Systems Engineering

WILEY

REGULAR ARTICLE OPEN ACCESS

Traceability Support for Engineering Reviews of Horizontal Model Evolution

Johan Cederbladh[1] | Eduard Kamburjan[2,3] | David A. Manrique-Negrin[4] | Rakshit Mittal[5] | Thomas Weber[6]

[1]Mälardalen University, Västerås, Sweden | [2]IT University of Copenhagen, Copenhagen, Denmark | [3]University of Oslo, Oslo, Norway | [4]Eindhoven University of Technology, Eindhoven, Netherlands | [5]University of Antwerp - Flanders Make, Antwerp, Belgium | [6]Karlsruhe Institute of Technology, Karlsruhe, Germany

Correspondence: Johan Cederbladh (johan.cederbladh@mdu.se)

Received: 25 July 2024 | **Revised:** 19 March 2025 | **Accepted:** 6 June 2025

Funding: This work was supported by funding from the DFG SFB 1608 (grant no. 501798263). This work was funded by the topic Engineering Secure Systems of the Helmholtz Association (HGF), the RCN project *PeTWIN* (grant no. 294600), the AIDOaRt project, an ECSEL Joint Undertaking (JU) under grant agreement No. 101007350, and the projec SM4RTENANCE, an EU H2020 project under grant agreement No. 101123423.

Keywords: engineering review | knowledge graphs | model-based systems engineering | SysML | traceability

ABSTRACT

At its very core, model-based systems engineering uses models to enable a multidisciplinary view on a system design in the early stages. These early stage models evolve *horizontally*: new diagrams for further perspectives and disciplines are added, using the same notation and the same abstraction level. Just as any other process in systems engineering, horizontal model evolution is subject to guidelines and standards, and the multidisciplinary view on a horizontal evolution, involving at least two disciplines, requires referring to multiple guidelines. Despite the significant effort invested in this process, there is no framework or tool support for engineering reviews of horizontal model evolution. In this paper, we aim to support engineering reviews by providing traceability for engineering activities that evolve a model horizontally. Our contribution is a process-agnostic framework that relies on capturing the intent of model changes in addition to the changes themselves. We group the model changes into transactional units called *deltas*, which are subsequently annotated with the engineer's intent to perform these specific changes. We give a methodology to integrate such *intent-annotated deltas* into engineering reviews and audits, an ontology to capture intent, and a meta-model for the deltas to achieve a language- and guideline-independent framework. We use an example from the earth moving machinery domain to exemplify the need for horizontal model evolution and provide a prototypical proof-of-concept implementation in the SysML Papyrus Plugin for Eclipse and a SysML case study using a machine brake system.

1 | Introduction

1.1 | Motivation

Models are becoming increasingly central to Systems Engineering (SE), notably in the earlier stages [1, 2], where models provide important capabilities, such as early design validation, a multidisciplinary view on design, or traceability throughout the system life-cycle [3, 4].

Early stage models do not necessarily support automation, but rather stakeholder communication and traceability by providing a single source of truth with different perspectives. They must provide precise meaning to enable communication and further

https://doi.org/10.1002/sys.70001

use while having enough flexibility to provide the margin-in-meaning necessary in early high-level designs to, for example, handle uncertainty management [5].

Early stage models are expressed in semi-formal notations and languages such as SysML [6], which support modeling in different engineering disciplines, by providing a metamodel that can be extended with domain concerns [6, 7]. Domain-specific modeling efforts during development of models is furthermore subject to internal guidelines for modeling, as well as subject to different standards and processes that should be applied in parallel by different stakeholders, such as electrical, software or safety engineering standards.

Such multidisciplinary models, with numerous diagrams for the corresponding views, however, pose a challenge to another central element of SE: audits and reviews. SE heavily depends on standard-driven processes and development for verification, validation [8], and accreditation, where a large emphasis is put on reviews of artifacts and eventual external audits of systems [1]. Internal engineering reviews are often used before external reviews and audits [9, 10] to reduce the risk of errors in the artifacts or processes that have been developed.

Engineering reviews are performed on various reports or documentation generated while developing or designing systems. As the understanding of the system expands, naturally, the model will *evolve*: either *vertically*, through refinement toward the final product, or *horizontally*, through integration of new disciplines or perspectives at a similar level of abstraction.

In this work, we focus on the latter–*horizontal model evolution*: the derivation of new views and diagrams from existing ones, for example, to derive a diagram for the software components from the mechanical diagram. In contrast to model refinement, horizontal evolution does not change the abstraction level, and providing evidence for its correctness has additional challenges due to the multidisciplinary nature of SE models.

1.2 | Challenge

Evolving a model requires training and expertise in both the process and the used tool, and is a laborious task in terms of modeling effort, but also in terms of documentation and actual reviewing effort.

Reviews and audits in horizontal model evolution require the expression of the intent of the engineer in a heterogeneous landscape and refer to several standards from multiple disciplines. To support horizontal model evolution, traceability mechanisms must account for the rationale of model modifications while retaining the advantages of semi-formal modeling notations. Yet, modifications performed as part of the evolution should be explicitly represented and reproducible.

From our experience working with companies in the context of Model-Based Systems Engineering, MBSE, there is no framework available to answer the need for systematic support of horizontal model evolution [11, 12]. Instead, unstructured notes such as screen-captures, natural language comments, or general text, are used. Neither are the actual modifications or model elements referred to (only the start and end states of the overall evolution), nor is it possible to automatically relate a modification and its corresponding guidelines/standards. The same gap can be found in SE literature, where the existing reviews are mostly performed in manual fashion for general SE processes [9, 10], but the need to go from reviewing (semi-structured) documents to reviewing data [13] has been discussed.

As reviewing and auditing is a critical step occurring throughout the development life-cycle [1], better structured support for reviews could reduce overall development effort spent while avoiding potential ambiguities in natural language review artifacts [14, 15].

1.3 | Approach

We present a language- and process-agnostic framework to provide traceability in horizontal model evolution to support engineering reviews. At its core, we superimpose a conceptual structure over the modifications performed during evolution, called *deltas*. These *deltas* are annotated with the engineer's *intent* in performing the modifications. The intent is an explicit reference to the standards and guidelines of the disciplines used by the engineers in creating/deriving the diagram. To be language-agnostic, the intent and delta models in our framework are presented as a metamodel and a corresponding ontology that can be instantiated for any platform. In more detail, our framework is based on two main ideas regarding the structure of the *deltas* in horizontal model evolution:

1.3.1 | Evolution Modularity

Each evolution consists of a number of atomic operations, for example, the addition and removal of SysML blocks in block diagrams. Each model evolution is composed of a hierarchical sequence of *intent-annotated deltas*. An intent-annotated delta is an annotated sequence of the aforementioned atomic operations (i.e., *atomic deltas*) performed during horizontal evolution of a model.

1.3.2 | Intent Annotation

Each of the *intent-annotated deltas* is annotated with its eponymous *intent*, which relates the contained sequence of atomic operations to a (rule or sentence in a) guideline or standard. Thus, the operations within a delta are not grouped by the internal subject of its modification (i.e., not grouped by *what* is modified), but by their relation to a computer-represented guideline (i.e., grouped by *why* the modification happens).

Breaking down the evolution into smaller steps and capturing their intents enables *traceability* to support engineering reviews, and to truly connect to the current industrial SE workflow. In our framework, we propose the following methodological stages for the integration of the proposed annotation of *intent* without committing to a specific tool or standard:

1.3.3 | Intent Capturing

To capture the intent and connect it with operations on the model, we propose to extend an integrated development environment (IDE) with the capability to record the deltas of an evolution, and annotate its intent when the evolution ends.

1.3.4 | Intent Lifting

The engineer's annotations are given semantic meaning to model the intent in an ontology. This allows us refer to different standards and guidelines from heterogeneous domains.

The approach uses a Knowledge Graph (KG) [16] (generated during intent recording and extraction) containing knowledge of both, the modifications committed by the engineer and their intent. This KG is consequently connected with external documents, for for example, modeling guidelines, and standards, and used as a support for engineering reviews, where the reviewer can query the KG, to investigate and comprehend the process/es that resulted in a specific model.

1.4 | Contributions and Structure

Our main contributions are (1) a structured framework for managing horizontal model evolution and traceability in engineering reviews, and (2) a demonstrator implemented in the Eclipse Papyrus IDE with SysML v1.6.

The rest of this work is structured as follows: Section 2 presents the necessary background and related work. We illustrate our approach in three steps. In Section 3, we summarize a case-study for one stage of the horizontal model evolution of an earth-moving vehicular braking system and the corresponding review/audit as it is done currently in practice based on our previous examples working in this domain [5, 17]. Then, we give an overview of our methodology and how it enhances the reviewing process, by precisely defining the concepts of the proposed framework in Sections 4, 5. Finally, a more detailed version of the example is described and our approach is prototyped as a proof-of-concept tool plug-in in Section 6. This example considers several steps of evolution, demonstrating a wider application. Section 7 discusses further details and Section 8 concludes the paper.

2 | Background & Related Work

In this section we introduce background and state-of-the-art in SE-related model evolution, traceability, reviews, and audits.

2.1 | Model-Based Systems Engineering and Guidelines

Systems Engineering is defined by the International Council on Systems Engineering (INCOSE) as "*a transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineering systems, using systems principles and concepts, and scientific, technological, and management methods.*" [1]. It is concerned with coordinating multiple engineering teams from different disciplines that work on a common system. Due to the increasing complexity and heterogeneity of systems [2], as well as current trends of technological advances, there are numerous incentives for SE practitioners to adopt model-based technologies in the form of MBSE, partly as an enabler for other technologies [2, 18], but also for the underlying benefits in development [2, 19] as a means to structure and coordinate trans-disciplinary work. This is in line with the current paradigm shift due to the overarching changes in the engineering landscape [1], with MBSE being listed as a potential enabler and emerging technology to tackle the challenges in terms of sustainability and digitalization in the 2035 INCOSE SE vision[1].

Although most engineering disciplines are most often discussed in terms of *problem solving*, SE revolves around *decision-making* [20]. Engineers are continuously tasked with making decisions while managing the inherent attributed risks [1]. As a result, an important aspect of SE is the development and application of best practices to adequately eliminate elements of risk during the overall processes for decision-making. Standards emerge as a natural solution, condensing existing expert knowledge and presenting useful heuristics and practices through systematic means. Less strict recommendations might also exist in terms of guidelines, for example, in the application of modeling languages [5]. Apart from supporting practitioners in applying disciplines or activities through best practices, standards, and guidelines enable verification of the engineering process through compliance checks. Governing standards can, as such, be incorporated in overall quality control through explicit review of adherence and alignment to provide evidence in the overall SE practice, and are built into most development processes [1].

