


Light-Weight MBSE Approach for Construction Equipment Domain - An Experience Report *

1st Johan Cederbladh *Mälardalen University*
Västerås, Sweden
johan.cederbladh@mdu.se 

2nd Jagadish Suryadevara *Volvo Construction Equipment*
Eskilstuna, Sweden
jagadish.suryadevara@volvo.com

December 15, 2023

Abstract

Model-Based Systems Engineering (MBSE) has emerged as a de facto standard practice for complex systems development. However, despite standard frameworks, methods, and tools, the MBSE practices in industrial contexts are far from mature. This is particularly true for traditionally hardware-intensive industrial systems and complex products such as heavy construction machinery. This paper outlines a light-weight MBSE methodology developed within the construction equipment domain. The approach is based on customizing a general MBSE methodology, such as RFLP, guided by a core ontology for architecture modeling. The ontology defines the main architecture concepts and corresponding modeling views to develop an architecture baseline for the corresponding system model. The initial architecture baseline is developed bottom-up instead of a traditional top-down approach. As we demonstrate in this paper, the minimalist approach paves the way for faster deployment of MBSE in industrial contexts with low modeling experience and legacy development silos across electronics, software, and hardware domains.

MBSE, Methodology, Architecture, Bottom-up

1 Introduction

Volvo Construction Equipment is a global manufacturer of heavy machinery and equipment in the Construction domain. Construction Equipment (CE)

*This is a pre-print of a paper in the APSEC30 conference. This work was partly funded by the AIDOaRt project, an ECSEL Joint Undertaking (JU) under grant agreement No. 101007350.

is utilized in a wide array of operational contexts and environments, such as mining, forestry, road paving, excavation, etc. Traditionally, the Systems Engineering (SE) approach within the CE domain has been hardware (HW)-centric despite the multi-disciplinary contexts involved, e.g., software (SW), electronics, electrical domains, etc. However, with the increasing complexity of machine functionality and the increase of SW concerns, the traditional HW-centric SE approach is increasingly ineffective. The CE domain is transforming towards digitalization with increased SW systems (embedded or inter-connected), ever-increasing functionality (e.g., business services), and corresponding data needs, in this context, establishing traceability (so-called *digital thread* [31]) through various system elements enable continuous development (e.g. DevOps [26]) of the involved systems and functionality throughout the machine lifecycle.

Model-Based Systems Engineering (MBSE), where digital models represent system specification and design artifacts, is becoming the standard industry practice during development and operational phases [10, 34]. Compared to traditional document-centric development (especially in the CE domain), the SE processes in model-based approaches are centered around developing various visual and machine-readable models, even from the very early stages of development [15]. Increased use of digital models enables enhanced SE capabilities such as traceability, early analysis, etc. [8, 17, 18]. Models additionally can help leverage abstraction for cross-domain integration and more holistic processes [31]. However, as experience and various studies have established, the application of MBSE, especially for industrial-scale projects, requires a steep learning curve and a long-term enterprise investment [6, 33, 36]. At the same time, MBSE is seeing a significant increase in industrial attention with corresponding success stories [2, 7]. So although MBSE is becoming a standard practice with several observed benefits, it still remains a challenge for industrial adoption. Particularly non-modeler engineers struggle to conform to languages and tools used with traditional model-based methods.

In this paper, we outline a light-weight MBSE methodology developed within the CE context and discuss the experience of introducing a model-based approach in an industrial setting. The methodology is based on adopting general systems modeling principles, however, customized for the CE context. The methodology facilitates an incremental and iterative approach in developing an architecture baseline for a system model, later used for detailed design and development. The approach emphasizes method and process-centric rather than learning specific modeling languages or tools. Experience shows that this has facilitated a faster understanding of modeling techniques and communication between non-expert modelers and other stakeholders. To facilitate the languages agnostic approach, we re-use a core ontology for CE and later highlight a practical application in a real system using a standard modeling language.

The rest of the paper is structured as follows. Section 2 presents an overview of SE concepts and some standard methodologies in SW and HW domains in addition to related work. Section 3 presents the motivation and overview of the proposed systems (architecture) modeling methodology. Section 4 describes the deployment of the methodology in a project context. The overall experience

and related issues are discussed in Section 5 followed by conclusions in Section 6.

2 Background and Related work

Systems Engineering (SE) is a multi-domain discipline that considers the entirety of a system life-cycle, from needs and requirements to finalized products and their operation [34]. The following standards govern SE practices. ISO 15288 "Software and systems engineering"¹, defines standard SE processes. ISO 26550 "Reference model for product line engineering and management"² extends the previous standard for product line development. ISO 42010 "Systems and software engineering — Architecture description"³, defines a reference model for architecture description for a System-of-Interest (SoI). As defined by the architecture standard, we consider the notion of architecture *views* central to a MBSE methodology. An architectural view addresses a particular stakeholder concern and describes the system elements and their relationships with respect to the concern. In the proposed light-weight methodology, we define a set of architecture views as a baseline for the system model.

Architectural description languages (ADLs) are widely used in SW engineering domains. For example, the EAST-ADL language is a de-facto standard for embedded SW development in the automotive domain [11]. The EAST-ADL language defines SW architecture views-stack (based on abstraction and refinement). The implementation layer of the stack is defined by AUTOSAR standard⁴. AADL (Architecture Analysis and Description Language) is used for the specification of both SW and HW (electronic) architecture views primarily in the Avionics domain [13]. These mentioned SW views, mainly EAST-ADL, apply in the CE domain. However, the methodology in this paper focuses on defining system-level architecture views.

Generally, "models" are abstract representations of reality used for some cognitive purpose [24]. The use of models is quite widespread in design engineering, and typical examples might include CAD, SW models, schematics, and various physics-based simulation models [21, 23, 25, 29]. In the MBSE context, the use of models has increasingly been motivated for early phases of system development prior to detailed system designs [1, 28]. SysML⁵ is the *de facto* standard language for system modeling [14] and provides visual notation for documenting various system aspects such as requirements, behaviour, structure, and parameters. Correspondingly, various diagrammatic views can be created using the above notation. SysML is a general-purpose modeling language that requires domain-specific methodologies and tool support for industrial applica-

¹ISO 15288 standard available at: <https://www.iso.org/standard/63711.html>

²ISO 26550 standard available at: <https://www.iso.org/standard/69529.html>

³ISO 42010 standard available at: <https://www.iso.org/standard/74393.html>

⁴<https://www.autosar.org/>. AUTOSAR enables integration of multi-vendor SW and electronics components to implement EAST-ADL specifications

⁵SysML standard available at: <https://www.omg.org/spec/SysML/>

tions. Besides general methodologies [14], [27], there exist some domain specific frameworks such as ARCADIA [30], and others [12].

While many approaches, methods, and methodologies are defined for MBSE [10, 27, 35], it is often difficult to apply for new adopters. In this paper, we specifically consider the CE domain, and although many works present domain-specific extensions for SysML, there is little work in our considered domain. In previous work, we have presented the experience of adopting MBSE in CE [33]. The work has highlighted the perceived usefulness of MBSE but also pinpointed the lack of added analytical capabilities in off-the-shelf tool solutions and the overall poor integration in tooling landscapes. Similarly, the other works of MBSE in the CE domain have shown initial success. However, these works have also demonstrated the need for more robust integration and development of underlying methodologies and frameworks [32, 33]. In this regard, we aim to use previous knowledge and extend the existing approaches to accommodate expressed limitations and realize perceived benefits.

