openCAESAR: Balancing Agility and Rigor in Model-Based Systems Engineering

Conference Paper · October 2023

DOI: 10.1109/MODELS-C59198.2023.00051

6 authors, including:

Maged Elaasar
NASA
15 PUBLICATIONS **133** CITATIONS
SEE PROFILE

Nicolas Rouquette
California Institute of Technology
53 PUBLICATIONS **733** CITATIONS
SEE PROFILE

David Wagner
California Institute of Technology
18 PUBLICATIONS **180** CITATIONS
SEE PROFILE

Bentley James Oakes
Polytechnique Montréal
50 PUBLICATIONS **243** CITATIONS
SEE PROFILE

openCAESAR: Balancing Agility and Rigor in Model-Based Systems Engineering

Maged Elaasar*, Nicolas Rouquette*, David Wagner*,
Bentley James Oakes†§, Abdelwahab Hamou-Lhadj‡* and Mohammad Hamdaqa§
*NASA Jet Propulsion Lab, California Institute of Technology, Pasadena, USA
Email: {maged.e.elaasar,nicolas.rouquette,david.a.wagner}@jpl.nasa.gov
†DIRO, Université de Montréal, Montreal, Canada
Email: bentley.oakes@umontreal.ca
‡Electrical and Computer Engineering, Concordia University, Montreal, Canada
Email: wahab.hamou-lhadj@concordia.ca
§Computer and Software Engineering, Polytechnique Montréal, Montreal, Canada
Email: {bentley.oakes,mhamdaqa}@polymtl.ca

***Abstract*—Model-Based System Engineering (MBSE) employs models and formal languages to support development of complex (systems-of-) systems. NASA Jet Propulsion Laboratory (JPL) sees MBSE as a key approach to managing the complexity of system development. However, balancing agility and rigor in MBSE has been reported as a challenging task not yet addressed by modeling tools and frameworks. This is because existing MBSE approaches may enable agility but compromise rigor, or enhance rigor but impede agility. We discuss the challenges of balancing agility and rigor in MBSE across seven systems engineering architectural functions defined by the JPL Integrated Model-Centric Engineering (IMCE) initiative. We demonstrate how openCAESAR, an open-source MBSE methodology and framework created at JPL, can strike a balance between agility and rigor through a case study of the Kepler16b project and discussion of lessons learned from past projects.**

***Index Terms*—Systems Engineering, Model-Based Systems Engineering, Ontology-based Modeling, OML, openCAESAR**

I. Introduction

Model-based Systems Engineering (MBSE) methodologies employ models and formal languages to support requirements, design, analysis, verification, and validation steps during the development of complex systems and system of systems [1]. Rather than a document-centric approach, models are regarded as first-class entities and central artifacts in MBSE for representing the system, information about a system, and the MBSE process itself. This reduces time and cost, and improves communication in the systems engineering process [2].

NASA Jet Propulsion Laboratory (JPL) is a research development center in California under contract with NASA to focus on the development and operation of planetary robotic spacecraft, Earth-orbit, and astronomy missions. With over 85 years of experience in engineering complex systems, JPL has embraced MBSE as a key approach to managing the complexity of systems and their development.

A. Balancing Agility and Rigor in MBSE

Systems engineering is a multi-disciplinary field that involves integrating models that have been created by authors with expertise from different domains into complex systems. [2], [3]. Even a small-scale project, such as the illustrative *Kepler16b* project discussed here, involves concepts such as electrical, mechanical, optics, orbital mechanics, and risk management, among others.

The experience of JPL on past projects on these complex systems of systems reveals the inherent challenge in utilizing existing MBSE approaches to strike the right balance between *agility* and *rigor* in system development. JPL's requirements go beyond the Agile Manifesto [4], focusing on the changeability, rate, and speed of system development, to also include reproducibility and trust in the entire *modeling pipeline*. This pipeline connects models originating in different tools, where the life-cycle is from model creation and evolution to deployment and operations. Additional DevOps principles [5] are therefore required at the modeling level to ensure that the system development process supports frequent integration between components from different domains, address scalability, model configuration and dependency management concerns, supports model federation, distribution, and versioning, and enables continuous model delivery and on-demand reporting.

The MBSE process at NASA JPL also requires intense *rigor* to achieve high-quality system development. For example, JPL identifies the need for scalable multi-domain analyses which tackle the consistency, correctness, and completeness of model elements. Knowledge from multiple domains must be captured and formally reasoned about to remove errors and inconsistencies introduced during the system's development.

Existing MBSE approaches (see Section V) that prioritize flexibility and adaptability can *enable agility* but may *compromise rigor*. Approaches that prioritize formalization and structure to ensure consistency, correctness, completeness, and traceability can *enhance rigor* but may *impede agility*.

For example, a project developed by a small group of JPL engineers is able to develop a complex mission in months, demonstrating agility. However, this face-to-face, non-rigorous communication style will not scale up to globally distributed teams of hundreds of engineers.

B. MBSE Methodologies at JPL

To handle these competing requirements and more efficiently support system engineers, JPL established the *Integrated Model-Centric Engineering* (IMCE) project in 2010 to develop tools, methodologies, training, and documentation for MBSE [6]. This led to the internal *Computer Aided Engineering for Spacecraft System Architectures Tool Suite* (CAESAR) platform, used on the Mars2020, Europa Clipper, Psyche, and Sample Return Lander projects at JPL.

Here, we introduce an open-source version of CAESAR called *openCAESAR*[1]. It is an MBSE methodology and framework developed at JPL to support systems engineers in the development, management, integration, deployment, and operations of complex models. It balances between an *agile approach* suitable for the challenges of the JPL environment and a *rigorous methodology* suitable for developing complex systems (and systems-of-systems). At the heart of openCAESAR is the Ontology Modeling Language (OML) [7]. OML allows engineers to directly specify their knowledge as *ontologies* with precise syntax and semantics, without the usual incidental complexity found in semantic web technologies and tools [8]. These ontologies then form the backbone of the openCAESAR MBSE methodology, enabling systems engineers to consider rigor and agility across seven systems engineering architectural functions: *representation*, *authoring*, *federation*, *configuration*, *integration*, *analysis*, and *reporting*. We demonstrate openCAESAR and how it strikes a balance between agility and rigor in MBSE through the Kepler16b example, and present lessons learned from past JPL projects.

