Software and Systems Modeling (2022) 21:2133–2169
https://doi.org/10.1007/s10270-021-00958-y

THEME SECTION PAPER

Requirements document relations

A reuse perspective on traceability through standards

Katharina Großer[1] · Volker Riediger[1] · Jan Jürjens[1,2]

Received: 18 November 2020 / Revised: 8 October 2021 / Accepted: 22 November 2021 / Published online: 25 January 2022

Abstract
Relations between requirements are part of nearly every requirements engineering approach. Yet, relations of views, such as requirements documents, are scarcely considered. This is remarkable as requirements documents and their structure are a key factor in requirements reuse, which is still challenging. Explicit formalized relations between documents can help to ensure consistency, improve completeness, and facilitate review activities in general. For example, this is relevant in space engineering, where many challenges related to complex document dependencies occur: 1. Several contractors contribute to a project. 2. Requirements from standards have to be applied in several projects. 3. Requirements from previous phases have to be reused. We exploit the concept of "layered traceability", explicitly considering documents as views on sets of individual requirements and specific traceability relations on and between all of these representation layers. Different types of relations and their dependencies are investigated with a special focus on requirement reuse through standards and formalized in an Object-Role Modelling (ORM) conceptual model. Automated analyses of requirement graphs based on this model are able to reveal document inconsistencies. We show examples of such queries in Neo4J/Cypher for the EagleEye case study. This work aims to be a step toward a better support to handle highly complex requirement document dependencies in large projects with a special focus on requirements reuse and to enable automated quality checks on dependent documents to facilitate requirements reviews.

Keywords Requirement document relations · Requirement reuse · Standards · Space engineering requirements · ECSS · Traceability

1 Introduction

Relations between individual requirements have been described intensively in past research and are part of nearly every requirements engineering approach, although in practice their purpose—which is very important for beneficial use [1,2]—often remains implicit or undocumented. Goodrum et al. [3], e.g., quote the usage of the vague term *"links"*. There appears to be a lack of common understanding or agreement on the semantics of reoccurring relation types and the set of relations to be used. Also, there is still little support of specific relationships in common requirements engineering tools [4], even though there exists a need for high-end traceability [5], e.g., in highly standardized and safety critical embedded systems domains, such as automotive, defense, or aerospace. Relations between requirements *views*, such as documents, are also rarely considered. Gotel and Finkelstein [6] showed that there exist different relations on different

Communicated by J. Araujo, A. Moreira, G. Mussbacher, and P. Sánchez.

✉ Katharina Großer
grosser@uni-koblenz.de

Volker Riediger
riediger@uni-koblenz.de

Jan Jürjens
juerjens@uni-koblenz.de
http://jan.jurjens.de/

1 Institute for Software Technology (IST), University of Koblenz-Landau, Universitätsstraße 1, Koblenz 56070, Germany

2 Fraunhofer Institute for Software and Systems Engineering (ISST), Dortmund, Germany

levels of representation granularity, which interact for composite artifacts, such as documents. While they demonstrated this for stakeholder contribution relations, this layered structure is not addressed for other relation types and even still rarely reflected at all in common tools.

This might be due to the fact that views are not fixed hierarchical structures but aggregate parts of different elements in new contexts, which also allows to partially derive them automatically from the requirement model. However, these initial views are altered or complemented by additional information when used, as already described by Leite et al. [7–9]. These alterations are part of the socio-technical nature of views as a human interface to the requirement model. Thus, the content, structure, and relations of views express semantics beyond the requirement model, what makes them traceable objects on their own. Some approaches inversely use existing documents to derive models from them, for example Ramesh and Jarke [5]. Here, documents are mostly treated as sources to justify requirements objects, again without further relations on their own. There exist many meta-models for requirements [5,10–12], and document structures [13,14] (and also templates and guidelines for both [15–19]) but to our knowledge no combined meta-model covering requirements, document views, and *interrelated* relations of both. This is remarkable, as requirement documents and their structure are a key factor in requirement reuse [20]. Re-occurring structures, e.g., sections provided by document templates or information on project characteristics, provide *abstraction*. Standards or reference specifications are designed to be reused, making this explicit. Such abstractions guide the *selection* and *specialization* [21] of appropriate reusable documents. Yet, there is a lack of structured processes for *integration*. Requirement reuse in the past has received relatively little attention [22], although requirements from similar domains for similar tasks are more likely to be similar than the components implementing them [23]. Research and industry increasingly discover this potential [22]. Explicit formalized relations between documents can help to ensure consistency, improve completeness, and facilitate reviews especially in, but not restricted to, the context of reuse. Interacting requirements are a common source for errors [24] and manual analysis is time-consuming and costly [25]. In addition, there is a need for clearly defined relationships to foster more formal methods, e.g., in the relation between requirements and goals [26] and with respect to a broad use of the term traceability [3]. To address this, the following research questions are investigated:

RQ-I *Which types of relations are relevant for requirement reuse from documents?*

RQ-II *How can requirements reuse from documents be formalized into a structured process that supports integration?*

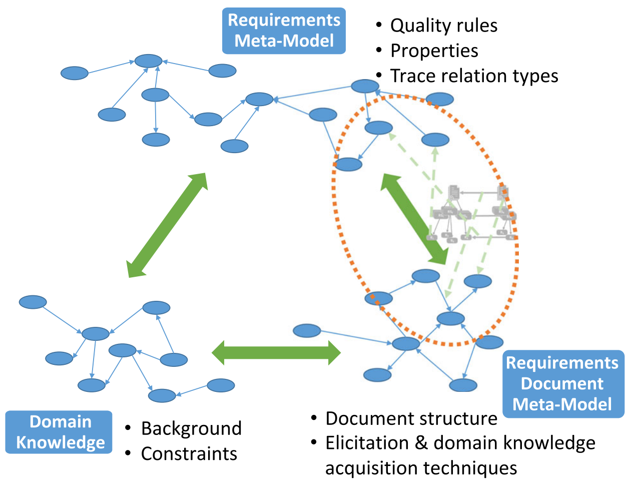


Fig. 1 Bridging requirements meta-model and document meta-model

The goal is to handle highly complex requirement document dependencies better in large projects to facilitate reviews and structured reuse by enabling automated quality checks on these dependent documents. Figure 1 illustrates that these research questions aim at the bridge between requirements meta-models (the properties, relations, and quality rules of requirements) and requirements document meta-models (the structure and properties of specification documents and, by relating it to specific topics, requirements elicitation and domain knowledge acquisition techniques). This is one major part of the overall integration of different ontology or meta-model types to support requirements engineering as identified by Castañeda et al. [20], which we address with our *T-Reqs* framework.

To achieve this, as depicted in steps (A) and (B) of Fig. 2, we collected knowledge in the respective areas from expert interviews, general literature, and guidelines in requirements engineering, as well as analysis of specification documents from different space engineering projects, among them the *EagleEye* earth observation reference mission. Resulting from an ontological analysis guided through the research questions RQ-I & RQ-II, we define a conceptual model (C). It provides a reference frame to define dependent relations on different layers, but special focus lays on its applicability to the different document reuse scenarios observed in space engineering, in particular the application of standards. The steps of a case study conducted with an excerpt from EagleEye are depicted in green in the lower right half of Fig. 2. Data extracted semi-automated from the original EagleEye PDF documents (D) are used to populate an instance of the conceptual model implemented in *Neo4J* (E). The graph contains a requirement model of EagleEye and its related ECSS standards as well as model representations of their document views analogous to the original PDFs. Graph queries demonstrate how quality constraints of the conceptual model can be

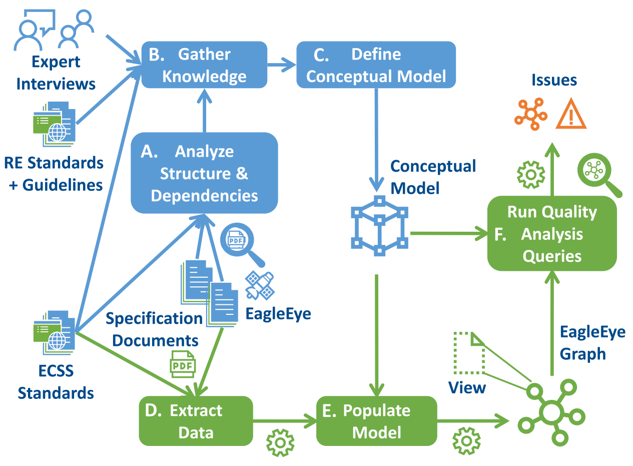


Fig. 2 Process to define model and conduct case study

checked and issues in dependencies to reused documents can be identified (F).

Section 2 distinguishes the work presented in this paper from some related approaches and in Sect. 3 we introduce a motivating running example from space engineering—the *EagleEye* satellite. Section 4 gives an overview on different types of requirements relations and general background on traceability and document-based reuse. While Sects. 5 and 6 present the conceptual model, the case study is described in Sect. 7. Finally, Sect. 8 summarizes the conclusions.

2 Related work

Already early tracing tools as presented by Pinheiro [27] support evolution and treat requirements and relations among them as objects. The approach does consider the situation that relations between different entities can depend on each other (and thus considers the kind of higher-order relation we aim at), but does not differentiate between the presentation and observation levels at doing so and does not pre-define such dependent relations. Similarly, this is treated in the approach of Schwarz et al. [12], which is based on graph technologies, as we also use them in our case study. The reference model of Ramesh and Jarke [5] differentiates documents as sources, but without trace relations among them. Building on these general reference models, we additionally include this type of traceability relation. Our refined reference model furthermore addresses specific dependencies between requirement documents and reused standards. Goknil et al. [28,29] use a formal requirements notation and typed relations to enable consistency checking between the links and inference of new relations. While this is in general the type of inter-dependencies, we are looking at, Goknil et al. do not consider documents or requirements sets as traceable entities on their own, and thus not the interdependencies between links on different layers. Furthermore, the approach depends on its formal notation of requirements. This enables inference of additional dependencies between requirements based on their content. Yet, formal notations put additional burdens on the requirements engineers and constitute a language barrier, which hinders stakeholder involvement and is not always applicable. We consider requirements as black box in our approach to enable support for multiple representations. Derivation rules for specific traceability relation types based on some notation can be combined with our framework, to assert those relations automatically. For example, Samer et al. [30] propose a method to automatically identify dependencies between textual requirements.

Maletic et al. [13] define document relations that connect fine-grained entities within software documents. The purposes of those relations are mainly document conformance and consistency. They build links between structural parts of documents such as paragraphs or sections, e.g., to define conformance or agreement relations between sections in multiple translations of a document. However, they do not consider requirements or other content as model elements outside of these documents. Espinoza et al. [14] present an approach that combines trace information and a traceability metaschema to define trace types with document models. Yet, they do not provide specific definitions of inter-dependencies of relation types and do not address different representation layers and their dependencies. Gotel and Finkelstein [6] explicitly describe these layers. However, they only examine in detail the impact to *contribution* relations and do not explicitly distinguish between model and view.

Requirements reuse is, i.a., addressed in current research on *Product Line Engineering*. For example, Abbas [31] or Reinhartz-Berger and Kemelman [32] present approaches to identify reusable requirements within the software product line requirements from customer requirements for individual products. Reusable documents as we consider them, e.g., standards, already define the reusable requirements. However, approaches like these could help standardization bodies or initiatives on generic specifications to specify or extend their catalogues or to determine from initial product requirements which domain standards should be made applicable. Goldin et al. [33] present a practical approach to requirements reuse, based on comparison between products of a product family and identification of reusable components. However, they do not consider generic reusable requirement sets and standards or automated rules to evaluate consistency and conformance. Naish and Zhao [34] propose a generalized framework to support the organization of requirements patterns for reuse. Their focus is on organization of reusable information, which would be applicable, e.g., to the structure of a standard database and the selection of applicable standards. Contrarily, our approach focuses on how to apply reusable requirements to the new project. The framework of Naish and Zhao could support the first steps of tailoring guiding the selection of standards, which we assume to be finished for our approach.

We look at compliance to reused higher-level requirements, e.g., from standards, in the sense of correct integration with the other project requirements at different levels of abstraction. The work of Siena et al. [35,36] and Zeni et al. [37] spuds in one step earlier to elicit this kind of requirements from sources, such as laws and norms, and to ensure compliance through their intentional alignment with these sources. Guo et al. [38] aim with their approach at automated tracing between requirements documents and *regulatory codes*. Regulatory codes for them can be higher-level requirements, laws—or standards, as we consider them in our approach. Yet, the conformance analysis of Guo et al. is based on general untyped trace links expressing assumably rationalization or refinement. Furthermore, standards concerned with “how to write requirements” are not considered explicitly, as opposed to our approach, where we introduce the concept of *meta-requirements*. Guo et al. treat individual regulatory codes as documents that need to be addressed in the project requirements. However, the containment structure of whole documents and different link types on different representation layers are not considered. The focus is on automated trace retrieval by semantic similarity supported by term matching. We did not consider this aspect of analyzing the requirements content in our work, but focused on the conceptual model to type and relate the traces. Similar to Guo et al., Wang et al. [39] present an approach for automatic trace retrieval from higher-level requirements. Yet, they focus on *vertical*[1] relationships among requirements on the same abstraction level, which should be reflected on the next *horizontal* level of requirements. They differentiate several sub-types of *dependency* to link requirements on one level. After an automated tracing of higher-level requirements to lower level requirements, similar to Guo et al., the dependency links are used to retrieve corresponding links between the lower level requirements. The main goal is not conformance analysis, but to enable better security testing by preservation of dependencies. However, the authors claim that the approach could support compliance analysis, too. Relations are established between individual requirements, documents or sets are not explicitly reflected as representation layers with links on their own. Furthermore, the conformance or refinement links between the levels do not consider meta-requirements either. Yet, the rules to shift vertical relations during horizontal refinement should be considered in future extension of our work addressing more general refinement relations. Especially temporal, causal, or resource dependencies among requirements also influence

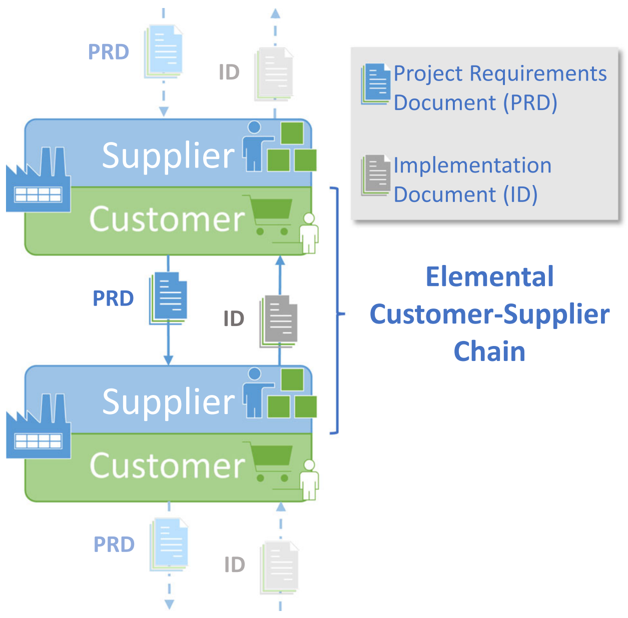

PRD
ID
Supplier
Customer
PRD
ID
Supplier
Customer
PRD
ID
Project Requirements Document (PRD)
Implementation Document (ID)
Elemental Customer-Supplier Chain

Fig. 3 Elemental customer–supplier chain [43]

conformance to higher-level requirements. Also Renault et al. [40] mention that dependencies among generic requirements from pattern catalogues can be propagated to the solutions requirements specification to improve traceability or facilitate elicitation. Automated trace retrieval techniques such as presented by Guo et al., Wang et al., and Samer et al. [30], or conflict detection by Ramadan et al. [41] could be combined with our approach in future work. Our conceptual model provides the rules to check, which requirements should be related to ensure compliance and infer additional relations, while automated analysis can retrieve the actual relations existent. We do not distinguish in our model, if trace links are asserted manually or by some automated approach.

3 Running example: EagleEye

We use a motivating running example from the space engineering domain. The focus of this paper is thereby on European space engineering. Yet, insights are expected to be generalizable to other comparable business domains. Space engineering is well suited as a demonstration domain for requirements engineering. As an embedded systems domain, the requirements have to deal with complex dependencies between several subsystems and the macro-system [9] with functions distributed over software and hardware components [42]. Furthermore, major projects always involve several cooperating industry and agency partners. They are organized hierarchically in the so-called *customer-supplier chain* [43]. Typically, a customer provides a set of interrelated requirements documents to several (potential) suppliers, i.e.,

[1] *vertical* means here at the same level of abstraction. This notion can be misleading depended on how the levels are presented (top-down or left to right). Hence, it is not used in the remainder of this paper.

technical, management, engineering, and product assurance requirements [44]. Actors at intermediate levels of the hierarchy are both supplier and customer.[2] Figure 3 shows the elemental customer-supplier chain. To handle this complexity, there is a strong need for standardization [46]. The *European Cooperation for Space Standardization* (ECSS[3]) is a standardization body providing a comprehensive system of standards for the European space industry. ECSS provides rules and guidelines for organized reuse of their standards called “tailoring” [43].

The *European Space Agency* (ESA[4]) introduced *EagleEye* as a virtual earth observation reference mission that is never meant to fly. EagleEye serves as a reference case study to test and demonstrate systems and software engineering approaches [47,48]. The satellite carries only one payload—a simple camera taking images of our planet’s surface from low earth orbit. The mission is prepared in the same way as usual missions and therefore represents a realistic example. The specification is expected to suffer from the same drawbacks like real missions, potentially even more as it is, as a virtual case study, prepared with low budget and low priority. Additionally, some examples are taken from the EUCLID[5] mission. Its development approach is claimed to be “the first attempt to apply an MBSE approach at mission level for a major science project under development in ESA” [42].

A simplified excerpt of the EagleEye document structure is used to investigate document and requirements dependencies. Figure 4 shows the main requirements documents and applicable ECSS standards used. The excerpt covers four levels of abstraction from the on-board software perspective, named after the typical document types found: *Mission Requirement Document* (MRD), *System Requirement Document* (SRD), the software system level called *Software System Specification* or *Software Requirements Baseline*, and the *Software Technical Specifications*, each with different sets of requirements depicted as linked boxes and associated documents containing these. Those sets colored in white have already undergone a consolidation activity. The links to the not yet consolidated sets, colored in grey, are broken, e.g., respective documents still link to old versions of other documents, as the outdated *Mission Requirements Document* (MRD) [49]. We use for the examples in this paper mainly the *Mission and System Requirements Document* (MSRD) [50] the *Software Requirements Specification* (SRS) [51], *Software Specification* (SWS) [52], and the *Central Software Requirements* (CSWR) [53]. The non-consolidated sub-tree for *Data Handling Software System Requirements* (SSR) and *Interface Control Documents* (ICDs) is not addressed in detail. Furthermore, the presented requirements documents are linked to several ECSS standards: E-60A [54], E-ST-60-10C [55], and E-ST-60-20C [56]—all on control engineering—E-ST-40B Part 1&2 [57,58], as software engineering standards, E-ST-50-13C [59], E-50-04A [60], and E-70-41A [61]—all concerned with telecommand and communication protocols.