Commonly, when performing systems modeling there is an inherent heterogeneity due to the many different underlying stakeholders. As a result, there is often a need for multiple perspectives or *views* when performing the eventual modeling activities of systems. Several paradigms have emerged related to this notion, such as multi-paradigm modeling and multi-view modeling [21, 22].

2.2 | Engineering Reviews and Traceability

One of the more commonly reported benefits and sources of motivation for MBSE adoption are the improved traceability and re-use capabilities [23]. Notably, capturing system details in a model from the very start paves the way for linking data and performing queries directly on artifacts as opposed to requiring external documents. In the SE context, traceability is a means of justifying various implementations by tracing them throughout the system life-cycle [1, 3]. Likewise, traceability should be incorporated to provide a means of attaching different rationales to decisions throughout the development progress. Models act as a powerful way of interlinking various information elements captured in formal notations and support engineers in navigating the data while supporting different analysis and information management [24]. In the SE context, traceability is a foundation for the management of product lines/families, which is a common way of managing variable products [25]. More recently, the notion of a *digital thread* that links data and artifacts throughout several projects, from the very start of design throughout operation, is seeing increased interest [26].

The use of external reviews is strictly enforced by many types of standards inside the SE and MBSE scope [1]. However, these kinds of activities are often a *final* step before entering a decision gate or development check-point. Before presenting a set of system artifacts for such review, there is often a need to internally review artifacts during the development, performed as a technical peer-review of engineers. These kinds of review can range from ad hoc to systematic, depending on the company and process observed [10]. In the end, internal reviews should align design artifacts with various modeling guidelines, in turn referring back to higher-level documents such as various standards. Most often such reviews are performed manually, and captured in various reports [14]. Additionally, it is often the case in larger industries that several governing standards are applied in parallel and need to be reviewed in conjunction [27]. As a result, these activities end up as a large part of the engineering process through several iterations of manual interaction and documentation.

Reviews are part of general engineering processes, and there is an emphasis on their systematic application [10]. There are also several approaches in the literature aiming to automate part of the reviewing process [28]. Specially, several domains are interested in automatically determining whether a design is compliant with governing standards [29, 30]. For example, Zhou et al. use deep learning and ontology based rules to automatically perform compliance checking in the construction domain [31]. Nawari et al. present a framework to formalize and apply reasoning for whether a building design is compliant or not with regulations [32]. Although these approaches in the literature can assist with internal review of compliance with various standards and governing documents, there is little work on applying reasoning for whether model evolution and modification intents are compliant with modeling guidelines.

Traceability is one of the cornerstone values of applying models in a SE context [2]. As information becomes formalized and captured in formats that are automatically readable by various tools in the engineering landscape, traceability becomes increasingly powerful [15, 33]. At the basic level, traceability is the action of navigating through a model in the same tool and language with various expressive links between modeling elements, such as a SysML model in a standard modeling tool [34]. Further maturity additionally enables the use of heterogenous model artifacts (in various notations and origins) to be used in the same tool, often leveraging an information back-bone in a standard language like SysML [35]. Finally, as traceability mechanisms are further created, there is an expectation that this will cut through various tools in the entire organization from where data can be read and written as per user needs (sometimes referred to as the digital thread [3]) [36]. Overall, the focus on traceability in the SE landscape is between various design artifacts and surrounding documentation, to support engineers in navigation and in maintaining traceable relationships between them. Although important, there has been less work on the individual artifacts themselves to sustain traceable rationales for model changes toward governing modeling guidelines/principles.

2.3 | Reviewing in Software Engineering

Code reviews are an established part of software engineering, also performed in the form of peer-reviews focusing on the reviewing changes between artifacts (commonly referred to as *Modern Code Reviews* [37]). Change management tools are a useful support for the Code review process [38]. However, software code reviews are rarely related to standards and guidelines, with the exception of code style guidelines and adhering to architectural constraints [39], but are concerned with the correctness of the resulting code [40, 41]. Indeed, their lightweightness is seen as a major advantage [40], however, it inhibits their application in the more standards- and process-driven field of SE.

Nonetheless, the rich body of studies in software engineering shows that reviews must be supported by methodologies and address the specific information needs of the review process [42, 43]. Among others, these are, in the terminology of Pascarella et al. [42], the ability to provide the *rationale* of a change, that is, its intent, and to be able to split it into self-contained units according to these rationales. Furthermore, Ram et al. [38] identified specific attributes that facilitate the review process. These include a precise change description that elucidates the rationale behind the modification, and code churn, which correlates with both the scale and the extent of the alterations. Consequently, it is crucial to implement changes in a manner that ensures they are concise, coherent, and well-defined in scope [38]. Other studies found similar requirements–we refer to the survey of Davila and Nunes [44] for a detailed overview. Our framework is designed according to these ideas (rationale/intent and self-contained units), but in contrast to software, version control and textual approach are less wide-spread in SE, so we cannot rely on an existing commit mechanism for simple difference visualization, or other textual tools to support reviews.

2.4 | Tool Chains and Ontologies

A tool-chain is defined as a collection of programming tools utilized to facilitate the development of a system [18]. Specifically, MBSE tool-chains aim to ease the creation of an SE workflow [18]. Ma et al. [18] highlight that MBSE tool-chains are required to possess integrating capabilities, interoperability, and traceability. However, it is noted that conventional MBSE tool-chains often lack functionalities such as change management or trace model evolution. Currently, this task, related to a peer-review, can only be done in an ad-hoc manner, by using external tools which is error-prone.

An approach to achieving knowledge management are ontologies. An ontology serves as a data model utilized to represent the semantics of domain concepts through entities (concepts) and their relationships (properties) [45]. Ontologies require a shared and consensual terminology, facilitating standardization of concepts. In addition, they offer rich semantic and syntactic expressiveness. Finally, they consist of semi-structured natural language. When data is incorporated into an ontology, it transforms into a KG [46]. The ontological characteristics inherent in the KG enable it to provide consistent domain knowledge that can be extracted through queries.

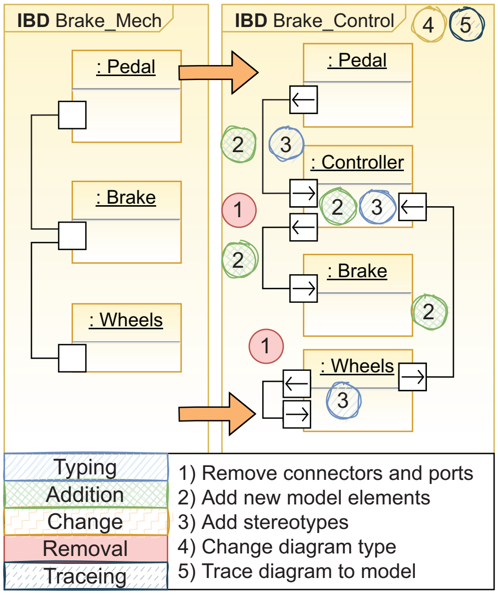

FIGURE 1 | Conceptual example of horizontal model evolution.

Concerning intent, several ontologies for modeling of intentions have been proposed, for example for business process modeling [47] and more foundational perspectives [48].

2.5 | Conclusion

As we can see, there is currently no unified methodological framework for assisting SE peer-reviews: while critical and time-consuming, they are performed ad hoc without specialized tools. At the same time, ideas from software engineering cannot be directly carried over due to the differences in the development processes and semantics of artifacts.

Our proposed framework aims to address this gap by enhancing the traceability of model evolution and establishing links between these evolutions and the standards or guidelines defined by the organization itself. To accomplish this linking task, effective management of the knowledge contained within standards or guidelines is necessary. Our approach aims to represent the knowledge derived from standards or guidelines within an ontology. Additionally, our objective is to construct a KG based on model evolution, adhering to a specific ontology of deltas. Consequently, the KG can be interrogated by a third party (i.e., a reviewer) to assess the compliance of model changes with established standards.

3 | Running Example

3.1 | Horizontal Evolution Case Study

We demonstrate our framework through a simplified example of a brake unit from an earth-moving machine as illustrated in Figure 1. The figure uses an abstract block diagram notation to describe a possible evolution of block diagrams concerning the brake-unit. The example has been inspired from previous work in MBSE adoption for the construction equipment (CE) domain [12].

3.1.1 | Mechanical Engineer's Diagram

The first diagrammatic view created is in the mechanical engineering domain, that is, the main components and their connectors. The aim of the mechanical engineer is to communicate the context of the brake sub-system, that is, which sub-systems interact within it at the interface level (here simplified), from a mechanical perspective. These interfaces describe physical connections without direction (i.e., they are acausal) and are possibly typed with some stereotype for mechanical interfacing, such as size and type of mechanical connector. In the example, a mechanical architectural diagram is the first diagram made due to the mechanical concerns heavily impacting other domains, thereby directly impacting the solution space in the design.

3.1.2 | Control Engineer's Diagram

Following the mechanical diagram definition, the following task is to create the control engineer's diagrammatic view in the model. The control engineer needs to understand the influencing signals in the control loop to help design an appropriate electronically-controlled behavior of the brake-unit. The control engineer clones the mechanical engineer's diagram, and modifies it w.r.t to their own concern, similar to clone-and-own variability [49]. This is especially true when the involved diagrams have similar structures and the components refer to the same higher-level element, in this case the brake-unit. The control engineer deals with causal and directional digital control signals *from* sensors and *to* actuators in contrast to the mechanical engineer. Hence, they may choose to either change the type of the connectors or delete and create new connectors based on the methodological guidelines of the team/organization.

3.1.3 | Traditional Engineering Review

After the horizontal evolution, for each view, the new diagram must be reviewed and audited. Indeed, a large part of SE is the reviewing and auditing of products *and the processes that create them* [1].

In our experience, the traditional method of performing peer-reviews is through the inspection of development artifacts at regular snapshots in time [13, 15]. Often, the artifacts might be semi-formal or semi-complete, and the eventual peer-review is intrinsically a manual activity consisting of inspecting and comparing artifacts with guidelines.

Following this, various interactions will have taken place between the creator and reviewer of the mechanical and control artifacts, generating documentation in the form of peer-review reports. The artifacts (but also the process of development) can, in this way, be assessed for correctness and compliance with standards and guidelines. The peer-review reports can then be assessed by higher-level stakeholders to assess the maturity and overall progress. Such documentation usually includes written summaries supported by various screen-grabs or figures describing the status and potential shortcomings of artifacts, to be used for assessment and planning of proceeding efforts, such as the figures presented here.

