Cutting through the Jungle: Disambiguating Model-based Traceability Terminology

Jörg Holtmann*, Jan-Philipp Steghöfer†, Michael Rath‡, David Schmelter*
*Software Engineering & IT Security, Fraunhofer IEM, Paderborn, Germany
Email: joerg.holtmann@iem.fraunhofer.de
david.schmelter@iem.fraunhofer.de
†Chalmers | University of Gothenburg, Gothenburg, Sweden
Email: jan-philipp.steghofer@gu.se
‡Technische Universität Ilmenau, Ilmenau, Germany
Email: michael.rath@tu-ilmenau.de

***Abstract*—Traceability, a classic requirements engineering topic, is increasingly used in the context of model-based engineering. However, researchers and practitioners lack a concise terminology to discuss aspects of requirements traceability in situations in which engineers heavily rely on models and model-based engineering. While others have previously surveyed the domain, no one has so far provided a clear, unambiguous set of terms that can be used to discuss traceability in such a context. We therefore set out to cut a path through the jungle of terminology for model-based traceability, ground it in established terminology from requirements engineering, and derive an unambiguous set of relevant terms. We also map the terminology used in existing primary and secondary studies to our taxonomy to show differences and commonalities. The contribution of this paper is thus a terminology for model-based traceability that allows requirements engineers and engineers working with models to unambiguously discuss their joint traceability efforts.**

***Index Terms*—Requirements Traceability, Model-based Engineering, Terminology**

I. Introduction

Natural language remains the dominant documentation format for requirements specifications for software-intensive systems, e.g., in the automotive sector [1]. At the same time, models are increasingly applied in the engineering of such systems: Embedded systems development heavily relies on model-based systems engineering for interdisciplinary communication and model-driven software development for early automated analyses and code generation [2]. In combination, the development processes for software-intensive systems include intertwined development phases producing, among others, informal system requirements, semi-formal system design models, informal software requirements, formal software design models, code, and tests.

At the same time, traceability is demanded by many development and safety standards for software-intensive systems (e.g., [3], [4], [5]) and has to be established throughout the development lifecycle. The term traceability and its definition originate in the Requirements Engineering (RE) community [6], which also provides a terminology for this area. However, existing definitions partially contradict each other. For example, *vertical* and *horizontal* traceability are defined differently in the RE community by Ramesh & Edwards [7], Pfleeger & Bohner [8], and Gotel et. al [9]. Ramesh & Edwards, e.g., distinguish trace links associating artifacts belonging to different or the same lifecycle phase(s). Pfleeger & Bohner also distinguish this aspect, but switch the meanings for the terms vertical (same lifecycle phase) and horizontal (different lifecycle phases) due to a different visualization of the lifecycle. In contrast, Gotel et al. distinguish trace links associating artifacts based on their level of abstraction.

On the other hand, the model-based traceability literature uses the same terms with different meanings. For example, Bianchi et al. [10] and De Lucia et al. [11] redefine vertical and horizontal traceability in a model-based context, both citing Pfleeger & Bohner [8] (see above). According to their definitions, vertical traceability is the ability to trace artifacts belonging to the same model and horizontal traceability is the ability to trace artifacts belonging to different models. In contrast, other model-based engineering literature describes the same differentiation as intra- and inter-model traceability [12].

What's more, additional aspects that go beyond existing traceability terminology are relevant when working with models. For instance, when models are transformed during development to yield other models with more fine-grained granularity and additional details, trace links between the source models and the target models can be established automatically. In addition, trace links are often stored in trace models [13] or are included in models in languages such as the Unified Modeling Language (UML) [14] or the Systems Modeling Language (SysML) [15]. The traceability definitions from the RE community do not yet cover such cases. Thus, there is the need to extend these definitions to cover cases from model-based engineering.

Whereas existing secondary studies (e.g., [16]) list the terminology, they do not resolve such conflicts. They also circumscribe concepts rather than naming them concretely while reusing existing terminology with additional identi-

2332-6441/20/$31.00 ©2020 IEEE
DOI 10.1109/RE48521.2020.00014

fiers. For instance, Aizenbud-Reshef et al. [17] write about "explicit links or mappings that are generated as a result of transformations, both forward (e.g., code generation) and backward (e.g., reverse engineering)", thus conflating different aspects in the same definition.

As Cabré points out: "Ambiguous terminology [...] presents obstacles to communication among specialists and inevitably frustrates efforts to order thought." [18] Thus, a domain terminology is the common ground for a set of people communicating with each other in this domain. A scientific community defines their own terminology as the basis for efficient communication and research conduct. In the case of requirements traceability, Gotel et al. [19] consequently outlined a research road map that included developing a common terminology as the foundation of any further research. In the same year, the same authors published a basic and generic terminology [9], which is established in the RE community and which we use as a basis in this paper. However, with the inclusion of a new research community from the modeling area it becomes necessary to extend this basic terminology, include new terms, and describe their relationship to the existing ones. Eight years after the publication of Gotel et al. [19], it is also time to revisit the terms and re-evaluate their usefulness.

The contribution of this paper is a terminology to disambiguate the different terms used in model-based traceability. As a side effect, this terminology serves as a classification scheme for how trace links are used in model-based environments. Since primary studies rarely define the properties of the traceability approach they propose, our work helps comparing and contrasting different approaches.

We choose the term "model-based" here to include all development efforts that include modeling, regardless of how the models are used and how much automation is in place. We focus on those parts of the terminology that are relevant for model-based engineering, based on a refined and disambiguated foundational terminology for traceability in general. We do not address use cases for traceability and thus do not discuss aspects such as how trace links are created, or why development organizations trace. Our results are based on and validated with a tertiary literature review and samples from primary literature. We include a mapping to how the secondary and primary studies in the review use the concepts in our terminology. Our results aim at simplifying discussions about model-based traceability and bridge the gap between the RE and model-based engineering communities.

II. Related Work

Maybe the most complete set of traceability terms is provided by Gotel et al. [9]. Our understanding of the base terminology of traceability is founded on this chapter and we refer the reader to it as a baseline for our own definitions. However, Gotel et al. do not consider model-based traceability and define no terminology that relates specifically to the challenges found in situations in which models play an important role in the engineering process.

Winkler and von Pilgrim [16] collect terminology from both the requirements management and the model-driven software development literature. However, they only point out the differences but do not resolve the conflicts. Furthermore, they consider model-driven software development literature but do not cover broader model-based engineering aspects like from the systems engineering community.

Paige et al. [20] present a traceability terminology and classification in model-driven engineering based on existing literature. However, they conceive this terminology to precisely describe their main contribution, which is an approach for identifying and encoding trace links. Thus, they only identify terms that their traceability approach covers and do not aim at disambiguating the broader corpus of terminology provided by the literature.

In the context of systems engineering, Königs et al. [21] propose a terminology and classification scheme for traceability. However, the terminology is geared towards two traceability approaches by the authors and does not aim at unifying terminology definitions from requirements and modeling literature. Furthermore, the proposed terminology is only partially based on existing literature.