II. Agility and Rigor in MBSE Methodology Functions

JPL's experience across past projects has led to the identification of seven key MBSE functions to be well-supported in MBSE methodologies and frameworks [6]: *representation*, *authoring*, *federation*, *configuration*, *integration*, *analysis*, and *reporting*. Here we discuss at the conceptual level these functions identified by JPL and their relation to the competing concepts of *agility* and *rigor*. In Section III, we explain at the technological level how openCAESAR supports each of these methodology functions using a running example.

A. Meaning of Agility and Rigor

Before discussing the MBSE methodology functions, this section introduces our meanings of *agility* and *rigor*. The essence of these two terms is that *agility* is about the ability to *reconfigure* and *adapt* the system and the development process as needed, while *rigor* is about ensuring that the system and development process are *sound*. In the context of MBSE, agility is the ability to adapt the MBSE models and associated analysis and reporting to reflect unique project details with minimal effort. Rigor instead includes the precision and accuracy or general lack of ambiguity.

We have selected the terms 'agility' and 'rigor' based on our understanding, supported by the discussion of Tolentino and Wood [9]. However, synonyms may be preferred by readers such as "changeability" and "robustness" [10].

1) Agility: Our understanding of *agility* is reflected in a quote from Willett *et al.* [11]: "Agility is the ability to move quickly and easily; speed with quality.". More precisely, they identify four areas where agility is relevant for systems engineering [11]: a) agile systems-engineering (*process*), b) agile-systems engineering (*technology*), c) agile-operations (*environment*), and d) agile-workforce (*people*). In the context of MBSE, these areas are broadly relevant [12]. However, MBSE methodologies must also be able to support concrete functions such as supporting stakeholder collaboration, communication, and on-boarding [3] to achieve this *agile-workforce*.

Notably, these agility concerns may not be adequately addressed in current MBSE tool-chains [13]. Ma *et al.* report that only 53% (138/260) tool-chains address *interoperability*, 43%(112/260) address *reusability*, and 43% (113/260) address *scalability*. Modularity and reusability, and tool dependency and integration are also challenges for adopting MBSE [14]. Another agility aspect specific to system engineering is that systems and processes can be project-specific. At JPL, every project will have some unique aspects including novel system capabilities. Thus the MBSE development process cannot move "quickly and easily" without addressing these concerns.

2) Rigor: MBSE relies upon the concept of *rigor* which sits at the heart of the systems engineering process [9]. Ramos *et al.* state that MBSE is the "formalized application of modeling principles, methods, languages, and tools to the entire life-cycle of large, complex, interdisciplinary, sociotechnical systems" [15], implying that rigor must be maintained in both the system and its development process [16]. The related properties "reliability" and "safety" are most often mentioned in MBSE tool-chains [13]. These dimensions of rigor, including *traceability*, *consistency*, *correctness*, and *completeness* are therefore applicable to the system and also to the system development process itself. At JPL, traditional testing techniques are no longer sufficient as missions become more complex, demanding increased rigor. For example, mission functions can no longer be tested on Earth, meaning that discovering flaws late in the development process carries major risks.

3) Trade-offs between Agility and Rigor: Tolentino and Wood discuss how the concepts of agility and rigor can be at odds in systems engineering [9]. They identified three areas of conflicts between agile development and systems engineering: documentation, prioritization of non-functional requirements versus development speed, and change management. For example, improving documentation increases traceability and the ability to perform trade-off analyses (improving *rigor*), but the process of producing documentation can be slow and inflexible to change (decreasing *agility*).

[1]https://www.opencaesar.io/

In MBSE, these aspects also have to be balanced. For example, Denil *et al.* identify one solution whereby rigor can be addressed during "sprints" (fixed time-periods in development) in an agile process [17]. During these periods, more rigorous processes can be integrated, with the final sprint fully adhering to applicable safety standards.

B. MBSE Methodology Functions at JPL

This section explains the JPL MBSE methodology functions and how they impact the *agility* and *rigor* of the system and the system development process. Section III offers technical details of how openCAESAR supports these functions.

1) Representation: In MBSE, *representation* refers to how information is structured and organized to support the system design and development process. Representation entails identifying the concepts, relationships, and constraints that comprise the system's architecture such that stakeholders can easily understand, communicate, and analyze. Representation reduces the cognitive load in system design as it determines the choice of the initial modeling constructs, how they are depicted, composed, combined, extended, fragmented, and configured to possess certain system characteristics while ensuring logical consistency and correctness. Thus, as the model representation becomes more expressive, concise, and effective in capturing the system requirements, then this enables a more efficient and effective system engineering process.

The link between representation and *agility* is that good representation enables a clear and structured view of the system, enables the team to quickly identify areas of the architecture that may need to be improved, and to experiment with different design options.

On the other hand, *rigor* is related to how representations should be sufficiently unambiguous and self-explanatory that they can be safely interpreted by many users. They should support *lucidity* (i.e., clarity and simplicity), *laconicity* (i.e., brevity and conciseness), *soundness* (i.e., correctness and consistency) and *completeness* (i.e., coverage and comprehensiveness). This helps ensure the system behaves as intended to meet performance, reliability, and safety requirements.

2) Authoring: The model *authoring* function facilitates what users write and mean to say. In system modeling, this entails following a methodology that supports using predefined representations of specific vocabularies and viewpoints to create and customize models that accurately describe systems from a set of supported domains.

For *agility*, model authoring can accelerate model creation and customization through tool features such as mixed concrete syntax (textual and visual), multi-level views, model layers, provider-specific view-points, and supporting collaboration, communication, and rapid feedback. Additionally, the ability to scale and extend the models allows for flexibility and adaptability in response to changing requirements.

Authoring has *rigor* when the models created are ensured to be accurate, consistent, and adhere to the specified methodology and predefined representations. Model authors can then know that the models accurately represent the system being modeled and adhere to any relevant standards or regulations. Additionally, verification methods provide accurate and timely feedback, enhancing rigor for the modeling process through quick identification and resolution of model issues.

3) Federation: Model *federation* refers to the ability to organize models and information flow between parts of the system to automate the project's workflow. It involves breaking down the system model into well-defined sub-models and organizing information in separate repositories to match that content's ownership. Thus, the *model organization* reflects the *process organization* to avoid collisions of authority. This approach enables parallel work on the system without unintended impacts. Breaking down the system model into fragments can introduce challenges by making it more fragile and harder to coordinate changes. However, this is preferable to having a single model which can make it hard to give authority or establish traceability. Therefore, federation allows for the integration of information and models from different sources to create a complete system model while maintaining its integrity and traceability.