Several model management approaches known as "grids" exist in literature [3, 22, 27]. At this stage, these approaches are common practice for using MBSE, and combines abstraction with stakeholder concerns to create intersecting grids that can drive development through various standard views. While separating concerns is useful, the existing methodologies are often hard to apply "as is" due to the overly abstract definition and high-level application descriptions. Similarly, the approaches expect traditional top-down workflows, which is not always suitable in a industrial SE context. The typical grids are also overly comprehensive and inhibit the creation of baseline views by overly complex methods often mapped to specific languages like SysML. As such, the need for more light-weight and, in the case of some HW-intensive systems, bottom-up approaches could improve adoption efforts. Our work additionally focuses on enabling a holistic light-weight approach to MBSE. Literature has few examples of light-weight approaches. Hanna *et al.* highlight a modular light-weight MBSE approach [16]. Their work focuses on the process of modeling methodically, and the main contribution is how to perform the modular approach systematically. The work emphasizes the need for processes and data to be linked and traced through development. Their work has similarities with what is presented in this paper, but our focus is different as the application of the approaches is not the same. Also, our work is based on the industrial focus and needs. While light-weight MBSE is more scarce, holistic approaches exist for other domains [4, 19, 20, 37]. These approaches are seeing increasing interest with the improvements to MBSE maturity, particularly tool integration and exchange of data. While these works are closely related, ours focuses on the CE domain, particularly highlighting a bottom-up rather than a typically top-down approach.

3 Methodology motivation and overview

In the CE domain, system development has traditionally been HW-centric, and HW modularity has been a driving factor for cost-effective design and modular

production. The HW modules are the main building blocks for designing configurable product platforms and overall management of corresponding product families. The individual product types are configured (through modules and containing design elements) from corresponding product platforms. Fig. 1 captures the core architecture ontology behind the modular architecture designs, albeit from the HW perspective, but applicable for other solution domains. The other solution domains of increasing interest are SW and electronics development. The latter solution domains have corresponding architecture methodologies, as described in Section 2. The mentioned solution domains have become development silos hindering the successful deployment of MBSE methodologies and realizing the stakeholder functionality required at the system level. In this paper, we present a light-weight MBSE approach for developing an architecture model at the System level that fulfills the following objectives:

- Traceability of SW and HW elements to stakeholder needs and system requirements.
- Specification of functionality at the system level.
- Specification of system (logical) architecture, i.e., logical partitioning of system functionality.
- A system model for stakeholder (non-expert modelers) communication.

Many standard methodologies described in the literature cater to the above objectives in general. However, as the experience points out, the industrial deployment of MBSE has not been very successful often due to the following reasons.

- Legacy methodologies disconnected from traditional top-down approach prescribed by MBSE.
- The commonly used general purpose modeling language SysML is perceived as SW-centric and incomprehensible.
- Non-modeling system stakeholders often prefer office-tools such as Powerpoint, Excel, Visio artifacts
- Lack of a unified architecture methodology with clear separation-of-concerns (e.g., architecture vs design).
- Lack of principles and guidelines to handle traceability, variability, etc
- Lack of leverage/synergy with HW modularity principles and methodologies, especially in the CE domain.

The light-weight approach described in this paper considers the above pain-points in defining a modeling approach, albeit leveraging existing methodologies. The architecture ontology described in Fig. 1 provides the conceptual framework for defining a minimal set of modeling views to develop a system

model. The approach emphasizes the quick adoption for non-modelers via the bottom-up approach that bases the models on already well formed machine solutions. Using existing machine solutions as a blueprint, the model represents a compact architecture view of the System-of-Interest (SoI), which can later be extended/complimented, as this paper shows, with other views, e.g., functional (behavior) as prescribed by standard modeling methodologies. In the rest of this section, we describe the transformation of the ontology into a baseline of system (architecture) views and corresponding building blocks. The approach is aligned with traditional RFLP modeling techniques, as shown in Fig. 2 and Fig. 3.

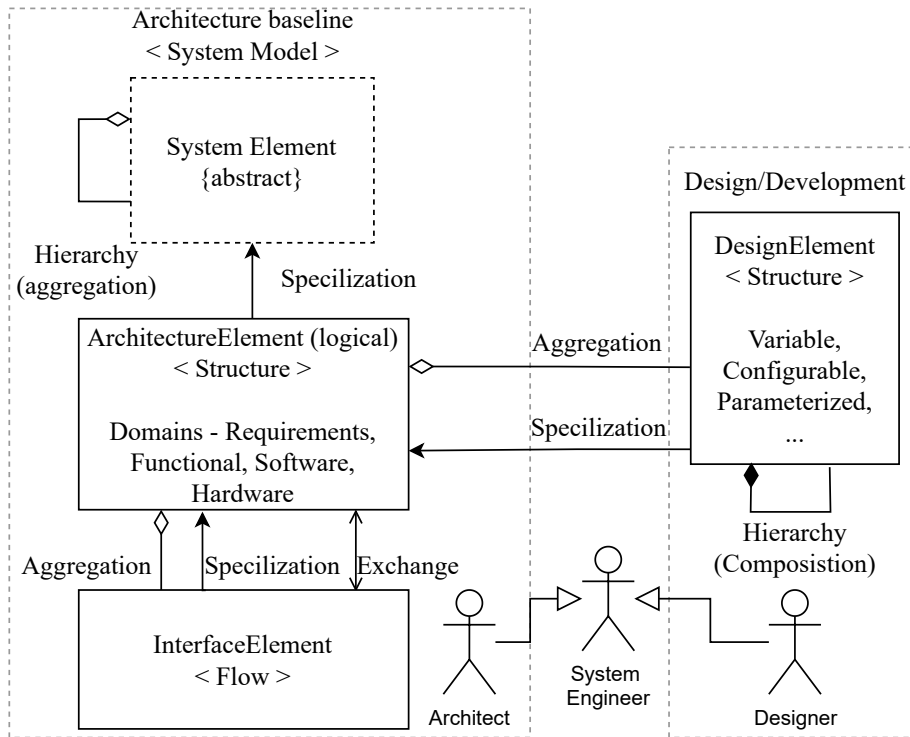


Figure 1: A core ontology for architecture baseline and design/development.

3.1 Architecture Ontology to a Baseline Model

The architecture ontology, as shown in Fig. 1, was defined in an earlier work [9], as a foundation for a cross-domain unified framework for developing Product Service Systems (PSS). This paper uses this ontology to define an architecture baseline model for a SoI (or Constituent System for a PSS) to be realized in HW and SW. The ontology is the basis for modeling the architecture baseline described further in the remaining part of this section.

ArchitectureElement and corresponding specialization **DesignElement** are the main building blocks for developing architecture views corresponding to specific stakeholders, e.g., System Architect (or Design Engineer) concerns. The **DesignElement** belongs to the leaf-level of logical decomposition of an **ArchitectureElement**. Special kinds of **ArchitectureElement** are defined corresponding to solution domains, e.g., **SoftwareBlock** for SW development, **Module** for HW engineering, etc. The logical architecture elements at the system level are concepts such as **System** and **SubSystem**. The domain-specific architecture elements are further decomposed in terms of detailed design elements, e.g., **SoftwareComponent** in the SW domain, **DesignUnit**, **Part**, **HW_Interface** etc in the HW domain. It can be observed that **ArchitectureElement** is based on the principle of low-cohesion and low-coupling (corresponding to L in Fig. 2), whereas the **DesignElement** is based on high-cohesion and high-coupling (corresponding to P in Fig. 2) [5]. The detailed designs in corresponding solution domains are out of the scope of the light-weight methodology described in this paper. The methodology is aimed at developing an architecture baseline (model) with views consisting of **ArchitectureElements** and its specializations in corresponding solution domains. The baseline architecture views are further detailed for *System Architecture* and *Traceability*.