4 Background

According to Gotel and Finkelstein [62]

> *Requirements traceability* refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction [...]

Traceability of a requirement back to its origin is also called *pre-RS* traceability, while the forward direction to its implementation in design and code is also called *post-RS* traceability, where RS stands for *requirements specification* [62]. Similarly, the ISO/IEC/IEEE Systems and software engineering - Vocabulary [63] defines *requirements traceability* as

> the identification and documentation of the derivation path (upward) and allocation/flow-down path (downward) of requirements in the requirements hierarchy. and “[a] discernable association between a requirement and related requirements, implementations, and verifications.”

These definitions already cover different aspects of traceability. Yet, they spare any detail about what other “discernable associations” and their application might be. Different stakeholders are interested in different types of relations, thus the interpretation of their semantics depends on theses stakeholders [2] and the intended use. To enable automated analysis and to provide hints of possible problems to requirements engineers, the quality and reliability of the relations have to be as high as possible. According to Pinheiro [27], “the use of formality increases traceability efficiency”. A formal mapping between formally described objects can enable automatic derivation of relationships, enhancing the functional aspects of traceability [27].

[2] Other than in the similar concept of *prosumer* in “industry 4.0” [45], the shared roles within the customer-supplier network are business-to-business relations and not explicitly considering end-consumers.

[3] The *European Cooperation for Space Standardization* is an initiative of different European space agencies and the *Eurospace* association to develop standards for use in all European space activities (http://ecss.nl, visited on 11/05/2019).

[4] The *European Space Agency* is an inter-governmental organization of 22 European member states dedicated to the exploration of space. (www.esa.int, visited on 11/05/2019).

[5] EUCLID “is a space-based optical/near-infrared survey mission designed to investigate the nature of dark energy, dark matter, and gravity by observing their signatures on the geometry of the Universe” [42] (http://sci.esa.int/euclid/, visited on 11/05/2019).

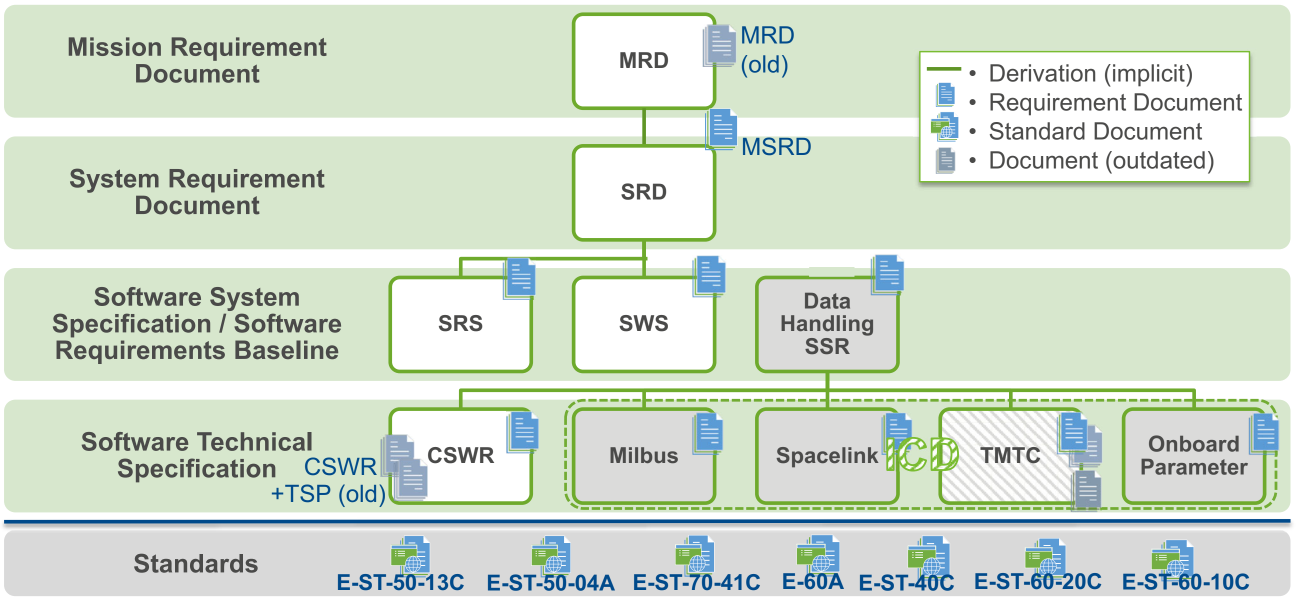


Fig. 4 Documents of the EagleEye example (excerpt)

Whilst the specific purpose and semantics of the relations is important for their beneficial usage, the respective representation in industrial application is shallow and remains often implicit. This has already been acknowledged at the turn of the millennium by Gotel and Finkelstein [64] and Dick [1], but has not changed since, as assessed, e.g., by Goodrum et al. [3] for respective tool support and Mavin et al. [26] for the use of goal-oriented approaches.

4.1 Traceability relation types

Many approaches define relation types, mostly as part of their specific meta-model, e.g., [5,10,11,14,29,42,65–67], or [12], but despite similarities there is no common terminology or semantics, as already criticized by Espinoza et al. [14,68] and listed as traceability challenge by Winkler and Pilgrim [69]. Spanoudakis and Zisman [2] identify commonalities and describe eight main relation types, complemented by Espinoza et al. [14] with two additional from Letelier [66], namely:

1. Dependency,
2. Generalization/Refinement,
3. Evolution,
4. Satisfaction,
5. Overlap,
6. Conflicting,
7. Rationalization,
8. Contribution,
9. Validation, and
10. Verification.

In the following exemplary those types, which we later explicitly use in our model and case study excerpts, are shortly described with examples from EagleEye.[6]

Generalization/Refinement Comprises the aggregation of composite elements and the representation of logical entities at different levels of abstraction [2].[7] It implies dependency.

This opens a wide range of possible interpretations. While formal approaches as KAOS [11,65] provide a clear definition of refinement within their context, in practice there is often no common interpretation [26]. According to Salay et al. [70], the following three types of refinement can be isolated:

1. To *enlarge scope* by adding new concepts.
2. To *detail* by adding more facts/constraints, but with the same concepts.
3. To *specialize*, where concepts are broken down to subtypes that are more specialized.

Example The SRD requirement EE-MR-0350 *"[...] shall account for the following sensors: Star tracker, Three-axis gyros, Sun sensors, Magnetometers, GPS"* [50] is *detailed* by the SWS requirement ATB-SR-EO-0580 *"[...] shall have the following sensors: 2 Star trackers, 2 Three-axis gyros, 2 Sun Sensor, 2 Magnetometers, 2 GPS"* [52]. The level of detail increases by quantifying the sensors.

[6] Examples are manually identified. Currently, only untyped traceability tables are recorded for a sub-set of the documents.

[7] Can be identified top-down as well as bottom-up [11].

Satisfaction An element e_1 *satisfies* an element e_2, if e_1 meets the expectation, needs, and desires of e_2; or if e_1 complies with a condition represented by e_2 [2].

Requirements are satisfied, e.g., by design or implementation artifacts, if their conditions hold for the given artifact. Furthermore, a requirement can contribute to the satisfaction of (higher level) requirements or goals [2].[8] The full propagation of satisfaction depends on the type of *refinement* relation between these requirements [71]:[9] [10]

a) If a requirement r_1 is *satisfied*, then its *specialized* requirements $\{r_1, \ldots, r_n\}$ should also be *satisfied*. However, *satisfying* a *specialized* requirement $r_i \in \{r_2, \ldots, r_n\}$ does not mean that the requirement r_1 from which it was *specialized* is also *satisfied*.
b) For *detailing*, the converse holds. If a requirement r_1 *details* some requirement r_2, *satisfying* r_1 implies *satisfying* of the original requirement r_2.

Example Take the example from *refinement* above: If the SWS [52] requirement ATB-SR-EO-0580 is satisfied, the SRD [50] requirement EE-MR-0350 is also satisfied. Whereas it alone does not fully satisfy the following SRD requirement EE-MR-0370 *"The AOCS subsystem shall account for redundancy of some hardware component to avoid critical and/or catastrophic consequences for the mission."* [50], as there might be other hardware components besides the sensors that need to be redundant.

Conflict Signifies conflicts between two elements e_1 and e_2 [2].

Example The redundant layout of certain sensors in System requirement EE-MR-0370 [50] (see above) could be in conflict with requirement EE-MR-0110 [50] limiting the total mass of the satellite to 250kg.

Rationalization Represent and maintain the rationale behind the creation and evolution of elements, and decisions at different levels of detail [2].

Example The EagleEye requirements documents do not provide any rationales, although this is strongly recommended. For ESA's EUCLID mission in contrast, justifications for each requirement are mandatory [42]. Justifications are mainly explanatory texts.

[8] Can also be negative for conflicting requirements [10].

[9] Pinheiro and Goguen [71] refer to a) as *derivation* and b) as *refinement*.

[10] KAOS [11] defines a refinement conjunction as *complete*, if the satisfaction of all contributing goals/requirements implies the satisfaction of the refined goal. This is always true for b) and a special case of a).

4.2 Requirement document relations

To answer the research question RQ-I "*Which types of relations are relevant for requirement reuse from documents?*", it is necessary to move from individual requirements relations to requirement sets and documents. We consider documents as kinds of *views*

> "An excerpt from an artifact, containing only those parts one is currently interested in. A view can abstract or aggregate parts of the artifact. [72]"

In our modeling context, as for Bruneliere et al. [73], "it is an instance of a particular view-type and consists of a set of elements coming from one or more [...] models. It is eventually complemented with some new interconnections between them and additional data that are manually entered and/or computed automatically." A view-type is a meta-model that determines the types of elements that can appear in a view [73].

View-types for requirements can be, for example, a use case diagram, dependency graph, or specification *document*—"[a] piece of written, printed, or electronic matter that provides information or evidence or that serves as an official record [74]." In this sense, we do not exclusively focus on classical text documents like PDF or Word/Writer files, as we use them exemplary from EagleEye, but also structured, mainly textual, representations as can be found in industrial requirements engineering tools, e.g., *LiveDocs* in Polarion[11], Visure Requirements[12] *documents*, or *modules* in DoorsNG[13]. Such document views are directly connected to the underlying requirement model, thus more flexible and more suitable for fully integrated model-based development approaches.

Like every view, each document or section within a document is a representation of a specific subject matter from the perspective of a given *viewpoint* [73]. According to the International Requirements Engineering Board (IREB) [72]:

> "A certain perspective on the requirements of a system. Typical viewpoints are perspectives that a stakeholder or stakeholder group has [...] there can also be topical viewpoints such as a security viewpoint."[14]

[11] By Siemens (https://polarion.plm.automation.siemens.com, visited on 10/05/2021).

[12] By Visure (https://visuresolutions.com/requirements-engineering, visited on 10/05/2021).

[13] By IBM (https://www.ibm.com/products/requirements-management, visited on 10/05/2021).

[14] Note that this definition is somewhat different from the definition of an *architectural viewpoint* as defined in ISO/IEC/IEEE 42010:2011 [75] and that there exist different appropriations of the terms *viewpoint* and *perspective*, which lead to different non-disjoint categories of views. For more details see, e.g., Silva and Leite [7].

Complementary, Bruneliere et al. [73] see it as

"the description of a combination, partitioning, and/or restriction of concerns from which systems can be observed." It governs the view-type definition.

According to ISO/IEC/IEEE 42010:2011 [75], "[t]here are two aspects to a viewpoint: the concerns it frames for stakeholders and the conventions it establishes on views."

Example The *Interface Control Document* (ICD) on *Telemetry and Telecommand* (TMTC) [76], part of the EagleEye specification, is a specific view on the EagleEye requirements. It is driven by the stakeholder perspective of satellite operations. Only those requirements relevant to define the interface to control the satellite and download telemetry data (e.g., housekeeping) are aggregated. The viewpoint implies a specific terminology as well as document style and format. For example, requirements have the form of packet layouts and service-type selections.

Dependent on the model type, different views and their dependencies can be automatically generated from a source model. Leite et al. [7,9] and Lamsweerde [11] demonstrate this for requirements. For example, the TMTC document [76] could be generated from a requirement database where the set of TMTC relevant requirements is marked as such, or from a model of relevant service-types and packet layouts.

In practice, however, we observe a "documents first" approach [69,77]. The document centered attitude is one of the barriers to adopt formal approaches in requirements engineering. To meet practitioners' needs better and to lower these barriers, the importance of views, such as documents, and their relations has to be taken into account. For example, *Document Requirements Lists* (DRLs) [44] define which documents are required by the time of specific review activities or project stages. *Document Requirement Definitions* (DRDs) [43] or other forms of document templates define the structure and content domains of specifications and thereby *guide* the requirements definition—they "structure the thinking". Such predefined categories can serve as checklists to guide the elicitation. In doing so, different kinds of requirements should be contained in different documents or document sections, as recommended by ISO/IEC/IEEE 29148 [19] or ECSS-M-ST-10 [44,78]. These different documents form an overall combined specification and contractual agreement [43]. However, the relationships between these parts are important for completeness and consistency checks. Furthermore, Gotel and Finkelstein [6] show, how relations on different representation layers of composite artifacts influence each other. In addition, Winkler and Pilgrim [69] acknowledge that "development artifacts exist in very different forms of representation, and that prior to recording fine[-]grained traceability links, it should be recorded how artifacts themselves are created, evolved, and transformed at a macro-level".

From EagleEye there are two types of document relations known:[15] So-called *applicable* documents (AD) "form part"[16] of the referencing document while *reference* documents (RD) "amplify or clarify its contents" [50,52]. While RDs have an informative character in terms of comments or justifications, ADs have a more binding character. Still, the given definitions are very vague. These relation types are similar to the *adopting* and *referencing* relations defined by Gotel and Finkelstein [64].

The documents from the EagleEye case study show an inconsistent usage of RD and AD relations, as can be seen in comparing Figs. 5 and 6. For example, the Central Software Requirements (CSWR) use an RD link to indicate refinement from the more abstract Mission and System Requirements (MSRD), while the Software Systems level documents (SWS and SRS) use an AD link for the same purpose. The same can be observed for the links to the ECSS standards, although we assume all of them should be applicable[17] here. The AD link between the Software System Specification (SWS) and the Software Requirements Specification (SRS) is bidirectional, as both mutually list the other as applicable document. As we do not see such mutual links between other documents, the meaning remains obscure. The CSWR completely ignores this next higher level and solely reference the MSRD. We assume that such inconsistent usage, as also observed in other projects, is caused by a lack of common understanding—shared semantics—between different involved organizations. This increases the complexity of the documentation and restrains tangibility of the overall dependencies. It makes the specification process error-prone and demands for time consuming consolidation activities. If documents evolve, these problems become even more serious as the change impact remains unclear in the same way.

The document relations as such are not expressive enough to clarify all of these dependencies. For example, standards have to be tailored when applied, what affects the contained requirements and their relation to the requirements in the project document. Thus, conformance can only be checked on individual requirement level, but depends on the document links. Gotel and Finkelstein [64], too, notice that "[...] more detailed semantics have implications for selective traceability[. Yet], consensus was found difficult to establish at that level [...]". If at all, most approaches only consider the relation types, introduced in Sect. 4.1, between individ-

[15] In the PDFs, references are explicitly listed in tables.

[16] The ECSS Glossary of Terms [79] defines an *Applicable Document* as a "document that contains provisions which, through reference in the source document, constitute additional provisions of the source document."

[17] In some projects, explicitly *normative document* links are introduced to denote applicable standards.

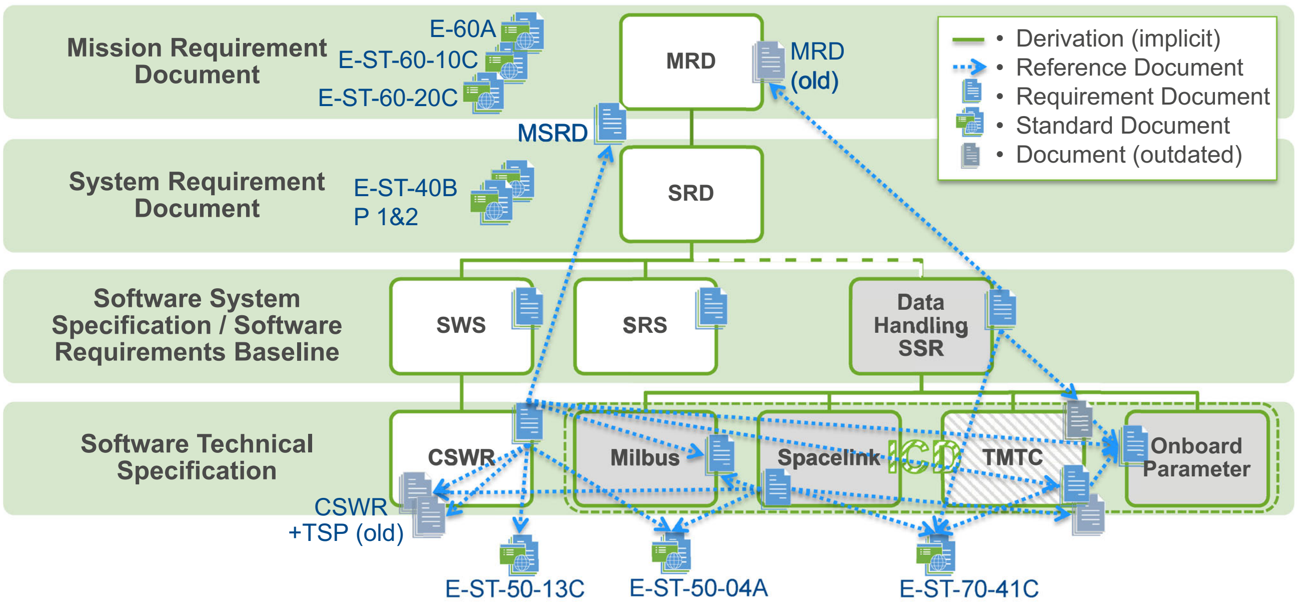


Fig. 5 Reference document relations of EagleEye (excerpt)

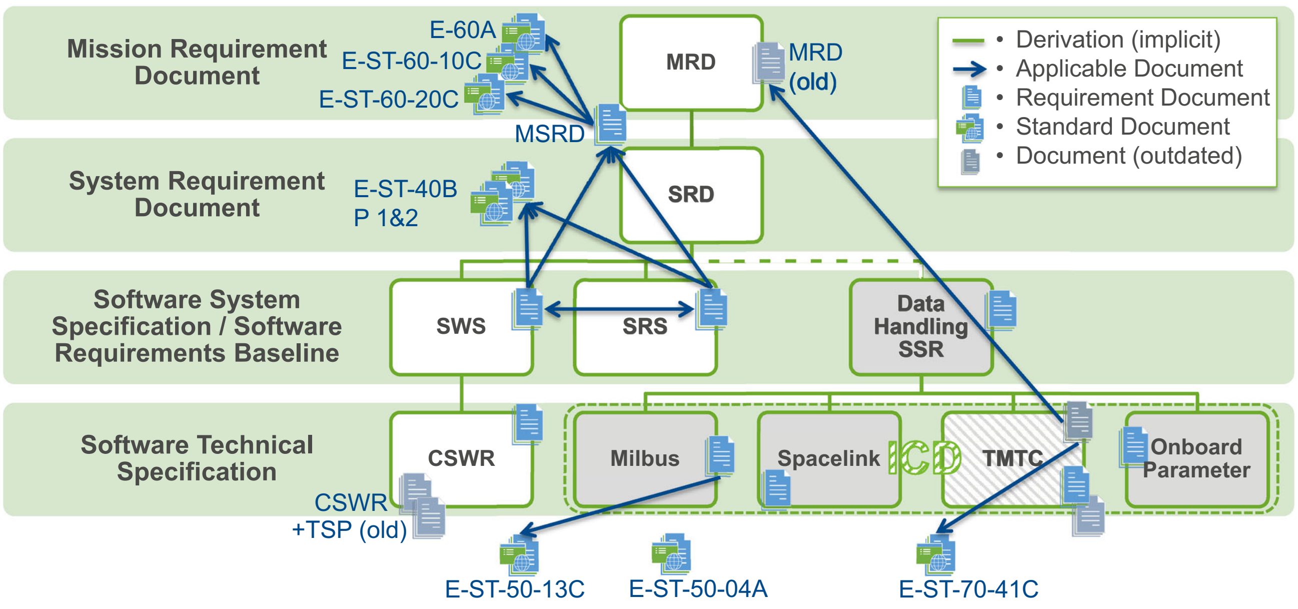


Fig. 6 Applicable document relations of EagleEye (excerpt)

ual requirements. Yet, they can also appear between sets of requirements or requirements documents.