As we have described previously, the above described traditional engineering peer-review might be complicated by several shortcomings.

- The compositional and hierarchical structure of the transformation is not preserved. The information about a change is described in terms of differences between the original and final artifact, not the intermediate steps, even though the intent of the steps is related to different process steps.
- The intent is not systematically captured. During the review, there is no support to query this information from the perspective of the used guideline. Comments or ad hoc notes might be present in the diagram.
- Review documents and the models are not meaningfully linked. A common example is that reports are written in a separate tool and stored as a textual artifact without any means of accessing either one from the other.

3.2 | Proposed Framework

In this context, a valuable support for the engineers is to capture the *intent* behind model modifications and provide rationales about the corresponding evolutions [50]. Considering the simplified case described earlier in the section, an engineering review process should include the motivation and rationale attached to why anything on the original diagram is modified to reach the new state. By gathering this set of modifications, the engineering review process can be structured in a manner to reflect the intent of the modeler (see, e.g., the intent catalog by Amrani et al. [51]). In this way, any ambiguity as to why something has been modified can be checked by the intent in addition to the actual modification. This will make sure that any modifications that are made are not only correct, but for the correct reason.

The composition of addition, removal, and modification operations into evolution steps is according to intent–this is different to other approaches that decompose the overall changes, for example, in version control [52], according to the location of the change in the model.

Our approach is based on (1) capturing the steps of the model evolution and grouping them according to intent (delta extraction), and (2) connecting the intent to the semantic data layer (intent annotation), where it can be used to support an engineering review. The framework is illustrated graphically in Figure 2. The model evolution is a sequence of diagrams $D_1, \dots, D_n$, where D_1 adheres to some perspective (such as mechanical engineering) and D_n to another perspective (such as electrical engineering). Each change in the diagram, that is, each step from D_i to D_{i+1}, is modeled as a delta. Deltas are then grouped into transactional deltas that are annotated with some intent. The entire information of deltas and intent is then added to the KG, which the reviewer can access. The KG provides additional information while maintaining references to all deltas, intents, and guidelines.

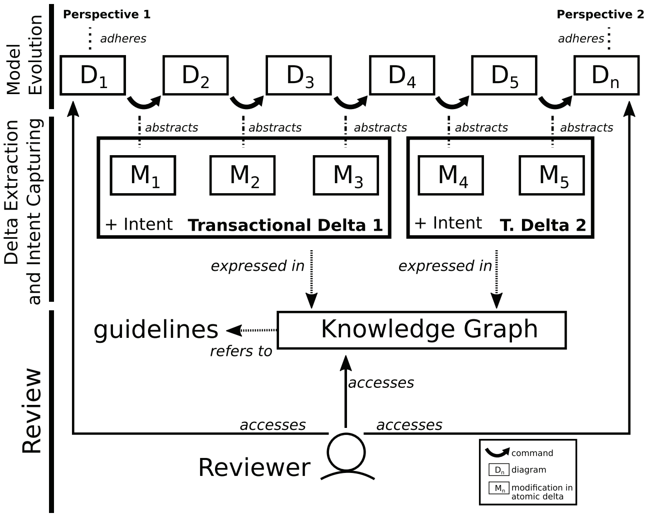

FIGURE 2 | Overview over the framework. The atomic deltas around the modifications are omitted.

3.2.1 | Delta Extraction

A delta[2] (Δ) is either a model transformation that transforms a diagram D into another diagram D' (a so-called *atomic delta*), or a sequence of such transformations (a so-called *transactional delta*). A sequence of diagrams $D_1, \dots, D_n$ is a *model evolution* if the perspective of D_n is not the same as that of D_1.

The intermediate diagrams $D_2, \dots, D_{n-1}$ may not conform to any perspective: the deltas associated with them contain information about the model transformation, grouped by their *intended* effect. Let us consider the evolution from the mechanical diagram to the control diagram in the running example. As a first step, the engineer removes the mechanical connectors between blocks. The resulting intermediate diagram is neither correct from the mechanical perspective, nor from the control perspective, yet still it is an important grouping of the modifications, since all three actions were performed with the same intent: *to remove components unrelated to the control loop*. Thus, these deltas form a unit, unified by a common intent. The core of the first step of our approach is to identify and extract such intentional units in the horizontal model evolution.

Indeed, deltas may be hierarchical to express hierarchical intent. In the running example, delta Δ_1 with the intent to remove mechanical connectors and ports, is followed by delta Δ_2 that adds the control ports and connectors - Δ_2 has a different intent, namely the addition of relevant control components. Both deltas compose a higher-level delta, namely Δ_3 with the intent to adhere to the guideline that *the control diagram must contain only relevant control-related components*.

To provide tool support, the engineering diagramming tool must be able to 'extract delta units', and enable the engineer to relate the deltas explicitly to their intents. We foresee two different techniques to do so: *delta recording* (so-called operation-based) and *delta derivation* (so-called state-based). In a typically common workflow: the engineer *starts* recording, performs the delta on the diagram, and then *ends* the recording. At the end, the engineer is prompted to provide an explicit intent annotation for the corresponding delta.

Delta recording groups all the performed commands and saves them together with the intent. On the other hand, *Delta derivation* is used when the tool does not have access to the specific delta-related commands' stack. In this scenario, derivation of the delta, by comparing the before and after of the diagram, becomes necessary.

Thus, a delta is not the same as a set of commands, but a tool-agnostic abstraction. We return to its details in Section 4.

3.2.2 | Intent Capturing

To ultimately utilize the structure provided by deltas for engineering reviews, they are annotated with intents.

The intent of a delta, in our framework, is a textual annotation with a set of explicit references to guidelines and standards. Such documents are often not uniform, even within the same discipline, and are not usually structured as well. Nonetheless, we provide a generic way to refer to documents via an ontology that can refer to both unstructured text, by referring to for example, specific page and paragraphs, and structured requirement documents, such as DOORS[3].

In the running example, the intent of Δ_1 *to remove the mechanical ports and connectors* might be related to more than one guideline, perhaps an external standard (e.g, ISO 3450:2011 stating *which* elements should be present) as well as the organization's internal modeling guidelines (e.g., "only relevant control components should be in the diagram"). In particular, a delta must be able to relate several guidelines that may contradict each other when applied to one single model view, but are necessary to relate *source and target* model views to each other. For example, Δ_1 is performed to satisfy the guidelines in the control engineering domain, while it contradicts the mechanical engineering guidelines. Hence, in our framework, the intent of a delta is situated *on a multidisciplinary level*–it is the *intent to replace* some structure, from a *source* to a *target* guideline. Although the target guideline is more relevant, the engineer must potentially refer to the source guideline as well, to make explicit which prior intent they are contradicting.

The presented framework makes the domain knowledge of the engineer explicit. It does so by enabling the engineers to capture their reasoning and intent behind performing certain modifications to the system. Intent itself can be captured in a number of ways, either (1) manually, where the engineer is prompted to provide references to standards at pre-defined intervals of the SE workflow, (2) the engineer may be assisted in the choice of such references by logical reasoning, (3) the engineer may have the option to specify a natural language annotation that is automatically matched to phrases in a guideline or (4) with enough data, the intent management framework may possibly automatically be able to assign intent based on SE patterns from previously captured intent.

3.2.3 | Intent Lifting

Indeed, the model of deltas annotated with intent will be specific to the modeling language and guidelines practiced by the organization. However, in our approach we incorporate ontologies to unify cross-domain concerns relating many standards to each other, while making the framework's core ontology extensible for different languages, tools, and workflows.

Instead of committing to any one design choice, modeling language, or standard, our framework provides a basic ontology that is easily extended or replaced by the ontology of choice in a concrete scenario. To implement intent lifting, the captured intent data is added to a KG based on the chosen/designed domain-specific ontology for deltas. Instantiating our framework merely requires matching [53] the concepts of our ontological layer, the modeling languages, and domain-specific ontologies. The ontologies and explicit representations of the semantics of intent are described in greater detail in Section 5.

3.2.4 | Traceability and Review Support

The reviewer can now execute queries on the KG to check the adherence of the horizontal model evolution to different engineering standards and guidelines. They can adopt different workflows based on their focus, such as verifying consistency across artifacts, or ensuring compliance with guidelines. They may take a document-driven approach, systematically reviewing standards and linking them to relevant evidence by asking "Which deltas have intent that satisfy this guideline?", or an issue-driven approach, by asking "Which guidelines have not been explicitly satisfied by any SE delta?" or "Which guidelines have been contracted, and by which deltas, resulting in which models?" to identify potential gaps or inconsistencies and investigate their root causes. They could also ask, "Do changes propagate correctly across related artifacts?".

4 | Intent-Annotated Deltas

Moving beyond the overview provided in the previous section, we turn to the technical intricacies of intent-annotated deltas.

Our delta metamodel, that is, the definition of the elements and their relations, is illustrated in Figure 3. A delta, per se, is a representation of a modification of a diagram in a model. Thus, each delta explicitly specifies at least one *pre* or *post* reference. *Pre* is a reference to the originating diagram element, whereas *post* is a reference to the target diagram element, which were associated with that particular delta. We distinguish between two kinds of deltas: An atomic delta, which is a concrete modification, expressed as *AtomicDelta*, and sequences of deltas that have a common intent, expressed as *TransactionalDelta*.

4.1 | Atomic Deltas

An *Atomic Delta* is a single modification from one diagram to another. It has a *scope*, which describes the kind of modification

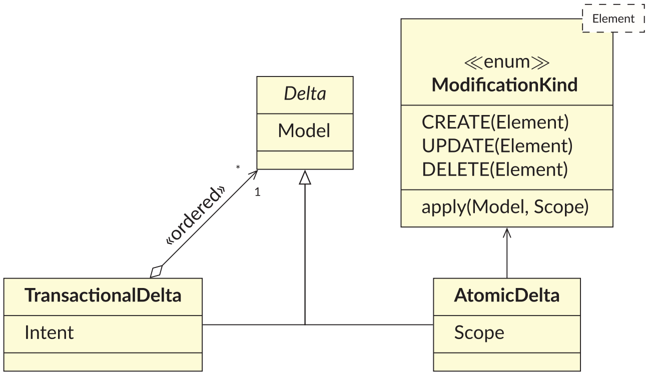

FIGURE 3 | The core Delta metamodel in our framework.

associated with that delta. There are three possible kinds of modifications: CREATE, UPDATE, and DELETE. Updates refer to update in specific properties of the diagram element, for example, the stereotype or name of a connector or component. In case of a deletion, there will be no *post* reference associated with the *Atomic Delta*. Similarly, in case of an addition of a component (or creation of the initial diagram), the *pre* reference shall be empty.