Model federation increases *agility* by allowing stakeholders to collaborate on the system at the same time and rapidly iterate and adapt, which is critical in a large organization such as JPL. Model federation can also support *rigor* by breaking the model into sub-models and repositories to make it easier to check the model's consistency or quality through specific techniques targeted for each type of model. Traceability throughout system development is also enhanced by assigning models and repositories to specific owners. These owners are then responsible for coordinating development, verification, and reporting for their models.

4) Configuration: Configuration refers to the ability to track various versions of valuable information produced by users during authoring, analysis and reporting. This information may have different variants which must be related to each other and clearly communicate to authors and stakeholders.

An *agile* systems engineering solution thus requires a *configuration management system* (CMS), resembling Git functionality, to streamline creating new versions, reviewing and approving new changes, handling conflicting versions, and merging the changes into a baseline branch.

In addition, *rigor* in configuration is required to ensure consistency and data integrity of the information across versions. Consistency should also be maintained for all models across the entire system with every change, which is challenging to achieve. The CMS should detect conflicting information provided by authors and provide effective conflict resolution mechanisms. This requires an effective way to measure and report differences between model/report versions and to reconcile those differences through a merge/resolution process.

5) Integration: The systems engineering development process is iterative and incremental, where models produced by users during authoring is constantly analyzed, and the analysis results are occasionally reported and published. This information authoring, analysis, reporting process requires the

integration of multiple tools to perform continuous operations as new data is added.

For an integration process to be both agile and rigorous, there are several challenges. First, the entire process should be triggered automatically with no user intervention. The second challenge relates to how dependencies are managed between the information models that describe different perspectives of a system. For example, a model describing an electrical circuit could depend on information provided in a model authored by a mechanical engineering team. The integration between the two models should ensure future versions of these models do not introduce inconsistencies.

Another challenge involves tool interoperability throughout the entire integration pipeline. Adapters must be created to render interoperability seamless and increase the *agility* to adopt new tools. This is a challenge when multiple authoring, analysis and reporting tools are used, which is common for complex systems that cover multiple domains. Third-party tools may also require constant updates as their APIs change, or their use is constrained due to different licensing models.

6) Analysis: A key aspect of system modeling is to *analyze* the produced models to ensure their *rigor*, in terms of *consistency*, *completeness*, and *correctness*. Consistency analysis checks that the descriptions of the system do not contain conflicting statements and that they are conform to any meta-models defined. The completeness analysis verifies that the system is adequately described in terms of its components, properties, and relationships. Correctness analysis checks whether a description satisfies its functional and non-functional requirements.

The modeling language choice directly impacts the level of rigor possible. Greater levels of rigor can be attained through the use of formal languages with unambiguous semantics. Whenever it is not possible to fully verify the accuracy of the information, this can be encoded in the model, or engineering margins and a periodic review process can be implemented to ensure that the uncertainty is sufficiently managed. To support *agility* during analysis, tools are also needed such as model querying, model simulation and execution, solvers, and model checkers. Supporting analysis tools can be challenging due to the complexity of these tools and the associated learning curve. The repeatability of the analysis is also essential for the purposes of auditing and regulatory compliance.

7) Reporting: In systems engineering, reporting is an activity that takes place throughout the entire life cycle of the system modeling, including authoring and analysis. Reporting enables effective review by tailoring the information presented to a user's expertise. This can include reporting identifiable discrepancies or just projecting model information into a view that allows readers to see the information in context or in a viewpoint they understand. These reports may include a combination of text, graphical, and analytical representations (e.g., tables, diagrams, charts, etc.). An *agile* reporting system should allow users to view reports in a variety of ways depending on their expertise. It should also be flexible enough to integrate with different reporting and analytical platforms used in other fields via APIs or adapters.

Reporting is subject to *rigorous* requirements. The generated reports must be factual, consistent, complete, and correct. Reports should also enable requirement traceability by mapping the models to the appropriate requirements or products. At JPL, reporting is a critical activity because of government requirements to produce and maintain persistent records.

III. Supporting Rigor and Agility in openCAESAR

This section explains the openCAESAR MBSE methodology and framework in the context of rigor and agility. We overview openCAESAR and detail the Kepler16b running example. Then, we use this example to demonstrate how the openCAESAR methodology and framework supports the seven MBSE functionalities described in Section II-B.

A. Introduction to openCAESAR

Figure 1 shows the openCAESAR architecture, emphasizing three categories of information functionalities involved in the end-to-end workflow from information-producing users (*authors*) to information-consuming users (*stakeholders*).

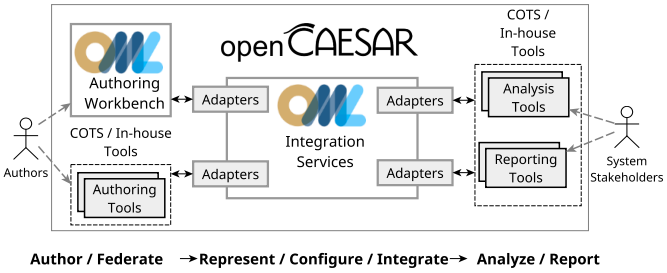

Fig. 1. Architecture of openCAESAR.

For authors unfamiliar with semantic web practices, the architecture accommodates using application-specific workbenches to provide *domain-specific interfaces* for specialized application domains. Such a workbench maps the application-specific model into the common *Ontological Modeling Language* (OML)-based representation using an application-specific adapter. The *representation*, *configuration*, and *integration* functionalities relate to managing information represented and configured in OML. A key focus of this stage is ensuring the logical consistency of the configuration of OML models to ensure validity of all analyses downstream. The *analysis* and *reporting* functionalities then focus on producing consumable artifacts for users, i.e. stakeholders. For authoring and cross-referencing content in a federated context, models use abbreviated and unambiguous Internationalized Resource Identifiers (IRIs) placed in unique namespaces.

Figure 2 shows the two openCAESAR methodology phases. First, the *methodologist* creates or imports the required vocabularies. Then, viewpoints are created based on these vocabularies and authoring tools are built to express these viewpoints. The second phase is an iterative development process where the methodologist works closely together with authors and stakeholders to continuously build and refine the vocabularies, descriptions, queries, and reports required for the project.