Example The different requirements documents on different levels of abstraction along different project phases, as shown for EagleEye in Figs. 5 and 6, *refine* each other. For example, the *Software Requirements Specification* (SRS) [51] and *Software Specification* (SWS) [52] refine the *Mission and System Requirements Document* (MSRD) [50]. While inside the MSRD the *System Requirements* refine the *Mission Requirements*. SWS and SRS both follow the guidelines given by E-ST-40B Part 1&2 [57,58] and therefore *satisfy* this document. Currently, such relations are not recorded on document level within EagleEye.

4.3 Document reuse

Reuse is becoming an integral part of product development [21], e.g., through product line engineering [32,80] or standardization. Besides achieving more cost-effective design and development by a cut-down on development resources

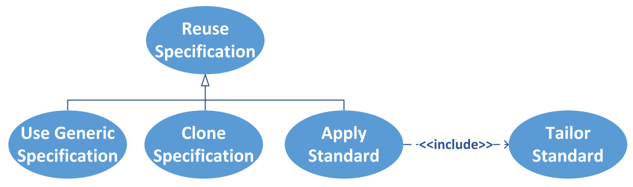


Fig. 7 Requirement document reuse scenarios

and time [33], reuse is expected to lead to improved quality and safety by applying proven requirements and methods [43]. Yet, reused requirements have to be carefully checked for consistency with the new context—thus, correct *integration* [21]. To solve conflicts, developers rely on backward traceability to get to the sources of requirements [3]. To support this, we need to know RQ-I "*Which types of relations are relevant for requirement reuse from documents?*" Fig. 7 shows three cases of document-based requirement reuse we identified from analyzing several space engineering projects.

4.3.1 Clone specification

An extensive type of reuse is to clone a complete specification document from a similar previous project. Though some provisions in standard procedures might prohibit such practice, this seems to be rather commonplace in the real world [22,33,40,81]. Reasons to do so are manifold. In the first instance, cloning existing documents instead of compiling new ones from scratch saves a lot of time in setting up the new project [33]. Additionally, for example, well-established requirements and lessons learned are automatically conveyed to the actual project, although they are not intentionally generalized.

Yet, it can lead to serious problems. For example, the findings about the reasons for the disastrous Ariane 5 failure of 1996 [82] indicate that a proven specification and design were reused without taking into account all relevant conditions of the new mission that differed from the prior ones. It is a common problem that previous requirements are "highly depend[e]nt on the context of their project and on the engineers that created them" [40]. Of course, copying the documents is always only the first step. Identifiers and links to other documents have to be created or updated. The resulting document requires time-consuming consolidation and reviews to make sure that no leftovers from the source introduce problems or inconsistencies. Also new boundary conditions introduced by the changed context might have an impact on the specification. Thus, the specialized parts need to be adapted and integrated to the new project. Given that cloning is considered a viable reuse approach, the indispensable consolidation should be supported, e.g., by automated checks for consistency. This is considered the most complex, because less formalized, of the three reuse cases.

4.3.2 Apply standard

Standards are a well-known method to provide requirements for a specific application domain in a reusable way, as can be seen in EagleEye. They aim to facilitate clear and unambiguous communication between all stakeholders, and are suitable for reference or quotation in legally binding documents, as they are clearly identified by their released versions. Furthermore, they are expected to reduce risk and to guarantee interoperability and interface compatibility [43]. A standardization body of high reputation presupposed, standards are considered to provide requirements of high quality.

The ECSS standards [43] impose functional requirements, but also contain management requirements describing what has to be covered by the project specification from the viewpoint of the specific engineering domain, and which form these requirements have to take. Furthermore, they contain *Document Requirement Definitions* (DRDs) which define the structure of the respective specification documents. Dependent on project characteristics such as cost, schedule, and technical drivers, identified risks, and the covered technical domain, applicable standards are selected. Requirements from standards have to be *tailored* for the specific project. Tailoring means, for example, to decide for each requirement if it is applicable for the project or not and whether it has to be modified—thus, specialization of the abstract document. We describe this in more detail in Sect. 6. In ECSS, this process is formalized to some extend and therefore suits as a reference case to support reuse by formalized relation dependencies. We build our approach on this and research different approaches how to integrate these requirements with other project requirements, as described in Sect. 6.3. Standards can be seen as an approach to reuse requirements based on patterns, like described by Palomares et al. [22], where the whole set of standards constitutes a pattern catalogue. Currently a new ECSS requirements management system (E-RMS) [83] is under development[18] to enhance accessibility of this catalogue in the future.

4.3.3 Generic specification

The goal of initiatives as, e.g., SAVOIR[19] is to provide a predefined set of reusable generic specifications. As standards, such documents contain collections of requirements usually applicable in a specific domain. They, too, are designed to standardize and therefore increase interoperability and reduce complexity. Thus, the focus lies on

[18] The E-RMS project was officially kick-offed in October 2018 (https://indico.esa.int/event/263, visited on 16/11/2020).

[19] "Space AVionics Open Interface aRchitecture is an initiative to improve the way that the European Space community builds avionics sub-systems." (http://savoir.estec.esa.int, visited on 11/05/2021)

providing predefined building blocks to increase efficiency, cost-effectiveness and flexibility [84], while standards rather aim at providing guidelines instead of concrete components.

In contrast to requirements from standards, these generic requirements are already phrased in the same style as project requirements, but have to be instantiated for a specific project. Yet, this specialization process has strong similarities with the tailoring process for standards, as these requirements also have to be assessed for applicability and potentially necessary modifications. Furthermore, generic requirements more likely contain placeholders that have to be assigned with specific values during specialization. As standards, the generic documents can reference other documents introducing additional dependencies, e.g., to standards. Similar to ECSS standards, SAVOIR requirements can also be seen as a pattern based approach to reuse requirements [22]. With their placeholders and product oriented writing style they resemble even more closely the patterns as presented by Renault et al. [40]. As the general dependencies are similar to the tailoring of ECSS standards, we plan in future work to extend our formalized description to match to SAVOIR, too. The SAVOIR reference requirements are not yet used for EagleEye, but different generic specifications could be applied to the SRD and SRS.

5 A document-aware traceability model

In the following, the *T-Reqs traceability model* is presented. It addresses in particular the challenges introduced by RQ-I & RQ-II and is developed based on *Ontological Analysis* of documents from space projects, ECSS standards, and known traceability models, e.g., by Ramesh and Jarke [5] or Schwarz et al. [12], as well as interviews with experts (steps A–C in Fig. 2). *The Conceptual Schema Design Procedure* (CSDP) [85, p. 13] is used together with the *Object-Role Modelling* (ORM) [86,87] conceptual modeling language. ORM comprises a graphical notation and the controlled natural English *Formal ORM Language* (FORML). The graphical notation captures the main language features, merely derivations and a few constraints can only be expressed textually. Additionally, FORML is used to verbalize the semantics. We use this in the remainder of the paper to explain our model. In ORM, ObjectTypes are related via predicates in which they play a specific role, constrained by **logic expressions**—together this constitutes a *Fact-Type*. Instances of object types populate the model. A very brief overview of basic ORM graphical elements is given in “Appendix 1”.

5.1 Traceability relations

The excerpt from our conceptual model depicted in Fig. 8 shows the objectified ternary fact type “TraceableEntity$_1$ relates to TraceableEntity$_2$ via TraceabilityRelationType[20]”. For example, TraceableEntity RequirementATB-SR-EO-0580 relates to TraceableEntity RequirementEE-MR-0350 via TraceabilityRelationType Detailing for the two requirements from the refinement example in Sect. 4.1. The objectification as TraceabilityRelation, similar to the respective class by Schwarz et al. [12], allows to attach further attributes to relation instances, e.g., meta-data necessary for trace management. Such relation instances have a direction as indicated by [source] and [target] roles. This source-to-target direction is not necessarily equivalent to the *forward* or *backward* direction as described above from [62] , but depends on the relation type definition. However, navigation is possible in both directions.

Specific TraceabilityRelationTypes have to be carefully specified, as they may have dependencies to different entities and even other relations. The types listed in Sect. 4.1 constitute an *extensible seed population*, whereas custom types are mostly subtypes of these. In the following examples of type specifications are provided by populating the so-called *Traceability Relationship Type Template* (TRTT)[21], introduced by Schwarz et al. [88].

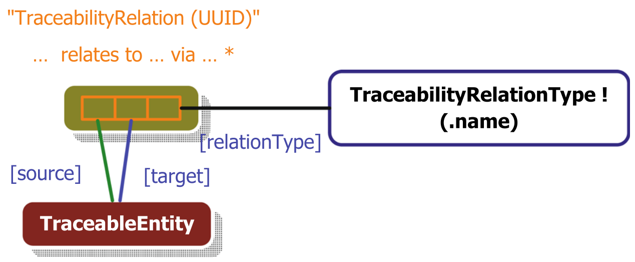

"TraceabilityRelation (UUID)"
... relates to ... via ... *
TraceabilityRelationType ! (.name)
[relationType]
[source]
[target]
TraceableEntity

Fig. 8 Typed traceability relation

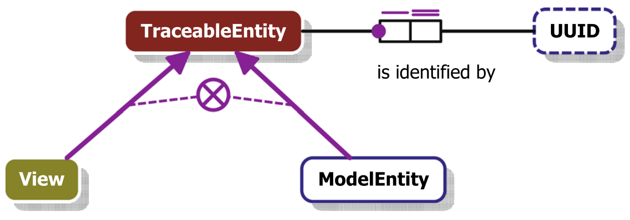


Fig. 9 Traceable entities

5.2 Traceable entities

Within the *T-Reqs* traceability model, any development artifact that can be identified through a *Universally Unique*

[20] The ! marks the object type as *independent*, meaning instances not necessarily have to play a role in some fact type.

[21] Espinoza et al. [14] define a similar *Traceability Meta Type*.

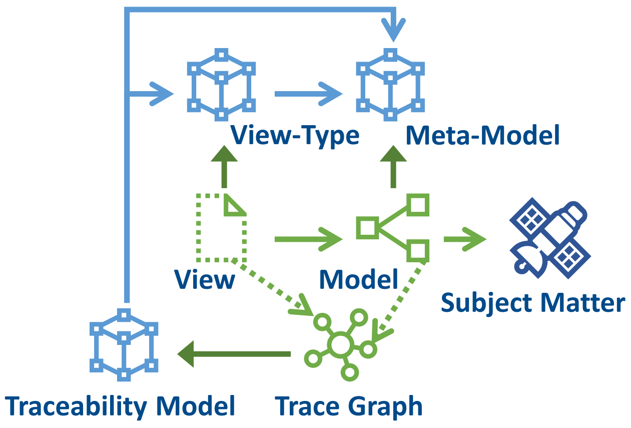


Fig. 10 Model views and their traceability

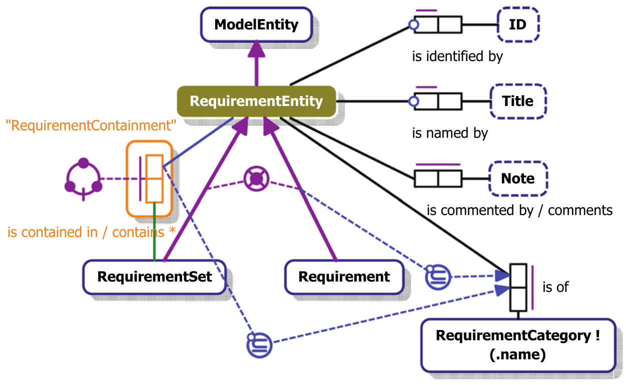


Fig. 11 Requirement entities with examples of properties

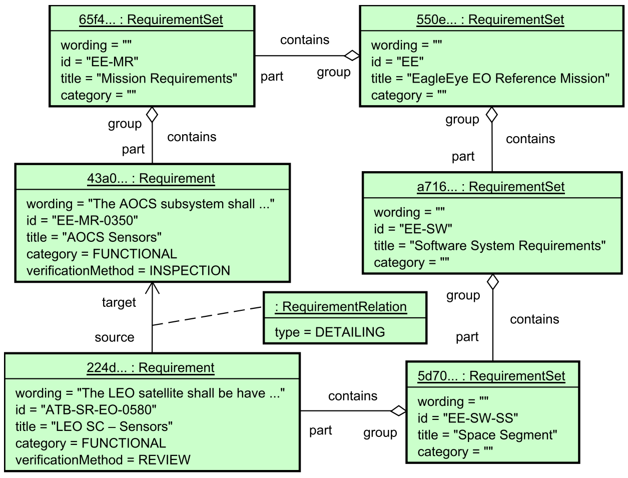


Fig. 12 Example excerpt of requirement model

Identifier (UUID)[22] can be traced as a TraceableEntity.[23] Figure 9 shows that View and ModelEntity are both modeled as sub-types of TraceableEntity, while most tracing approaches only consider the model side. The integration of meta-models is illustrated in Fig. 10. View-types and meta-model elements are linked together in the *T-Reqs* traceability model to enable the tracing of macro level view artifacts together with fine-grained model tracing. In the following models and views and their relation in this context are introduced in more detail.

5.2.1 Model entities

A ModelEntity is some model or any element part of such. It describes some subject matter[24] [73]—e.g., a real world object, development artifact, system or component under development, or related processes. The language or formalism used is defined by the model's meta-model. Models are the core artifacts in *Model-Based Software/System Development* (MBSD). Tracing within models or between different models is widely used for quality analysis. For example, to analyze security and privacy, Mouratidis and Jürjens [90] or Ahmadian et al. [91] trace requirements to design models while Peldszus et al. [92] trace design models to code. Although the *T-Reqs* traceability model is generally extensible for different kinds of models, the work presented here is focused on *requirement* models.

Figure 11 shows the conceptual model excerpt that defines a RequirementEntity as a ModelEntity and some relevant parts of the requirement meta-model used. Here, the *composite pattern* [93] is employed to express hierarchical structures in sets of requirements, similar as in the trace model by Schwarz et al. [12]. Thereby requirements can be grouped in RequirementSets on different levels. **It is possible that some** RequirementEntity is contained in **more than one** RequirementSet[25], to prevent the "tyranny of the dominant decomposition" [94].

Furthermore, requirements and sets have specific attributes, such as, exemplary depicted in Fig. 11, an ID, a Title or potentially Notes. Each requirement should belong to at least one RequirementCategory and requirements inside a RequirementSet that is of a specific category should also be of that category, as expressed through the deontic subset constraints. Further attributes, such as wording and verification method, are part of the detailed requirement meta-model, but omitted in Fig. 11 for conciseness.

[22] A *Universally Unique Identifier* (UUID) is a 128-bit identifier number specified in [89]. As they are generated locally and not by a central authority, it is possible that duplicates exist, but the probability is close enough to zero to be negligible.

[23] Thus, also TraceabilityRelations can be traced for trace management, although, we do not address this explicitly.

[24] The term *system* used by Bruneliere et al. [73] can be misleading, as there exist different context sensitive definitions of what a system shall be, especially in embedded systems engineering [33]—e.g., macro-system vs. sub-system [9]. A specification can comprise requirements not directly concerned with "the system" or its sub-systems. Therefore, the more generic *subject matter* is used.

[25] Yet, **no** RequirementEntity **may cycle back to itself**. ()

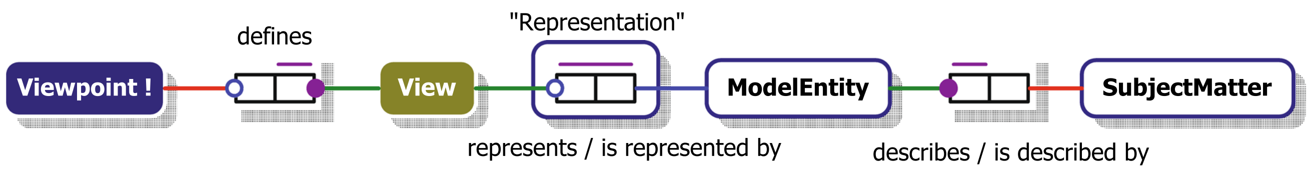


Fig. 13 Models can be represented by views

Model-specific interconnections establish traceability relations among model entities.

Example Figure 12 illustrates an excerpt of a requirement model noted as UML object diagram [95] covering the example from above with EagleEye requirements ATB-SR-EO-0580 and EE-MR-0350.

