, Semantic Web Best Practices and Deployment Working Group, January 15, 2005. http://www.cs.man.ac.uk/~rector/swbp/simple-part-whole/simple-part-whole-relations-v0-2.html
51. Odell, J.J.: Six Different Kinds of Composition. Journal of Object-Oriented Programming, Vol 5, No 8, January 1994. http://www.conradbock.org/compkind.html

52. Kendall, E.: Ontology Definition Metamodel, second revised submission. OMG ad/05-04-13, May 31, 2005. http://www.omg.org/cgi-bin/doc?ad/2005-04-13
53. Chandra, C.: Analytical Modeling of Logistics for Re-Configurable Supply Chain in Mass Customization of Vehicles. Report Brief. Inst. for Adv. Vehicle Systems, College of Eng. and Comp. Science, Univ. of Michigan, Dearborn, MI. Technology Day, June 9, 2004.