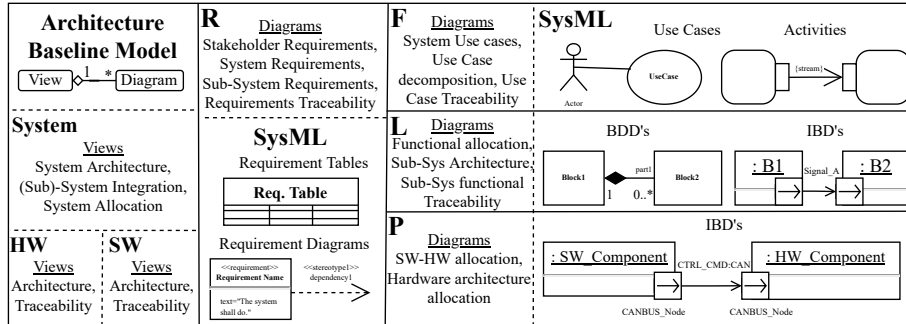


Figure 2: Proposed reduced grid approach following the RFLP classification for architecture baseline models. The baseline consists of Views which are based on diagrams. The grid details which diagram belongs to which modeling abstraction and are mapped to the SysML language.

System Architecture Views The logical partitioning of the SoI in terms of **System** and **SubSystem** elements. In the proposed methodology, the initial version of this architecture view is influenced by the HW architecture often established early (in the CE domain) during the development phase. The view consists of largely HW-centric subsystems (whose dynamic behavior is controlled by corresponding SW blocks). The architecture views of the subsystems are also modeled in terms of SW and HW architecture elements (seen in the core ontology). The stakeholder concerns of these views correspond to the SW and HW integration and establishing the logical interfaces between the subsystems.

It should be noted that the SW-centric subsystems, e.g., HMI, Telematics etc are not part of the baseline architecture.

Traceability Views The architecture baseline model includes the traceability and allocation views. These views (largely based on the tool support) enable stakeholder communication and overall project management. The traceability views are based on the stakeholder allocation of requirements and system behavior onto the architecture elements seen in Fig 1. These views influence the overall quality and effectiveness of the MBSE process.

The following subsection describes the modeling approach to develop the architecture baseline model described above.

3.2 Architecture Baseline Modeling

As illustrated in Fig. 2, the light-weight approach is based on developing the architecture baseline within general top-down approaches such as RFLP. The architecture baseline model (mainly system structure views, i.e., logical, HW, SW) described in previous sections is complimented with the functional view (i.e., the system behavior) and corresponding decomposition and allocation to architectural elements in the baseline model. Fig. 3 illustrates the overall modeling process/workflows of developing the baseline model.

As shown in Fig. 3, the baseline model consists of system and subsystem architecture views and the HW and SW subsystem integration views. The requirement traceability views are included in the baseline. Unlike the traditional top-down approach followed in general methodologies, the initial baseline model is developed in a bottom-up fashion, i.e., HW and SW integration views are modeled based on preliminary architecture data available from design engineers based on pre-existing/legacy designs. The preliminary HW designs (at the architecture level) are often available as far as the CE domain is concerned (e.g. so called A-Release phase) due to established modular product platforms. In CE context, mainly system development activities consist of complimenting the HW platform with corresponding functional and SW concerns.

The initial version of the baseline model acts as a reference model for project management, model organization, stakeholder communication, etc. The various architecture elements of the model may represent the "place-holders" for revised/updated elements resulting from the normal top-down process shown in Fig. 3. The bottom-up approach is amenable to the legacy processes and supports quicker adoption of MBSE methodologies. In essence, the baseline is created based on already well-documented physical designs. The transition to model-based from legacy methods is therefore minimal, and the non-modeler audience can use their existing expertise, while producing a valuable baseline for further development of missing concerns in the current setup (e.g, functional concerns or SW). The baseline then acts as the product-line representation which can be leveraged for top-down methods of capturing specific SoI requirements and specifications while engaging non-modeling engineers.

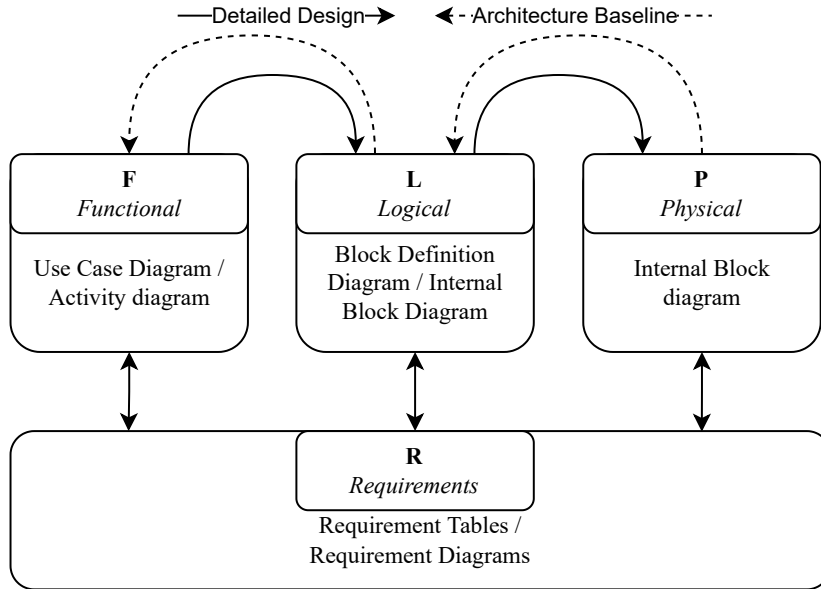


Figure 3: The bottom-up flow applied for the architecture baseline. Conversely, the detailed design considers a traditional top-down approach.

An architecture "view" consists of one or more SysML diagrams (it requires defining a model-management approach described later in this paper). The baseline architecture views described above are mapped onto SysML diagrams as illustrated in Fig. 2. We describe below the general modeling workflows (Fig. 3) corresponding to the RFLP methodology.

Modeling Physical Views The HW architectures corresponding to base subsystems are modeled first using BDDs (Block Definition Diagrams) and IBDs (Internal Block Diagrams). The subsystems consist of HW architecture elements such as `Modules` and `DesignUnits` described earlier. These correspond to the preliminary HW designs, e.g., modular HW platforms (often documented using *Microsoft Visio* tool) and are part of proven HW development methodology.

Modeling Logical Views As shown in Fig. 2, the logical architecture view, i.e., system and subsystem architectures, are modeled using SysML BDD and IBDs. Initially, the base subsystems correspond to those identified in the HW architecture views in the previous section. However, in the modeling context, the mechanical interfaces are replaced with logical interfaces capturing the energy, material, and information flows, if any. Further, the subsystem architectures include logical entities `SoftwareBlocks` described in previous sections. These serve as SW-HW (logical) interfaces in the context of subsystem behavior.

The views may be referred to as "deployment" as well as "integration" views as the functional entities, i.e., requirements (resp. behavior) are traced (resp.

deployed) to subsystem and architecture elements (both HW and SW) modeled in these views. It should be noted that the behavior diagrams, such as Use Cases (UCs) and Activities, are modeled (as described in coming sections) after the baseline model is established and agreed upon among the project stakeholders.

Modeling Functional Views The traditional MBSE methodologies follow a top-down approach which implies modeling system behavior (for an SoI) in black-box (e.g., *use cases* modeling) and in grey-box (e.g., *statemachine* and/or *activities* modeling). However, as the baseline architecture is available, the functional entities can be readily deployed onto the logical architectures. Specifically, the functional modeling is performed via use case and activity diagrams. UCs capture stakeholder requirements and customer needs, while the activities decompose these UCs further. Besides the system behavior modeling, architecture elements in logical views are assigned detailed behavior specifications, e.g., *statemachine* modeling for **SoftwareBlock** (for development of SW architectures during the design phase).