5.2.2 Views

Following Bruneliere et al. [73], a model can be "used in and referenced from a given view",[26] as illustrated in green in the middle of Fig. 10. Depending on the corresponding viewpoint and defined view-type, a view can possibly gather elements coming from one or more models. The view-type defines which elements can appear in a view, thus its meta-model. It therefore references elements from the model's meta-model, as illustrated in blue on top of Fig. 10. Figure 13 depicts the respective conceptual model excerpt: **It is obligatory that each** View represents **some** ModelEntity **that** describes **exactly one** SubjectMatter. **Each** View is defined by **exactly one** Viewpoint. **It is possible that some** View represents **more than one** ModelEntity **and that some** ModelEntity is represented by **more than one** View [7,73]. Elements may be part of the model entity's meta-model or may be specifically defined for the view-type [73]. For example, it is possibly complemented with view-specific interconnections between the model elements and additional data that are manually entered and/or computed automatically, as defined by the viewpoint [7,73]. View-types are defined sub-types of View. A given view-type can be relevant for several viewpoints, and usually a Viewpoint defines **more than one** View of potentially different view-types. The viewpoint is explicitly included in our conceptual model, as depicted in Fig. 13, to be able to relate views sharing the same such. However, so far, our tracing approach exploits only view-type definitions and viewpoint definitions are thus not elaborated. Possible extensions in this direction could potentially be based on guidelines for architectural viewpoint definitions [75].

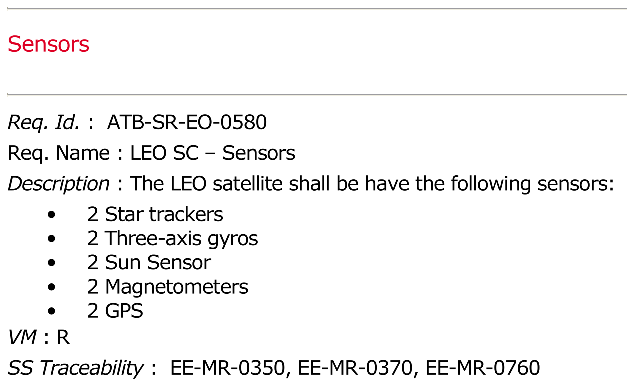
Generally, the dependencies presented here are applicable for any type of view. Yet, we focus on *document views*, to address RQ-II "*How can requirements reuse from documents be formalized into a structured process that supports integration?*" In particular, we investigate in Sect. 6 reuse processes for standards, which are usually provided in document form. As described in Sect. 4.2, this is not limited to classical document formats, but also interactive document like views, e.g., in requirements engineering tools or wikis [96]. Within the *T-Reqs* conceptual model, documents are regarded as *views* on (sets of) requirements. Some views can be composed of sub-views, e.g., documents can consist of chapters, sections, and paragraphs, or can contain diagrams. Following again the *composite pattern* [93], **for each** View, **exactly one of the following holds: that** View is **some** Container **or that** View is **some** ViewElement. **It is possible that some** Container contains **more than one** View **and**—to break "the tyranny of the dominant decomposition" [7,94]—**that some** View is contained in **more than one** Container. Document is a sub-type of Container. Specialized document types, as described by *Document Requirement Definitions* (DRDs)[27] or standards, form further (custom) sub-types of Document. Standardization manuals, as ECSS-D-00-01C [98] for ECSS or the *ESA Standardization Manual* [99], define commonalities of standards and by that part of those viewpoints and view-types. Such container view-types consist of specific sub-view-types like requirement paragraphs.

Example Figure 14 shows an excerpt of the EagleEye *Software Specification* (SWS) [52] representing the requirement ATB-SR-EO-0580 from the example above. The paragraph in this document is a view on this requirement. While the graphical model representation shown in Fig. 12 also gives some view on the used requirements model, it is not adequate for all requirements engineering viewpoints. For example, the textual representation in Fig. 14 is more accessible for non-technical stakeholders. Besides the textual listing of the requirement's attribute values of ID, title, wording,[28] and verification method, on the bottom there is also a representation of some of its relations and related requirements. Yet, in this specific view, there is no representation of the relation type. Related other requirements are only represented by their ID. The same requirements are represented in detail in another view—the *Mission and System Requirements Doc-*

[26] There exist different other appropriations of the terms *view*, *viewpoint*, and *perspective*, c.f. Sect. 4.2 or [7].

[27] For example, in ECSS-E-ST-40C [97].

[28] Depending on the underlying base model, this wording can also be interpreted as a set of five requirements—one for each bullet point—which are aggregated in this view.



Sensors

Req. Id. : ATB-SR-EO-0580

Req. Name : LEO SC – Sensors

Description : The LEO satellite shall be have the following sensors:

- 2 Star trackers
- 2 Three-axis gyros
- 2 Sun Sensor
- 2 Magnetometers
- 2 GPS

VM : R

SS Traceability : EE-MR-0350, EE-MR-0370, EE-MR-0760

Fig. 14 Example excerpt of requirement document

ument (MSRD) [50]. The requirement ATB-SR-EO-0580 is listed together with other requirements under the section heading "Sensors". This grouping may imply a related requirement set in the model, as we assumed for our case study presented in Sect. 7.

Grouping of model entities represented in a view not necessarily has to, but often reflects sets within the model. However, it may be only one among several non-disjoint set affiliations of the involved model entities or views may aggregate in a way only relevant in the specific view context. In particular, each view inherently displays some ordering, while this might not be identical to the ordering within the models data structure or not even relevant to model semantics.

Views are treated in the *T-Reqs* traceability model as "first-class citizen" with relations on their own. This is important, as views carry additional information through the way they aggregate or complement model elements. Furthermore, as workflow artifacts views constitute points of reference for involved stakeholders. As illustrated in Fig. 10, the traceability model integrates view-types and meta-models to trace among model entities as well as views and enable typed relations among all these entities.

Views can be manually designed or automatically derived from the model, as demonstrated among others by Silva and Leite [7]. We do not distinguish that explicitly, as we focus on the traceability of views after they came to existence. Therefore, we partly exploit the same relations between model and view elements, necessary for view generation or transformation, but also include explicit relations between views. For example, *refinement*, as explained in more detail in the next section.

5.3 Levels of abstraction

According to Li et al. [77], current requirements engineering tools, as DOORS, pay little attention to requirement refinement. Requirements can be of *different abstraction or level of detail* [77,100], as it is implied by a chain of refinement relations. As these levels are blurred, there exist different categorizations, e.g., [100–102]. More levels that are more detailed can be elicited within the development team or by the supplier. Thereby, the specification tree depends on the business agreement structure [44], leading to very divers results. The *T-Reqs* model abstracts from such concrete view levels and focuses on the reoccurring relationship between two consecutive levels, as seen in Fig. 3. Here, the customer pushes the higher-level *Project Requirements Document* (PRD) to the supplier.[29] The corresponding *Implementation Document* (ID) contains a refined set of requirements that is the basis of the business agreement. The TRTT in Table 1 summarizes this relation also depicted in Fig. 15. This does not include refinement through *evolution* of documents. To ensure consistency over several levels, it has to be made sure that all necessary higher-level requirements are included in the project-requirements Document and completely considered in the implementation Document. This includes all relevant requirements the customer received from her/his customer–the next higher-level implementation document, or at least the parts relevant for the lower level. This can be achieved through diverse refinements between requirement entities on these different levels. For example, Li et al. [77] describe such more detailed refinement relations and processes for their identification. Thus, the refinements between individual requirements depend on the document refinements. These inter-relations among artifacts of different granularity are covered in the following section.

5.4 Representation layers

According to Winkler and Pilgrim [69], "it has to be acknowledged that software development artifacts exist in very different forms of representation, and that prior to recording fine-grained traceability links, it should be recorded how artifacts themselves are created, evolved, and transformed at a macro level." This is also true for requirements documents as macro level representations of requirements, as already described by Gotel and Finkelstein [6]. Figure 16 illustrates this concept of *"layered traceability"*. The representation layer hierarchy is orthogonal to the hierarchy established by refinement relations, as described in Sect. 5.3 above.

There are relations, refinement among them, between artifacts on each layer of representation granularity. Figure 16 illustrates this with relations in different shades of green. For example, views can be related to other views, illustrated by documents D_1 and D_2 in Fig. 16. Besides document

[29] Additional and more detailed *contribution* relations [6] can be defined for the involved documents and all their parts. For example, the customer should be a motivating *principal* for the implementation document.

Table 1 TRTT for the DocumentRefinement relation

Property	Value
Name	DocumentRefinement
Description	Project-requirements Document is refined by implementation Document
Super-types	Refinement
Meta-model fragment	See Fig. 15
Attributes	[customer] Agent and [supplier] Agent, both *mandatory*
Constraints	**For each** implementation Document, **at most one** project-requirements Document is refined **by that** implementation Document. Relation is *acyclic* and *strongly intransitive* : **No** Document **may cycle back to itself via one or more traversals through** project-requirements Document is refined by implementation Document. **If** project-requirements Document_1 is refined by **some** implementation Document_2 **then it is not true that** Document_1 **is indirectly related to** Document_2 **by repeatedly applying this fact type.**
Impact designators	(not considered in this paper)
Examples	EagleEye requirements documents on the different levels presented in Fig. 4, 5, and 6. DocumentRefinement should be used instead of applicable or reference document links or implicit, immaterial refinement

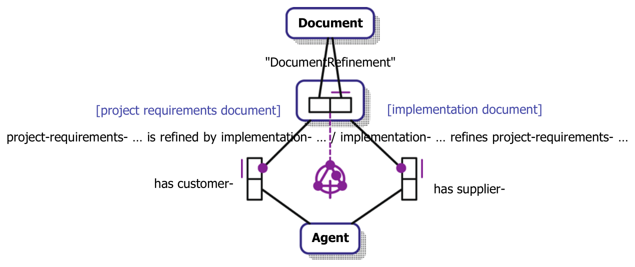

Document
"DocumentRefinement"
[project requirements document]
[implementation document]
project-requirements- ... is refined by implementation- ... / implementation- ... refines project-requirements- ...
has customer-
has supplier-
Agent

Fig. 15 Document refinement

refinements, as discussed in the previous section, relations as discussed in Sect. 4.2 or any other view-type specific relation type is possible. These relations describe the macro-level dependencies of these artifacts without disclosure of the details of represented model entities. Figures 4, 5, and 6 illustrate inter-document relations for EagleEye. In the same way, there can be relations between different more coarse grained model entities, such as requirement sets, illustrated in Fig. 16 by RS_3 and RS_5 or RS_1 and RS_2. Finally, also fine-grained model entities such as different requirements can be related

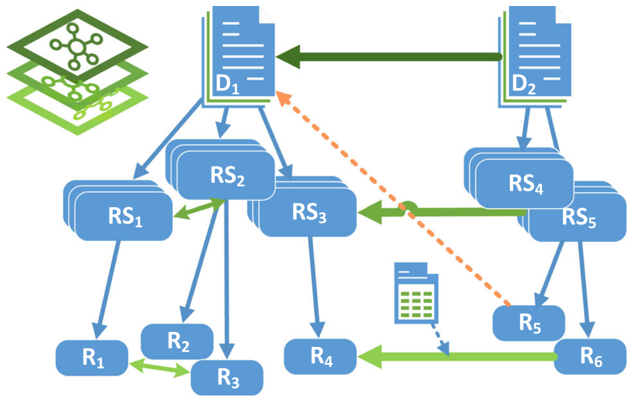


Fig. 16 Relations on different representation layers

among each other, as discussed in detail also with examples from EagleEye in Sect. 4.1. In Fig. 16, this is illustrated by R_1 and R_3 or R_4 and R_6. The hierarchical relations implied by containment and representation, of either views or models, bridge between the layers. These are depicted in blue in Fig. 16. Sub-views and nested sets can constitute additional intermediate layers, left out in the illustrating figure for conciseness.

Different stakeholders work with representations of artifacts and relations of different granularity. For example, a project manager needs to know that certain standards are applied, but is at that time not interested in the individual requirements. By masking details, the complexity can be reduced for such perspectives and specific views are created—here, e.g., a *specification tree*. Although it is possible to define and visualize relations explicitly on all layers independently, obviously they impose constraints on each other. Gotel and Finkelstein [6] showed that for contribution relations. Such constraints play a central role for consistent integration of (reused) requirements documents. In the following, we give examples for different types of dependencies between relations on different layers.

5.4.1 Derivation between layers

Some relations can be derived from relations that are registered on other layers. For example, *conflicts* between requirements arise from their content. In the same way, two requirement sets are in conflict with each other if their content, thus one or more of the contained requirements, are in conflict. Similarly, two requirement documents are inconsistent with each other, if the requirements entities they represent are in conflict.

The TRTT of RequirementSetConflict in Table 2 shows in constraint 2. the derivation rule for the semi-derived fact type RequirementSet conflicts with RequirementSet, depicted on the right in Fig. 17. The RequirementEntity is contained in RequirementSet fact type used in this rule is depicted in Fig. 11. It represents the transitive closure of the contain-

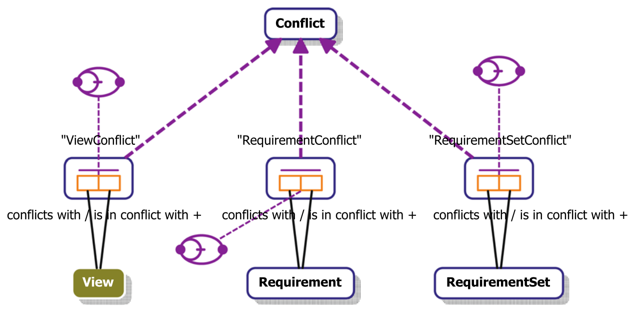

Conflict
"ViewConflict"
"RequirementConflict"
"RequirementSetConflict"
conflicts with / is in conflict with +
conflicts with / is in conflict with +
conflicts with / is in conflict with +
View
Requirement
RequirementSet

Fig. 17 Different types of conflict on different layers

Table 2 TRTT for the RequirementSetConflict relation

Property	Value
Name	RequirementSetConflict
Description	A RequirementSetConflict links two conflicting RequirementSets
Super-types	Conflict
Meta-model fragment	See Fig. 17
Attributes	(not further considered here, i.a., state as *solved*)
Constraints	1. Relation is *symmetric* and *irreflexive* : **If** RequirementSet$_1$ conflicts with RequirementSet$_2$ **then** RequirementSet$_2$ conflicts with RequirementSet$_1$. **No** RequirementSet conflicts with **the same** RequirementSet. 2. Relation is *semi-derived* (+): either asserted by a reviewer, or derived automatically dependent on the given derivation rule: +RequirementSet$_1$ conflicts with RequirementSet$_2$ **if** **that** RequirementSet$_1$ contains **some** RequirementEntity$_1$ **that is some** Requirement$_1$ **that** conflicts with **some** Requirement$_2$ **that is some** RequirementEntity$_2$ **that** is contained in **that** RequirementSet$_2$ **or that** RequirementSet$_1$ is in conflict with **that** RequirementSet$_2$.
Impact designators	(not considered here)
Examples	A set of security requirements in conflict with a set of performance requirements, because of a conflict between two individual requirements from each set.

ment hierarchy. The last inclusive **or** derives the symmetric counterparts, so the relation only needs to be asserted in one direction. Conflicts between sets that do not arise from conflicts between contained requirements, but from other properties of the sets, can be asserted in addition to the rule, as it is semi-derived (**+**). The derivation of ViewConflicts works analogous.

5.4.2 Consistency between layers

Other types of relations require consistency between relations that are asserted on different layers. For example, *refinement*. Similar to *conflicts*, refinement can be derived from refinement relations between contained or represented entities on lower layers. Yet, in contrast, relations asserted on higher layers are not independent, but imply completeness toward the refinement of contained entities, as described in Sect. 5.3. There is a need to check, if the relations defined on different layers are consistent with each other. As described in Sect. 4.1, this depends on the type of refinement.

An example for refinement relations on different layers from EagleEye: Before the requirements of the Software Specification (SWS) [52] are actually written, it is already defined that the SWS refines the Mission and System Specification (MSRD) [50], as currently indicated by listing the MSRD as *applicable document*[30] within the SWS [52]. In Sect. 4.1, two examples for a *refinement* relation between individual requirements are presented: The System Specification requirement EE-MR-0350 is *detailed* by the Software Specification requirement ATB-SR-EO-0580 and ATB-SR-EO-0580 *specializes* the System Specification requirement EE-MR-0370. The requirements are listed in a traceability matrix mapping all requirements represented in the SWS [52] with all from the MSRD [50]. However, it remains undocumented for these explicit links within the original EagleEye documents, as well on requirements level as on documents level, of what type the relation is and if the combination of all refinements between individual requirements is consistent or complete toward the intended refinement between the documents. This type of analysis is targeted with the presented approach.

5.4.3 Crossing layers

As illustrated in Fig. 16 by the orange dashed arrow between R_5 and D_1, some relation types can cross the layer hierarchy directly, not only implicitly by derivation. One of those is the *Satisfy* relation, where RequirementEntity can be satisfied by a Document. For example, when the document follows the requirements of a *Document Requirement Definition* (DRD). ECSS-E-ST-10-06C [78] requires for *Technical Requirements Specifications* (TS) *"The technical requirements shall be grouped."* and *"The specification shall be identifiable, referable and related to a product or a system."* A given technical specification Document can satisfy these requirements. In a similar way, the mere existence of another Document, RequirementEntity or its properties can satisfy another RequirementEntity, if the existence of specific content is

[30] As discussed in Sect. 4.2, an explicit *refines* relation would be more favorable from the point of view of traceability analysis.