In addition, several papers define some terminology for traceability as part of their background sections. For instance, Bianchi et al. [10] define a number of terms such as horizontal and vertical links. Likewise, Broy [22] introduces *intra- and inter-artifact* links, Jaber et al. [23] introduce *causal* and *non-causal* links, and Knether et al. [24] define *implicit* and *explicit* links. However, these papers often cite each other and propose conflicting definitions such as the ones pointed out in the introduction. Our aim is to disambiguate these terms and extend the common traceability terminology with terms helpful in describing trace links in model-based environments. Additionally, we organize the terms in a widely applicable trace link classification scheme.

III. Research Method

Our research method consists of several iterative refinements of the terminology. The process is described in the following and also depicted in Figure 1.

A. Stage 1: Identification of common definitions

As a first stage, we have used well-known primary sources such as [9] to get a first understanding of traceability terminology as a whole. We have also used primary sources on model-based engineering such as [25] to establish common definitions within this community. Finally, we have consulted highly-cited works in model-based traceability such as [17] to identify concepts that are missing from the standard definitions of traceability when model-based engineering is concerned. We then created draft definitions based on the understanding in the author team. The outcome of this first stage was a draft taxonomy containing preliminary definitions of the most important concepts.

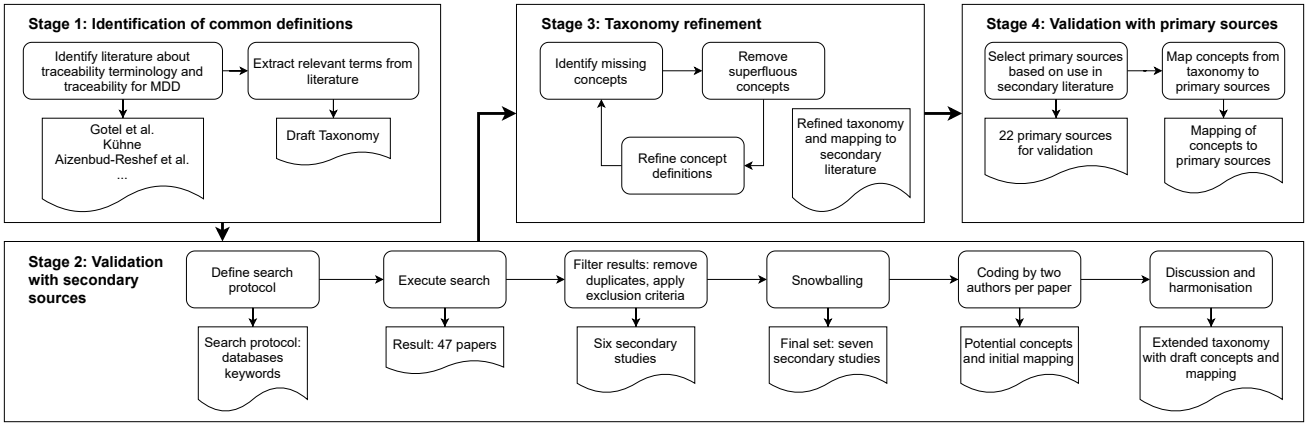


Fig. 1. The multi-stage research methodology applied to define and refine the terminology.

B. Stage 2: Validation with secondary sources

In order to validate our initial terminology from the first stage, we conducted a *tertiary literature review* in the second stage in which we systematically analyzed secondary sources to better understand how traceability terminology is used in overview papers for model-based traceability. Since the number of primary studies that address model-based traceability is staggering (Pilgrim and Winkler's review [16] from 2010 cites more than 200 papers), we opted to instead use survey and review papers to gain an understanding of the consensus within the field. Since these papers already aggregate the terminology of the papers they reviewed, we also expected to be able to base our own work on previous efforts.

We utilized three electronic databases to search for relevant papers: ACM Digital Library, IEEE Xplore, and Web of Science. Furthermore, we defined a generic search query that we modified subsequently to match the requirements of the concrete platform, as shown in Table I. Referring to the PICO-Method [26], the search term *traceability* represents the **p**opulation for our search, i. e., the specific software engineering discipline. The terms *model-driven* and *model-based* serve as **i**ntervention and tie the search to this area of software and systems engineering. Our research objective does not require a **c**omparison so that we did not include any terms for this part. At last, the **o**utcome is relevant to *survey* or *review*.

Executing theses queries returned a total of 47 papers. Of those, ten papers appeared in multiple databases and were removed as duplicates. We carefully read the abstracts and scanned the remaining papers to *(a)* exclude those which were not secondary literature and *(b)* papers not matching our definitions and research context. After this step, six papers [17], [27], [28], [29], [16], [30] were selected as secondary sources. We also performed snowballing by identifying other relevant studies not found by our search queries in the reference lists of the selected papers. In this step, we found one additional paper by Zikra et al. [31]. It is interesting to note that the resulting seven papers only cover the period between 2006 and 2014, although we did not restrict the publication years.

In a round-robin fashion, each paper was then assigned to two of four authors for coding. We used a number of predetermined codes based on Stage 1 that were extended with emergent codes when necessary. The coding results were discussed and harmonized. This yielded trace link concepts and a mapping of our terminology captured in our supplemental material [32] which were used in the third stage to refine the terminology.

C. Stage 3: Taxonomy refinement

As the third stage, we used the information gathered in stage two to refine our taxonomy. This included removing concepts, adding concepts we had missed, and refining concepts by changing their names and/or changing the definitions. As an example, we added the definition for supra-model trace links after identifying this relevant concept in the secondary sources. We also identified conflicting definitions and tried to resolve them by, e.g., introducing new terms or by mapping one of the conflicting definitions to other, existing terms. For instance, we introduced the new term transformation trace links to specifically denote trace links that are automatically created from model transformations. The term allows us to differentiate such links from other computed links that are, e.g., the result of using a trace recovery algorithm. The final version of our taxonomy is described in Section V.

D. Stage 4: Validation with primary sources

As a fourth and final stage, we validated our taxonomy from the previous stage. For this purpose, we followed 22 references from secondary sources to primary sources and compared the concepts with our terminology. This enabled us to identify discrepancies between published papers and our consolidated taxonomy. As an example, we mapped Pfleeger & Bohner's notion of *horizontal* trace links ("horizontal traceability addresses the relationships

TABLE I
SEARCH QUERIES, RETURNED HITS AND SELECTED PAPERS FOR EACH LIBRARY.

Origin	Specific search query	Hits	Selected
ACM	("model driven" "model based") AND (+traceability) AND (survey review)	6	
IEEE	(("All Metadata":model−driven OR "All Metadata":"model based") AND ("All Metadata":traceability) AND ("All Metadata":review OR "All Metadata":survey))	11	[27], [28]
Web of Science	ALL=(model−driven OR "model based") AND ALL=(traceability) AND ALL=(survey OR review)	30	[17] [29], [30], [16]
Snowballing			[31]

of these components across pairs of workproducts" [8]) to *inter-model* or *supra-model* trace links (cf. our supplemental material [32]). Further examples of definitions not conforming to our concepts are mentioned when we introduce the terminology in Section V.