Modeling Requirements Views Requirements are modeled using the SysML requirement elements, and subsequent relevant diagrams and tables. Traceability is achieved by using various "links" (e.g, satisfy, allocate, trace, etc.) in requirement diagrams between requirement elements and other model elements (such as UCs or activities). The functional requirements are modeled later in the development phase, i.e., after the baseline architecture (driven by HW platform design) is established. The requirements decomposition is handled during the functional modeling described in previous sub-sections. The derived requirements are assigned to subsystems and other architecture elements already modeled. Both the functional and requirements traceability views are developed (with tool support) and included in the subsequent releases of the architecture baseline.

As shown in Fig.3 and described above, the overall development of the baseline architecture model (in the CE domain) follows non-sequential iterative phases due to the pre-existence of many system artifacts.

4 Modeling architecture views - examples

In this Section, we demonstrate the methodology on an example in the form of an Autonomous Hauler seen in Fig. 4.

We highlight our modeling methodology through the application of the machine in an autonomous site context and demonstrate the proposed approach through model views discussed in the previous Section. The machine considers many interconnected domains and acts as a representation of typical CE development. We limit our reporting to the SW and HW domains in the paper, but the approach is not limited to only these domains. Furthermore, we use the machine brake function/system as a recurring model element to make the reporting cohesive. In the excerpts of the models in later Sections, the "TA"



Figure 4: The Autonomous Hauler machine.

or "machine" naming refers to the autonomous hauler machine. The rest of this Section highlights the previously defined modeling views through SysML diagrams.

4.1 Physical hardware architecture view

Physical HW architecture views are created with IBDs, visualized in Fig. 5. Corresponding BDD's are made to detail the decomposition, we refer to the next Section for an example.

The modularity concept drives physical HW, and `Modules` are created based on `DesignUnits`. A `Module` is a re-usable configuration of HW `DesignUnits`. In Fig. 5, a `Module` is represented as a container of `DesignUnits`. The diagram consists of `Modules` and `DesignUnits` for the brake system HW-architecture. In the SysML language, blocks are used for the components/parts and flow ports for exchanges between blocks with corresponding connectors. In particular, each of the two `Modules` considers a brake system `Module`, instantiated for the front and rear brake respectively. Exchanges connect `DesignUnits` and should have a physical meaning, for example, electrical or hydraulic, and flow ports are typed with respective properties. This type of diagram is useful to detail the physical architecture of a system and attribute physical properties, such as the type of exchanges between `Modules` and `DesignUnits`. We note that this diagram only specifies HW concerns, as that is the standard practice for CE architectures, while later views add SW concerns as well.

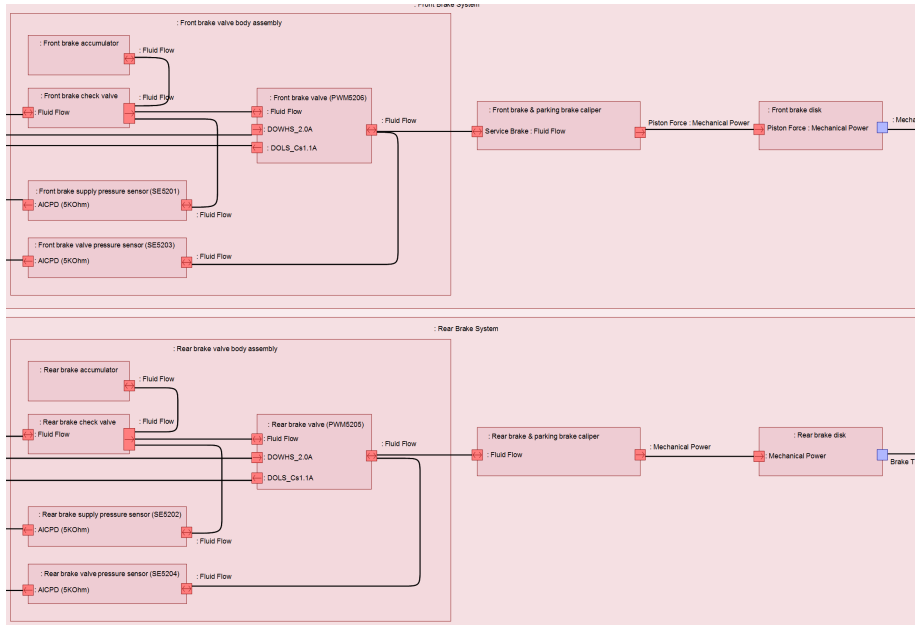


Figure 5: A IBD that shows the Physical HW architecture. The figure extracts a few Modules of the Brake System, internally made up of physical components and parts. An exchange between parts is modeled with flow ports.

4.2 Logical architecture Views

Logical architecture views highlight the components composing various systems and sub-systems and the corresponding interfaces. In the SysML language, decomposition is made using BDDs, and we highlight a BDD excerpt from the Machine in Fig. 6. We note that the physical HW BDD's are modeled the same way.

The extracted BDD is somewhat simplified, as there are more sub-systems for the entire machine, and there can in turn be a decomposition of sub-systems as well. The use of SysML associations defines that the machine consists of several sub-systems. In this figure, colors are additionally used to separate concerns for various development teams. While we do not highlight it in the diagram, different relationships can be used from the SysML language to highlight optional/variable blocks or multiplicity (for example aggregation). The logical interfaces of sub-systems are modeled in IBDs. Fig. 7 is an excerpt of the IBD for the Machine, considering, for example, the Brake System and Motion System.

The aim is to show logical interfaces and exchanges between the sub-systems, modeled with ports and connectors. At this abstraction, there are no physical properties allocated to the ports and signals compared to the HW architecture, even if the names can give a suggestion. The model abstracts the previously defined physical models to the logical domain in preparation for later functional

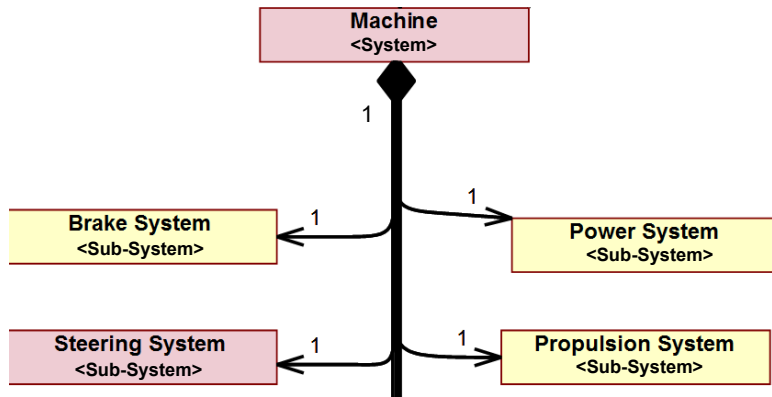


Figure 6: BDD decomposition highlighting the Machine system and some of its compositional sub-systems. The elements are modeled as SysML blocks and are connected with SysML associations, and different colors are used in the diagram to separate responsibilities for operational teams.