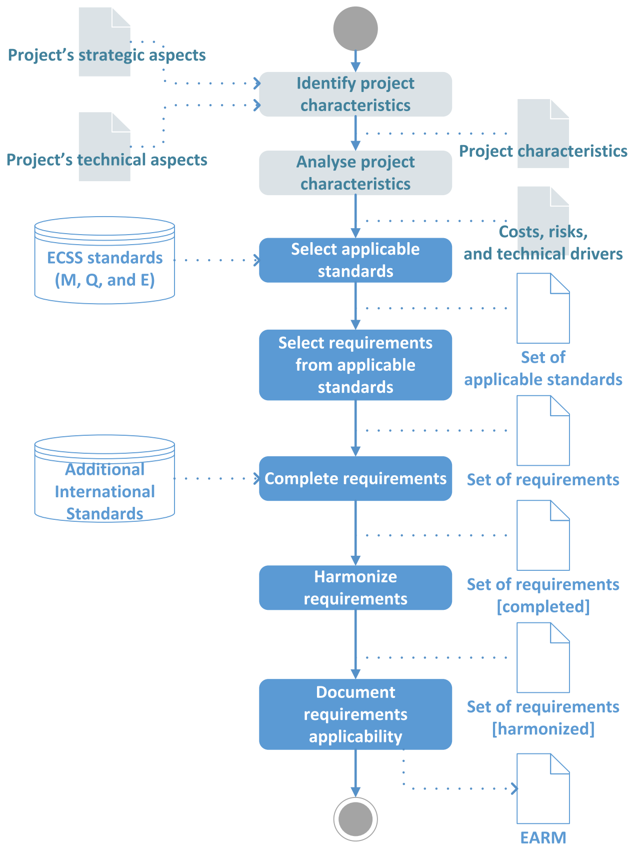
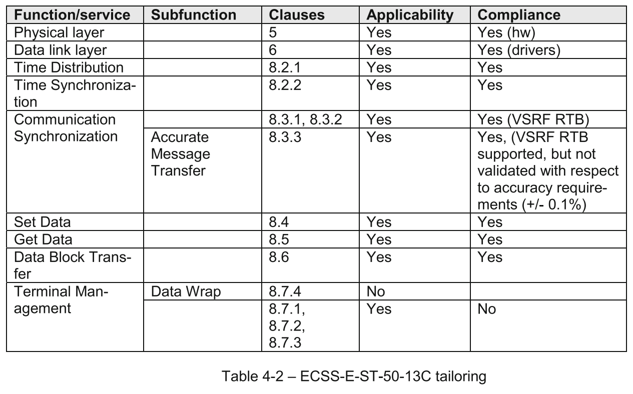

Project's strategic aspects
Project's technical aspects
Identify project characteristics
Analyse project characteristics
Project characteristics
ECSS standards (M, Q, and E)
Select applicable standards
Costs, risks, and technical drivers
Select requirements from applicable standards
Set of applicable standards
Additional International Standards
Complete requirements
Set of requirements
Harmonize requirements
Set of requirements [completed]
Document requirements applicability
Set of requirements [harmonized]
EARM

Fig. 18 ECSS tailoring process [43]

required, again, e.g., by DRDs or *Documents Requirements List* (DRL). ECSS-E-ST-40B Part 1 [57] requires *" [..] all software observability requirements to facilitate the software integration shall be specified by the customer."* Thus, the existence of "observability requirements" satisfies this. In the following, the kind of requirements that expresses constraints or properties to be satisfied by other requirements is called *meta-requirement* and, if targeting views, *view requirement* respectively.

6 Standard tailoring

We use the before introduced general concepts of our *T-Reqs* traceability model to define a more specific reference model to address RQ-II "*How can requirements reuse from documents be formalized into a structured process that supports integration?*". In particular, we concentrate on the application of standards, as there already exist some processes and guidelines, and dependencies are observed to be similar also for generic specifications (c.f. Sect. 4.3). If a standard is applicable to a project, it has to be specialized and integrated with the project requirements. In case of ECSS [43] this process,



Function/service	Subfunction	Clauses	Applicability	Compliance
Physical layer		5	Yes	Yes (hw)
Data link layer		6	Yes	Yes (drivers)
Time Distribution		8.2.1	Yes	Yes
Time Synchroniza-tion		8.2.2	Yes	Yes
Communication Synchronization		8.3.1, 8.3.2	Yes	Yes (VSRF RTB)
	Accurate Message Transfer	8.3.3	Yes	Yes, (VSRF RTB supported, but not validated with respect to accuracy require-ments (+/- 0.1%)
Set Data		8.4	Yes	Yes
Get Data		8.5	Yes	Yes
Data Block Trans-fer		8.6	Yes	Yes
Terminal Man-agement	Data Wrap	8.7.4	No	
		8.7.1, 8.7.2, 8.7.3	Yes	No

Table 4-2 – ECSS-E-ST-50-13C tailoring

Fig. 19 Tailoring of ECSS-50-13C [59] for the EagleEye Central Software Requirements (CSWR) [104]

depicted as BPMN [103] diagram in Fig. 18, is called *tailoring* and consists of these main steps:[31]

1. The extent to which the standard is made applicable is assessed against cost, schedule, and technical drivers, as well as the identified risks and their mitigation strategies. Each requirement is classified as: *(Y) Applicable without change*, *(M) Applicable with modification*, or *(N) Not applicable* (omitted).
   References to other standards may imply iterated additional tailoring. (Steps 1–4 in Fig. 18)
2. When a deficiency is identified in the standard that is not addressed by project specific requirements, it has to be complemented. Therefore, new *additional requirements (A)* are generated or adopted, preferably from a standard of another standardization organization.
3. The requirements have to be reviewed and harmonized to ensure the coherence and consistency of the overall set of requirements before they are applied to the project.
4. The results of this process are documented in a so-called *ECSS Applicability Requirements Matrix* (EARM), where all requirements are listed with their classification, for (M) and (A) with complete requirement wording and—as well as for (N)—justification. Figure 19 shows the EARM from the outdated version of EagleEye's Central Software Requirements (CSWR) [104] that tailors ECSS-50-13C [59]. By referencing whole clauses, several requirements are made applicable or omitted as a set.

[31] The ECSS tailoring process serves as an example. Other standardization bodies may define slightly different processes to adapt their standards or use other terminology. Yet, the main steps, as defined for ECSS, of assessing the applicability and possibly necessary modifications, harmonization, and documentation are expected to be similar and generally applicable.

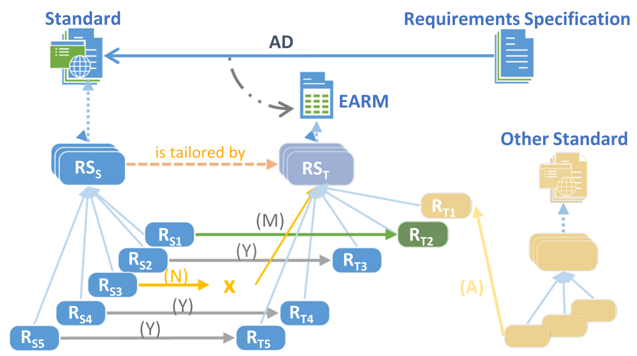


Fig. 20 Tailoring requirements dependencies

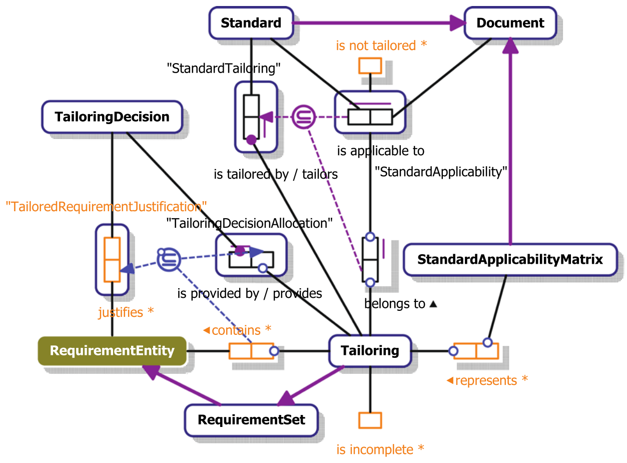


Fig. 21 Standard tailoring

6.1 Tailoring decision

With this process the reuse of requirements from standards is somewhat formalized and their traceability established. Figure 20 illustrates how this process leads to a new set of requirements—the tailoring RS_T of the standard's requirements RS_S. RS_T *depends on* RS_S as well as each of the tailored requirements (R_{T2-5}) depends on its counterpart standard requirement. These relations obviously differ in semantics: Whereas (Y) dependencies lead to copies or pointer objects, (M) dependencies link to new derived requirements. (A) Dependencies can either be copies or refinements and link to other sets.

Figure 21 depicts the respective conceptual model excerpt for a Standard being applicable to some other requirements Document. Table 3 summarizes the TRTT for the StandardApplicability relation linking the two documents, as depicted in the upper center of the model excerpt. **If some** Standard is involved in **some** StandardApplicability **where some** Tailoring belongs to **that** StandardApplicability **then that** Standard is tailored by **that** Tailoring. As expressed by the sub-set ⊆ constraint. **Each** Tailoring is an instance of RequirementSet. The deontic sub-set constraint ⊆ on the left checks, if all requirements contained in a tailoring are justified backward by an explicit decision in the tailoring:

Constraint 1 *(All Tailored Requirements Justified)*

It is obligatory that if some Tailoring contains **some** RequirementEntity **then some** TailoringDecision justifies **that** RequirementEntity
and is provided by **that** Tailoring.

The involved TailoringDecisionAllocation and TailoredRequirementJustification are thus sub-types of Rationalization, while StandardTailoring is a Refinement of standards. The derived fact type is not tailored at the top of Fig. 21 identifies StandardApplicabilitys without a tailoring at all (post-RS). The derived fact type depicted at the bottom can check the completeness of a Tailoring with respect to the Standard that is tailored:

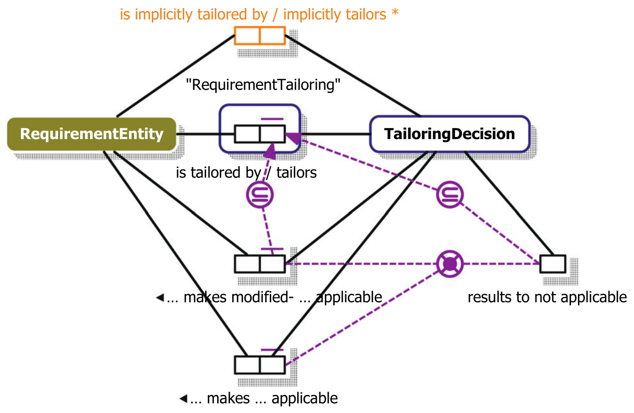

is implicitly tailored by / implicitly tailors *
"RequirementTailoring"
RequirementEntity
TailoringDecision
is tailored by / tailors
◂... makes modified- ... applicable
results to not applicable
◂... makes ... applicable

Fig. 22 Tailoring decision

Derivation 1 *(Tailoring Completeness)*

*Tailoring is incomplete **if and only if**
that Tailoring tailors **some** Standard **that** represents
some RequirementEntity
 that is tailored by **no** $\text{TailoringDecision}_1$ **that** is provided by
 that Tailoring
 or that RequirementEntity is implicitly tailored by **no**
 $\text{TailoringDecision}_2$ **that** is provided by **that** Tailoring.

Completeness means here that all requirements from the standard are addressed by a tailoring decision (post-RS). The semantic correctness of the decisions and the completeness with respect to potential deficiencies of the standard, which make additional requirements necessary, cannot be checked without additional context knowledge. Thus, only some *incompleteness* can be detected. Completeness cannot be asserted.

A TailoringDecision justifies RequirementEntity **if and only if that** TailoringDecision makes **that** RequirementEntity applicable (Y) **or** makes **that** modified Require-

Table 3 TRTT for the StandardApplicability relation

Property	Value
Name	StandardApplicability
Description	Standard is applicable to Document
Super-types	Dependency
Meta-model fragment	See Fig. 21
Attributes	`Boolean` is not tailored, Tailoring (obligatory)
Constraints	1. **It is possible that some** Standard is applicable to **more than one** Document **and that for some** Document, **more than one** Standard is applicable to **that** Document. **In each population of** Standard is applicable to Document, **each** Standard, Document **combination** 2. **occurs at most once.**
Impact designators	(not considered here)
Examples	Links between requirements documents and standards in Figs. 5 and 6. For example, ECSS-E-ST-50-13C is applicable to the *Central Software Requirements* document. The related tailoring documentation in Fig. 19.

mentEntity applicable (M) [...][32]. If a requirement is omitted (N), **that** TailoringDecision results to not applicable. Figure 22 shows the three different mandatory disjoint ⊗ options. Additional (A) requirements can either come from other source requirements or be completely new. In this case the decision tailors no requirement, but only makes one applicable. The two subset constraints ⊆ ensure that this is not possible for modifications or omissions. A decision can also be asserted for a set of requirements, as also seen in the example in Fig. 19. This is covered by the derived fact-type is implicitly tailored by on top of Fig. 22. Each decision option has its own constraints, which need to be enforced or checked for consistency. For example, **it is obligatory that** for all decisions other than (Y) **that** TailoringDecision is justified by **some** Justification, as exemplary shown for modifications (M) in Fig. 23 through the deontic subset constraint ⊆. Furthermore, a modified requirement always refines the original requirement tailored from the standard to establish backward traceability, expressed through the upper right ⊆ subset constraint. The second subset constraint on the left ensures that the modified requirement is contained in the tailoring.

[32] or makes some RequirementSet applicable that contains that RequirementEntity

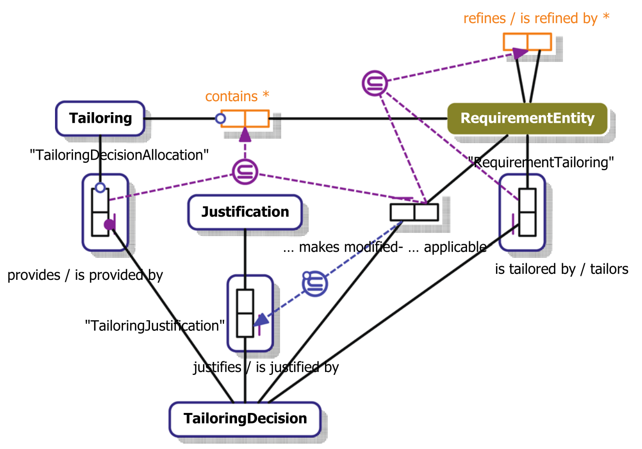

refines / is refined by *
contains *
Tailoring
RequirementEntity
"TailoringDecisionAllocation"
"RequirementTailoring"
Justification
... makes modified- ... applicable
provides / is provided by
is tailored by / tailors
"TailoringJustification"
justifies / is justified by
TailoringDecision

Fig. 23 Applicable with modifications (M)

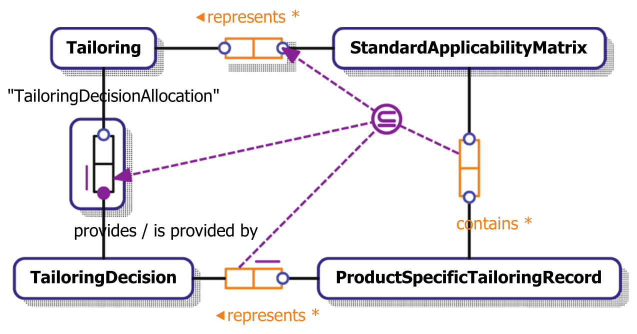

◂represents *
Tailoring
StandardApplicabilityMatrix
"TailoringDecisionAllocation"
provides / is provided by
contains *
TailoringDecision
ProductSpecificTailoringRecord
◂represents *

Fig. 24 Tailoring decision documentation

Such checks toward *justification* and *completeness* of tailored requirements enable automated analysis to support requirement engineers during tailoring or in reviews. Deficiencies can be reported without the necessity of manual comparison of tailoring documentations such as shown in Fig. 19 with the full list of requirements of the standard, in this example ECSS-E-ST-50-13C [59].

Specific view-types can represent Tailoring and TailoringDecision, like the already mentioned *ECSS Applicability Requirements Matrix* (EARM) with an example in Fig. 19. We generalized this view-type to StandardApplicabilityMatrix in our model as depicted in Fig. 24. The representation of an individual decision in a ProductSpecificTailoringRecord is a sub-view within the matrix Container—e.g., a table row in Fig. 19. The subset constraint ensures that the view containment structure is consistent with the model relations.

6.2 Tailoring recommendation

Some standards, ECSS-50-13C [59] among them, also provide a *Tailoring Recommendation* on how to tailor this standard consistently. Figure 25 shows this for ECSS-50-13C. It can be seen that the actual tailoring of this standard as shown in Fig. 19 is inconsistent with this recommendation for Clause 8.7.4.

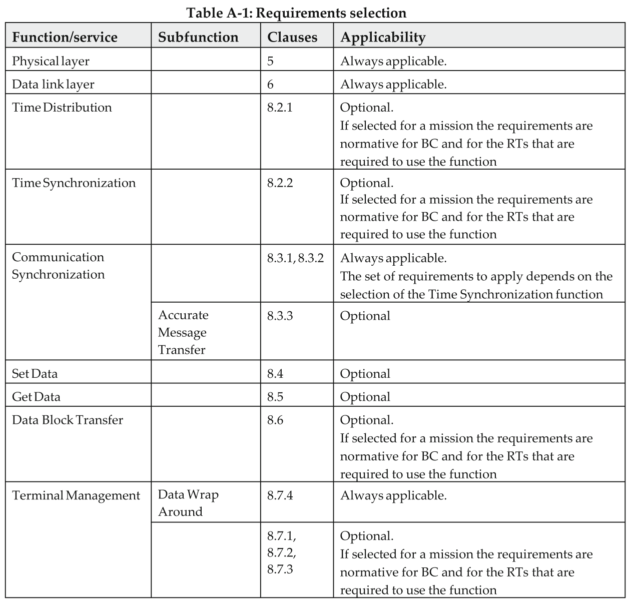


Table A-1: Requirements selection

Function/service	Subfunction	Clauses	Applicability
Physical layer		5	Always applicable.
Data link layer		6	Always applicable.
Time Distribution		8.2.1	Optional. If selected for a mission the requirements are normative for BC and for the RTs that are required to use the function
Time Synchronization		8.2.2	Optional. If selected for a mission the requirements are normative for BC and for the RTs that are required to use the function
Communication Synchronization		8.3.1, 8.3.2	Always applicable. The set of requirements to apply depends on the selection of the Time Synchronization function
	Accurate Message Transfer	8.3.3	Optional
Set Data		8.4	Optional
Get Data		8.5	Optional
Data Block Transfer		8.6	Optional. If selected for a mission the requirements are normative for BC and for the RTs that are required to use the function
Terminal Management	Data Wrap Around	8.7.4	Always applicable.
		8.7.1, 8.7.2, 8.7.3	Optional. If selected for a mission the requirements are normative for BC and for the RTs that are required to use the function

Fig. 25 Tailoring recommendation for ECSS-50-13C [59]

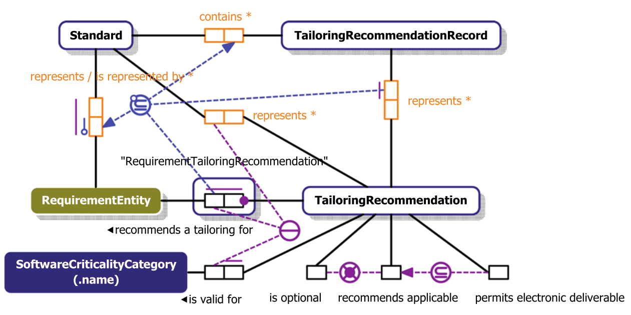

contains *
Standard
TailoringRecommendationRecord
represents / is represented by *
represents *
represents *
"RequirementTailoringRecommendation"
RequirementEntity
TailoringRecommendation
◂recommends a tailoring for
SoftwareCriticalityCategory (.name)
◂is valid for
is optional
recommends applicable
permits electronic deliverable

Fig. 26 Tailoring recommendation

Figure 26 shows the conceptual model excerpt for a TailoringRecommendation. There can be different recommendations for a given RequirementEntity, dependent on project properties, as, e.g., the SoftwareCriticalityCategory for the ECSS-E-ST-40C [97]. Yet, only one per category, enforced by the external uniqueness constraint ⊖. The options—recommends applicable or is optional—are modeled as exclusive unary fact types at the bottom right of Fig. 26. In some domains, where standards define specific process deliverables, a subset of recommends applicable decisions permits electronic deliverable, dependent, e.g., on the criticality level. The upper part of Fig. 26 depicts the representation of recommendations in TailoringRecommendationRecord views and that these should be part of the respective Standard document.