Note that a DELETE modification only deletes a model element from a view of the model and not from the model itself that is, it only expresses the difference (or Δ) between the current view and the previous view of the system. To maintain historical traceability, our proposed framework includes retaining all versioning data in a versioning-capable model management framework.

4.1.1 | Transactional Deltas

A *Transactional Delta* is composed of a sequence of *Atomic Deltas*, which share a common intent. Applying a *Transactional Delta* is to execute all its contained deltas in their sequence. A *Transactional Delta* must have an intent, that is, a description of why this delta was created. The intent annotation is provided by the developer when committing the deltas to reason about the purpose of the modifications performed.

It is transactional (a term we borrow from databases [54]) in the sense that all modifications within must be performed to realize the expressed intent: The states of the model/diagram before and after the application of the *Transactional Delta* are considered intermediate (or final, starting) states in a model evolution, such that they can be investigated on their own, but the state between the application of two atomic deltas is too rudimentary for that. *Transactional Deltas* are composable, that is, a *Transactional Delta* can contain other transactional deltas. Thus, it is possible to express both hierarchical modifications and hierarchical intent. This hierarchy allows modeling the different abstraction levels suitable to describe the system.

4.1.2 | Discussion

Our approach is not specific to any modeling language, but applicable to all kinds of models or diagrams for horizontal

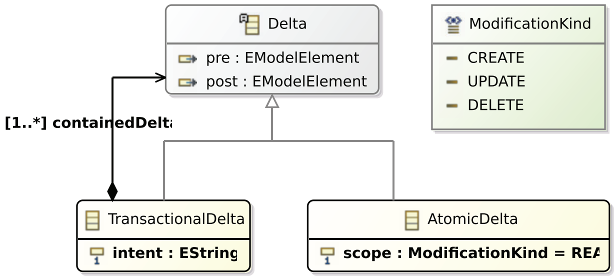

FIGURE 4 | Exemplary implementation of the Delta metamodel in EMF, used in the proof-of-concept implementation.

evolution scenarios, as long as they can express that a model has several diagrams, which in turn have elements. To use the proposed delta metamodel to track changes in the horizontal evolution of models in a specific modeling language, the range for its metamodel in turn has to be included in our delta metamodel, the range of *pre*, *post*, and *intent* references should be updated to reflect model elements in that modeling language or the utilized modeling framework.

The presented delta metamodel is deliberately minimalistic, focusing on capturing and structuring system engineering intent rather than providing a full-fledged model evolution framework. Existing solutions for model changes/evolution, such as Edelta [55] enable parallel and safe model evolution, and Khelladi et al. [56] propose mechanisms to detect complex changes. In contrast, the lightweight representation used here ensures that our approach can be integrated with others, more comprehensive delta-oriented frameworks if needed. Thus, the presented metamodel remains adaptable and complementary rather than competing with existing solutions.

In this way, the metamodel can be specialized for different scenarios to handle additional notions of deltas or modification. An example could be the addition of another *ModificationKind*, for example, *replace*. Replacement is not necessarily equivalent to deletion followed by creation–a commonly accepted notion in delta-oriented paradigms [57]; it can be used to express the replacement of an element with another element that retains some qualities, such as names. If a coarse-grained granularity is sufficient, even the whole horizontal evolution scenario could be defined as one possible *Modification*. The metamodel may be extended by hand or with delta model generation approaches [58]. Figure 4 describes the implemented metamodel in EMF, for the running example. Note that EModelElement and Estring are EMF stereotypes. The granularity that the developer may choose is driven by the core purpose of the *Delta*, that is, to capture changes performed and their order. Such an extensible framework is amenable to generalizability or reusability, for example, to be able to define modification scenarios which can be applied to different models or scopes, that is, the possible identification and re-use of modifications across models and diagrams.

We do not suggest modification of the delta metamodel in production, but rather before our framework is instantiated or incorporated for a particular domain. We believe the question of co-evolving metamodels and models is out-of-context in the current work, and is well answered in literature [55, 56].

4.2 | Extracting Deltas

We leave the exact nature of intents to be discussed in Section 5, and discuss the two possibilities to extract deltas: recording and deriving.

Before we can annotate deltas with an intent, we have to extract them first. We assume that the IDE/tool used by the developer records every operation internally, as the developer performs the modifications (commonly implemented in a command stack, for example, using the Memento pattern [59]), for example, to implement undo-redo functionality. Although many tools have undo and redo functionality, they may not make the command stack available to external integrations.

If this stack is accessible, then *delta recording* can be used. Otherwise, we assume that the IDE/tool can be used to export the whole diagram, from which we then can *derive deltas*.

4.2.1 | Recording Deltas

Recording deltas is the preferred way to extract them, because it is more fine-grained and closer to the overall workflow [60].

If the aforementioned command stack is accessible, then every (diagram-editing) command can be expressed as an *Atomic Delta*. To implement *Transactional Deltas*, the IDE must be extended with the functionality to start and stop recordings. Once a recording is started, each (diagram-editing) command is packaged as an *Atomic Delta*, within a single *Transactional Delta* until the `stop recording` command is received from the engineer. Then the engineer is prompted to provide the intent for the transaction, where they may package the *Atomic Deltas* into lower-level *Transactional Deltas* contained within the single top-level *Transactional Delta* representing the entire transaction.

An example for the recording of modifications is the *EMF Change Recorder*[4], which records changes to a model in the *Eclipse* IDE[5].

Recording has the advantage of capturing the exact modifications a user has made. Although this may include superfluous modification, for example, adding an element and deleting it afterward, such modifications can be removed (either automatically or manually) and may even be due to standards demanding some error checking analysis, for example, checking whether the addition of an extra sensor shall not provide a significant gain in performance. This condition may be manually checked by adding a sensor, running some analysis, and deleting the sensor afterward.

4.2.2 | Deriving Deltas

An alternative to recording, which does not require the used IDE to provide its command stack, is the derivation of deltas. The derivation of deltas uses the two states of a model, the *original state* before the modification and the *modified state* after the modification. The difference between the two states is then expressed with modifications, which can be modeled as deltas: First all elements that are only in the first state but not the second are removed, then all elements that are in the second state but not in the first one are added. The execution of the transaction composed of these deltas on the original state results in the modified state.

An example for the derivation of deltas is Git[6], which computes the difference between the last committed state and the current state of a repository and creates a list of modifications from it. This list of modifications can also be annotated with an intent, that is, the commit message, but provides a fixed and coarse-grained level of granularity based on changed lines in a file. Our approach enables the annotation of intent to specific model modifications to improve the traceability support. Another example for the derivation of deltas is the work by Wittler et al. [61], based on the comparison algorithms implemented in the *EMF* framework[7].

The advantage of the derivation of deltas compared to the recording of deltas is its more universal applicability. The derivation requires the original and the modified state, which are provided by any tool that can save or export the model in a readable format. This functionality is available in all modeling tools, and thus we can improve traceability support. Moreover, the delta sequence created is minimal, because no superfluous deltas are created [61].

Derivation is less precise than recording, because the actual modifications performed may differ from the ones derived, for example, because of the aforementioned superfluous modifications [61]. Additionally, the deltas derived may be inaccurate, for example, the modification of an element that only has one attribute is indistinguishable from its deletion and the creation of a new element with the changed attribute. The derived deltas can afterward also be annotated with intent. We deem the recording of deltas as preferable, due to its closer connection to workflows.

5 | Semantics of Intent in Deltas

Intent must refer to documents, such as standards and guidelines. However, these documents are highly heterogeneous in their form and matter, even within a single domain, and can be either documents in natural language, semi-structured formats or, rarely, structured formats such as ontologies. Even in case of structured documents, the content lies in the natural language within.

In the following, we describe an ontology that formalizes intent to annotate the transactional deltas, by referring to documents, without formalizing them or their content. The purpose of this ontology is to be used by the metamodel presented in Figure 3. In our framework, the notion of an intent is modeled as an ontological concept, which is related to modeling guidelines and possibly overarching standards as an *adheres-to* or *diverges-from* relation. An adheres-to relation between a (transactional) delta and a modeling guideline or rule means that the delta adheres to the guideline. On the other hand, a diverges-from relation between a delta and a guideline or rule means that the delta contradicts that rule or guideline: An intent can be explicitly related to the rules in the rule-book of choice that the engineer follows while modifying a certain model. In our framework, intent can be captured:

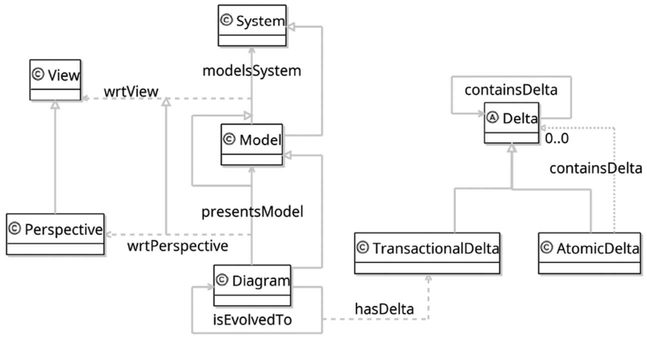

FIGURE 5 | Ontological representation of horizontal model evolution framework for systems engineering, with deltas, as an OML Vocabulary.

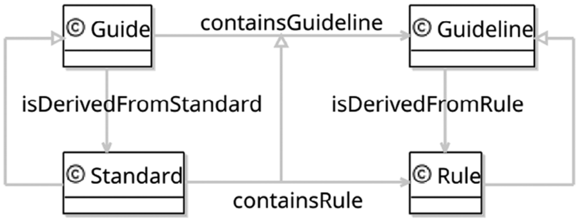

FIGURE 6 | Guides and standards as an OML Vocabulary. *Standard* specializes *Guide* and similarly *Rule* specializes *Guideline*.

- In the form of textual natural language descriptions of the transaction associated with the delta, including also its convergence or divergence w.r.t the organization's internal modeling guidelines or external standards.
- In the form of discrete and explicit ontological relations to internal SE workflow and model quality guidelines.
- In the form of discrete and explicit ontological relations to external SE standards for workflows and model quality guidelines.