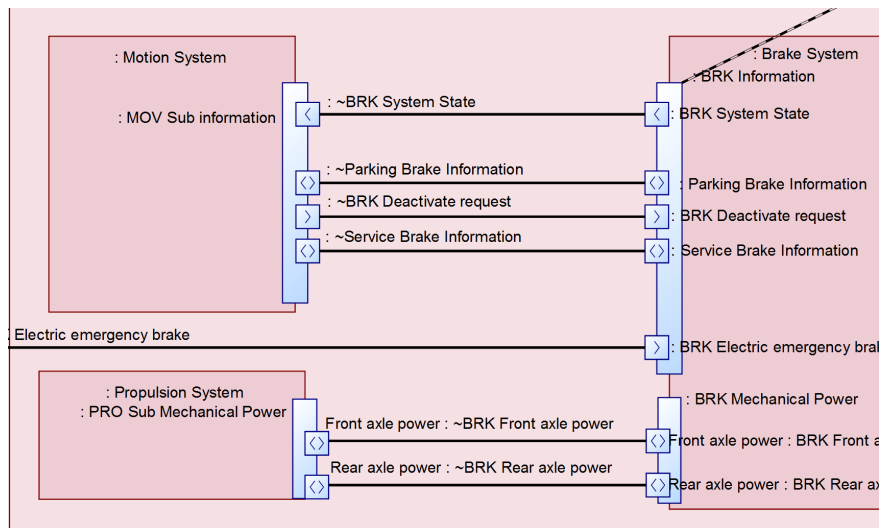


Figure 7: A IBD displaying the logical architecture in the Machine system. In the figure, the Brake System is shown to be interfaced with both the Motion System and the Propulsion System with logical interfaces.

views. When models become large, creating several IBDs at a given hierarchy is often useful. The diagram, for example, highlights the interfaces of the Brake System. It omits elements not interfaced with the system from the diagrammatic view. Another similar diagram might instead be focused on the Power System from Fig. 6, and in this case, omit the Brake System. It is important to reduce the duplication of information and instead aim to keep the overall model as a single source of truth. However, different views are often a valuable means of

better organizing the information instead of large cluttered diagrams.

Based on logical architectures like in Fig. 7, activities can be allocated to sub-systems via *swim-lanes*. Fig. 8 is an excerpt from the AD (Activity Diagram) **Regulate Machine Velocity** highlighting the activity partition and interfacing.

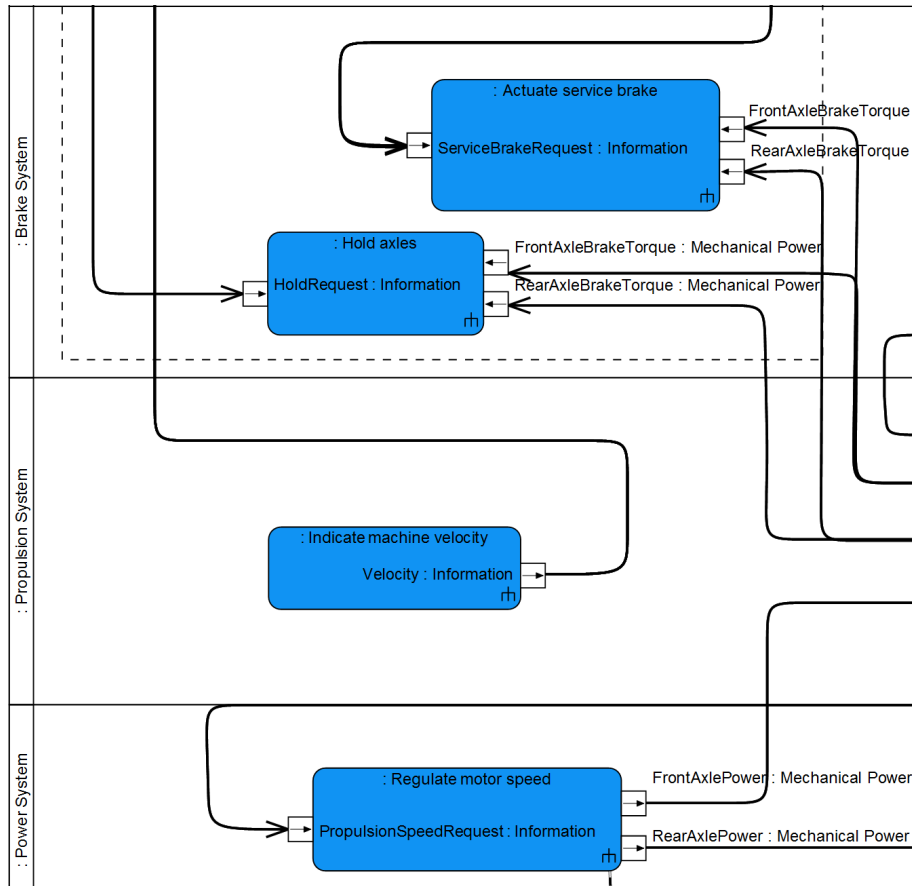


Figure 8: ADs highlighting the activities performed for a particular UC and the relation between them. Activities are connected via ports that have a direction and exchange of information. On the left-hand side, swim-lanes partition the activities into corresponding owning systems or sub-systems. The diagram highlights the behavior of a system/sub-system.

The different swim-lanes allocate different activities to a particular sub-system in the architecture. For example, the **Actuate Service Brake** activity is allocated to the **Brake System** in the AD. This way, the activities are created and allocated for the previously defined logical architecture defined in Fig. 7. This type of partitioning is used for HW concerns in the model to separate concerns further. At this stage, the system is considered from the logical per-

spective, and the diagram, for example, only specifies a signal to be of type "information" or "mechanical". It is important to note that activities often are defined in a hierarchy and decomposed. For example, Fig. 8 details an AD that is a sub-AD of a higher-level AD.

4.3 Functional Views

Functional concerns are modeled in two ways via UCDs and ADs. UCDs are first created from the breakdown of the stakeholder requirements and customer needs, while activity diagrams decompose the UCs. The functional views should match the high-level requirements (in UCs) with the logical architecture (in ADs) for complete traceability. Top-level ADs are, in turn, mapped from UCs. UCs are defined in UCDs and are decomposed by high-level activities. An example is provided in Fig. 9, where we consider the remote control for the autonomous machine.

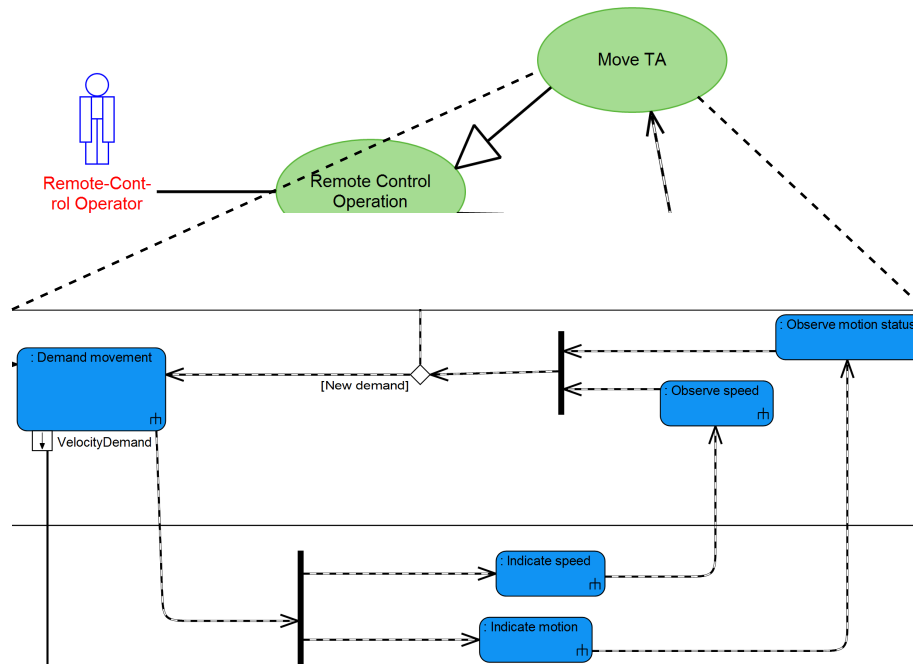


Figure 9: Excerpt from a High-level UCD depicting the autonomous operation of the vehicle. The `Remote-Control Operator` actor is linked with the `Remote Control Operation`, which has the sub-UC of `Move TA`.