Figure 27 shows the specific conflict type TailoringRecommendationViolation we find for ECSS-50-13C in EagleEye, and which can be detected by the following derivation rule:

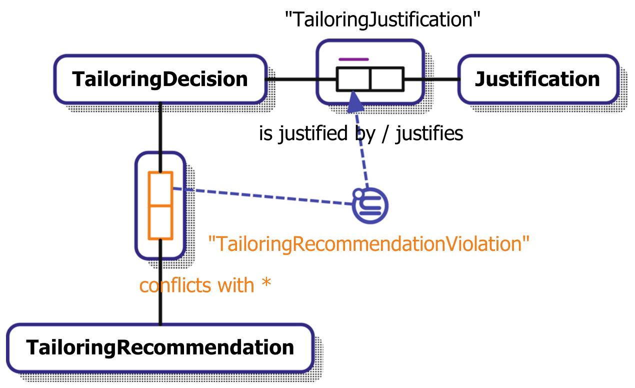


Fig. 27 Tailoring recommendation violation

Derivation 2 *(Tailoring Recommendation Violation)*

*****TailoringDecision conflicts with
TailoringRecommendation **if and only if**
that TailoringDecision tailors **some** RequirementEntity
  where that TailoringRecommendation recommends
  a tailoring for **that** RequirementEntity
  and recommends applicable
  and that TailoringDecision results to not applicable
or that TailoringDecision implicitly tailors **some**
RequirementEntity
  where that TailoringRecommendation recommends
  a tailoring for **that** RequirementEntity
  and recommends applicable
  and that TailoringDecision results to not applicable**.**

[…or the same for implicit recommendations …].[33]

If a conflict with the recommendation is detected but the deviation is intended, the decision should be justified by some Justification ⊜.

More complex recommendations with dependencies or exclusions among each other could be expressed by integrated *feature models* [80]. Such dependencies are rarely within ECSS standards, but occur, e.g., within SAVOIR specifications. We plan to extend our model to address this in the future.

6.3 Integration to project

However, the ECSS tailoring as described above does not define how to apply these tailored requirements to the project requirements represented by the document that made the standard applicable. Palomares et al. [22] claim that one main barrier for reuse by pattern application is that organizations do not know how to incorporate this to their processes and have difficulties adapting pattern output to the organization's requirements specification format.

To *exploratory* [105] find out about this part of the process for ECSS tailoring we analyzed excerpts of space

[33] Properties such as the Software Criticality Category are not considered to simplify the example.

project specifications, e.g., from IXV,[34] EUCLID, EPS-SG/Metop-SG,[35]), and EagleEye. Additionally the first author conducted *semi-structured interviews* [105,106] with five Software Engineers and three System Engineers from different projects and sections at ESA. The interviews took place in *face-to-face mode* [107] and the following *main themes* and *follow-up questions* [106] where used to guide the interviews:

1. How does the typical structure for document abstraction levels look like? (E.g., Specification Tree)
2. When does the tailoring take place?
   (a) At project level? (at project initialization)
   (b) Early selection of standards at system level or later for subsystems?
   (c) Is there a central harmonization step? (standards with each other and/or project requirements)
   (d) Are there different tailorings of the same standard for different subsystems?
3. How is the tailored set related to the project requirements?
   (a) Unification? (standard requirements become part of the project requirements)
   (b) Does it remain implicit? (only in EARM)
   (c) If explicit, refined requirements or clones?
   (d) How are meta-requirements addressed?
   (e) Are tailoring recommendations followed?
   (f) Who checks conformance?
4. Is SAVOIR tailored similarly?
   (a) How to deal with SAVOIR references to ECSS standards?

The answers of the interviewees are not statistically representative due to the small sample. Yet, they showed, as expected, a different use and perception of the tailoring process.[36] All variants we considered in our guiding questions also appear in practice.

Standards are usually not tailored centrally for one project, but “progressively tailored by each customer in the customer-supplier chain to reflect the type and phase of the project covered by the business agreement, as well as the scope of the suppliers’ tasks [...].” [44]

[34] www.esa.int/Our_Activities/Space_Transportation/IXV (visited on 11/05/2019).

[35] https://www.eumetsat.int/our-satellites/metop-series (visited on 06/01/2021).

[36] This not only motivates clear defined semantics covering different use cases, as we aim at with our model, but additionally thorough training of the user community. In turn, such learning can profit from support through modeled semantics, shown for security knowledge by Schneider et al. [108].

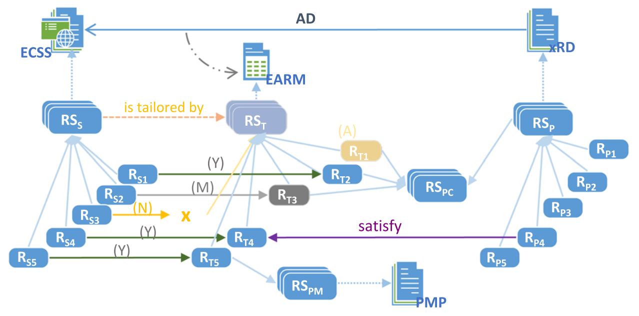


Fig. 28 Tailoring dependencies to project requirements

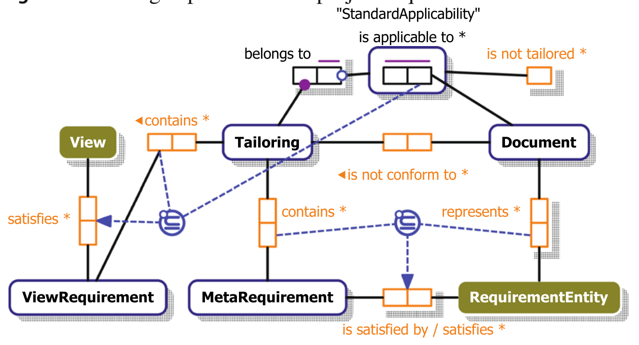


Fig. 29 Conformance checks for tailoring

Dependent on the specific (sub-) domain and the style and content of the standards relevant to it, different approaches how to combine the tailored requirements with the project requirements are common. The impact of the applicable standard to the project strongly depends on the type of requirement contained in the standard. Figure 28 extends Fig. 20 to illustrate the different possible cases:

1. Functional and non-functional requirements directly concerned with the product can be joined to a unified set with the project specific requirements (RS_{PC}) or alternatively be refined by project requirements. For example, ECSS-E-ST-60-20C [56] contains such functional requirements.
2. ECSS standards also often contain *meta-requirements* describing form or content of requirements that have to be directly satisfied by the project requirements (R_{T4} - R_{P4}). For example, in ECSS-E-ST-60-10C [55]: *“In the general case all probabilities shall be expressed as fractions or as percentages.”*
3. Project management related requirements (RS_{PM}) that have to be addressed in separate but related management documents of the project (PMP), in case project requirements document (xRD) is a technical specification. For example, the ECSS-E-ST-40 [97] contains mostly process requirements standardizing the software engineering processes. If a standard with management requirements is made applicable to a management requirements document, they become part of a unified set equivalent to functional requirements for technical specifications.

Yet finally, the combined project and standards requirements have to be reviewed for consistency and completeness. This conformance review is in many cases non-transparent, especially to un-experienced engineers. This is because at the current state most of these dependencies remain implicit and requirements are not typed in this sense. In this context, the explicit relations of these imported requirements to other project requirements are of special interest to perform automated conformance, consistency, and completeness checks. Figure 29 illustrates some possible constraints and rules to check the conformance. For example, the deontic subset constraint ⊜ on the right, which checks if meta-requirements are post-RS *satisfied* by the respective requirements of the document:

Constraint 2 *(Meta-Requirements Satisfaction)*

It is obligatory that if some Tailoring contains **some**
MetaRequirement **that** applies to requirements of **some**
RequirementCategory
and that Tailoring belongs to **some** StandardApplicability
that involves **some** Document **that** represents **some**
RequirementEntity **that** is of **that** RequirementCategory
then that MetaRequirement is satisfied by **that**
RequirementEntity**.**

As we distinguish model from view, requirements that describe properties of the view, e.g., document structure, form a separate sub-type—ViewRequirement. Their satisfaction can be checked with the subset constraint on the left. It is to note that *satisfaction* means here the existence of a respective traceability link, not how this link is established, e.g., through a linked verification process [5], manually or automated.

In combination with checks, if the remaining requirements are represented in the project document, it can be evaluated if a Document is not conform to Tailoring. The fact type is depicted in Fig. 29 and is defined by the following derivation rule:

Derivation 3 *(Tailoring Conformance)*

*Document is not conform to Tailoring **if and only if**
that Tailoring contains **some** MetaRequirement
  where some RequirementEntity$_1$ misses to satisfy **that**
  MetaRequirement
  where that Document represents **that** RequirementEntity$_1$
or that Tailoring contains **some** ViewRequirement
  where some View$_1$ misses to satisfy **that**
  ViewRequirement
    where that View$_1$ **is some** Container$_1$ **that** is **that**
    Document **or some** View$_2$ misses to satisfy **that**
    ViewRequirement
    and is contained in **some** Container$_2$ **that** is **that**
    Document
or that Tailoring contains **some** RequirementEntity$_2$
  where that Document misses to represent **that**
  RequirementEntity$_2$.

This check only determines non-conformance on document level. However, to resolve conflicts and non-conformances, users rely on more detailed fact types. For example, to evaluate, which requirements are not reflected, which view should satisfy which view requirement, or which project requirements should satisfy which meta-requirements. The derived fact type RequirementEntity misses to satisfy MetaRequirement can be used to determine the individual requirements violating the subset constraint defined in Constraint 2:

Derivation 4 *(Unsatisfied Meta-Requirement)*

*RequirementEntity misses to satisfy MetaRequirement
if and only
if that RequirementEntity is of **some** RequirementCategory
and some Document represents **that** RequirementEntity
  and is involved in **some** StandardApplicability
  where some Tailoring belongs to **that** StandardApplicability
  and contains **that** MetaRequirement **that** applies to
  requirements of **that** RequirementCategory
and it is not true that (**that** RequirementEntity satisfies **that**
MetaRequirement).

This rule searches for the applicable meta-requirements for the requirement category of an entity and finds entities that do not satisfy those meta-requirements. Analogously, offending views that should satisfy type-specific view requirements can be identified.

Example ECSS-E-ST-60-10C [55] requires as *"Elements of a performance requirement"* among other things a *probability* (confidence level) and *"[t]he [statistical] interpretation of this probability"*. While the EagleEye system requirement EE-MR-0255 [50] entitled as *"AOCS Performance"* provides a probability, it lacks a statistical interpretation.

A tool supporting our framework should provide requirements engineers with backward information about which meta- (or view-) requirements should be satisfied. Ideally, such checks would be executed as soon as an affected requirement or view is edited.

Other requirements, only implicitly part of the specification through the tailoring, should automatically be included in consistency and completeness checks. As expressed by Derivation 5, those requirements should also be shown to the engineers either by representing them directly or in a refined form within the product specification.

Derivation 5 *(Non-Represented Requirements)*

*Document misses to represent RequirementEntity$_1$ **if and only if**
that Document is involved in **some** StandardApplicability **where**
some Tailoring belongs to **that** StandardApplicability
and contains **that** RequirementEntity$_1$
  where it is not true that (**that** RequirementEntity$_1$ **is some**
  Requirement **that** is **some** MetaRequirement)
  and it is not true that (**that** RequirementEntity$_1$ **is some**
  Requirement **that** is **some** ViewRequirement)
  and it is not true that (**that** Document represents **that**
  RequirementEntity$_1$)
  and that RequirementEntity$_1$ is cloned by **no**
  RequirementEntity$_2$
    where that Document represents **that** RequirementEntity$_2$
  and that RequirementEntity$_1$ is refined by **no**
  RequirementEntity$_3$
    where that Document represents **that** RequirementEntity$_3$.

7 Case study

To evaluate the validity of our *T-Reqs* conceptual model and demonstrate a proof of concept for the quality checks based on it, we partially implement and populate it in a *case study* [105].

State of the art requirements engineering tools, as *DoorsNG*, *Polarion*, *SE Suite*,[37] *Visure Requirements*, and others, mostly provide the possibility to define custom data models and relation types. To implement our reference model, as presented in Sect. 5 and 6, it can be adapted in one of such tools. Yet, capabilities to define constraints and queries for consistency checks are mostly limited and thus intensive scripting and a thorough evaluation of tool limitations is necessary. Furthermore, domain knowledge is necessary to transform existing requirements and traceability information to the suggested fine-grained representation. In the context of the E-RMS project, ECSS is at the time of writing still in the process to identify an appropriate tool candidate to reform their standard requirements and related processes. Thus, input data and an appropriate real-world tool environment are missing.

Therefore, we perform as a first step an evaluation with a lightweight graph implementation. Graph representations are well suited to capture and retrieve traceability information [12,109]. We used the data from the EagleEye specifications specifications, as presented in Sect. 3, to build a trace graph with the native graph database Neo4J[38]. Neo4J comes with a graph database server that can be used via APIs in various programming languages. We use the Cypher[39] graph query language to create and query a traceability graph and to show that our constraints and derivations are suited to discover deficiencies in tailoring of standards.

In the following, we explain first the simplified *implementation schema* and representation of constraints we adopted for Neo4J to implement parts of our conceptual model relevant to the tailoring process analyzed in the case study. Then, we describe how we extracted data from the original EagleEye specification specification documents in PDF format to populate this graph, introduced as steps (D) and (E) in Fig. 2. Next, we present the selected quality checks implemented as Cypher graph queries and their application to the case study data—step (F). Finally, we discuss the results and threats to validity.

[37] TRC https://www.reusecompany.com/systems-engineering-suite, visited on 05/20/2021.

[38] https://neo4j.com/, visited on 10/25/2020

[39] https://neo4j.com/developer/cypher/, visited on 11/11/2020.

7.1 Graph schema

The graph model of Neo4J is a labeled property graph with binary relations. There is no explicit graph schema. Instead, the schema *emerges* from the existing nodes, relations, and properties in the graph. Neo4J imposes no restrictions on the labels of relations or linked nodes. This approach is very flexible since new information can be added at any time to an already existing graph. On the other hand, without knowledge about the various node types and the relation types that connect the nodes, meaningful and efficient querying is impossible. The typical approach to query previously unknown graphs is to perform a manual graph exploration or schema reconstruction based on samples of the data. In our approach, coming from a formalized conceptual model in ORM, this step can be skipped since we know exactly what is related and how. However, the expressiveness of ORM schemas substantially exceeds that of schemas for labeled property graphs. Hence, it is necessary to derive an *implementation schema*. This step would also be necessary to create a custom data model for an existing requirements engineering tool. Since the foundations of ORM were laid in the data-engineering domain, transformation from ORM into relational database schemas are available. Yet, in implementations on hand, derived and semi-derived facts and most constraints are not automatically transformed at all. To our knowledge, only first premature work exists on mapping of an ORM schema to graph databases, e.g., [110] or [111]. Our manually derived implementation schema for the traceability graph in this case study is explained in the following sections. We also realize some exemplary derivation rules by manually translating them into equivalent Cypher queries. These queries only depend on the schema, not on the data of our case study and are thus project independent.

To improve processing performance and to make the graphs easier to understand, we do not map all ORM schema elements, but only incorporate those that contribute to answer the user's questions. We concentrate on those parts of the ORM model that relate to the *tailoring process*. The schema is divided in three sections: document structure as the view part, requirement model, and tailoring model. Figures 30, 31, and 32 show a simplified UML representation of these. We use UML classes to denote nodes and associations to denote directed relations. The class names and the association names resemble the object types and fact types of the ORM schema. Attributes are omitted to improve understandability.[40] The names are used as node and relation labels in the graph. Since labeled property graphs do not support generalization, the generalization hierarchy had to be flattened, and relations that encounter super-types in the ORM schema had to be copied to all non-abstract sub-types.

[40] The complete schema can be obtained at [112].

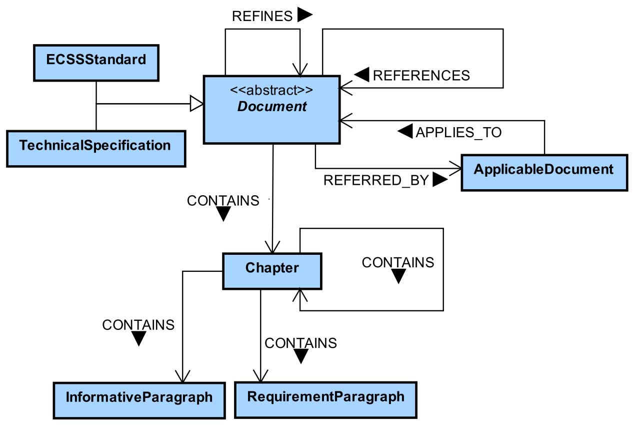


Fig. 30 Graph schema for the document structure

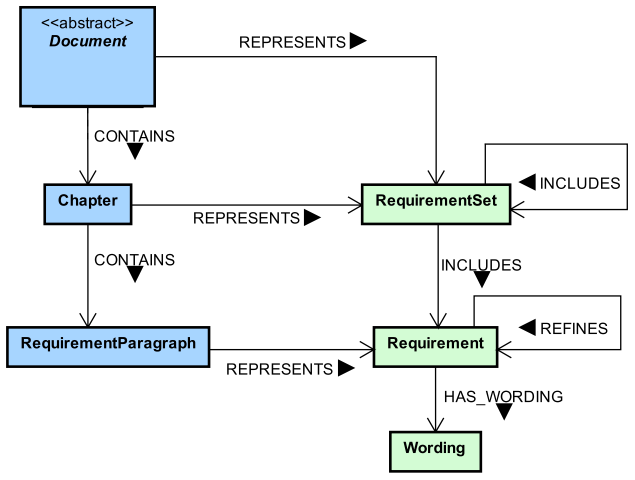


Fig. 31 Graph schema for the requirement model (green parts)

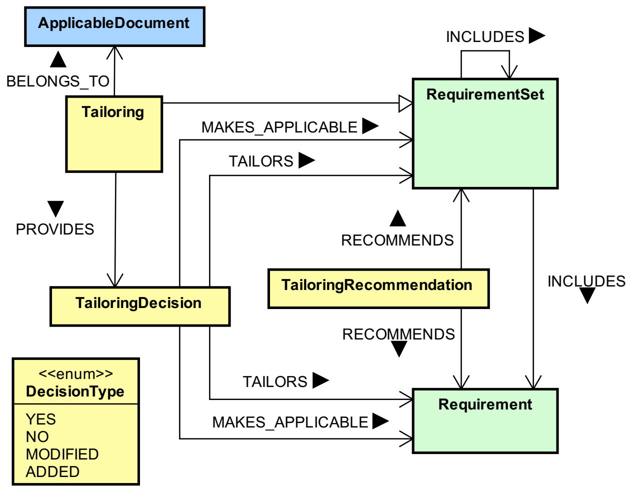


Fig. 32 Graph schema for the tailoring model (yellow parts)

A Document, as shown in Fig. 30, consists of chapters and paragraphs. The ordered CONTAINS relation allows for arbitrary nesting and is used to represent the sequence of chapters and paragraphs in a document. Requirements are represented in RequirementParagraph while other content is stored in InformativeParagraphs. The two document types used are ECSSStandard and TechnicalSpecification. There are three high-level document trace links: REFERENCES, APPLIES_TO, and REFINES. The first is a simple reference; the second denotes stricter semantics to include the requirements of the applicable document into the referrer document, depending on a tailoring decision. ApplicableDocument is modeled as a node type since it has further relations to a tailoring. Documents on more detailed abstraction levels connect to their parents by REFINES relations.