5.1 | Intent Ontology

Figure 5 presents an ontological representation of the general linguistic elements associated with our framework. A *model* models a *system* w.r.t a *view*; a generic interpretation of the conceptual model of architecture description given by the IEEE/ISO/IEC 42010[8] standard. To this generic standard ontology we introduce the notions of *diagram* and *perspective* to represent the context of horizontal model evolution. A *diagram* presents a model, and is a graphical abstraction of that model, w.r.t a certain perspective. For instance, in the running example, the complete brake system has an architectural model that is represented by three different diagrams based on three different perspectives, mechanical, software, and electrical. A diagram can be evolved to another diagram (i.e., horizontal model evolution). Such an evolution is ontologically related to the ontological representation of a *Transactional Delta*.

Figure 6 presents a basic ontology of guidelines, guides, rules, and standards. Internal SE guidelines are usually low-level documents, whereas external SE standards are usually more abstract and general. If a guide is derived from a higher-level standard, the *isDerivedFromStandard* relation is used to capture that ontologically. Similarly, the precise guidelines contained in the guides are related to the various rules in the standards. We find it important to explicitly mention the difference between internal SE guidelines and external SE standards since, as explained earlier, there is high heterogeneity in the MBSE of a complex industrial system. In such a scenario, different internal guidelines may exist within the different specializations/domains, while adhering to the same external standards. This is a strong use-case for ontological models: the ability to relate concepts from different domains and silos in a singular framework. This becomes important especially in the context of horizontal model evolution which more often than not, entails the transformation of models between different views, each view with possibly its own set and hierarchies of guidelines and standards.

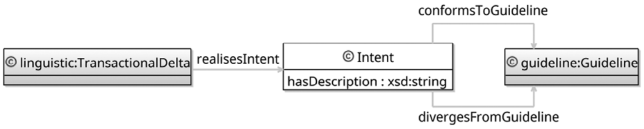

FIGURE 7 | Ontological representation of intent, as an OML Vocabulary that links the linguistic and guideline ontologies.

Figure 7 presents the intent ontology that links a transactional delta to a guideline through the concept of *Intent*. A *Transactional Delta* realizes a certain *Intent*. The *Intent* may conform to a certain guideline and/or diverge from one/another. As shown in the metamodel described in Section 4, the Intent can be described by a string as well. The string may be natural language, the location of another document, etc.

The idea is that a local (e.g., company-specific) ontological implementation of this framework can be connected to the other ontologies of the team, that describe the semantics of the artifacts that are evolved horizontally, in the form of properties or inference rules. These semantics will later be used in Section 5.3, which describes how the KG can be used in audits.

5.2 | Annotating Intent

Aided with the ontological representation, and the engineering team's choice of recording or deriving deltas, as described in Section 4, the intent can be recorded and added to the knowledge base in a number of ways. The proposed framework allows to accommodate horizontal model evolution carried out in many different organizations and engineering teams, and by various engineers with differing viewpoints and levels of experience.

Once a horizontal model evolution transaction is complete, the engineer (who performed the transaction) can be prompted to describe the intent using an *explicit ontological* annotation (i.e., using the terminology of the ontology) of the transaction in one of the following ways.

(1) Through an annotation relating the intent to an exact rule of the team's *internal SE guideline*; a higher-level team of the organization creates and maintains these guidelines with explicit

ontological relations to the standards they are based on, to unify and ease the review/auditing process, while controlling the engineers' interpretation of international standards. (2) Similarly, the engineer can provide an explicit annotation (an ontological relation) relating the transactional delta directly to a definition in a standard. This could be the case, for example, if no internal guideline, as described in the previous point, is available.

The above requires that the ontological implementations of standards and guidelines, as well as the engineer's familiarity with them, are high enough that annotating does not disrupt the overall workflow. However, as the vocabulary is rather small, and does not require formalizing standards and guidelines, but only to refer to substructures, we consider this as a realistic assumption.

If the structure of standards or guidelines is not yet structured in a way that it can be referred to from the intent ontology, or an engineer does not have the ability to relate the transactional deltas to the available guidelines, then the engineer can provide a textual description of the intent. This textual natural language annotation could then be analyzed and interpreted automatically manually by another engineer (who may be) an internal reviewer, or a higher level member coordinating with the auditors, before submission for audit.

5.3 | Supporting Engineering Review

As described in Section 2, ontologies and KGs provide certain advantages compared to traditional metamodels and models for querying, inferencing, and reasoning. There are three ways in which the knowledge captured in our framework can be used to make SE more efficient at multiple levels of reviews:

5.3.1 | Mechanized Conformance Constraints

Given an ontological representation of the guidelines and standards that also formalizes their content, then checking conformance requirements can be supported through checking the structural features of model elements as described by the standard; once the engineer relates the transaction to a certain guideline. Usually, such requirements on the structural features of the KG elements are expressed as a constraint pattern over (a subset of) KG elements. For RDF-based KGs, SHACL[9] (SHApes-Constraint Language), a W3C recommendation, is the most popular choice of such a pattern language. This implements a light-weight check to catch mistakes already during annotation, or supports reviews when the requirements can be described structurally in full.

5.3.2 | Querying Support

In the traditional setting of an audit or engineering review, an auditor can query the KG with a query that represents a pattern of KG element types and their properties, and their relations. For example, a pattern could represent something as simple as *"a component with no contained subcomponent"* which matches with all the leaf components (if the pattern is specified correctly), or *"all evolutions of this model diagram"*, or it could be as complex as *"all evolutions of this diagram corresponding to this viewpoint, made by a specific engineer, where the evolution diverged from a specific guideline…"*. For RDF-based KGs, SPARQL[10] (SPARQL Protocol and RDF Query Language), another W3C standard, is the most popular language of choice.

The auditor can systematically use the pattern matches to inspect the model repository and confirm if the standard workflows were truly followed by the organization / team. Such queries can be either generic, that is, be defined as part of the workflow (such as the above *"a component with no contained subcomponent"*), or specific and used in a particular situation (such as the above *"all evolutions of this model diagram"*).

5.3.3 | Document Linking

The KG, specifically the deltas, contain links to the elements in the diagrams that they are modifying. An engineer, thus, does not just query the KG for information that we store, but can also follow this link to open the diagram it refers to and see the full context of the change, for both source and target diagram.

We demonstrate all three kinds of support in the section below, where we apply our framework in full detail to the SysML model of the brake system described above.

6 | Demonstration

Let us return to the simplified example of a brake unit from an earth-moving machine described in Section 3. We illustrate the example in the Papyrus SysML 1.6 notation to describe the evolution of internal block diagrams concerning the brake-unit. A (non-exhaustive) description of the related standards that the engineers have to follow are also provided. The overall view is provided in Figure 8.

We have implemented the running example, within a demonstrative prototypical tool-chain based on our framework, using the eclipse modeling framework (EMF). The corresponding Eclipse project artifacts can be found in this Zenodo repository[11]. The accompanying video demonstrates how to use the tool-chain to inspect the artifacts and query the resulting KG.

We first implemented the brake unit sub-system diagrams using Papyrus with the SysMLv1.6 plugin. The diagrams are shown in Figure 8. Then we implemented the linguistic delta metamodel as an Ecore model using Eclipse Modeling Tools 2024-03 release. The corresponding diagram is shown in Figure 3. EMF automatically creates an EMF-based model editor allowing rapid prototyping of the delta instances.

We manually created/instantiated the *Transactional Deltas* and *Atomic Deltas* with explicit references to the corresponding SysML diagram elements. Each *Transactional Delta* contains a corresponding natural language intent annotation. This was done for each evolution: Creation of Pneumatic Diagram, Pneumatic Diagram to Control Diagram, and Control Diagram to Electric Diagram. Eclipse CAPRA[12] was used to visualize these delta traces.

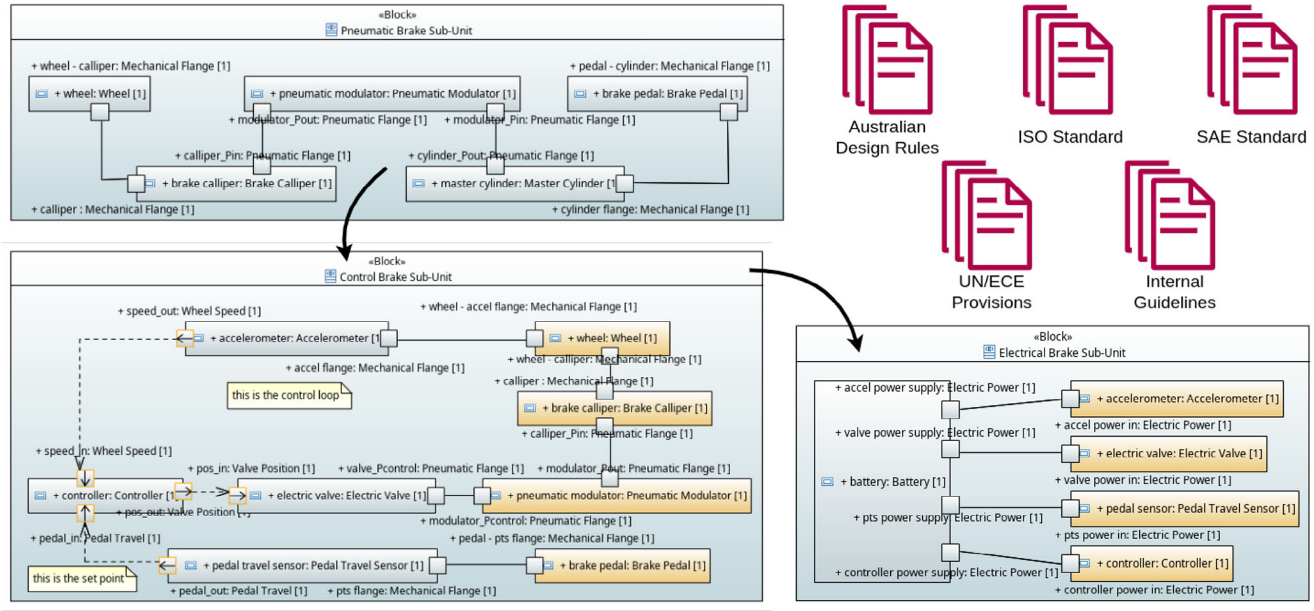
«Block»
Pneumatic Brake Sub-Unit
+ wheel - calliper: Mechanical Flange [1]
+ pedal - cylinder: Mechanical Flange [1]
+ wheel: Wheel [1]
+ pneumatic modulator: Pneumatic Modulator [1]
+ brake pedal: Brake Pedal [1]
+ modulator_Pout: Pneumatic Flange [1]
+ modulator_Pin: Pneumatic Flange [1]
+ calliper_Pin: Pneumatic Flange [1]
+ cylinder_Pout: Pneumatic Flange [1]
+ brake calliper: Brake Calliper [1]
+ master cylinder: Master Cylinder [1]
+ calliper : Mechanical Flange [1]
+ cylinder flange: Mechanical Flange [1]
Australian Design Rules
ISO Standard
SAE Standard
UN/ECE Provisions
Internal Guidelines
«Block»
Control Brake Sub-Unit
+ speed_out: Wheel Speed [1]
+ wheel - accel flange: Mechanical Flange [1]
+ accelerometer: Accelerometer [1]
+ wheel: Wheel [1]
+ wheel - calliper: Mechanical Flange [1]
+ accel flange: Mechanical Flange [1]
+ calliper : Mechanical Flange [1]
this is the control loop
+ brake calliper: Brake Calliper [1]
+ calliper_Pin: Pneumatic Flange [1]
+ speed_in: Wheel Speed [1]
+ pos_in: Valve Position [1]
+ valve_Pcontrol: Pneumatic Flange [1]
+ modulator_Pout: Pneumatic Flange [1]
+ controller: Controller [1]
+ electric valve: Electric Valve [1]
+ pneumatic modulator: Pneumatic Modulator [1]
+ pos_out: Valve Position [1]
+ modulator_Pcontrol: Pneumatic Flange [1]
+ pedal_in: Pedal Travel [1]
+ pedal - pts flange: Mechanical Flange [1]
+ pedal travel sensor: Pedal Travel Sensor [1]
+ brake pedal: Brake Pedal [1]
this is the set point
+ pedal_out: Pedal Travel [1]
+ pts flange: Mechanical Flange [1]
«Block»
Electrical Brake Sub-Unit
+ accel power supply: Electric Power [1]
+ accelerometer: Accelerometer [1]
+ accel power in: Electric Power [1]
+ valve power supply: Electric Power [1]
+ electric valve: Electric Valve [1]
+ valve power in: Electric Power [1]
+ battery: Battery [1]
+ pedal sensor: Pedal Travel Sensor [1]
+ pts power supply: Electric Power [1]
+ pts power in: Electric Power [1]
+ controller: Controller [1]
+ controller power supply: Electric Power [1]
+ controller power in: Electric Power [1]