In this type of diagram, the stakeholders are visualized and linked with related UCs. UCs can also be linked. In the figure, the site UC `Remote Control Operation` generalizes the `Move TA` which belongs to the machine. UCs can, in turn, be decomposed into corresponding high-level functional activities. In the figure, several activities are performed in a loop to maneuver the machine

continuously. One of these seen activities is further decomposed into the diagram in Fig. 8, defining a traceable hierarchy of UCs and ADs which can be used to navigate across diagrams and the model browser.

4.4 Integration views

Integration views can be created by leveraging the physical and logical architecture. Integration views aim to explicitly partition SW and HW, indicating what activities and components are allocated to SW and HW. Fig. 10 is an excerpt from the **Brake System** integration IBD.

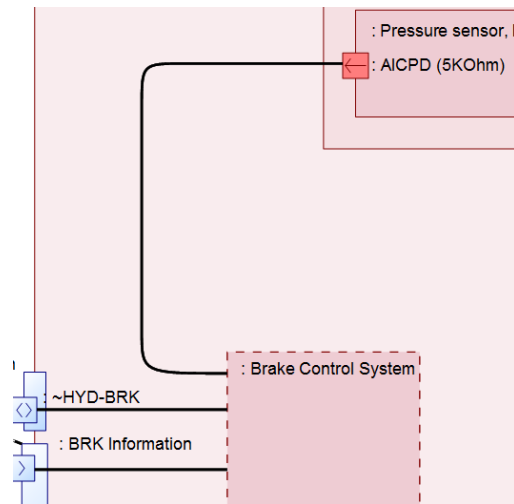


Figure 10: A IBD displaying the HW-SW integration of the **Brake System**. Internal elements are connected via ports and signals. The signals and elements are partitioned between SW and HW, where SW is visible in the bottom left and HW in the top right.

In the diagram, we see a similar view as in Fig. 5, but for the internal of the **Brake System**. Largely the aim of the view is the same, but there is added information in the interfaces and signals with the allocation of HW and SW. The diagram only shows two elements, the **Pressure Sensor** component (HW) and the **Brake Control System** component (SW). They are interfaced with an electrical signal (typed on the sensor port). Adding explicit partitions between domains such as SW and HW also significantly increases the model detail and emphasizes the design of required interfaces and signals. Integration views can also be made for the system's logical architecture, for example, in the case of functional views such as Fig. 11. These views act as a final piece of the architecture baseline.

In this case, the system's activities are partitioned instead of the components and modules from the physical architecture. The AD is modeled similarly to Fig. 8, only that the level of detail is increased with explicit exchanges between

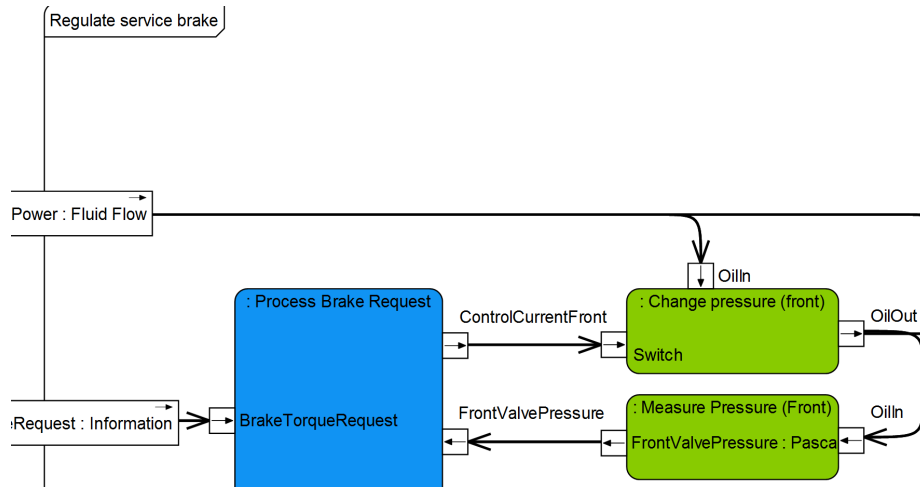


Figure 11: Functional allocation to SW and HW via AD partitioning. The signals are all described via physical and information properties, and SW/HW is visible and distinct via the activity color.

activities and clear partition. Visually the partition is performed with separate colours, but the types of the involved elements are also specific for their particular domain. The diagram highlights the **Regulate Service Brake** activity, and we see SW (in blue) and HW (in green) integration in terms of activities. In the case of HW this might also detail the physical properties, such as **Fluid Flow** in the figure, while for SW it can instead be **Information flow**.

In this way, the model is "living" and is refined based on the continuous efforts of the involved engineers. Practically it is not feasible to consider the development of any model view completely sequentially due to the often large collaborative effort and continuous requirement/design refinements.

4.5 Requirements views and traceability

Requirements are modeled as SysML requirements in requirement diagrams. Requirements are allocated to model elements via links. An example of how such a diagram is used is given in Fig. 12.

The diagram highlights the requirements **Emergency stop activates parkingbrake**, and its allocation to the **Brake System** package. Additionally, the requirement is also traced to the activity **Actuate Parking Brake**. The requirement has a corresponding table view, highlighted in the figure, where additional information can be more easily read. Importantly, the allocate association enables the requirements to be gathered in specific tables, while the traces allow for backward and forward traceability. Requirements are allocated continuously to the elements of the model, and tables can be generated dynamically. Types and stereotypes can be created and added per the user needs. By enforcing language extension mechanisms in addition to linkage with other model elements, the re-

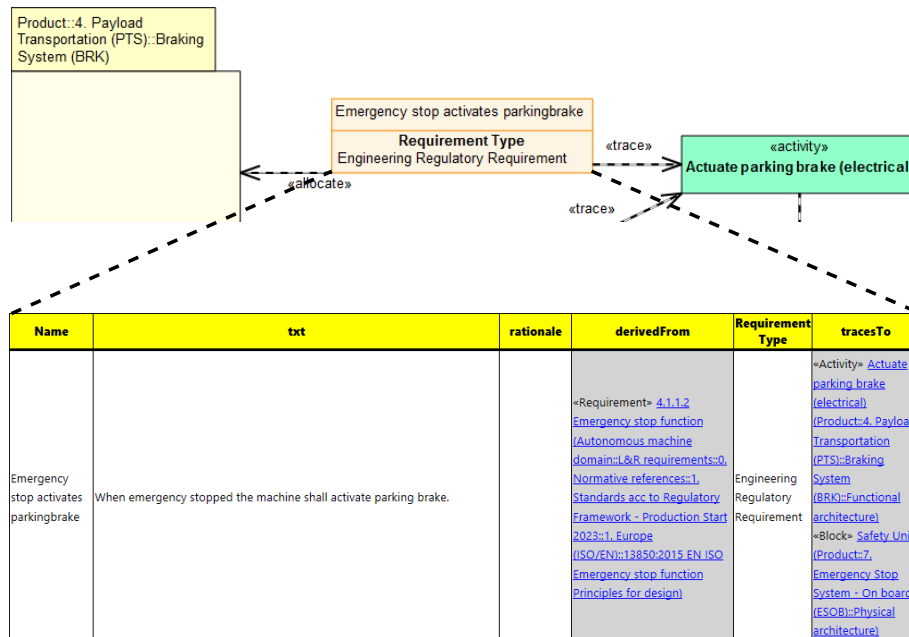


Figure 12: Requirements are modeled as requirement elements in SysML and allocated to different packages. Tables are then generated detailing the structured information whenever the user retrieves the information.

requirements become a powerful traceability tool, and the requirement diagrams can be leveraged for traceability views. For example, the diagram shows the requirement allocated to a particular model package while it is also traced to a particular activity. Requirements can be represented in tables or diagrams per the user needs.

4.6 Architecture baseline model management

We highlight the management of modeling elements and diagrams via the use of the model browser tool. Fig. 13 shows an excerpt of the corresponding Architecture baseline model management for the previously extracted diagrams and views. The annotations on the figure highlight where the elements originally are located, but we have moved them from their original packages for visualization purposes.