E. Threats to Validity

Our main measure to address *external validity* was to map our terminology to that used in other studies (cf. Table II and Table III). By showing that our terminology can be mapped to the one used elsewhere, we demonstrate that our results are generalizable. In terms of *reliability*, we acknowledge a certain subjectivity in constructing the terminology and the mapping. To address this, two researchers have coded each secondary study and the independent coding was compared and discussed until both converged on the same opinion. We also published the full mapping available for external validation [32]. We mitigated threats to *construct validity* by reviews of our search protocols and by broadening our search terms as widely as possible.

IV. WHAT IS A MODEL?

In order to define model-based traceability unambiguously, we need to differentiate conceptual and technical definitions of models. The difference is important since *(a)* these definitions do not always match up and *(b)*, we need an unambiguous understanding of model boundaries, that is, whether trace artifacts belong to the same model.

A number of *conceptual definitions* from academic literature [33], [25] allow to decide whether a given thing is a model. For instance, according to Wasowski and Berger, "A model is an abstraction of reality made with a given purpose in mind" [34]. This generic definition encompasses the characteristics defined by Stachowiak [35] that a model has to be based on an original, reduce the original, and serve a purpose. However, this conceptual view does not help us to differentiate whether two things are indeed two different models or part of the same model.

In order to decide whether trace artifacts belong to the same model, we need to take a more technical approach to models. Some literature seems to assume that all information relevant in a specific life-cycle phase is captured in a single model, e. g., requirements in [10]. However, the practical reality in a modern development environment is quite heterogeneous: Product visions might be described in natural language in a text file created in a word processor, system and software requirements can be captured as structured text in separate linked IBM DOORS modules or as a goal model in a notation such as KAOS [36], and design artifacts are captured in separate linked SysML and UML design models. Even more technical definitions, such as the one used by the OMG in the MDA Guide [37] explicitly allow bundling heterogeneous artifacts together in one model, as long as they address the same "concern": "[...] a model of a software system could include a UML class diagram, E/R (Entity-Relationship) diagrams, and images of the user interface [...]"

For the purposes of traceability and this paper, we need to differentiate between these artifacts and treat them as different models. The reason is that many of the terms used in model-based traceability rely on the ability to clearly distinguish two different models. Whether trace links are *intra-model* or *inter-model* (cf. Section V-B1), e.g., can only be decided if it is unambiguous whether two artifacts are treated as different models or not. Leaving such a decision to subjective choice introduces ambiguity again and makes it more difficult for engineers to discuss the trace links they work on together. Thus, we define models as follows:

A model represents an aspect of a system under development captured in a specific instance of a formal language that serves a purpose within the development lifecycle.

This means that for our purposes, some of the artifacts discussed above reside in models since they are expressed in instances of formal languages [38]. Since IBM DOORS or ReqIF [39] give natural language text a structure by providing a meta-model for requirement items (similar to KAOS), we consider the resulting structured requirement texts instances of a formal language and thereby models. Thus, a trace link between such structured requirements and SysML/UML models should be categorized as inter-model. Natural language as part of a general-purpose text file for the product vision does not constitute a model in our sense, because the word processor does not provide a meta-model for structuring the contents. So, we categorize trace links between the product vision and the other artifacts as supra-model. Since the definition refers to *formal language*,

code artifacts can be treated like models as they are based on a grammar. Please note that natural language text pasted en bloc into structured requirements documents are a boundary case. If the structural elements of the formal language are not used to represent the content, we would not consider such a document a model.

V. Terminology

In this section, we first introduce foundational terminology and subsequently add more precise definitions for the model-based aspects of traceability. These definitions are based on *Stage 3* in our methodology (cf. Section III-C), and thus represent the final, refined definitions. Please note that we do not make assumptions about trace links such as their directionality, cardinality, or whether they are typed or not. Our terminology applies regardless of such distinctions and is written in the most abstract way possible by only referring to "artifacts" (anything created during development) or "elements" (specific parts of an artifact, such as a line of code or a class in a UML model) that are linked to each other.

Figure 2 depicts a classification scheme specified by means of a feature model [40] for trace links, which we use to conceptually visualize the structure of our terminology. In a concrete trace link variant, all features can be arbitrarily combined as long as the mandatory features and the cross-feature dependencies are respected.

A. Foundational Terminology

In this section, we present the existing terminology introduced by the RE and model-based traceability literature (e.g., [9], [41]) and refined by us to remove ambiguities. All changes to the originals are pointed out in the following. The feature model in Figure 2 represents this part of the terminology by mandatory features of any trace link. Please note that horizontal and vertical traceability are not included here (cf. Section VI-B1).

1) Basic Terms: Gotel et al. [9] present a common, generic terminology for traceability: "*Traceability* is the potential for *traces* [...] to be established (i.e., created and maintained) and used". A trace encompasses *trace artifacts* as well as a *trace link* associating the trace artifacts, where the latter ones "are traceable units of data". Trace artifacts as well as trace links have a *trace artifact type* and *trace link type*, respectively. These types are labels characterizing artifacts or links "that have the same or similar structure (syntax) and/or purpose (semantics)". A *traceability information model* defines "the permissible trace artifact types [and] the permissible trace link types", often by means of a traceability meta-model.

2) Specificity: Previous works [41], [11], [20] identified the traceability aspect termed *specificity* in our feature model (cf. Figure 2). We define two kinds of specificity:

Implicit trace links exist whenever naming conventions or similar techniques are used without explicit syntactical elements to establish relationships.

Explicit trace links establish relationships between model elements by means of abstract syntax elements of a formal language.

An example for implicit links is tracing from text documents including requirements, design documents, or error logs to source code using keywords or meaningful names for code identifiers such as method names. In the context of models, naming conventions are applied to implicitly associate trace artifacts, e.g., a state machine implementing a requirement has the same name as the requirement. Trace links are thus not explicitly modeled.

In contrast, tools with direct traceability support such as IBM DOORS use explicit links. In the context of models, modeling means are applied to explicitly associate trace artifacts. For instance, in SysML [15], a trace link between a state machine implementing a requirement is established by means of an instance of the meta-class Dependency with the stereotype «satisfy».

3) Storage: This feature (cf. Figure 2) refers to the applied persistence strategy, and has an impact on how to work with trace links: "Storing traces in dedicated traceability models external to the models containing the actual trace artifacts has the advantage that the traces do not unnecessarily 'pollute' these models" [20].

Volatile trace links are not stored for later retrieval, but rather only kept in memory until an operation utilizing them is completed.

Persisted trace links are stored for later retrieval.

- **Internal** trace links are stored within one of the traced artifacts.
- **External** trace links are stored separately from the traced artifacts.

Volatile trace links are, e.g., generated by model transformations that keep track of mapped artifacts only during execution of the transformation algorithm (such as ATL model transformations [42]). Another example are on-the-fly trace recovery mechanisms for implicit trace links [11].

Examples for internal trace links are issue IDs in Java files, UML/SysML trace links contained in the same model as (at least one of) the traced artifacts, or the storage mechanism of OSLC [43]. In contrast, external trace links are persisted in dedicated trace models, databases, or files.

4) Automation: This feature (cf. Figure 2) refers to the origin of a trace link, that is, whether it is established by a human or by an algorithm:

Manual trace links are established by a human user.