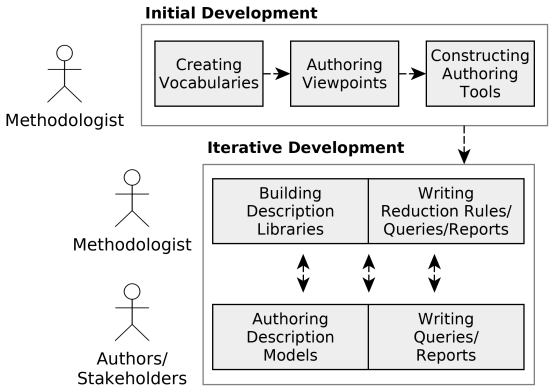

Fig. 2. openCAESAR methodology with two development phases.

B. Kepler16b Running Example

Kepler16b is an exoplanet orbiting the binary star system Kepler16 approximately 245 light-years from Earth[2]. Using the lessons and tools from past JPL projects, NASA JPL is describing an illustrative mission 'Kepler16b' to act as a case study for the openCAESAR platform[3]. There are two main spacecrafts: an *orbiter* and a *lander*. Each of these have scientific missions to accomplish, such as characterizing the atmosphere, environment, and gravitational field. To accomplish these missions and objectives, the spacecrafts are made up of subsystems such as electrical, thermal, telecom, mechanical, and propulsion and their components. Thus, this example represents a complex system requiring an agile and rigorous MBSE methodology and tool support.

C. Representation

The core functionality of openCAESAR allows authors to describe their system of interest by using OML as the key formalism for *representing* domain knowledge in models. Past work discusses technical OML details [7], while here we provide high-level explanations focusing on how OML supports agility and rigor.

As with any space mission, the Kepler16b example is multi-disciplinary, involving low-level concerns such as *mechanical* components and high-level concerns such as *mission operations*. OML addresses this heterogeneity by first creating specialized *vocabularies* for each concern, then using them in the *descriptions*. Kepler16b components represented in OML include an *orbiter* mission (launch system, spacecraft, ground datasystem), a *lander* mission (launch system, spacecraft, ground data system), and a *mission operations system*.

1) Vocabularies and Descriptions: A *vocabulary model* (similar to OWL's T-box) defines the terms (classes and properties and their restrictions) and inference rules required to describe a particular domain. Through layered specialization, these vocabularies can be targeted towards different domains and levels of abstraction. For example, in Listing 1, the concepts of a *Mission* and an *Objective* are defined along with a *Pursues* relation between them.

A *description model* (similar to an OWL A-box) uses one or more vocabularies to describe knowledge in a given context at a given time with a set of instances (individuals) of some types and their (property value) assertions. Unlike classes, instances cannot be specialized or restricted further. Therefore, description models should be used when the intent is to describe (individual) instances in the real world that we want to reason on. For example, the *missions* description model in Listing 2 asserts that there exists a *Lander Mission* mission which *pursues* the *characterize-atmosphere* objective.

Listing 1. Excerpt of the OML missions vocabulary model.

```
@rdfs:label "Mission"
concept Mission :> base:IdentifiedElement
@rdfs:label "Objective"
concept Objective :> base:IdentifiedElement
@rdfs:label "Pursues"
@dc:^description "A Mission pursues >=0 Objectives."
relation entity Pursues [
    from Mission
    to Objective
    forward pursues
    reverse isPursuedBy
    asymmetric
    irreflexive
]
```

Listing 2. Excerpt of the OML missions description model.

```
ci lander : mission:Mission [
    base:hasIdentifier "M.02"
    base:hasCanonicalName "Lander Mission"
    mission:pursues objectives:characterize-atmosphere
    mission:pursues objectives:characterize-environment
    mission:deploys components:lander-launch-system
    mission:deploys components:lander-spacecraft
    mission:deploys components:lander-ground-data-system
    mission:deploys components:mission-operations-system
]
```

OML also supports the ability to multi-classify instances in description models. This enhances agility by allowing classifying instances with types belonging to different vocabularies in the same or different descriptions (e.g., classifying *orbiter-launch-system* as a *mission:Component* in *components* description and as a *mechanical:MechanicalComponent* in *masses* description). Thus, models can be smaller and more coherent, with concerns separated into different domains.

Unlike OWL, which allows mixing T-box and A-box statements in the same ontology, OML enforces rigor by putting them in either vocabulary (T-box) or description (A-box) ontologies. This separation makes it easier to establish whether an ontology is representing the domain(s) itself, or the real-world instances within the system-of-interest. This duality of modeling is very powerful and allows a simpler and homogeneous multi-level modeling stack.

Figure 3 shows the vocabularies and descriptions in the Kepler16b example, along with the *extends* and *uses* relationships

[2]https://exoplanets.nasa.gov/exoplanet-catalog/1814/kepler-16b/
[3]Available at https://github.com/opencaesar/kepler16b-example

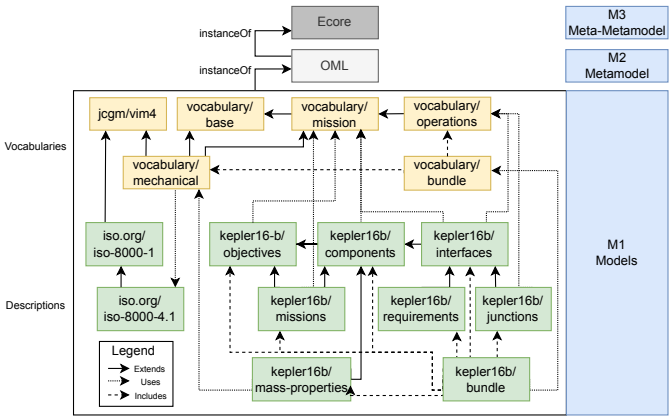

Fig. 3. Vocabularies and descriptions in the Kepler16b example

between them. *Bundles* (described below) can also *include* other ontologies. Both vocabularies and description ontologies are instances of the OML language, which is itself an instance of the Ecore layer. This is represented on the right-hand side of Figure 3 where vocabularies and descriptions are at the *M1 model* level and OML is placed at the *M2 metamodel* level.