This hierarchy is useful for navigation and managing the abstraction in clear boundaries in the system, which enables useful partitioning of requirements for the different hierarchy levels. The model management enables separation of concerns, assisting configuration and change management which is a typical shortcoming of MBSE. Particularly the separation of concerns facilitates modular view creation, with clear boundaries and traceability of the model elements. Implementing a clear model hierarchy makes it easy for users to navigate ef-

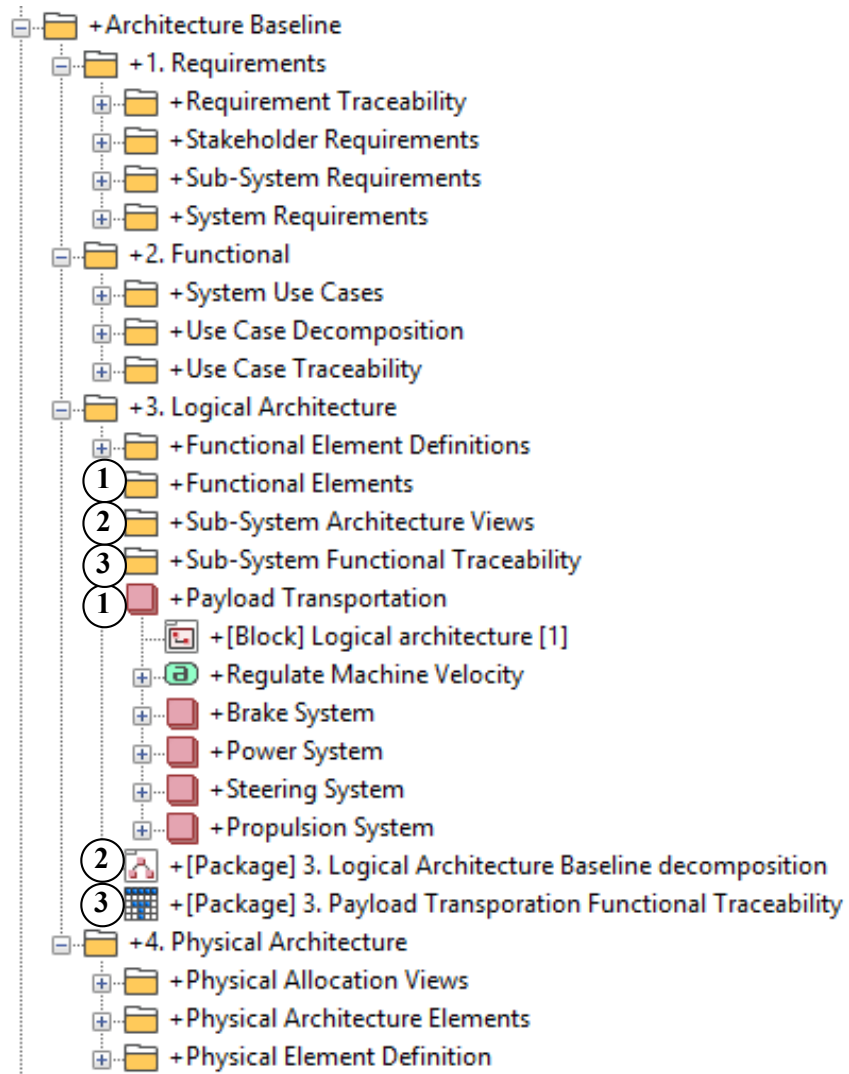


Figure 13: The model browser highlighting some previous examples in the architecture baseline model management. RFLP is separated into packages, containing the corresponding diagrams and elements.

ficiently, especially if the same structure is kept across different development efforts.

5 Discussion

The paper results from work aiming to introduce a robust adoption of MBSE in the CE domain. Previous efforts have faced several challenges when applying industrially proven methodologies and frameworks. Experience has shown that when the target engineers come from legacy domains, such as HW-intensive system development, applying top-down best practice approaches is met with resistance. Adding functional views and requirements traceability is one of the main motivations for applying MBSE in the CE domain, but also significantly different from legacy methods. It is intuitive to start from similar views to what already exists, for example, Visio views on the HW architecture. Our experience shows that it is sometimes *easier* to adapt the modeling views bottom up, even though it goes against traditional MBSE knowledge, as it lowers the learning curve for the initial phases in adoption (particularly for non-model-based practitioners). While the architecture development phase is well-emphasized in both SE and modeling standards, it is low-prioritized in project processes due to a shorter time-to-deliver and other project constraints. Other reasons are development silos in solution domains, e.g., HW, SW. The architecture ontology and the corresponding system baseline model described in the light-weight MBSE methodology emphasize on the logical model to bridge the architecture and design phases besides enabling traceability views. Further, the logical models provide deployment and system integration benefits. The time saved later in the design phase is the return on investment (in terms of time and effort) in the architecture phase as experienced by the project stakeholders.

Partly the need for a bottom-up approach is the already defined concepts and modular design-units in the HW domain. In this context, it is counter-productive to start from a top-down design approach without considering the already-in-place well formed design elements. Furthermore, a bottom-up approach is a means of creating an architecture suitable for a product-family rather than a specific product based on the existing design artifacts. The baseline architecture can then be used for designing new systems as part of a larger product-family (which is the norm for most CE products). By utilizing our proposed MBSE methodology we are aiming to allow non-modelers to develop systems similarly to what is done in the existing HW process, that is leveraging in place modular design-units for system definition and design. By centering the approach around modular re-use principles from HW legacy, it paves the way for eventual SW modularity by defining re-usable SW functionality and design units. Our approach as such not only promotes modeling via light-weight legacy inspired methods but also paves the way for more integrated holistic architectural views across stakeholder domains.

The approach presented in the paper has the aim of a fast *light-weight* model and architecture baseline deployment. Current efforts of adopting MBSE have fallen short in part due to the steep learning curve and poor fit match of methods compared to legacy (partly due to perceived complexity). Therefore the quick definition of an architecture baseline, which maps well to traditional artifacts, can act as a first milestone for further MBSE development efforts. Due to

the scope of an architecture baseline, the eventual architecture baseline model is somewhat limited in flexibility as it is based on already existing products. However, as the aim is to get a model's first definition, this lack of flexibility is not seen as a significant detriment, and the benefits of reaching an architecture model, as described in the earlier sections, are significant. In the same way, the simplification and scalability of these architecture models can be somewhat reduced, and again, we emphasize that the aim is to reduce complexity at this stage for easier dissemination of models and reproducing similar artifacts to the current non-model-based processes. With an architecture baseline model in place, it is expected that the model will evolve and have additional views created as development and design progress in a typical top-down workflow like best practice MBSE approaches. With the deployed baseline, existing MBSE development approaches are expected to address scalability, flexibility, and model simplifications with well-proven and tested methods and principles.

6 Conclusion

This paper presents our experience developing and applying a light-weight MBSE approach for the CE domain. We highlight the difficulty of adopting MBSE in industrial processes and the experience that light-weight approaches are more beneficial than modeling-heavy patterns and methodologies. As a first step, we defined an architecture (core) ontology for the development of a system (architecture) baseline model, separating the architecture concerns from design (implementation) details. Next, we defined the architecture baseline model by mapping the proposed architecture ontology onto standard modeling views using a sub-set of the SysML language within standard RFLP modeling workflows. We describe the deployment of the approach in an industrial project and demonstrate the approach in practice for an autonomous hauler machine. In contrast with the typical top-down approaches in the system modeling domain, the overall modeling workflows within the proposed light-weight approach are iterative and bottom-up, leveraging pre-existing system artifacts for the dissemination of an architecture baseline model. In future work, we foresee using MBSE modeling approaches to enable architecture descriptions that orchestrate co-simulation for early validation of system (architecture) designs using heterogeneous models from domains such as HW, SW, and electronics development. A necessary next step is the inclusion of more refined SW views to capture and integrate the corresponding concerns in the wider system context.