Computed trace links are established by an algorithm.

- **Transformation** trace links are computed trace links that are established by a model transformation.

Examples of human-established trace links are found, e.g., between a state machine and a requirement using a manually created SysML «satisfy» trace link. Computed trace links are, for example, established by applying automated reasoning such as an information retrieval algorithm on requirements in natural language (cf. [27], [44]).

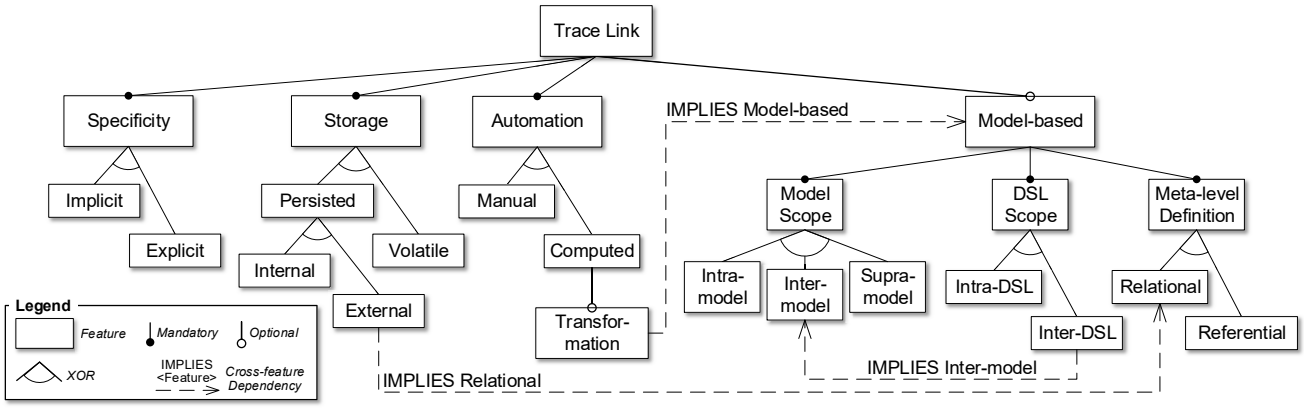


Fig. 2. Classification scheme for trace links specified by means of a feature model.

A model transformation (e.g., one that automatically generates one or more target UML sequence diagrams for a source UML use case) can establish transformation trace links between source and target model elements automatically as a "by-product" [17], [16] so that an engineer can track the transformation outcome. Model transformations can also establish transformation trace links only for the purpose of enabling the "target-incrementality" [45] feature of the model transformation approach [17], [16]. Transformation trace links are a special kind of computed trace links, so that Transformation is an optional sub-feature of the feature Computed in Figure 2. In the case of a transformation trace link created by a model transformation, the link is also model-based so that the particular sub-features of the feature Model-based in Figure 2 apply (cf. cross-feature dependency of the feature Transformation).

B. Specific Terminology for Model-based Traceability

In the following, we present the terminology specific to model-based traceability. In terms of the feature model (see Figure 2), it is represented by mandatory features of the optional Trace Link sub-feature Model-based.

1) Model Scope: As outlined in the introduction, trace links in a model-based environment can connect elements within the same model, elements that are part of different models, or elements that are not part of a model (cf. Fig. 2):

Intra-model links associate elements within the same model (i.e., the same instance of a formal language).

Inter-model trace links associate elements of different models (i.e., different instances of a formal language or instances of different formal languages).

Supra-model trace links associate elements of which at least one is not a model.

For example, refinement relations between requirements within the same ReqIF model [39] are intra-model trace links (cf. case A in Figure 3). Which internal relationships are useful as trace links has to be decided on a case-by-case basis. Containment (e. g., a class that is contained in a UML package) is one of the most common relationships and might not be useful when performing a typical traceability activity such as program comprehension. However, it can be exploited in an overall traceability approach to enable impact analyses as proposed by [46].

Inter-model trace links reside, e.g., in one model and associate trace artifacts from another model by importing and referencing them (cf. case B in Figure 3). Other examples associate elements of different models or between a model element and text-based formal languages via a trace model (cf. cases C and D in Figure 3). When considering source code and tests, we define each code file to be a separate instance of the formal (programming) language. Thus, a trace link between a test and a class or method that is established via naming conventions counts as an (implicit) inter-model trace link.

Supra-model trace links associate, e.g., model elements and images (cf. case E in Figure 3). Another example are links between informal requirements as part of unstructured documents (e.g., using naming conventions or stating the identifier of the associated requirement).

2) DSL Scope: Two different models connected by an inter-model trace link either are instances of the same formal language (DSL) or of different ones.

Intra-DSL trace links associate elements in models based on the same language.

Inter-DSL trace links associate elements from different formal languages.

Illustrative examples for intra-DSL trace links are depicted in the cases A – C in Figure 3. This includes means to specify trace links that are defined in the formal language itself, such as SysMLs' «satisfy», «allocate», or «derives» trace link types [15] (cf. cases A and B in Figure 3). We consider trace links associating elements of different models that are instances of the same DSL via a separate trace model as intra-DSL trace links, even if that trace model is based on a different formal language (cf. case C in Figure 3 connecting models 1 and 2 via model 3).

An example for an inter-DSL trace link is one that associates a requirement in ReqIF [39] with a class in

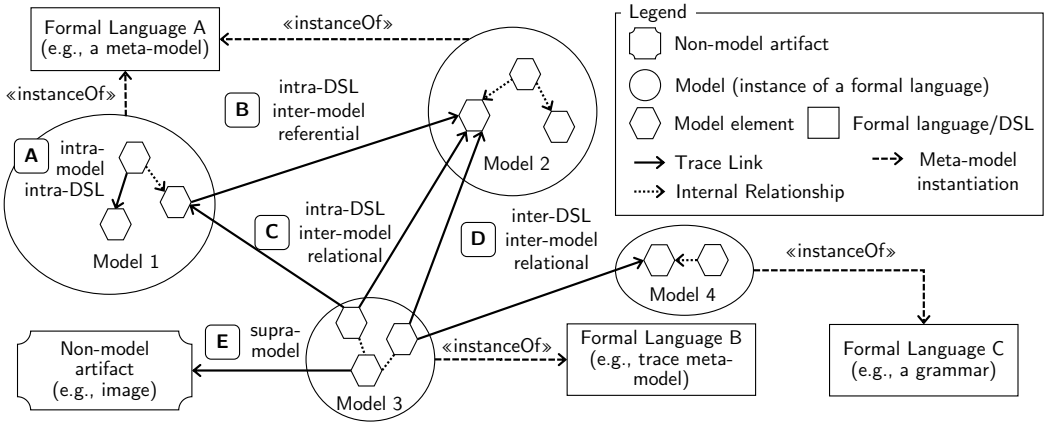


Fig. 3. An illustration of exemplary combinations of the trace link features Model Scope, DSL Scope, and Meta-level Definition.