FIGURE 8 | The earth-moving machinery brake-unit case-study. The model horizontally evolves, from a pneumatic perspective, to a control-engineering perspective, and then to the electrical power perspective. The orange components are components preserved during an evolution. The deltas in both evolutions are intended to follow or diverge from multiple standards and internal guidelines.

Next, we used the Ontological Modeling Language (OML), a language developed by the Jet Propulsion Laboratory at NASA to bring ontologies closer to SE, and its Eclipse-based tool Rosetta v2.6 to create the ontological metamodel. The corresponding diagrams are shown in Figures 5, 6, 7. We then manually instantiated the ontological concepts (called Descriptions in OML), equivalent to the linguistic delta model. Each of the ontological deltas is explicitly related to its equivalent linguistic delta using XPath[13] annotations. We also created the ontological instances of the relevant segments of various standards and mock guidelines. Each of these ontological guidelines is explicitly related to the rule by rule/paragraph/page number. Finally, we instantiated the intent ontology allowing us to relate the deltas to guidelines.

Following the OML workflow, "building" the OML project transforms the OML Descriptions ultimately to RDF-triples: the KG, while also checking for inconsistencies using ontological reasoners. The final step was to design the SPARQL queries to query the resulting KG.

6.1 | Mechanical Engineer's Diagram

The first diagrammatic view created is in the mechanical engineering domain, that is, the pneumato-mechanical components and their interfaces. The diagram is displayed in Figure 8, labeled `Pneumatic Brake Sub-Unit`. At the given abstraction, the aim of the mechanical engineer is to communicate the context of the brake sub-system, that is, which sub-systems interact within it at the interface level, from a mechanical perspective. These interfaces describe physical connections without direction (acausal) and are possibly typed with some stereotype for mechanical interfacing, such as size and type of mechanical connector. In the case-study these are acausal connectors of the `mechanical flange` and `pneumatic flange` types.

Engineers follow many standards, for example, that the connections have no direction is commonly part of guidelines for modeling of mechanical diagrams [12]. Similarly, it is a requirement to stop the vehicle in a reasonable distance in case the electronic control unit fails as specified in the Federal Motor Vehicle Safety Standards (FMVSS)[14]. The interpretation of this rule by the engineer is that a functional mechanical design independent from the control unit satisifies the rules in the standards. Note that this interpretation is implicit knowledge.

The FMVSS also mandates the presence of a modulator while the ISO 3450:2011 standard [15] mandates the presence of a master cylinder. Additionally, the United Nations Economic Commission for Europe (UN-ECE)[16] recommends that brake linings should not contain asbestos. The Australian Design Rules (ADR)[17] state that there should be a single control lever for activation of the brake-unit.

6.2 | Control Engineer's Diagram

The next task is to create the control engineer's diagrammatic view. The control engineer needs to understand the influencing signals in the control loop to help design an appropriate electronically-controlled behavior of the brake-unit. The control engineer clones the mechanical engineer's diagram, and modifies it w.r.t to their own concern, similar to clone-and-own variability [49]. This is especially true when the involved diagrams have similar structures and the components refer to the same higher-level element, in this case the brake-unit. In the `Control Brake Sub-Unit` internal block diagram in Figure 8, the orange-colored

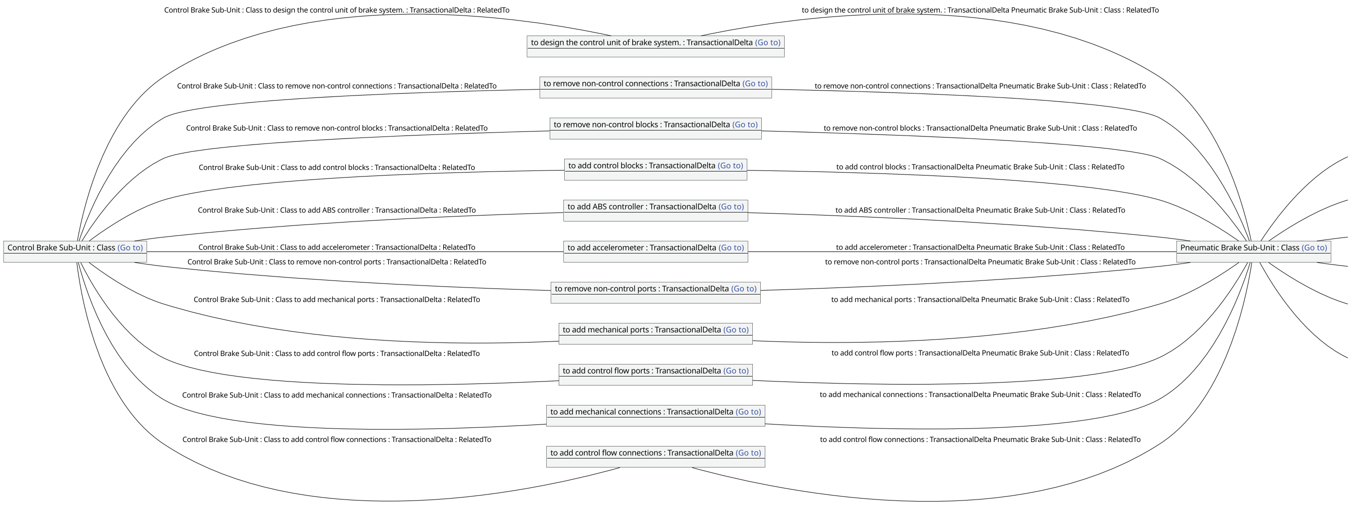
Control Brake Sub-Unit : Class to design the control unit of brake system. : TransactionalDelta : RelatedTo
to design the control unit of brake system. : TransactionalDelta (Go to)
to design the control unit of brake system. : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Control Brake Sub-Unit : Class to remove non-control connections : TransactionalDelta : RelatedTo
to remove non-control connections : TransactionalDelta (Go to)
to remove non-control connections : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Control Brake Sub-Unit : Class to remove non-control blocks : TransactionalDelta : RelatedTo
to remove non-control blocks : TransactionalDelta (Go to)
to remove non-control blocks : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Control Brake Sub-Unit : Class to add control blocks : TransactionalDelta : RelatedTo
to add control blocks : TransactionalDelta (Go to)
to add control blocks : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Control Brake Sub-Unit : Class to add ABS controller : TransactionalDelta : RelatedTo
to add ABS controller : TransactionalDelta (Go to)
to add ABS controller : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Control Brake Sub-Unit : Class (Go to)
Control Brake Sub-Unit : Class to add accelerometer : TransactionalDelta : RelatedTo
to add accelerometer : TransactionalDelta (Go to)
to add accelerometer : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Pneumatic Brake Sub-Unit : Class (Go to)
Control Brake Sub-Unit : Class to remove non-control ports : TransactionalDelta : RelatedTo
to remove non-control ports : TransactionalDelta (Go to)
to remove non-control ports : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Control Brake Sub-Unit : Class to add mechanical ports : TransactionalDelta : RelatedTo
to add mechanical ports : TransactionalDelta (Go to)
to add mechanical ports : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Control Brake Sub-Unit : Class to add control flow ports : TransactionalDelta : RelatedTo
to add control flow ports : TransactionalDelta (Go to)
to add control flow ports : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Control Brake Sub-Unit : Class to add mechanical connections : TransactionalDelta : RelatedTo
to add mechanical connections : TransactionalDelta (Go to)
to add mechanical connections : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo
Control Brake Sub-Unit : Class to add control flow connections : TransactionalDelta : RelatedTo
to add control flow connections : TransactionalDelta (Go to)
to add control flow connections : TransactionalDelta Pneumatic Brake Sub-Unit : Class : RelatedTo

FIGURE 9 | Traceability graph generated in the demonstration, showing some of the *Transactional Deltas* in the horizontal model evolution from the pneumatic engineer's diagram to the control engineer's diagram of the exemplary brake unit.

blocks represent the components preserved during the evolution from the pneumatic sub-unit diagram.

The control engineer deals with causal and directional digital control signals *from* sensors, and *to* actuators. Hence, they may chose to either change the type of the connectors or delete and create new connectors based on the methodological guidelines of the team/organization. Item flows (a SysML stereotype for directed connections) will be used.