References

- [1] O. Aiello, D. S. D. R. Kandel, J.-C. Chaudemar, O. Poitou, and P. de Saqui-Sannes, "Populating mbse models from mdao analysis," in *2021 IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2021, pp. 1–8.

- [2] T. Amorim, A. Vogelsang, F. Pudlitz, P. Gersing, and J. Philipps, “Strategies and best practices for model-based systems engineering adoption in embedded systems industry,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 203–212.
- [3] M. Bouaicha, N. Machkour, I. E. Adraoui, and M. Zegrari, “Mbse grid: Operational analysis for the implementation of hydroelectric group health monitoring and management unit,” in *International Conference on Smart Applications and Data Analysis*. Springer, 2022, pp. 402–410.
- [4] L. Bretz, C. Tschirner, and R. Dumitrescu, “A concept for managing information in early stages of product engineering by integrating mbse and workflow management systems,” in *2016 IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2016, pp. 1–8.
- [5] H. P. L. Bruun, N. H. Mortensen, and U. Harlou, “Plm support for development of modular product families,” in *DS 75-4: Proceedings of the 19th International Conference on Engineering Design (ICED13), Design for Harmonies, Vol. 4: Product, Service and Systems Design, Seoul, Korea, 19-22.08. 2013*, 2013.
- [6] A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, “Grand challenges in model-driven engineering: an analysis of the state of the research,” *Software and Systems Modeling*, vol. 19, pp. 5–13, 2020.
- [7] A. Bucchiarone, F. Ciccozzi, L. Lambers, A. Pierantonio, M. Tichy, M. Tisi, A. Wortmann, and V. Zaytsev, “What is the future of modeling?” *IEEE Software*, vol. 38, no. 2, pp. 119–127, 2021.
- [8] K. X. Campo, T. Teper, C. E. Eaton, A. M. Shipman, G. Bhatia, and B. Mesmer, “Model-based systems engineering: Evaluating perceived value, metrics, and evidence through literature,” *Systems Engineering*, 2022.
- [9] J. Cederbladh and J. Suryadevara, “Towards a unified architecture methodology for product service systems,” in *Asia Oceanic Systems Engineering Conference*, October 2023. [Online]. Available: <http://www.es.mdu.se/publications/6806->
- [10] P. De Saqui-Sannes, R. A. Vingerhoeds, C. Garion, and X. Thirioux, “A taxonomy of mbse approaches by languages, tools and methods,” *IEEE Access*, 2022.
- [11] V. Debruynel, F. Simonot-Lion, and Y. Trinquet, “East-adl—an architecture,” in *Architecture Description Languages: IFIP TC-2 Workshop on Architecture Description Languages (WADL), World Computer Congress, Aug. 22-27, 2004, Toulouse, France*, vol. 176. Springer Science & Business Media, 2005, p. 181.

- [12] J. A. Estefan *et al.*, “Survey of model-based systems engineering (mbse) methodologies,” *IncoSE MBSE Focus Group*, vol. 25, no. 8, pp. 1–12, 2007.
- [13] P. H. Feiler and D. P. Gluch, *Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language*. Addison-Wesley, 2012.
- [14] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.
- [15] J. Gregory, L. Berthoud, T. Tryfonas, A. Rossignol, and L. Faure, “The long and winding road: Mbse adoption for functional avionics of spacecraft,” *Journal of Systems and Software*, vol. 160, p. 110453, 2020.
- [16] M. Hanna, L.-N. Schwede, J. Schwenke, F. Laukotka, and D. Krause, “Methodical modeling of product and process data of design methods using the example of modular lightweight design,” in *ASME International Mechanical Engineering Congress and Exposition*, vol. 85604. American Society of Mechanical Engineers, 2021, p. V006T06A035.
- [17] K. Henderson, T. McDermott, E. Van Aken, and A. Salado, “Towards developing metrics to evaluate digital engineering,” *Systems Engineering*, vol. 26, no. 1, pp. 3–31, 2023.
- [18] 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.
- [19] G. Hoepfner, I. Nachmann, T. Zerwas, J. K. Berroth, J. Kohl, C. Guist, B. Rumpe, and G. Jacobs, “Towards a holistic and functional model-based design method for mechatronic cyber-physical systems,” *Journal of Computing and Information Science in Engineering*, vol. 23, no. 5, p. 051001, 2023.
- [20] Y. Hooshmand, D. Adamenko, S. Kunnen, P. Köhler *et al.*, “An approach for holistic model-based engineering of industrial plants,” in *DS 87-3 Proceedings of the 21st International Conference on Engineering Design (ICED 17) Vol 3: Product, Services and Systems Design, Vancouver, Canada, 21-25.08. 2017*, 2017, pp. 101–110.
- [21] N. Hughes, “Functional modelling of electrical schematics,” 2004.
- [22] D. Krob, “Cesam: Cesames systems architecting method-a pocket guide,” 2017.
- [23] A. M. Law, “How to build valid and credible simulation models,” in *2019 Winter Simulation Conference (WSC)*. IEEE, 2019, pp. 1402–1414.
- [24] E. A. Lee and M. Sirjani, “What good are models?” in *International Conference on Formal Aspects of Component Software*. Springer, 2018, pp. 3–31.

- [25] J. Ludewig, “Models in software engineering—an introduction,” *Software and Systems Modeling*, vol. 2, pp. 5–14, 2003.
- [26] R. W. Macarthy and J. M. Bass, “An empirical taxonomy of devops in practice,” in *2020 46th euromicro conference on software engineering and advanced applications (seaa)*. IEEE, 2020, pp. 221–228.
- [27] A. Morkevicius, A. Aleksandraviciene, and Z. Strolia, “System verification and validation approach using the magicgrid framework,” *INSIGHT*, vol. 26, no. 1, pp. 51–59, 2023.
- [28] C. Nigischer, S. Bougain, R. Riegler, H. P. Stanek, and M. Grafinger, “Multi-domain simulation utilizing sysml: state of the art and future perspectives,” *Procedia CIRP*, vol. 100, pp. 319–324, 2021.
- [29] A. Parkinson, “Robust mechanical design using engineering models,” 1995.
- [30] P. Roques, “Mbse with the arcadia method and the capella tool,” in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.
- [31] V. Singh and K. E. Willcox, “Engineering design with digital thread,” *AIAA Journal*, vol. 56, no. 11, pp. 4515–4528, 2018.
- [32] P. Sjöberg, L.-O. Kihlström, and M. Hause, “An industrial example of using enterprise architecture to speed up systems development,” in *INCOSE International Symposium*, vol. 27, no. 1. Wiley Online Library, 2017, pp. 401–417.
- [33] J. Suryadevara and S. Tiwari, “Adopting mbse in construction equipment industry: An experience report,” in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 512–521.
- [34] D. D. Walden *et al.*, “Systems engineering handbook: A guide for system life cycle processes and activities,” (*No Title*), 2015.
- [35] T. Weikiens, A. Scheithauer, M. Di Maio, and N. Klusmann, “Evaluating and comparing mbse methodologies for practitioners,” in *2016 IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, 2016, pp. 1–8.
- [36] S. Wolny, A. Mazak, C. Carpella, V. Geist, and M. Wimmer, “Thirteen years of sysml: a systematic mapping study,” *Software and Systems Modeling*, vol. 19, pp. 111–169, 2020.
- [37] Y. Zhang, G. Hoepfner, J. Berroth, G. Pasch, and G. Jacobs, “Towards holistic system models including domain-specific simulation models based on sysml,” *Systems*, vol. 9, no. 4, p. 76, 2021.