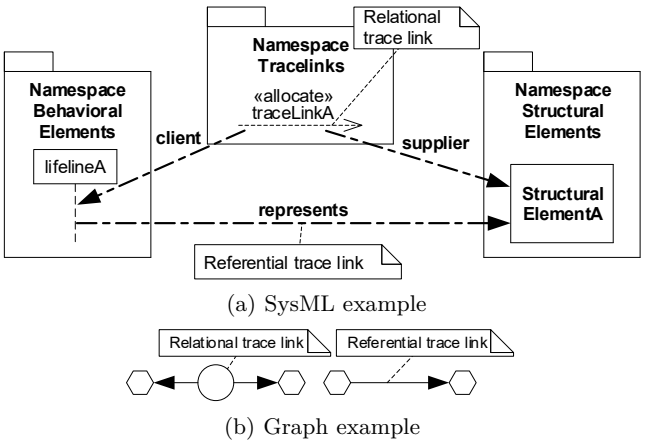
UML (cf. case D in Figure 3 connecting Models 2 and 4 via Model 3). We assume that an extension of a formal language such as a UML profile always creates a new formal language. That means that a trace link from a model element in a "vanilla" UML model to a model element in a model that uses UML extended with a profile is inter-DSL traceability. Since inter-DSL trace links associate trace artifacts from different models, their application always implies inter-model trace links (cf. cross-feature dependency of the feature Inter-DSL in Figure 2).

3) Meta-level Definition: Aizenbud-Reshef et al. [17] distinguish relationships that "are kept either as properties in the representation of the artifacts [...] or as 'first-class citizens,' having their own representation [...]". We refer to this aspect as the *meta-level definition* of trace links (cf. Figure 2). Thereby, we distinguish between traceability provided by modeling languages to express traceability relationships in a dedicated manner and traceability that is an effect of other considerations in the meta-model design as part of the inherent model coherence:

Relational trace links associate model elements by means of an instance of a meta-class provided by the meta-model of a modeling language or of a traceability management tool.

Referential trace links associate different model elements by means of a direct reference that is an instance of an association between two meta-classes.

In terms of UML/SysML [14], [15], relational trace links are instances of the meta-class Abstraction and its specialization Dependency, which have association ends pointing to generalizations of the trace artifact meta-classes. Such dedicated objects reference the corresponding trace artifact objects and have a concrete syntax as well as their own namespace. For example, the object traceLinkA (cf. Figure 4(a)) has two abstract syntax links client and supplier to its trace artifacts and is owned by Namespace Tracelinks. In terms of graphs, a relational trace link is represented by a dedicated node with incident directed edges pointing to the trace artifact nodes



(a) SysML example

(b) Graph example

Fig. 4. Examples for relational and referential trace links.

(e.g., the node depicted on the left of Figure 4(b) as well as the cases C and D in Figure 3). Traceability management tools typically provide trace link types in a traceability meta-model. Since external trace links reference trace artifacts in other models from their own namespace, their application always implies relational trace links (cf. cross-feature dependency of the feature External in Figure 2).

In UML/SysML, referential trace links are instances of an association between two trace artifact meta-classes. Such links associate the trace artifact objects directly and have neither a concrete syntax nor a separate namespace. In Figure 4(a) the lifelineA has an abstract syntax link represents to StructuralElementA, where this link manifests as a property of the lifeline. In terms of graphs, a referential trace link is a directed edge from one trace artifact node to another trace artifact node (e.g., the directed edge depicted on the right of Fig. 4(b) and case B in Fig. 3). Referential trace links also appear in the OSLC standard [43], where the Uniform Resource Identifier of the target trace artifact is stored as a property of the source trace artifact.

VI. Discussion

Based on the terminology presented in the previous section, we illustrate how the terminology can be mapped to some secondary as well as primary literature papers, discuss notable absences, and explore some ways in which the terminology can be used in scientific discourse.

A. Mapping the Terminology

One of our goals was to be able to map our terminology to existing work in the area to allow discussing this previous work using the new terminology and thus resolving ambiguities. Table II shows how the terminology presented in Section V is used in three of the papers we included in our tertiary literature review. An empty cell indicates that this terminology was not used in the paper.

Additionally, we also validated the terminology by looking into primary literature. Similarly to Table II, Table III shows how the terminology maps to these papers. Every table cell presents a quote from primary literature and the respective citing secondary literature. The highlighted phrases indicate key information used to assign the quote to a feature in our taxonomy. A complete mapping to 7 secondary and 22 primary sources and more details for the ones listed here can be found in [32].

B. Notable Absences

We did not add the following terms to our terminology:

1) Horizontal and Vertical Trace Links: The most notable absence might be *horizontal* and *vertical* trace links. Whereas these characteristics for trace link types are fairly common in the literature, their definitions are contradicting as outlined in the introduction. In addition, they are based on other concepts that are hard to define. Overall, this makes the definition hard to apply and severely limits its usefulness as we will outline in the following.

Our main reason for not defining horizontal and vertical trace links is that the terms represent concepts that visualize development process models. They thus strongly rely on an individual's visual ideas of the process model. As outlined in the introduction, both Pfleeger & Bohner [8] and Ramesh & Edwards [7] define horizontal and vertical trace links in terms of lifecycle phases, but define the particular terms in contradiction to each other due to different visual arrangements of the phases. Moreover, agile processes or Boehm's incremental spiral model [56] have much more complicated visualization options than conventional waterfall and V-models.

Further, the lifecycle phases themselves are poorly defined. Software process literature has acknowledged that "there is no unique, universal software lifecycle that can be used in any organization and for any product." [57]. The phase relevancy strongly depends on the system under development and on the organizational structure of the company developing the system. Therefore, coarse-grained definitions such as Sommerville's [58], e. g., who suggests to use "requirements definition", "system and software design", "implementation and testing", and "operation and maintenance" are rarely applicable in practice any only apply to a system that encompasses only software ("software systems" according to [59]). In addition, modern approaches such as the Scaled Agile Framework (SAFe) [60] contain many more phases such as "planning", "portfolio management", and "decommission". Furthermore, for a system that encompasses software as well as hardware ("software-intensive systems" according to [59]), there typically are dedicated system requirements analysis and system architectural design phases prior to the software-specific phases and additional validation phases.

We can easily define horizontal and vertical trace links referring to lifecycle phases through other terms in our terminology. For example, a work product according to Pfleeger & Bohner [8] is a software lifecycle object such as requirements, design, code, and test plans. Each work product can internally have relationships between its various parts. Pfleeger & Bohner call this vertical traceability. The set of work products is in turn connected via a graph, thus establishing horizontal traceability. These definitions are also used in other sources [10], [11] and correspond to our definition of the aspect *Model Scope* in Section V-B1.

A second branch of traceability literature defines horizontal and vertical traceability by means of abstraction levels. Whereas Gotel et al. [9] describe vertical traceability as "accommodat[ing] life cycle-wide or end-to-end traceability", they and others (see, e. g., [28], [9], [61]) define horizontal traceability linking artifacts with the same or different "levels of abstraction". We find this problematic since the term "level of abstraction" is not well-defined. None of the references provide a good definition that is easily applicable to a given set of artefacts. Is a domain model, e.g., a refinement of a requirements document and thus on a different level of abstraction? Or does it represent the problem domain in a way that is just as abstract as the requirements? If not, what would two artifacts on the same level of abstraction look like?