The control engineer also deals with many standards governing the functionalities of the controller. For example, the FMVSS and ADR require the controller to be able to control Anti-lock braking systems (ABS) while the FMVSS also mandates the presence of functionality of electronic stability control (ESC) which the ADR calls 'traction control'. Internal guidelines w.r.t using the correct stereotypes for signals and blocks are also important from the control engineer's perspective.

6.3 | Electrical Engineer's Diagram

Evolving the control view to the electrical view requires several modifications. For example, only the electricity-powered components should be retained in the evolution, while the rest of the mechanical components should be removed. An appropriate battery component should be added. Electrical ports and connections are considered directionless (unless electrical power supply is modeled as a transfer of electric power) according to the engineering guidelines. An important engineering guidelines associated with the electrical viewpoint is that all electrical components should be connected independently to the battery. This contributes to fulfilling the UN-ECE's requirements on electronic fault tolerance of the brake-unit.

6.4 | Intended Review Workflow

All of the above when encoded with deltas in our demonstrative prototype, yield graphs describing traceability links as illustrated in Figure 9. The graph describes (a subset of) the transactional intents associated with the horizontal evolution from the pneumatic engineer's diagram to the control engineer's diagram. This has the potential to be an informative tool to support internal reviews of SE, by showing clearly which intentional transactions have taken place in the model evolution.

On the other hand, external auditors, who are typically more interested in formal guarantees, want to be able to deeply and (preferably) formally verify that systems adhere to standards. In such a scenario a tool that allows writing and performing queries on the stored KG of horizontal model evolution and intent can be useful. The SPARQL query for a simple question, for example, "Are there any rules in the ADR standard that were not satisfied during horizontal model evolution?", is as follows:

```
SELECT DISTINCT?rule

WHERE {

 ?rule a guideline:Rule .

 ?rule guideline:statedIn guideline:ADR.

 FILTER NOT EXISTS {

  ?intent intent:conformsToGuideline?rule .

 }

}
```

With the SPARQL queries, reviewers can verify whether all guidelines were fulfilled at some point in the evolution, and if not, which guidelines were not. Similarly, the reviewer can also check which delta satisfies or goes against which guidelines, etc. This opens the possibility of developing custom verified and domain-specific model verification tools to abstract the complexity of writing SPARQL queries from auditors.

7 | Discussion

The presented framework for using intent-annotated deltas focuses on solving the concrete problem of support for engineering reviews, based on our experiences with such practices. Our notions of intent and deltas are fitted for this purpose. In the following, we discuss possible limitations and their resolutions.

7.1 | On Intent

In our intent model, we do consider that a modification can have multiple intents, but assume that the structure of intents and modifications is tree-like and hierarchical. There may be situations where modifications, that is, deltas, may interleave. For example, the modifications may not be ordered by intent, but user habits (for example, remove all edges first, even if it is for different purposes). In our experience, the presented tree-model is sufficient for most situations occurring in practice, and leave a study of more complex topologies for later investigations.

Capturing and recording user intent and actions is a foundation for several technologies. Particularly, a systematic gathering and collection of user actions can enable subsequent recommendations or assistance of user actions, based on the collected data. In particular, given enough extracted deltas with annotated intent, one may investigate the possibility of developing a recommender system that, given an intent, recommends a modification or deltas. Such an approach utilizing a recommender system based on user actions has been developed for the earth-moving machinery context [17], and we see the merge of these approaches toward unified support for traceability of horizontal model evolution as future work.

7.2 | On Deltas

The extracted deltas can be useful in contexts other than reviews: SE is intrinsically related to variability and product line engineering, and deltas are a recognized variability management approach [57], mostly considered for software with only limited investigation of applying them to variability management of systems, for example, by Kowal et al. [62].

Once sufficiently many deltas are extracted, it may be possible to associate them with a feature model to describe which combinations of deltas are allowed. This would require a language extension to be able to automatically program deltas, for which Haber et al. [63] provide a framework.

7.3 | On Vertical Evolution

Model evolution naturally also encompasses refinement, that is, the development of model of lower abstraction, which we denote as *vertical model evolution*. Indeed, commonly used MBSE approaches expect that a system model will be developed top-down, usually starting from high-level requirements and customer needs/expectations, emphasizing vertical evolution.

The presented framework targets horizontal evolution. Nonetheless, the principles in this paper translate to some extent to vertical evolution as well, with some key differences that should be addressed; (1) Vertical integration might consider movement to different notations, for example to simulation specific notations from a descriptive system model. Our work relies on the fact that both models are expressed in the same notation. Thus, a different mechanism than deltas must be used (2) The technical audience in horizontal evolution most certainly differs between the two models, but this might not be the case for horizontal integration. Thus, the intent only targets one set of guidelines and rulebooks.

7.4 | On Requirements

The necessity to comply with standards or guidelines is not the only application of annotating intent. Additionally, other factors that influence the system directly or indirectly might be part of the intent of a delta, for example, requirements for the system. Although it is also quite useful to annotate them to deltas, we focus in this paper on the support for engineering reviews, where the part of the intent that relates to a requirement is not relevant, that is, developing a system conforming to a standard is independent of the developed system itself. Nonetheless, the proposed framework can also be used to support requirement traceability.

8 | Conclusion

Engineering reviews are a critical yet time-consuming activity in MBSE, particularly in horizontal model evolution, where ensuring cross-disciplinary consistency and compliance is essential. Despite its significance, horizontal model evolution lacks *structured traceability and review* support, leading to a manual, error-prone review process. In this paper, we introduce a *process-agnostic framework* to improve traceability in horizontal model evolution by:

1. *Structuring model changes into transactions as Deltas*, grouping atomic modification based on intent.
2. *Annotating deltas with explicit references to engineering guidelines and standards*, ensuring compliance and auditability.
3. *Leveraging ontologies and KGs* to support automating querying, reasoning, and review workflows.

To demonstrate the framework's applicability, we applied it to a SysML- based case study of an earth-moving machine and implemented a prototype in Papyrus for Eclipse. Our results indicate that intent-annotate deltas provide a structured, traceable representation of model evolution, reducing ambiguity and improving review efficiency.

For future work, we plan to investigate the connection of system engineering reviews, ergonomics, and intent-annotated deltas for further purposes, in particular, their use as a basis for product line engineering and recommender systems.

Acknowledgments

This work was supported by funding from the DFG SFB 1608 (grant no. 501798263). This work was funded by the topic Engineering Secure Systems of the Helmholtz Association (HGF), the RCN project *PeTWIN* (grant no. 294600), the AIDOaRt project, an ECSEL Joint Undertaking (JU) under grant agreement No. 101007350, and the project SM4RTENANCE, an EU H2020 project under grant agreement No. 101123423. The paper reflects the authors' views and the Commission is not responsible for any use that may be made of the information it contains. We thank the organizers of the 4th WAKERS workshop. We thank Volvo Construction Equipment for their valuable input.

Data Availability Statement

The data that support the findings of this study are openly available in Zenodo at https://zenodo.org/records/12784499.

Endnotes

[1] The INCOSE 2035 vision: https://www.incose.org/publications/se-vision-2035

[2] The term delta is inspired by Delta-Oriented Programming [57]

[3] https://www.ibm.com/products/requirements-management

[4] https://download.eclipse.org/modeling/emf/emf/javadoc/2.4.2/org/eclipse/emf/ecore/change/util/ChangeRecorder.html

[5] https://eclipseide.org/

[6] https://git-scm.com/

[7] https://eclipse.dev/modeling/emf/

[8] https://standards.ieee.org/ieee/42010/6846/

[9] https://www.w3.org/TR/shacl/

[10] https://www.w3.org/TR/sparql11-query/

[11] https://doi.org/10.5281/zenodo.12784498

[12] https://projects.eclipse.org/projects/modeling.capra

[13] https://www.w3.org/TR/xpath/

[14] https://www.ecfr.gov/current/title-49/subtitle-B/chapter-V/part-571

[15] https://www.iso.org/standard/42076.html

[16] https://digitallibrary.un.org/record/635146

[17] https://www.infrastructure.gov.au/infrastructure-transport-vehicles/vehicles/vehicle-design-regulation/australian-design-rules/third-edition

References

1. D. D. Walden, T. Shortell, G. Roedler, et al., *INCOSE Systems Engineering Handbook*, 5th edn. (Wiley, 2023).

2. A. M. Madni and M. Sievers, "Model-Based Systems Engineering: Motivation, Current Status, and Research Opportunities," *Systems Engineering* 21, no. 3 (2018): 172–190.

3. V. Singh and K. E. Willcox, "Engineering Design With Digital Thread," *AIAA Journal* 56, no. 11 (2018): 4515–4528.

4. P. D. Saqui-Sannes, R. A. Vingerhoeds, C. Garion, and X. Thirioux, "A Taxonomy of MBSE Approaches by Languages, Tools and Methods," *IEEE Access* 10 (2022): 120936–120950.

5. J. Cederbladh, A. Cicchetti, and J. Suryadevara, "Early Validation and Verification of System Behaviour in Model-Based Systems Engineering: A Systematic Literature Review," *ACM Transactions on Software Engineering and Methodology* 33 (2023): 1–67.

6. S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language* (Morgan Kaufmann, 2014).

7. J. A. Estefan, "Survey of Model-Based Systems Engineering (MBSE) Methodologies," *Incose MBSE Focus Group* 25, no. 8 (2007): 1–12.

8. R. Mittal, R. Eslampanah, L. Lima, H. Vangheluwe, and D. Blouin, "Towards an Ontological Framework for Validity Frames," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (IEEE, 2023), 801–805.

9. C. A. Drewien, G. Artery, K. Eras, W. Ballard, and J. F. Nagel, "Effective System Engineering Peer Reviews," *Incose International Symposium* 25 (2015): 724–738.

10. L. P. Chao, I. Tumer, and K. Ishii, "Design Process Error-Proofing: Engineering Peer Review Lessons From NASA," in *Proceedings of the ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 46962 (ASME, 2004), 793–803.

11. J. Cederbladh, R. Eramo, V. Muttillo, and P. E. Strandberg, "Experiences and Challenges From Developing Cyber-Physical Systems in Industry-Academia Collaboration," *Software: Practice and Experience* 54 (2024): 1193–1212.

12. J. Cederbladh, "Light-Weight MBSE Approach for Construction Equipment Domain-An Experience Report," in *30th Asia-Pacific Software Engineering Conference (APSEC)* (IEEE, 2023), 51–60.

13. W. Vaneman, R. Carlson, and C. Wolfgeher, *Defining a Model-Based Systems Engineering Approach for Milestone Technical Reviews. tech. rep*. (Naval Postgraduate School, 2019).