2) Description Logic: OML ontologies have description logic (DL) semantics, which allows using logical reasoners (like Pellet) to check the satisfiability of types (that a type can be instantiated) in vocabularies, the self-consistency of descriptions (the lack of logical contradictions), and the consistency of descriptions with (the semantics of) the used vocabularies. In the Kepler16b example, this allows detection of logical inconsistencies that can be introduced by mistake, such as a spacecraft component modeled as containing itself.

The DL semantics also allows for inferencing of *entailments* (axioms that are inferred from others via rules) from assertions (what users declare to be true). This allows assertions to be very concise, minimizing the work of systems engineers, while generating a much richer set of entailments to be used for analysis. In addition to the inference rules built into DL, OML allows specifying custom inference rules in vocabularies to extend the set of analysis entailments even further.

Entailments help simplify analysis via SPARQL queries (see Section III-I) since we can shift some of their pattern matching logic to inference rules, leaving them more simple and direct. Queries can then match instance types directly and not have complex type checking logic. For example, in Kepler16b, classification reasoning can entail all possible *Component* types for a given instance. A query can then select all instances of type *Component* despite the fact that such instances were typed by *SpacecraftComponent*.

3) Open-world versus Closed-world: The logical semantics of OML support two kinds of assumptions. The *open-world assumption* is extremely agile as it states that a statement may be true irrespective of whether or not it is known to be true. For example, a vocabulary could define a concept called *Component* with no supertypes, and another vocabulary can add a supertype *IdentifiedElement* to it. This can also be performed with instances in description models. This modeling agility is not possible in representations with closed-world semantics (e.g., MOF and UML) since types and instances are closed and a new type or instance would have to be defined.

While making an open-world assumption is very useful for agility, it cannot be used to perform verification, which requires a *closed-world assumption*, similar to schemas and metamodels. OML can *bundle* both vocabularies or descriptions. A vocabulary bundle aggregates a set of vocabularies and partially closes the world on them using a special algorithm. In particular, extra assertions are added such that types without a common subtypes are marked as *disjoint*. This enhances rigor by preventing an instance in a description from being typed by disjoint types at the same time, which would be acceptable under an open-world assumption. Similarly, OML supports description bundling with an associated algorithm to close the world and reason about the cardinality of values. This algorithm adds checking the *minimum* cardinality restrictions that are not checked by the DL reasoner (which checks the *maximum* cardinalities) due to the open world assumptions, as the lack of values does not necessarily mean they do not exist.

D. Authoring

Once a set of OML vocabularies are defined, they can be used in OML descriptions that use those vocabularies' terms. openCAESAR supports methodology-aware OML authoring tools through *adapters* for existing systems engineering tools.These adapters translate information back and forth between a non-OML authoring tool and OML, using openCAESAR's authoring tool and OML APIs. This allows for greater agility in using tools by providing an adapter.

openCAESAR also supports OML authoring through OML workbenches: *Rosetta* (Eclipse-based) and *Luxor* (VSCode-based). These workbenches allow modeling with OML directly using its textual grammar formalized in EBNF. EBNF frameworks like Xtext and Langium can then generate APIs to load and save models in that grammar.

In contrast, MOF's textual syntax, called XMI, is described with informal mapping rules from MOF to XML, which can be used to infer an XSD schema for a MOF-based language. This method makes the process of obtaining the XSD schema error-prone, and the mapping rules provide several variability points leading to interoperability issues[4].

Rosetta also allows authoring through OML's graphical syntax as shown in Figure 4, and can be used to develop methodology-specific UI viewpoints that makes the experience of the authors much easier. For example, Figure 5 shows a table-based viewpoint for editing missions. Viewpoints can support various UI widgets, and address methodology-specific functional and non-functional concerns.

E. Federation

openCAESAR organizes OML models in different Git-based *repositories* to support federated work in a project.

[4]For example, see https://www.omgwiki.org/model-interchange/

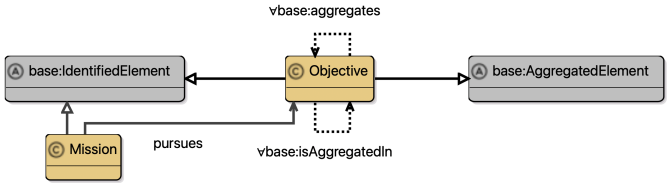

Fig. 4. A subset of the mission vocabulary in OML graphical syntax

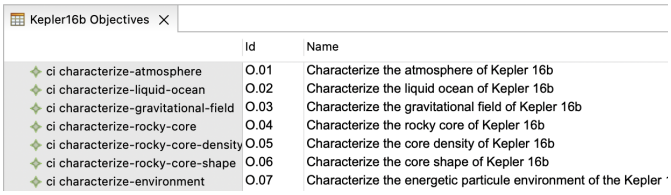

Fig. 5. A tabular view for authoring mission objectives

This enhances both agility and rigor by separating concerns based on authority, concern, and tool boundaries, improves scalability by allowing for focused models, and enables collaborating in a parallel and asynchronous manner without unintended impacts. As described in the *configuration* section, these models can then be integrated by repositories declaring dependencies on artifacts published by other repositories.

In the Kepler16b example, there exists a dependency on an OML library called *core-vocabularies* that provides reusable vocabularies like the well-known Dublin Core, OWL, RDFS, and XMLSchema. Such a library is managed in a separate Github repository and published to MavenCentral. The Kepler16b project declares a version dependency on this library. The build process then downloads a read-only copy and makes it available for the project's own OML models.

Further, ontologies can thus be organized in multiple folders in an OML project. For example, ontologies could be organized in different folders based on criteria (manually authored vs generated by tools, pertaining to a particular subsystem, main information vs example information, etc.). The agility to do this organization comes from the OML syntax allowing statements about elements (even ones defined elsewhere) to be stated in multiple ontologies. That is, a vocabulary or description can import another and add extra statements about members of that ontology, effectively extending the imported members. This is not necessarily the case for other modeling languages that rely on statements about a subject being nested under the subjects in a single model.

The Kepler16b example heavily relies on ontology importing. The description is organized into multiple ontologies that import each other to keep them smaller and easily reusable. For example, the *components* defined in one ontology is imported by others to describe component masses and their interfaces.

F. Configuration

Many system modeling tools have their own Content Management System (CMS) capabilities that have some (but not all) of the required functionality. In the case of openCAESAR, OML models are textual files which can be stored in Git repositories. This leverages Git's powerful configuration management capabilities and provides users with familiar software engineering features such as providing tracking revisions through commits and tags, allowing parallel and exploratory work through branches, and code review before merging changes onto a protected main (baseline) branch.