In summary, we believe that the notions of horizontal and vertical trace links are too ambiguous to be useful and thus suggest to not use them anymore when discussing traceability. Both the "lifecycle scope" and the "level of abstraction" interpretations rely on other, ill-defined terms and are thus subject to interpretation and can not only differ from organization to organization, but also from team to team and product to product within an organization.

2) Structural and Cognitive Traceability: Bianchi et al. [10] as well as De Lucia et al. [11] identify the aspect of structural vs. cognitive traceability. Structural knowledge is derivable from the source code. Thus, a "structural link [is] the traceability associated with syntactic dependency relationships between software artifacts" [10]. Trace links based on knowledge about the semantics of the system are "cognitive links". These definitions distinguish based on which "evidence" links were created. When coding the secondary studies, we found it very difficult to understand

TABLE II
MAPPING OF THE TERMINOLOGY PROPOSED IN THIS PAPER TO THE TERMINOLOGY USED IN THREE OF THE SURVEYED ARTICLES.

	Aizenbud-Reshef et al. [17]	Galvão and Goknil [28]	Winkler and v. Pilgrim [16]
Implicit	"Interconnections are largely implicit [...]"	"[...] implicit relationships between models [...]"	"[...] implicit traceability links [...] between two elements [...] not explicitly connected [...]"
Explicit	"End-to-end integration can make these relationships explicit [...]"	"[...] store and represent trace information [...] explicitly as trace models."	"[...] links which are captured directly in models themselves using a suitable concrete syntax [...]"
Volatile	"[...] internal traceability, which is maintained by the transformation engine during the transformation [...]"	"Such a form of traceability need not persist after executing a transformation."	
Internal	"[...] tools store link information within the artifacts themselves."	"[...] links are embedded inside the target models [...]"	"[...] models can contain the links themselves [...]"
External	"Keeping linking information separate from the artifacts [...]"	"[...] external traceability links, stored in separate models [...]"	"Trace models are usually stored as separate models [...]"
Manual	Manual relationship	"[...] add trace links manually [...]"	"Human interaction [...] to record [...] traces [...]"
Computed	Computed relationship; analysis relationship	"[...] automate the discovery and the generation of trace relationships [...]"	"explicit traces [...] which can be unambiguously reconstructed at any time [...]"
Transform.	Derivation relationship	"[...] model transformation mechanisms for generating trace information."	"[...] traces can be recorded automatically as a by-product of model transformations."
Intra-Model		"[...] addition of [...] transformation traceability links into UML models [...]"	
Inter-Model		"trace relationship between model elements [...] in different models."	"[...] technical coupling of different models [...]"
Supra-Model		"[...] relationships between application requirements and models [...]"	Pre-model and post-model traceability
Intra-DSL		UML traceability standard stereotype	SysML trace link types
Inter-DSL			"[...] model-to-[code] traceability can be facilitated [. . .]."
Relational	"[...] 'first-class citizens,' having their own representation [...]"	Trace metamodels; UML traceability standard stereotype	Traceability metamodels; SysML trace link types
Referential	"[...] properties in the representation of artifacts [...]"		

which evidence was used in many cases. In addition, we doubt the explanatory power of these definitions: Even when inspecting a specific link, they do not allow to reconstruct if an engineer created this link based on syntactical considerations or due to their knowledge of the "semantics of the system".

3) Causal and Non-Causal Trace Links: For similar reasons, we did not use the definitions proposed by Jaber et al. [23]. They distinguish between *causal* and *non-causal* trace links. The former ones "[r]epresent relationships that have an implied logical ordering between source and sink. For e.g., bug reports cannot be produced unless the source code is available." The latter ones "[r]epresent relationships that must agree with each other, but the causality cannot be clearly determined. For e.g., multiple versions of the same document in different languages must agree, but there need not be a causal relationship among them." In order to decide whether a specific link is causal or non-causal, the link creation process needs to be understood.

4) Backward and Forward Traceability: Finally, we omitted *backward traceability* ("that is, to previous stages of development") and *forward traceability* ("that is, to all documents spawned by the [requirements specification]") as defined in [62]. These definitions refer to the aspect of lifecycle scope discussed in Section VI-B1 and in particular to a waterfallish thinking about how development artifacts are created. In practice, the traceability meta-model often prescribes a semantic reading direction for the links (usually, links go from sources in earlier lifecycle phases to targets in later phases), but modern tools allow traversing such links in both directions [9]. Therefore, and since the terms did not play a major role in the secondary literature we reviewed, we omitted them from our terminology.

C. Using the Terminology to Characterize Traceability Approaches

An aspect that became apparent when coding the secondary studies, and in particular when coding Winkler and von Pilgrim [16] as well as Galvão and Goknil [28], is that our terminology can be used to characterize different traceability approaches. That is, it allowed us to easily assign terms based on the short descriptions of just a few sentences. For instance, Winkler and von Pilgrim describe an approach by Ghervasi and Zhowgi [63] where a formal logic model is generated from requirements in natural language along with corresponding trace links that are persisted in a database. We could easily tag this as creating trace links that are supra-model, computed, and persisted. In this way, our terminology does not only help

TABLE III
MAPPING OF THE TERMINOLOGY PROPOSED IN THIS PAPER TO THE TERMINOLOGY USED IN SELECTED PRIMARY LITERATURE.