14. C. Orlowski, P. Blessner, T. D. Blackburn, and B. A. Olson, "A Framework for Implementing Systems Engineering Leading Indicators for Technical Reviews and Audits," *Procedia Computer Science* 61 (2015): 293–300.

15. E. L. Parrott and K. J. Weiland, Using Model-Based Systems Engineering to Provide Artifacts for NASA Project Life-cycle and Technical Reviews (AIAA, 2017).

16. A. Hogan, E. Blomqvist, M. Cochez, et al., "Knowledge Graphs," *ACM Computing Surveys* 54, no. 4 (2022): 71:1–71:37.

17. J. Cederbladh, L. Berardinelli, H. Bruneliere, et al., "Towards Automating Model-Based Systems Engineering in Industry-An Experience Report," in *IEEE International Systems Conference (SysCon)* (IEEE, 2024), 1–8.

18. J. Ma, G. Wang, J. Lu, H. Vangheluwe, D. Kiritsis, and Y. Yan, "Systematic Literature Review of Mbse Tool-Chains," *Applied Sciences* 12, no. 7 (2022): 3431.

19. 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* 26, no. 1 (2023): 104–129.

20. R. Haberfellner, P. Nagel, M. Becker, A. Büchel, and v. H. Massow, *Systems Engineering* (Springer, 2019).

21. H. Vangheluwe, J. D. Lara, and P. J. Mosterman, "An Introduction to Multi-Paradigm Modelling and Simulation," in *Proceedings of the AIS'2002 conference* (AI, Simulation and Planning in High Autonomy Systems), (Lisboa, Portugal, Vol. 21, No. 1, 2002).

22. A. Cicchetti, F. Ciccozzi, and A. Pierantonio, "Multi-View Approaches for Software and System Modelling: A Systematic Literature Review," *Software and Systems Modeling* 18 (2019): 3207–3233.

23. 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* 160 (2020): 110453.

24. S. F. Königs, G. Beier, A. Figge, and R. Stark, "Traceability in Systems Engineering–Review of Industrial Practices, State-of-the-art Technologies and New Research Solutions," *Advanced Engineering Informatics* 26, no. 4 (2012): 924–940.

25. J. Jiao and M. M. Tseng, "Fundamentals of Product Family Architecture," *Integrated Manufacturing Systems* 11, no. 7 (2000): 469–483.

26. M. Amrani, R. Mittal, M. Goulão, et al., "A Survey of Federative Approaches for Model Management in MBSE," in *MODELS Companion '24: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems* (ACM. Association for Computing Machinery, 2024), 990–999.

27. M. Bernardo, M. Casadesus, S. Karapetrovic, and I. Heras, "Relationships Between the Integration of Audits and Management Systems: An Empirical Study," *The TQM Journal* 23, no. 6 (2011): 659–672.

28. A. F. Mehr, I. Y. Tumer, F. Barrientos, and D. Ullman, "An Information-Exchange Tool for Capturing and Communicating Decisions During Early-Phase Design and Concept Evaluation," in *Proceedings of the ASME 2005 International Mechanical Engineering Congress and Exposition*, vol. 42304 (ASME, 2005), 125–131.

29. T. H. Beach, J. L. Hippolyte, and Y. Rezgui, "Towards the Adoption of Automated Regulatory Compliance Checking in the Built Environment," *Automation in construction* 118 (2020): 103285.

30. Z. Zhang, L. Ma, and T. Broyd, "Rule Capture of Automated Compliance Checking of Building Requirements: A Review," *Proceedings of the Institution of Civil Engineers-Smart Infrastructure and Construction* 176, no. 4 (2023): 224–238.

31. P. Zhou and N. El-Gohary, "Semantic Information Alignment of BIMs to Computer-Interpretable Regulations Using Ontologies and Deep Learning," *Advanced Engineering Informatics* 48 (2021): 101239.

32. N. O. Nawari, "Generalized Adaptive Framework for Computerizing the Building Design Review Process," *Journal of Architectural Engineering* 26, no. 1 (2020): 04019026.

33. J. Suryadevara and S. Tiwari, "Adopting mbse in Construction Equipment Industry: An Experience Report," in *25th Asia-Pacific Software Engineering Conference (APSEC)* (IEEE, 2018), 512–521.

34. S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand, and T. Coq, "A SysML-Based Approach to Traceability Management and Design Slicing in Support of Safety Certification: Framework, Tool Support, and Case Studies," *Information and Software Technology* 54, no. 6 (2012): 569–590.

35. W. Wang, N. Niu, M. Alenazi, and L. D. Xu, "In-Place Traceability for Automated Production Systems: A Survey of PLC and SysML Tools," *IEEE Transactions on Industrial Informatics* 15, no. 6 (2018): 3155–3162.

36. F. Rinker, L. Waltersdorfer, K. Meixner, D. Winkler, A. Lüder, and S. Biffl, "Traceable Multi-View Model Integration: A Transformation Pipeline for Agile Production Systems Engineering," *SN Computer Science* 4, no. 2 (2023): 205.

37. D. Badampudi, M. Unterkalmsteiner, and R. Britto, "Modern Code Reviews - Survey of Literature and Practice," *ACM Transactions on Software Engineering and Methodology* 32, no. 4 (2023): 107:1–107:61.

38. A. Ram, A. A. Sawant, M. Castelluccio, and A. Bacchelli, "What Makes a Code Change Easier to Review: An Empirical Investigation on Code Change Reviewability," in *ESEC/FSE 2018* (AMC. Association for Computing Machinery, 2018), 201–212.

39. M. Paixão, J. Krinke, D. Han, C. Ragkhitwetsagul, and M. Harman, "The Impact of Code Review on Architectural Changes," *IEEE Transactions on Software Engineering* 47, no. 5 (2021): 1041–1059.

40. A. Bacchelli and C. Bird, "Expectations, Outcomes, and Challenges of Modern Code Review," in (IEEE. IEEE Computer Society, 2013), 712–721.

41. S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An Empirical Study of the Impact of Modern Code Review Practices on Software Quality," *Empirical Software Engineering* 21, no. 5 (2016): 2146–2189.

42. L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, and A. Bacchelli, "Information Needs in Contemporary Code Review," *Proceedings of the ACM on Human-Computer Interaction* 2, no. CSCW (2018): 135:1–135:27.

43. F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion Detection in Code Reviews," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)* (IEEE. IEEE Computer Society, 2017), 549–553.

44. N. Davila and I. Nunes, "A Systematic Literature Review and Taxonomy of Modern Code Review," *Journal of Systems and Software* 177 (2021): 110951.

45. D. Fensel, *Ontologies* (Springer Berlin Heidelberg, 2001), 11–18.

46. M. Yahya, J. G. Breslin, and M. I. Ali, "Semantic Web and Knowledge Graphs for Industry 4.0," *Applied Sciences* 11, no. 11 (2021), https://doi.org/10.3390/app11115110.

47. d. J. B. S. França, J. M. Netto, d. J. E. S. Carvalho, F. M. Santoro, F. A. Baião, and M. G. Pimentel, "KIPO: The Knowledge-Intensive Process Ontology," *Software and Systems Modeling* 14, no. 3 (2015): 1127–1157, https://doi.org/10.1007/S10270-014-0397-1.

48. F. Toyoshima, A. Barton, and O. Grenier, "Foundations for an Ontology of Belief, Desire and Intention," *Frontiers in Artificial Intelligence and Applications* 30, (IOS Press, 2020), 140–154.

49. J. Rubin, K. Czarnecki, and M. Chechik, "Managing Cloned Variants: A Framework and Experience," in *SPLC '13: Proceedings of the 17th International Software Product Line Conference* (ACM, 2013), 101–110.

50. D. P. Clausing and K. V. Katsikopoulos, "Rationality in Systems Engineering: Beyond Calculation or Political Action," *Systems Engineering* 11, no. 4 (2008): 309–328.

51. M. Amrani, J. Dingel, L. Lambers, et al., "Towards a Model Transformation Intent Catalog," in *AMT '12: Proceedings of the First Workshop on the Analysis of Model Transformations* (ACM, 2012), 3–8.

52. N. N. Zolkifli, A. Ngah, and A. Deraman, "Version Control System: A Review," *Procedia Computer Science* 135 (2018): 408–415.

53. P. Shvaiko and J. Euzenat, "Ontology Matching: State of the Art and Future Challenges," *IEEE Transactions on Knowledge and Data Engineering* 25, no. 1 (2013): 158–176.

54. G. Weikum and G. Vossen, *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery* (Morgan Kaufmann, 2002).

55. L. Bettini, D. D. Ruscio, L. Iovino, and A. Pierantonio, "Supporting Safe Metamodel Evolution With Edelta," *International Journal on Software Tools for Technology Transfer* 24 (2022), https://doi.org/10.1007/s10009-022-00646-2.

56. D. E. Khelladi, R. Hebig, R. Bendraou, J. Robin, and M. P. Gervais, "Detecting Complex Changes and Refactorings During (Meta)model Evolution," *Information Systems* 62 (2016): 220–241, https://doi.org/10.1016/j.is.2016.05.002.

57. I. Schaefer, L. Bettini, V. Bono, F. Damiani, and N. Tanzarella, "Delta-Oriented Programming of Software Product Lines," in *Software Product Lines: Going Beyond. SPLC 2010. Lecture Notes in Computer Science*, vol. 6287 (Springer, 2010), 77–91.

58. A. Haber, K. Hölldobler, C. Kolassa, et al., "Systematic Synthesis of Delta Modeling Languages," *International Journal on Software Tools for Technology Transfer* 17 (2015): 601–626.

59. E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley Professional, 1994).

60. R. Mittal, D. Blouin, A. Bhobe, and S. Bandyopadhyay, “Solving the Instance Model-View Update Problem in AADL,” in *MODELS ’22: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, (MODELS ’22. ACM. Association for Computing Machinery, 2022), 55–65.

61. J. W. Wittler, T. Saglam, and T. Kühn, “Evaluating Model Differencing for the Consistency Preservation of State-Based Views,” *Journal of Object Technology* 22 (2023): 1–14.

62. M. Kowal, C. Legat, D. Lorefice, C. Prehofer, I. Schaefer, and B. Vogel-Heuser, “Delta Modeling for Variant-Rich and Evolving Manufacturing Systems,” in *MoSEMInA 2014: Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation* (ACM, 2014), 32–41.

63. A. Haber, K. Hölldobler, C. Kolassa, et al., “Systematic Synthesis of Delta Modeling Languages,” *International Journal on Software Tools for Technology* 17, no. 5 (2015): 601–626.