While using Git as a CMS facilitates the revision, collaboration, and review use cases, this does not address the release and dependency management use cases. *Artifact management systems* (AMS) including Maven and Ivy allow publishing artifacts to repositories and making them accessible as dependencies of other artifacts. These capabilities allow an agile tool-independent approach to dependency management.

openCAESAR fully exploits this dependency management capability in OML projects, where every project declares its own version as well as versions of its dependencies. The versioning scheme used is *semantic versioning*[5], where a version is made up of "major.minor.patch" components. Using these scheme clearly communicates the intent of a new version to its downstream adopters. For example, dependency version ranges such as `1.2.+` indicate that only patch updates can be used, or the range `+` will receive all updates for continual tracking of a dependency. The Kepler16b example declares a dependency on the *core-vocabulary* artifact and specifically version 5.+, i.e., fixing the major revision to 5 while requesting the latest minor release.

G. Integration

In openCAESAR, *integration* is about ensuring that each of the models, tools, analyses, and reports are correctly working together, in the style of DevOps CI/CD [5]. Typically, the engineering knowledge is federated across authority and concern boundaries. This results in multiple models being managed and published in different CMS/AMS repositories with cross dependencies declared between them.

openCAESAR makes it easy for development to proceed in parallel with frequent integrations along the way. It utilizes Gradle (for analyses orchestration), Maven (for releases/dependencies with semantic versions), Docker (for provisioning CI environments) and CI tools (for running the entire process upon a change to the Git repo). The continuous integration process starts with a change pushed to a Git repository, which triggers an analysis pipeline to analyze the change. In this pipeline, fresh read-only copies of the dependencies are downloaded from their repositories, the models are transformed into inputs suitable for analysis tools, the analyses is run to produce reports, and the reports are published for review.

As with usual CI principles in programming, issues with openCAESAR models, tools, analyses, or reports will be detected and made visible in this process. The responsible party can then address the problems in a subsequent change to trigger the CI/CD process again. This enforces rigor by uncovering any problems (in the models themselves or their

[5]https://semver.org/

dependencies) as soon as possible. A change that is a candidate for release can be *tagged* in Git to trigger the extra steps of packaging the system models, adding the version to the package, and publishing it to an AMS repo. Once published, it can be used as a dependency of another model.

H. Analysis

In openCAESAR, *analysis* happens at two levels. First, the *logical consistency* of the models is checked using OML and description logic reasoners. Second, other analyses investigate other properties of the system. OML models can easily be checked for logical consistency with an off-the-shelf description logic (DL) reasoner, as noted in Section III-C. This check is typically done very early in an analysis pipeline as in the Kepler16b example CI pipeline. Completeness checks can be encoded in a vocabulary directly using minimum/exact cardinality restrictions on properties. They can also be captured using libraries of well-formedness rules that query consistent models. The advantage of the latter is the ability to separate those two concerns, and the ability to use other analyses frameworks than a DL reasoner.

Rigorous analyses involving functional and non-functional requirements can be performed using various frameworks including solvers, simulators, etc. depending on the nature of the description (uses mathematical equations, has operational semantics, etc.). openCAESAR uses Docker and Gradle for analysis orchestration. It allows fully declaring (and managing) the environment used for analysis including the version of analysis tools that will be used.

I. Reporting

openCAESAR provides frameworks to develop viewpoints from OML repositories to facilitate *reporting* tasks. These viewpoints can either collect data from the OML source models directly, or from data that was produced by analyzing those sources. Steps in an openCAESAR analysis pipeline dedicated to producing views or reports include queries to fetch the designed data patterns, reductions to transform them into a format closer to the view, and finally renderings to visualize them as static documents or interactive viewers. Such views are tagged using the same Git tag of the source OML model and published to a repository that makes them accessible to stakeholders. For example, one SPARQL query in the Kepler16b example produces a table for components in the system along with their mass in kilograms. As shown in Figure 6, a short Javascript code snippet using the D3 library can then visualize the masses and allow for interaction[6].

A web server can index these reports and makes them accessible and searchable. Stakeholders will be able search those views by keywords or tags, and can launch them to view them, comment on them, or download them. The full provenance of the views is also traceable so that stakeholders can determine the specific versions of the OML models and analysis tools involved in the report.

[6]http://www.opencaesar.io/kepler16b-example/

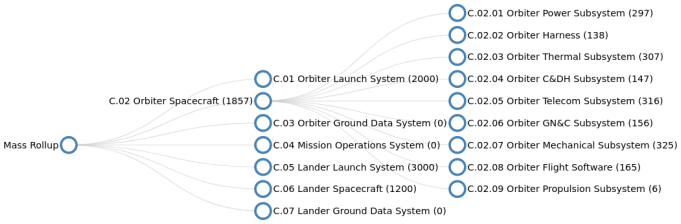

Fig. 6. An interactive mass roll-up visualization in an openCAESAR report

The openCAESAR methodology and framework thus dramatically improves the change request process from being ad-hoc and unrepeatable to a rigorous and systematic one. Despite the rigor of explicit processes, the user is also afforded agility by being able to easily define additional queries and reports on the system, and having those reports efficiently distributed to interested stakeholders.

IV. Lessons Learned and Open Challenges

This section discusses lessons learned from applying openCAESAR to past JPL projects, along with open challenges.

a) Strong Requirements for Agility and Rigor: As described through this article, JPL has demanding requirements for agility and rigor from both MBSE methodologies and tools. On the rigor side, the process from requirements to design requires time-intensive expertise from information architects. On the agility side, there is no silver bullet and any framework/methodology must be adapted to account for different contexts, teams, and stakeholders.

Challenge: Developing good and reusable vocabularies is very challenging especially in a multi-domain environment. OML helps to address this by being able to import core ontologies to form a layered ontology approach, but still the meaning of terminology and concepts may evolve over time. This requires enormous effort to define and describe concepts in a particular context and ensure that the meaning is retained throughout the project lifecycle.

b) Improving Adoption: A key lesson learned throughout the application of openCAESAR in JPL was the power of DevOps and CI/CD practices that can be applied to models and their ecosystem. These practices are common-place from software engineering but have not yet reached saturation across MDE and systems engineering. openCAESAR is an excellent starting point for these practices, but more can be done.