	Reference
Implicit	"Implicit traceability results from existing associations between elements of the system model. For example, **the use of the same identifier in an analysis and a design artefact expresses a dependency between both.** [...] In contrast to explicit traceability implicit traceability describes references between two model elements, without any additional properties. It is possible to search for implicit connections, to store them and make them explicit." [47] cited in Winkler et al. [16]
Explicit	"Explicit traceability is defined in terms of **trace links that are concretely represented in models.**" [48] cited in Winkler et al. [16]
Volatile	"A target element may for instance be referred to by using its corresponding source element as a key. [...]. Such a form of traceability **need however not persist after executing a transformation**. For this reason, we call it internal traceability." [49] cited in Galvão et al. [28] and Santiago et al. [29]
Internal	"In our view, storing traceability links in separate models is more preferable than **embedding the traceability information in the models they refer to**. This is because in addition to the aforementioned advantages, this strategy is able to capture both intra-model as well as inter-model links." [12] cited in Santiago et al. [29]
External	"Concepts from open hypermedia, such as n-ary first class links and lightweight hypermedia tool adapters, can be applied to facilitate the efficient capture and management of relationships between artifacts. Hypermedia links with lightweight tool adapters cross heterogeneous tools at reasonable development costs. **Stored outside the artifacts they connect**, n-ary first class links enable tracing heterogeneous artifacts that are maintained in diverse formats with different tools." [50] cited in Santiago et al. [29]
Manual	"Explicit Traceability results from the **establishing of connections between** two artefacts during the software development process **by a developer**. [...] **The creation** of explicit **traceability requires additional effort of the developer.** If one or both of two linked artefacts are changing, there is a risk that the traceability link is becoming inconsistent or invalid. It is necessary, to check explicit traceability links between changed artefacts, before they are used." [41] cited in Winkler et al. [16]
Computed	"This work introduces a new, strongly iterative approach to trace analysis. We will show that it is possible to **automatically generate new trace dependencies** and validate existing ones." [51] cited in Galvão et al. [28]
Transform.	"Implicit traceability involves **trace links that are created and manipulated by application of MDE operations.**" [48] cited in Winkler et al. [16]
Intra-Model	"Vertical traceability refers to the abiity to trace dependent artefacts within a model, while horizontal traceability refers to the **ability to trace artefacts between different models** [...]." [11]
Inter-Model	"Traceability metamodel is not fixed and traceability models can always be further processed into any format. As for range, TracerAdder could be easily modified to not add TGC to specific rules, for instance, and thus reduce trace model size. Although an EMF-specific feature was used to implement **inter-model linking**, other possibilities exist." [49] cited in Galvão et al. [28] and Santiago et al. [29]
Supra-Model	"An artifact is a piece of information produced or modified as part of the software engineering process [...]. Artifacts take a variety of forms including **models, documents**, [...] A link (a, a') represents an explicit relationship defined between two artifacts a and a'." [52] cited in Galvão et al. [28]
Intra-DSL	"The term traceability information applies to a wide spectrum, ranging from hand-crafted **intra-model links, such as stereotyped UML associations**, to toolgenerated inter-model links relating elements of the source and target models of an automatic transformation." [53] cited in Santiago et al. [29]
Inter-DSL	"Several requirements modeling languages such as the requirements package of SysML have been developed to improve the elicitation, analysis, validation and verification of requirements during project development life cycles. However, **none of these languages is generic enough to embed explicit traces to components of arbitrary system architecture languages** intending to provide a solution to the problem formalized by requirements specifications. For example, **systems engineers using the Architecture Analysis and Design Language (AADL) cannot broidge SysML requirements to their architecture models in the same way it is done for UML models**. The only way would be to define an external trace model linking the requirements to AADL model elements." [54] cited in Assar et al. [27]
Relational	"[...] a SysML requirement is usually linked to a UML NamedElement expected to satisfy the requirement by **creating a UML dependency link** tagged with the SysML *Satisfy* stereotype." [54] cited in Assar et al. [27]
Referential	"Our traceability mechanism addresses these points, by providing The use of relations **instead of simple links**. TOOR can link requirements to design documents, specifications, code, and other artifacts through user-definable relations that are meaningful for the kind of connection being made. This lets developers distinguish among different links between the same objects. Also, by using mathematical properties of relations such as transitivity, TOOR can relate objects thate are not directly linked. This makes the trace procedure more powerful because it allows identifying not only the objects linked to a given object, but also the objects related to a given object by some specific pattern of composed relations." [55] cited in Aizenbud-Reshef et al. [17]

us discuss model-based traceability, but can serve as a tool to characterize and differentiate traceability approaches based on which types of links they create. Further examples of this can be found in our supplementary material [32].

VII. CONCLUSION

In this paper, we provide definitions for traceability terminology in the context of model-based development. Based on existing terminology for requirements traceability, we develop a set of terms that allow us to describe how traceability is used and which properties trace links possess. Furthermore, we capture these terms in a classification scheme specified by means of a feature model. We used a tertiary literature review as well as samples from primary literature to validate and refine our terminology with a total of seven secondary studies and 22 primary studies.

Our terminology disambiguates and harmonizes literature from the RE and the model-based engineering communities. We have also refined the definitions of existing terminology and identified and removed ambiguous concepts. In this way, we believe our work will simplify discussions between requirements engineers and engineers working with models since they can now use a set of unambiguous terms to discuss traceability concepts. Furthermore, the classification scheme provides a way to characterize different model-based traceability approaches based on the type of trace links they produce.

Acknowledgements

This research has been partly sponsored by DFG grant MA 5030/3-1 as well as the ITEA3 project *Panorama* (https://panorama-research.org/) under Vinnova grant 2018-02228 and BMFI grant 01IS18057F.

References

[1] G. Liebel, M. Tichy, and E. Knauss, "Use, potential, and showstoppers of models in automotive requirements engineering," *Software and Systems Modeling*, vol. 18, no. 4, 2019.

[2] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice," *Software & Systems Modeling*, vol. 17, no. 1, 2018.

[3] Automotive Special Interest Group / VDA QMC Working Group 13, *Automotive SPICE Process Reference and Assessment Model, Version 3.1*, 2017.

[4] International Organization for Standardization (ISO), *Road vehicles—Functional safety (ISO 26262:2018)*, 2018.

[5] Radio Technical Commission for Aeronautics (RTCA), *Software Considerations in Airborne Systems and Equipment Certification (DO-178C)*, 2011.

[6] O. C. Z. Gotel and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," in *Proc. 1st Int. Conf. on Requirements Engineering (RE)*. IEEE, 1994.

[7] B. Ramesh and M. Edwards, "Issues in the development of a requirements traceability model," in *Proceedings of the IEEE International Symposium on Requirements Engineering*. IEEE, 1993.

[8] S. L. Pfleeger and S. A. Bohner, "A framework for software maintenance metrics," in *Proceedings. Conference on Software Maintenance 1990*. IEEE, 1990.

[9] O. C. Z. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder, "Traceability fundamentals," in *Software and Systems Traceability*. London, UK: Springer, 2012.

[10] A. Bianchi, A. R. Fasolino, and G. Visaggio, "An exploratory case study of the maintenance effectiveness of traceability models," in *8th Int. Workshop on Program Comprehension (IWPC)*. IEEE, 2000.

[11] A. De Lucia, F. Fasano, and R. Oliveto, "Traceability management for impact analysis," in *2008 Frontiers of Software Maintenance*, 2008.

[12] N. Drivalos, R. F. Paige, K. J. Fernandes, and D. S. Kolovos, "Towards rigorously defined model-to-model traceability," in *ECMDA Traceability Workshop (ECMDA-TW)*, 2008, pp. 17–26.

[13] S. Maro, A. Anjorin, R. Wohlrab, and J. Steghöfer, "Traceability maintenance: factors and guidelines," in *31st IEEE/ACM Int. Conf. on Automated Software Engineering, ASE*, 2016.

[14] Object Management Group, *OMG Unified Modeling Language, Version 2.5.1*, 2017. [Online]. Available: http://www.omg.org/spec/UML/2.5.1/

[15] ——, *OMG Systems Modeling Language, Version 1.6*, 2019. [Online]. Available: http://www.omg.org/spec/SysML/1.6/

[16] S. Winkler and J. von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Software & Systems Modeling*, vol. 9, no. 4, 2010.

[17] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, "Model traceability," *IBM Systems Journal*, vol. 45, no. 3, 2006.

[18] M. T. Cabré, "Terminology and standardization," in *Terminology: Theory, methods, and applications*, ser. Terminology and lexicography research and practice, J. C. Sager, Ed. Amsterdam: Benjamins, 1999, vol. 1, ch. 6.

[19] O. C. Z. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, and G. Antoniol, "The quest for ubiquity: A roadmap for software and systems traceability research," in *Proc. 20th IEEE Int. Conf. on Requirements Engineering (RE)*, P. Sawyer and M. Heimdahl, Eds. Piscataway, USA: IEEE, 2012, pp. 71–80.