In the requirement model in Fig. 31, the RequirementSet and Requirement form a directed acyclic sub-graph structure of nested sets by means of the INCLUDES relation.[41] REPRESENTS relations link both to the document structure. REFINES is an example of trace links between individual requirements. More trace link types can be added as needed. The textual content of requirements is stored in Wording nodes.

Finally, Fig. 32 shows how tailoring of standards is addressed in the graph. The relevant concepts are defined and explained in Sect. 6. A Tailoring TAILORS a standard and is a RequirementSet that consists of the tailored requirements. Since a standard usually is applied to many specifications, each tailoring BELONGS_TO one or more ApplicableDocument. Each tailoring PROVIDES a set of TailoringDecisions. Depending on the decision type, the source requirements may be copied (decision: YES), skipped (NO), or refined (MODIFIED). Another option is to add new requirements that could be incorporated into a future release of a standard. Additionally, some standards come with TailoringRecommendations that recommend whether a requirement set is intended to be incorporated to dependent documents, or if it is optional. All requirement entities influenced by a tailoring decision are linked by MAKES_APPLICABLE relations.

7.2 Constraints

Constraints in the implementation are intended to prevent inconsistent or corrupted data on a technical level. Constraints should be enforced by the underlying technology. For example, the relation types connecting nodes should be restricted to match the implementation schema. The schemaless approach in Neo4J contradicts such restrictions. Thus, it is the responsibility of the tooling to care for consistent data. Alternatively, user-defined triggers could be installed to check for violations before changes are committed to the database. We omit triggers, since in the case study, automated extractors create the graphs (see Sect. 7.3) and subsequently they are used mostly in read-only mode. Uniqueness of iden-

[41] Unlike CONTAINS, INCLUDES is not ordered.

tifiers can be checked and enforced automatically in Neo4J by defining uniqueness constraints on properties for specific node labels.[42]

Derived facts as presented in Sect. 5.4.3 can be used to describe a softer type of constraint. We use them do determine quality issues that can be reported to users. Graph queries with their ability to traverse nodes by matching a path description allow for a straightforward realization.

7.3 The EagleEye graph

To build the traceability graph for EagleEye, we automatically extract the document structure and requirements from ECSS and EagleEye documents. Of course, scanning textual documents to create the requirements model is not the intended approach. Instead, in an ideal setting, the documents would be created from the model and editors would provide editable views, where writing requirements simultaneously creates or modifies the view and model elements.

In this case study, only the public PDF versions of documents are used. For EagleEye, other formats were not available at the time. They are created by various stakeholders without using a requirements engineering tool. The plain text of the PDFs is extracted with the Apache PDFBox[43] library and processed line by line by an importer tool that creates the graph elements. In total, 20 ECSS standards and 4 technical specifications are scanned. The resulting graph has 17.476 nodes and 23.507 edges and contains 4.132 requirements.

7.3.1 Detection of the document structure

The importer extracts the document structure, i.e., chapter headings, nesting of chapters, and requirements paragraphs. For example, the following plain text fragment from ECSS-E-ST-60-20C [56] has to be processed:

```
4.1.3 Star tracking
4.1.3.1 Inputs
a. The minimum set of inputs to be supplied in
order to initialize the Star Tracking shall be:
1. the initial star position;
2. the angular rate;
3. validity date.
b. For aided tracking, data specified in
4.1.3.1a shall be supplied regularly by the
spacecraft, at an update rate and accuracy
agreed by the customer.
c. The unit of all inputs shall be indicated.
```

It contains a chapter heading *"Star tracking"*, a nested chapter *"Inputs"*, and three requirements starting with lower case letters *a*, *b*, and *c*. To find chapter headings in the plain text of a document, we use regular expressions. For example, the chapter headings of ECSS standards are found using Regular Expression 1.

Regular Expression 1 *(ECSS chapter heading)*

```
^(
  (Annex\s+([A-Z])(\s*(\([a-z]+\))?)) |
  ([A-Z](\.\d+)+) |
  ([<]?\d+(\.\d+)*[>]?)
)\s*
([^.;:-]*)$
```

In this regular expression, the first three parts match the numbering variants of a chapter, while the last line matches the heading text. The first numbering variant matches the main heading of an annex, like `Annex C (informative)`. The second variant deals with the hierarchical numbering inside an annex. An example plain text is `C.8.2 Star tracker`. Lastly, the third variant matches hierarchical numbering of chapters in the main sections of a standard. If a line of the plain text matches the regular expression, this indicates a *potential* chapter or paragraph heading.

The regular expressions are constructed after manual inspection of some of the documents, running a test scan, and then refining the expressions. However, even after refinement, we observed many false positive matches. To sort out those false positives, we used the following checks for each document:

a) A manually created list of pages to be ignored by the importer. For example, some pages contain tables with text matching the chapter heading expression, but not being a heading.
b) A manually created list of chapter numbers that can contain requirements. Only those chapters are analyzed for requirements. ECSS standards as well as technical specifications follow a common document structure. For example, in ECSS standards, requirements can only occur in Chapter 4ff, and in the annex.
c) A heuristic telling if a chapter number is a viable successor of the previously detected chapter number.

Regular Expression 2 *(ECSS requirement heading)*

```
^([a-z])\.\s+(.*)$
```

The Regular Expression 2 is used to detect and extract the heading and the first part of the wording of a requirement. If the plain text matches the expression, and the match occurs inside a requirements chapter, the corresponding view and model nodes are created.

Similar regular expressions and the same heuristics were used to import the technical specifications of EagleEye. We

[42] We found no way to enforce uniqueness of an attribute for all nodes. For example, a uuid attribute should be globally unique, not only within a subset with common labels.

[43] https://pdfbox.apache.org, visited on 11/11/2020.

are aware of the fact that this approach is only a preliminary step, and that the importers are one-time software.

7.3.2 Creating the view and model nodes

Documents, RequirementSets, and Requirements are identified by a unique *user ID*. In case of the standards, it corresponds to the chapter and section numbering. For the technical specifications of EagleEye, requirements have unique identifiers that can be used. To avoid name clashes, the document ID is added as a prefix.

Graph nodes and edges are created by executing Cypher queries via the Neo4J Java API. For each document, the importer creates a top-level ECSSStandard or TechnicalSpecification node as root of the CONTAINS hierarchy by executing Query 1. On the model side, a corresponding top-level RequirementSet is used as root of the INCLUDES hierarchy. Additionally, REPRESENTS edges are created between each document and its top-level RequirementSet. Query variables ($), e.g., `$title`, are set by the importer before execution.

Query 1 *(Creation of an ECSSStandard node)*

```
CREATE (d :ECSSStandard {
  uuid: $docUuid,
  userId: $userId,
  title: $title,
  filename: $filename})
-[:REPRESENTS]-> (rs :RequirementSet {
  uuid: $rsUuid,
  userId: $userId,
  title: $title})
```

Requirements are grouped in RequirementSets. They form a tree created according to the chapter nesting. As explained in Sect. 5.2.2, the one-to-one mapping of chapters to sets is not the sole possible or favored set hierarchy. Yet, it is the only one automatically derivable without additional information on requirements content and classification. Besides, tailoring recommendations can refer to chapters. Hence, it makes sense to create a set per chapter to provide target nodes for the recommendation and decision relations. Query 2 creates the respective nodes and edges and links the newly created Chapter and RequirementSet nodes to their parents in the CONTAINS and INCLUDES hierarchy.

Query 2 *(Creation of a Chapter node)*

```
MATCH
  (n {uuid: $parentUuid}),
  (rsp :RequirementSet {uuid: $rsParentUuid})
CREATE (n)
-[:CONTAINS {index: $index}]-> (c :Chapter {
  uuid: $chapterUuid,
  userId: $userId,
  title: $title,
  pageNumber: $pageNumber})
-[:REPRESENTS]-> (rs :RequirementSet {
    uuid: $rsUuid,
    userId:
    $userId, title: $title})
<-[:INCLUDES]- (rsp)
```

A similar query is used to create Requirement nodes and to link them to their parent sets. Since informative text in the documents is not relevant for the analyses of this case study, the importer currently does not create InformativeParagraph nodes.

Due to the heterogeneous formats and varying quality of the available documents, trace information beyond containment and representation could not be extracted automatically. Instead, the trace links are created manually by specifying the source and target user id and the required relation type (c.f. Sect. 4.2). We add around 40 nodes and 70 trace relations to store the document dependencies and the tailoring data. Relations between individual requirement, such as REFINES (c.f. Sect. 4.1), are not considered. Such explicit links are not present in the documents. Rather, trace information is only given on a coarse granularity level by referencing whole sections of standards. Even the few references to individual requirements found in the technical specifications cannot be exploited because of missing trace type information and unavailable referenced documents. Establishing correct trace links manually furthermore would require understanding the technical and scientific details of the space engineering domain. Because of this missing information, only analysis that do not rely on these relation types are implemented for this case study.

Converting PDF documents to plain text is not an ideal solution to populate a trace graph. This is because the PDF format is meant to perfectly reproduce the layout of a document, but not to represent the structure. The extractors have to deal with many exceptions, as explained in the previous section. We assume that better precision and more automation could be achieved when formalized sources, for example DOORS databases[44] would be available. Anyway, this step is only necessary to transform the legacy data of the case study. New projects using a tool that follows the reference model would produce this data while editing the requirements. Dependent on the tool capabilities, partially automated trace retrieval can be supported.

7.4 Consistency and conformance checks

We demonstrate the applicability of our approach by showing how the Derivation 2 for conflicting tailoring decisions (see p. 21) and the Derivation 5 for missing requirements (see p. 23) can be computed with graph queries on the implementation schema.

[44] Porting of the EagleEye specifications to DOORS was ongoing while the work presented here was performed.

7.4.1 Conflicting tailoring decisions

Derivation 2 uses the fact type tailors and the derived fact type implicitly tailors[45] that link tailoring decisions with requirement entities. When the referenced entity is a RequirementSet that can include requirements and nested sets, those nested entities are regarded as implicitly tailored by the decision.

Derivation 6 *(Implicitly Tailors)*

*RequirementEntity$_1$ is implicitly tailored by TailoringDecision$_1$
if and only if
that TailoringDecision$_1$ tailors **some** RequirementEntity$_2$ **that is some** RequirementSet$_1$ **that** contains **that** RequirementEntity$_1$
and no TailoringDecision$_2$ **exists**
  where that TailoringDecision$_2$ tailors **that** RequirementEntity$_1$
  or tailors **some** RequirementEntity$_3$
   that is contained in **that** RequirementSet$_1$
   and that RequirementEntity$_3$ **is some** RequirementSet$_2$
   that contains **that** RequirementEntity$_1$.

Hence, implicitly tailors is the transitive closure of the INCLUDES relation, where entities[46] that have an individual tailoring decision are not part of the closure. Derivation 6 can be computed[47] by Cypher Query 3.

Query 3 *(Implicitly Tailors)*

```
MATCH (td1 :TailoringDecision)
      -[:TAILORS]-> (re2 :RequirementSet)
      -[:INCLUDES*1..]-> (re1: RequirementEntity)
WHERE NOT EXISTS {
  MATCH (td2 :TailoringDecision)
  WHERE ((td2) -[:TAILORS]-> (re1))
    OR (EXISTS {
      MATCH (td2) -[:TAILORS]-> (re3 :RequirementSet)
            <-[:INCLUDES*1..]- (re2)
      WHERE (re3) -[:INCLUDES]-> (re1)
    })
}
RETURN td1, re1
```

The transitive closure is built by matching the INCLUDES relation multiple times. The condition in the WHERE part of the outer query prunes sub-graphs with individual tailoring decisions. In the query, the "**that** is **some** ..." parts of the derivation rule do not appear since the type hierarchy is flattened in the implementation schema. To compute Derivation 2, Query 3 would have to be incorporated as a sub-query. This would result in a very complicated query. We "materialize" the implicitly tailors facts by adding explicit relations between the nodes of the result set through preventive execution of this query.

Figure 33 shows an excerpt of our trace graph. At the top, an ApplicableDocument (dark green) between an EagleEye specification and ECSS standard is shown. The chapter structure (grey nodes with CONTAINS links) form a tree. The leafs of the structural tree are RequirementChapters (purple) that represent the Requirements (green) with their Wording (orange).

The central yellow node represents the Tailoring with some expanded TailoringDecision nodes. The bottom right yellow decision node labeled with NO refers to a requirement set (dark blue) that does not include subsets while the decision labeled with YES on the left tailors a set with deeper nesting. The IMPLICITLY_TAILORS edges were added to allow for simpler subsequent queries. In our example, 179 additional edges are created for a tailoring of one ECSS standard. Of course, this overhead in space can be traded for more complicated and less efficient queries. Depending on the concrete graph size, document structure, and frequency of violation detection, and target technology, such optimizations could be omitted or deferred. Similarly, the TailoringRecommendation propagates through the INCLUDES relations. The implicitly affected nodes are linked by IMPLICITLY_RECOMMENDS edges.

These materialized shortcut edges make the detection of violations very easy. Query 4 determines pairs of TailoringDecision and TailoringRecommendation nodes that are in conflict with respect to the Derivation 2 on page 21.

Query 4 *(Recommendation Violation)*

```
MATCH (td :TailoringDecision {decision: 'NO'})
  -[:TAILORS|IMPLICITLY_TAILORS]->
  (:RequirementEntity)
  <-[:RECOMMENDS|IMPLICITLY_RECOMMENDS]-
  (tr :TailoringRecommendation { mandatory: true})
RETURN DISTINCT td, tr
```

The graph excerpt in Fig. 34 shows three tailoring recommendations (red nodes). A true value in a recommendation specifies that the requirements in the referenced set (blue node) have to be applied, while false marks optional application. Hence, the leftmost tailoring decision (yellow node) labeled with YES as well as the rightmost one comply with the respective recommendations. In contrast, the tailoring decision labeled with NO excludes requirements that are mandatory since the recommendation holds a true value. Query 4 reports this conflict as a pair of the two conflicting nodes. In the figure example, only the simplest situation is depicted, where recommendation and decision refer to the same set. Yet, this is the only case contained in the EagleEye data. However, by considering possibly nested sets and requirements, nested decisions, and nested recommen-

[45] c.f. Fig. 22, p. 19.

[46] including their descendants

[47] Due to a bug in the Cypher engine, we actually use a less efficient work-around. It is provided at [112].

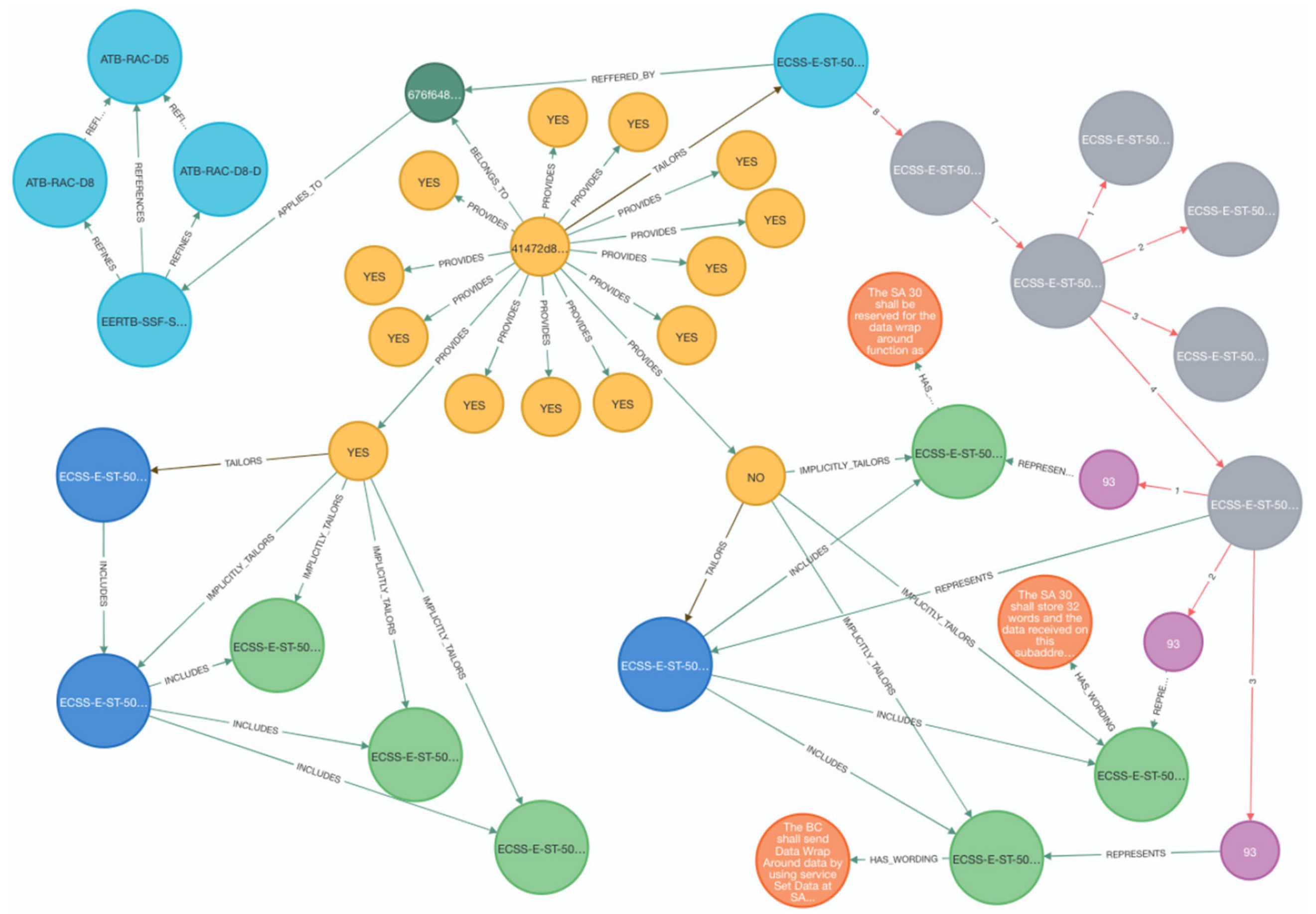

ATB-RAC-D5
ATB-RAC-D8
ATB-RAC-D8-D
EERTB-SSF-S...
REFERENCES
REFINES
APPLIES_TO
676f648...
REFFERED_BY
ECSS-E-ST-50...
41472d8...
BELONGS_TO
PROVIDES
TAILORS
YES
NO
IMPLICITLY_TAILORS
INCLUDES
REPRESENTS
HAS_WORDING
93
The SA 30 shall be reserved for the data wrap around function as
The SA 30 shall store 32 words and the data received on this subaddre...
The BC shall send Data Wrap Around data by using service Set Data at SA...