Challenge: Experiences showed that it was sometimes difficult to move to the openCAESAR methodology/framework from existing practices. Some teams were more ready than others. Analyses and maturity level assessments will be required to systematically determine when openCAESAR is beneficial for a project/team. This assessment must also take the use of MBSE existing tools into account, to ensure that productivity and experience is not lost during the transition.

c) Stakeholder Engagement: openCAESAR has a strong focus on providing analyses and reports for non-technical stakeholders, due to the nature of projects at JPL. This requires overcoming technical challenges to be able to create and

deploy queries and reports such that non-technical personnel could develop their own views onto the system and understand the impact of changes.

Challenge: Further work is required to make change requests easy to understand for reviewers, such as being able to understand the changes ("diffs") at a semantic level instead of a syntactic level [18]. This is crucial to allow non-technical personnel to understand the change. One current approach is that differences in the derived reports are presented alongside a change request. This strategy should be further integrated technically such that a stakeholder can dynamically create new views to understand a change at the time of a change request.

d) Standardization and Integration: OML was designed to be a language for capturing general descriptive models as opposed to other languages that are designed to enable certain particular kinds of analysis (e.g., Modelica). As a general modeling language, OML has broad expressiveness but has to be adapted (through vocabularies) to say anything.

Challenge: JPL has produced a set of vocabularies to describe specialized concepts and relations but these may not be generalizable to all system engineering since the processes for engineering models differs greatly between industries and organizations. Therefore, as a general language it cannot compete with specialized languages like Modelica in those particular analytical domains, or in expressing geometry such as in CAD tools. However, if one needs a system model to describe the architecture of a system and explain how the CAD model relates to the Modelica model, OML provides a flexible and adaptable way to do it.

Challenge: Enormous effort is required to normalizing different engineering tools into standard ontologies, as SysML has one vocabulary while DOORS has another. In the openCAESAR framework, adaptors can be fragile where it is difficult to perform back-and-forth editing and analysis between OML and other tools. One solution is to utilize chains of model transformations. However, it is unclear how to perform this in an agile and rigorous manner, especially to provide assurance that the model transformation has been correctly performed.

Challenge: Authors need to recognize their content in other formalism or tools. There is thus a need for traceability such as provide Internationalized Resource Identifiers (IRIs) to provide a Digital Thread throughout the project and across tools. However, this requires further development to lower cognitive effort and specify the precise semantics of element traceability.

e) Aligning openCAESAR/OML with SysML: SysML is a well-used MBSE language as seen in Section V. However, the ontological and text-based approach employed in openCAESAR provides a different balance of agility and rigor. In particular, the clean semantics of OML, formal consistency checks, and ability to quickly query models and report on them provides clear advantages over SysML.

Challenge: JPL is in discussions with OMG on how to integrate SysML at a deep level into the openCAESAR framework to obtain the best of the OML and SysML worlds. One solution is to build an OML ontology of SysML to aid the creation of adaptors. However, the semantics of SysML do not map directly onto the formal semantics of OML.

V. Related Work

Talentino and Wood [9] discussed their experience supporting a team of systems engineers and software developers in the development of a large number of naval applications. The authors argued that the main areas of conflicts between the systems engineering and the agile software methodologies are related to documentation rigor, prioritization of non-functional requirements such as compliance, and openness to change.

Navas *et al.* discuss how a model-based approach provides agility in the context of Capella/Arcadia [12]. An iterative and incremental development process is discussed whereby models are iteratively refined and verified between development sprints. However, they mention that system-level verification is not integrated into a continuous automatic process as in the openCAESAR framework.

Jenkins and Rouquette [19] argued that while SysML is a rich and flexible modeling language for systems engineering, it lacks the rigor and semantics required for logical reasoning in system engineering such as verification of consistency and satisfiability analysis. They propose combining SysML with the Web Ontology Language (OWL) through profiling and model transformations. The goal is to enhance SysML semantics with OWL's formal logic, and improve the adoption of ontology-based languages in systems engineering by leveraging SysML's graphical notations and tool support.

Krupa [20] applied an agile process to the design of a flight control system, showing that combining agility with SysML for modeling the system artifacts provides many advantages over traditional systems engineering processes, such as the ability to develop specifications that meet the system requirements and customer's needs. The author also showed that the incremental nature of agile processes, where each iteration is verified against requirements, results in improved design.

Do and Cook report on a SysML-based MBSE approach for designing a Ground-Based Air and Missile Defence system [21]. They report SysML-based MBSE supports flexibility in modeling, but that this flexibility imposes a need to carefully build the model and SysML diagrams to fit the project.

Spangelo *et al.* apply SysML-based MBSE to an example cube satellite mission *FireSat* [22]. They demonstrate the multi-disciplinary nature of space-based missions like FireSat and Kepler16b requires extensive domain modeling and separation of concerns between physical and logical components.

Rountree reports on the MBSE process as applied to the NASA Mars Ascent Vehicle [23]. A combination of SysML, MagicDraw, and Doors Next Generation were utilized to capture the requirements of the system. Similar to openCAESAR, specialized viewpoints were produced to separate concerns, with web-based reports produced for stakeholders.

Boggero *et al.* present an MBSE architectural framework for SysML for structuring the stakeholders, needs, and requirements of a complex aeronautic system [24]. This architectural framework focuses on providing the *agility* to define and

model complex systems. Similar to openCAESAR, enhanced agility is achieved through the extensive use of ontologies and viewpoints, packaged as the *AGILE4Profile* SysML profile.

Hennig *et al.* provide a detailed discussion on the application of an ontology for satellite design [25]. The ontology was created in the OWL2 ontology format manually from a UML system model. While this approach allowed for consistency and verification checks, there were issues identified by employing OWL2: a) closed-world checks could not be applied, b) data could be modeled in the A-Box which was out-of-scope of the T-Box, c) no built-in part-of/containment/aggregation relationships, and d) difficulty reasoning about numeric values. OML and available OML libraries address these issues.

These studies demonstrate that while other MBSE approaches are feasible, limitations exist either in their agility or rigor. In contrast, the openCAESAR methodology and framework attempts to bring together the best of these approaches to provide sufficient agility and rigor for MBSE practitioners.