[20] R. F. Paige, N. Drivalos, D. S. Kolovos, K. J. Fernandes, C. Power, G. K. Olsen, and S. Zschaler, "Rigorous identification and encoding of trace-links in model-driven engineering," *Software & Systems Modeling*, vol. 10, no. 4, 2011.

[21] 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*, vol. 26, no. 4, 2012.

[22] M. Broy, "A logical approach to systems engineering artifacts: semantic relationships and dependencies beyond traceability—from requirements to functional and architectural views," *Software & Systems Modeling*, vol. 17, no. 2, pp. 365–393, 2018.

[23] K. Jaber, B. Sharif, and C. Liu, "A study on the effect of traceability links in software maintenance," *IEEE Access*, vol. 1, 2013.

[24] A. Von Knethen and B. Paech, "A survey on tracing approaches in practice and research," Fraunhofer IESE, IESE-Report No. 095.01/E, 2002.

[25] T. Kühne, "Matters of (meta-) modeling," *Software & Systems Modeling*, vol. 5, no. 4, 2006.

[26] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering, Version 2.3," Keele University and University of Durham, EBSE Technical Report EBSE-2007-01, 2007.

[27] S. Assar, "Model driven requirements engineering: Mapping the field and beyond," in *IEEE 4th Int. Model-Driven Req. Engineering Workshop, MoDRE*, 2014.

[28] I. Galvão and A. Goknil, "Survey of traceability approaches in model-driven engineering," in *11th IEEE Int. Enterprise Distributed Object Computing Conf. EDOC*, 2007.

[29] I. Santiago, Á. Jiménez, J. M. Vara, V. de Castro, V. A. Bollati, and E. Marcos, "Model-driven engineering as a new landscape for traceability management: A systematic literature review," *Information & Software Technology*, vol. 54, no. 12, 2012.

[30] T. Yue, L. C. Briand, and Y. Labiche, "A systematic review of transformation approaches between user requirements and analysis models," *Requir. Eng.*, vol. 16, no. 2, 2011.

[31] I. Zikra, J. Stirna, and J. Zdravkovic, "Analyzing the integration between requirements and models in model driven development," in *EMMSAD*, 2011.

[32] J. Holtmann, J.-P. Steghöfer, M. Rath, and D. Schmelter. (2020) Cutting through the Jungle: Disambiguating Model-based Traceability Terminology—Terminology Mapping Table. [Online]. Available: https://doi.org/10.5281/zenodo.3895681

[33] J. Ludewig, "Models in software engineering–an introduction," *Software and Systems Modeling*, vol. 2, no. 1, 2003.

[34] A. Wasowski and T. Berger, *Principles of Software Language Design for Model-Driven Software Engineering*. Springer, 2020. [Online]. Available: http://mdsebook.org/

[35] H. Stachowiak, *Allgemeine Modelltheorie*. Springer, 1973.

[36] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde, "Grail/kaos: an environment for goal-driven requirements engineering," in *Proceedings of the 19th International Conference on Software Engineering (ICSE)*, 1997, pp. 612–613.

[37] Object Management Group, *MDA Guide Revision 2.0*, June 2014.

[38] S. C. Reghizzi, L. Breveglieri, and A. Morzenti, *Formal languages and compilation*. Springer, 2013.

[39] Object Management Group, *Requirements Interchange Format (ReqIF), Version 1.2*, 2016. [Online]. Available: https://www.omg.org/spec/ReqIF/

[40] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep. CMU/SEI-90-TR-21, 1990.

[41] P. Mäder, I. Philippow, and M. Riebisch, "Customizing traceability links for the unified process," in *International Conference on the Quality of Software Architectures*. Springer, 2007, pp. 53–71.

[42] Eclipse Foundation, *Atlas Transformation Language (ATL)*. [Online]. Available: https://www.eclipse.org/atl/

[43] Organization for the Advancement of Structured Information Standards, *Open Services for Lifecycle Collaboration (OSLC)*. [Online]. Available: https://open-services.net/

[44] N. Sannier and B. Baudry, "Toward multilevel textual requirements traceability using model-driven engineering and information retrieval," in *2nd IEEE Int. Ws. on Model-Driven Requirements Engineering (MoDRE)*, Sep. 2012, pp. 29–38.
[45] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, vol. 45, no. 3, 2006.
[46] P. Letelier, "A framework for requirements traceability in UML-based projects," in *1st Int. Workshop on Traceability in Emerging Forms of Software Engineering*, 2002.
[47] P. Mäder, O. C. Z. Gotel, and I. Philippow, "Getting back to basics: Promoting the use of a traceability information model in practice," in *Proceedings of the 5th ICSE Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*. IEEE, 2009.
[48] R. F. Paige, G. K. Olsen, D. Kolovos, S. Zschaler, and C. D. Power, "Building model-driven engineering traceability," in *ECMDA Traceability Workshop (ECMDA-TW)*. Sintef, 2010, p. 49.
[49] F. Jouault, "Loosely coupled traceability for ATL," in *Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany*, vol. 91, 2005, p. 2.
[50] H. U. Asuncion, "Towards practical software traceability," in *Companion of the 30th International Conference on Software Engineering*, 2008, pp. 1023–1026.
[51] A. Egyed, "A scenario-driven approach to trace dependency analysis," *IEEE transactions on Software Engineering*, vol. 29, no. 2, pp. 116–132, 2003.
[52] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 796–810, 2003.
[53] D. S. Kolovos, R. F. Paige, and F. A. Polack, "On-demand merging of traceability links with models," in *3rd ECMDA traceability workshop*, 2006, pp. 47–55.
[54] D. Blouin, E. Senn, and S. Turki, "Defining an annex language to the architecture analysis and design language for requirements engineering activities support," in *2011 Model-Driven Requirements Engineering Workshop*. IEEE, 2011, pp. 11–20.
[55] F. A. Pinheiro and J. A. Goguen, "An object-oriented tool for tracing requirements," *IEEE software*, vol. 13, no. 2, pp. 52–64, 1996.
[56] B. W. Boehm, J. A. Lane, S. Koolmanojwong, R. Turner, and F. P. Brooks, *The Incremental Commitment Spiral Model: Principles and Practices for Successful Systems and Software*. Addison-Wesley/Pearson, 2014.
[57] G. Cugola and C. Ghezzi, "Software processes: a retrospective and a path to the future," *Software Process: Improvement and Practice*, vol. 4, no. 3, pp. 101–123, 1998.
[58] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.
[59] ISO/IEC/IEEE, *Systems and software engineering—Vocabulary (ISO/IEC/IEEE 24765:2017)*, 2017.
[60] D. Leffingwell, *SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises*. Addison-Wesley Professional, 2018.
[61] L. C. Briand, Y. Labiche, and T. Yue, "Automated traceability analysis for uml model refinements," *Information and Software Technology*, vol. 51, no. 2, 2009.
[62] "IEEE guide for software requirements specifications," IEEE, IEEE Std 830-1984, 1984.
[63] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," *ACM Trans. on Soft. Eng. and Methodology (TOSEM)*, vol. 14, no. 3, 2005.