VI. Conclusion

This article has discussed the openCAESAR MBSE framework and methodology developed at JPL. The seven core MBSE functionalities identified by JPL [6] were discussed in the context of *agility* and *rigor*: *representation*, *authoring*, *federation*, *configuration*, *integration*, *analysis*, and *reporting*. The Kepler-16b case study was then used to explain how openCAESAR supports these functionalities. Lessons learned from past JPL MBSE projects using openCAESAR were presented along with future challenges.

Future Work: JPL is currently enriching the Kepler-16b example and performing further open-source development of openCAESAR. To encourage the adoption of openCAESAR, JPL is also creating a diverse ecosystem of community of tool developers and users (e.g. [26]) through the development of seminars and workshops.

Acknowledgment

The research described here was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

References

[1] A. M. Madni and M. Sievers, "Model-based systems engineering: Motivation, current status, and research opportunities," *Systems Engineering*, vol. 21, no. 3, pp. 172–190, 2018.

[2] K. Henderson and A. Salado, "Value and benefits of model-based systems engineering (MBSE): Evidence from the literature," *Systems Engineering*, vol. 24, no. 1, pp. 51–66, 2021.

[3] T. Huldt and I. Stenius, "State-of-practice survey of model-based systems," *Systems Engineering*, vol. 22, no. 2, pp. 134–145, 2019.

[4] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "The agile manifesto," 2001.

[5] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of devops concepts and challenges," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–35, 2019.

[6] D. Wagner, S. Y. Kim-Castet, A. Jimenez, M. Elaasar, N. Rouquette, and S. Jenkins, "CAESAR model-based approach to harness design," in *2020 IEEE Aerospace Conference*. IEEE, 2020, pp. 1–13.

[7] D. A. Wagner, M. Chodas, M. Elaasar, J. S. Jenkins, and N. Rouquette, "Ontological Metamodeling and Analysis Using openCAESAR," in *Handbook of Model-Based Systems Engineering*, A. M. Madni, N. Augustine, and M. Sievers, Eds. Cham: Springer, 2022, pp. 1–30.

[8] C. Hildebrandt, A. Köcher, C. Küstner, C.-M. López-Enríquez, A. W. Müller, B. Caesar, C. S. Gundlach, and A. Fay, "Ontology building for cyber–physical systems: Application in the manufacturing domain," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 3, pp. 1266–1282, 2020.

[9] G. Tolentino and J. Wood, "Balancing systems engineering rigor with agile software development flexibility," *Insight*, vol. 21, no. 2, pp. 25–28, Jan. 2018.

[10] O. L. De Weck, A. M. Ross, and D. H. Rhodes, "Investigating relationships and semantic sets amongst system lifecycle properties (ilities)," *ESD Working Papers;ESD-WP-2012-12*, 2012.

[11] K. D. Willett, R. Dove, A. Chudnow, R. Eckman, L. Rosser, J. S. Stevens, R. Yeman, and M. Yokell, "Agility in the future of systems engineering (FuSE)-a roadmap of foundational concepts," in *INCOSE International Symposium*, vol. 31, no. 1. Wiley, 2021, pp. 158–174.

[12] J. Navas, S. Bonnet, J.-L. Voirin, and G. Journaux, "Models as enablers of agility in complex systems engineering," in *INCOSE International Symposium*, vol. 30, no. 1. Wiley Online Library, 2020, pp. 339–355.

[13] J. Ma, G. Wang, J. Lu, H. Vangheluwe, D. Kiritsis, and Y. Yan, "Systematic literature review of MBSE tool-chains," *Applied Sciences*, vol. 12, no. 7, p. 3431, 2022.

[14] M. Chami and J.-M. Bruel, "A Survey on MBSE Adoption Challenges," in *INCOSE EMEA Sector Systems Engineering Conference (INCOSE EMEASEC 2018)*, Berlin, Germany, Nov. 2018, pp. 1–16. [Online]. Available: https://hal.science/hal-02124402

[15] A. L. Ramos, J. V. Ferreira, and J. Barceló, "Model-based systems engineering: An emerging approach for modern systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 1, pp. 101–111, 2011.

[16] D. Gibson, "Factors affecting systems engineering rigor in launch vehicle organizations," Ph.D. dissertation, University of Central Florida, 2019.

[17] J. Denil, R. Salay, C. Paredis, and H. Vangheluwe, "Towards agile model-based systems engineering," in *CEUR workshop proceedings*, 2017, pp. 424–429.

[18] M. Zadahmad, E. Syriani, O. Alam, E. Guerra, and J. de Lara, "DSM-Compare: domain-specific model differencing for graphical domain-specific languages," *Software and Systems Modeling*, pp. 1–30, 2022.

[19] J. S. Jenkins and N. Rouquette, "Semantically-rigorous systems engineering modeling using SysML and OWL," in *Proceedings of the 5th International Workshop on Systems & Concurrent Engineering for Space Applications*, 2012.

[20] G. P. Krupa, "Application of agile model-based systems engineering in aircraft conceptual design," in *Proceedings of the 31st Congress of the International Council of the Aeronautical Sciences*, 2018.

[21] Q. Do and S. Cook, "10.5. 1 an MBSE case study and research challenges," in *INCOSE International Symposium*, vol. 22, no. 1. Wiley Online Library, 2012, pp. 1531–1543.

[22] S. C. Spangelo, D. Kaslow, C. Delp, B. Cole, L. Anderson, E. Fosse, B. S. Gilbert, L. Hartman, T. Kahn, and J. Cutler, "Applying model based systems engineering (MBSE) to a standard cubesat," in *2012 IEEE aerospace conference*. IEEE, 2012, pp. 1–20.

[23] I. Rountree, "MBSE applications for the MSR SRC Mars Ascent Vehicle," in *Aerospace Conference (AERO)*. IEEE, 2022, pp. 1–14.

[24] L. Boggero, P. D. Ciampa, and B. Nagel, "An MBSE architectural framework for the agile definition of system stakeholders, needs and requirements," in *AIAA Aviation 2021 Forum*, 2021, p. 3076.

[25] C. Hennig, A. Viehl, B. Kämpgen, and H. Eisenmann, "Ontology-based design of space systems," in *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II 15*. Springer, 2016, pp. 308–324.

[26] M. O. Nachawati, G. Bullegas, A. Vasilyev, J. Gregory, A. Pop, M. Elaasar, and A. Asghar, "Towards an open platform for democratized model-based design and engineering of cyber-physical systems," in *Modelica Conferences*, 2022, pp. 102–114.