Fig. 33 Excerpt from EagleEye graph with implicit tailoring decisions

dations, the query is also capable to detect conflicts with a more complex structure.

7.4.2 Missing tailorings

When a standard is applied to a specification document, it needs to be tailored to determine the relevant requirement set. The tailoring does not necessarily have to belong to this document, but can also occur at a document further down the refinement hierarchy of a specification document tree. Thus, in complex specification trees, it is easy to miss a tailoring. While experienced project engineers usually are aware of the standard requirements that apply to a specification, missing explicit tailorings contradict completeness provisions, as defined by ECSS. In the available EagleEye documents, we encountered many applicable standard references without a tailoring.[48] In general, missing tailorings inhibit automatic computation of the complete effective requirement set of a project and thus hamper automated and manual consistency reviews. Derivation 7 for the fact type is not tailored[49] can be used to detect missing tailorings.

Derivation 7 *(Applicable standard is not tailored)*

```
*StandardApplicability$_1$ is not tailored **if and only if**
**no** Tailoring belongs to **that** StandardApplicability$_1$
**and that** StandardApplicability$_1$ involves **no** Document$_1$
  **where that** project-requirements Document$_1$ is refined by **some**
  implementation Document$_2$
    **where some** Standard is applicable to **that** Document$_2$
      **and** is involved in **that** StandardApplicability$_1$
    **and that** Document$_2$ is involved in **some** StandardApplicability$_2$
      **that** involves **that** Standard
      **and it is not true that** (**that** StandardApplicability$_2$ is not tailored).
```

The rule checks where a tailoring is missing on the level where the standard is applied for the first time (first part of the **and**) and that a tailoring cannot be found in any of the refining documents. Since the rule is recursive (last line), it traverses the transitive closure of the refines hierarchy. For

[48] We assume that those applicable standards would be tailored by more detailed documents on system or subsystem level, but such documents do not exist for the virtual mission. Furthermore, according to our interviews (c.f. Sect. 6.3) it seems to be common to assume that an experienced supplier "knows how to tailor these standards".

[49] see Fig. 21 on p. 19 and Fig. 29 on p. 22.

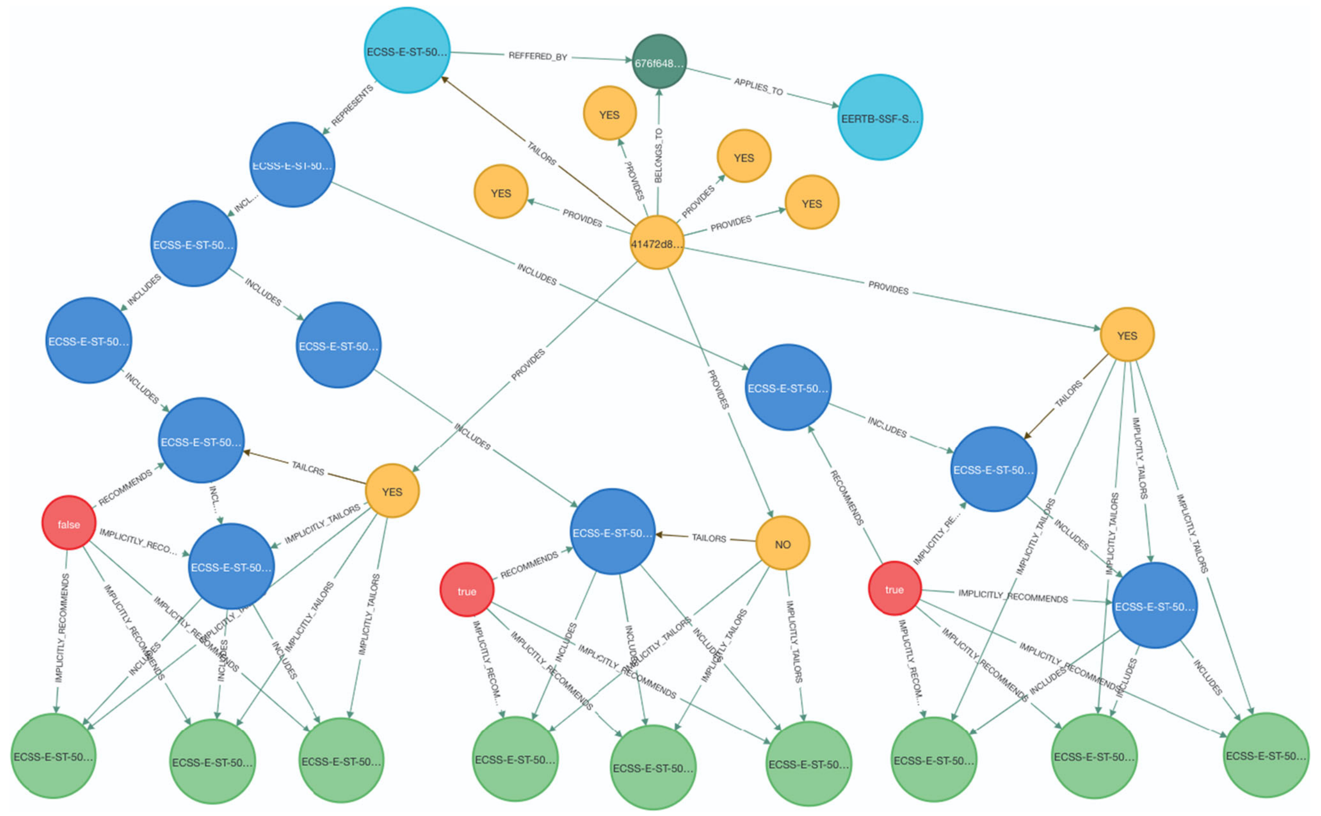


Fig. 34 Excerpt from EagleEye graph with tailoring recommendation violation

our traceability graph implementation, Query 5 computes Derivation 7.

Query 5 *(Applicable standard is not tailored)*

```
MATCH (st :ECSSStandard)
   -[:REFERRED_BY]-> (ad :ApplicableDocument)
   -[:APPLIES_TO]-> (d1 :TechnicalSpecification)
WHERE NOT EXISTS {
    MATCH (ad) <-[:BELONGS_TO]- (:Tailoring)
       -[:TAILORS]-> (st)
}
AND NOT EXISTS {
    MATCH (st) <-[:TAILORS]- (:Tailoring)
       -[:BELONGS_TO]-> ( ApplicableDocumet)
       <-[:APPLIES_TO]- (:TechnicalSpecification)
       -[:REFINES*1..]-> (d1)
}
RETURN ad
```

In this query, the nodes ad, d1, and st bind to any applications of a standard. They refer to the variables $\mathsf{StandardApplicability}_1$, $\mathsf{Document}_1$, and Standard of Derivation 7, respectively. The first condition of the WHERE part checks the absence of a tailoring of st on the same level as d1, while the second condition traverses the REFINES hierarchy. This second part checks that a tailoring of the standard does also not occur on one of the more detailed levels. For our case study, the query reports 7 out of 11 ApplicableStandard trace relations without an explicit tailoring.

We have shown by two non-trivial examples how graph queries can be used to compute derived facts of our ORM schema. Matching and path traversal in graphs facilitates efficient evaluation of such derived facts. By materializing derived facts in additional edges, queries for derivations that build on other derivations can be simplified. We did not measure evaluation times. For the graph size used in our case study (around 40.000 nodes and edges), the results were computed almost instantly with no observable delay. Practical performance will strongly depend on the target requirements engineering or traceability tool and its technology to implement such queries.

7.5 Discussion

The main goal of our approach is to support reviews and conformance analysis within projects with dependent and reused requirements documents such as standards. On one hand, recall on identified dependencies shall be enhanced by explicitly visualize otherwise overlooked dependencies to individual requirements, e.g., within applicable standards. On the other hand, we expect to reduce the effort for reviewers by limiting the number of dependency candidates by classification (meta-requirements) or reduce effort by classification of relation types [25,39]. To achieve this, a rich set of traceability relation types and traceable entities is defined. Pinheiro [27] notes on very rich sets of traces and traceable objects that "the complexity of the model may impair the tracing process". To determine the necessary relevant entities, dependencies and constraints in our detailed meta-model of individual relations we analyzed several project documentations, ECSS standards and drafting rules for such standards and verified our results in interviews and discussions with domain experts. To evaluate the usefulness and effectiveness of the approach, the approach has to be integrated into requirements engineering processes and tools of the target organizations. This would also enable to evaluate if model complexity can be concealed by adequate tool support, i.a., through automated trace retrieval, or if it has a limited scalability. Yet, the presented case study based on EagleEye only represents a proof of concept. This is due to a lack of an adequate tool environment, which covers ECSS standards as well as project requirements with the right level of granularity and several limitations of the dataset.

7.5.1 Threats to validity

Zogaan et al. [113] summarize possible threats to validity for datasets used for traceability studies, covering different aspects of *internal validity*, *external validity*, and *reliability* [105]:

Artificial Answer-Set Although our data- and answer-set is no *Student Dataset*, still it is an artificial example. A ground truth answer-set only exists for small proof of concept excerpts and other researchers may come to different conclusions on how to classify requirements and dependencies.

Real-World Data Although EagleEye is an artificial example with reduced complexity, it is to a certain degree representative for industrial projects. It is developed with industry involvement from ESA and several companies based on previous projects and for the simulation some real software deliverable is implemented. Furthermore, the standards used are real domain standards applicable in all similar European space engineering projects. However, due to confidentiality and the sometimes classified nature of safety and security critical missions, it is difficult to publish actual real-world examples from this domain. Here, EagleEye fills the gap for illustration and as a potential benchmark.

Limited Observations As only proof of concept examples for one case study are implemented, only limited observations can be claimed. Yet, the interviews showed that the dependencies and problems addressed in the case study occur in real life projects.

Domain The background research and case study are focused on only one domain. Yet, similar issues with standardized reuse and conformance can be observed in other highly standardized, security, safety, and/or privacy sensitive domains, such as aerospace, automotive, and healthcare [38,39,101]. Yet, different types of requirements in other domains can raise the need for additional dependencies to be considered.

Size The small size of the dataset implemented in the case study is one of the main threats to generalization. With only few examples and no established ground truth, it is not possible to measure the effectiveness of the approach in terms of precision and recall compared to manual reviews. In the same way, usability can only be evaluated with a substantially large enough—realistic—example and with integration to established tooling and workflows. However, previous research that incorporated very large graphs[50] with several millions of nodes and edges showed that the efficiency of recent query languages is sufficient to be used even in interactive tools. Current initiatives in the domain, as the E-RMS development and the OSMoSE[51] initiative to develop a space systems ontology, are expected to produce larger datasets with the required level of detail and appropriate extensible tool support in the near future. Yet, the challenge is, that only the ECSS repository data is not sufficient—to investigate the application to projects, compatible system specifications with adequate granularity and formality have to be available. In addition, the small group of interview partners is not representative. Yet, the aim of this qualitative investigation was to verify our assumptions about heterogeneity and lack of awareness for different tailoring procedures and conformance review approaches and to explore new facets of the topic. Semi-structured interviews are well suited for this [106], but due to their time consuming nature do not scale well for large samples.

Artifact Type Only natural language requirements and specification documents are considered. Generally, the approach is extensible for other artifact and relation types, which may raise the need for additional relation types and adjustment of derivation rules. For EagleEye also other artifacts, such as system models and source code are available.

Programming Language As we used a lightweight graph implementation for our evaluation, we expect it to be easily portable to other graph representations and query languages. Yet, consistency constraints cannot be preserved on the technical level this way. This can be achieved by directly using the XML based population feature of the NORMA ORM-tool. This, however, suffers from severe performance issues and does not scale to larger populations. An alternative are graph

[50] Those graphs were extracted as fine-grained representations of program structures of complex software systems.

[51] Kick-Off meeting in June 2019, https://indico.esa.int/event/310/ (visited 17/11/2020).

frameworks with schema support, such as JGraLab[52] which works on *TGraphs* [114]. However, manual meta-model transformation is tedious and error prone. It is possible to overcome such problems by an automatic model transformation from the ORM schema to a TGraph schema. Obviously, the transformation should convey the complete data model. A more ambitious goal is to transfer the constraints and derived facts from the conceptual level to the implementation level, too. First results in this direction were achieved in a prototype transformation tool [111]. However, an implementation of the reference model with its queries should be integrated to a full requirements engineering and/or traceability solution.

Selection Bias For proof of concept, only a subset of the available data is implemented as trace graph. To show the applicability we select examples representative for problem categories. We do not claim this to be statistically representative for occurring relation types or quality of real-world projects.

Information Bias As mentioned in Sect. 7.3, we assume that better precision is achieved when formalized sources, like DOORS databases or E-RMS [83], are used instead of PDF text documents. However, as we only aimed for a proof of concept, the current results are not negatively impacted by this. A *Negative Set Bias* is not applicable, as no automated classification is used. Concerning *reliability*, we provide the graph and queries of our experiment for replication and analysis [112]. However, the conceptual model it is based on may differ, if modeled by other researchers. Nevertheless, the general assumptions are likely to hold, as they conclude from domain expert knowledge and literature by application of the *Conceptual Schema Design Procedure* (CSDP) [85, p.13].

7.5.2 Challenges and future work

Our approach is based on the existence of explicit relations from which additional implicit relations can be derived. "It is also useful to incorporate extraction mechanisms not bounded to the existence of explicit links" [27]. Some relations can be derived through derivation rules over the properties of traceable entities. We address this in work in progress on an extended version of the meta-model covering more fine-grained properties of requirements. It is of special importance to correctly identify the content and targets of requirements, but this is also challenging for *Systems of Systems* (SoS) [115]. In Sect. 2, some related approaches for automated trace retrieval are discussed. Such approaches need to be tailored to match our set of relation types, as most automatic approaches to trace recording only support one general type of traceability link [69]. Besides this, our reference model is ready to be used together with automated trace retrieval techniques, as for our constraints, analysis, and additionally derived relations it is not important if the links are asserted manually or by some algorithm.

Furthermore, we plan to extend the meta-model with further relations and dependencies resulting from the refinement of documents within the customer supplier chain and to adjust it to SAVOIR's generic specifications. One challenge in this area is the relation between SAVOIR and ECSS documents. Our interviews (c.f. Sect. 6.3) show that it is not yet clearly defined, how applicable document links from SAVOIR specifications to ECSS standards should be resolved. The integration of *feature models* [80] would enable to express more complex dependencies in tailoring, as also mutually conditional recommendations can be expressed. Further work in this area could be the basis to build tool support that enables requirements engineers to better handle complex document dependencies in big projects and facilitate reuse by formalizing the import of existing requirements, not only from standards. Likewise, tailoring of other standards, generic sources, requirements from legal documents, and legacy requirements are expected to work in a similar way.

The lack of convenient tools with suitable requirements classification and good facilities for accessing the requirements repository is also identified as a critical factor hindering requirements reuse [22]. We therefore need to develop such tooling or integrate our approach to existing tools targeting an appropriate level of granularity, e.g., in the context of the E-RMS [83] or OSMoSE initiatives. One challenge here is the transformation of the existing standards to the new, more fine grained, format. Currently implicit properties of requirements, i.a., the differentiation in meta- and view-requirements, needs to be made explicit. *Machine learning* (ML) approaches for text classification [116,117] have been successfully applied for requirements classification problems [108,118] and seem promising for this task. Alternatively, heuristic rules on term patterns can be used, as Houmb et al. [119] or Gärtner et al. [120] demonstrate for the identification of security requirements.

Furthermore, such tooling also needs to address meta-data for trace management and *versioning* of relations and traceable entities. State of the art tools mostly already support configuration management, thanks to the integration of version control systems. Through baselines, different versions can be treated as individual traceable entities in the sense of our conceptual model. This is necessary, as several versions of a document might be used in parallel. For example, a project near to completion might not want to update the version of a used standard anymore. Thus, automated propagation of changes through different projects is challenging. It depends on the type of relation between the entities and the type of change. This could be handled via *event-condition-action* rules as by Schwarz et al. [88]. The main challenge is

[52] https://jgralab.github.io/jgralab/, visited on 10/29/2020.

to differentiate between pure editorial and semantic changes. We plan to research this for a future extension of the reference model.

8 Conclusion

The main goal of our approach presented in this paper is to support reviews and conformance analysis within projects with dependent and reused requirements documents, such as standards. To achieve this, we investigate the following research questions:

RQ-I. *Which types of relations are relevant for requirement reuse from documents?*

RQ-II. *How can requirements reuse from documents be formalized into a structured process that supports integration?*

In this paper, we provide an overview on different types of requirement relations with a special focus on inter-set and view relations. Documents are considered as views on sets of requirements with traceability relations on their own. To specify these relation types, the concept of "layered traceability" described by Gotel and Finkelstein [6] is used. Relations between artifacts of different representation granularity, e.g., requirements documents, requirements sets, or individual requirements, influence each other. We particularly focused on dependencies between relations of different types to handle implications through different representation layers.

As a reference for reuse scenarios tackling RQ-I and RQ-II the tailoring processes for standards of the European Cooperation for Space Standardization (ECSS) [43] are analyzed. We introduced specific relations amongst logical sets of requirements and requirements documents, based on known inter-requirements relations, which are relevant for conformance analysis with respect to standard tailoring. We showed their applicability in practice through usage examples from space industry.

Results are formalized as a conceptual model using the Object-Role Modelling (ORM) [86,87] notation. We expressed dependencies between different relation types via constraints and derivation rules.

The EagleEye case study, a virtual earth observation mission, is used to implement a proof of concept trace graph in Neo4J. We show example Cypher queries to detect violations of tailoring recommendations and missing tailorings.

Yet, our framework in its current state is mostly theoretical and we have not yet developed any tool support. Future work should therefore focus on implementing a practical tool to validate the framework. State of the art requirements engineering tools bring the capacities to define meta-models with customized artifact and relation types. Yet, often limitations exist in terms of defining the relation semantics, constraints, and especially dependencies among them. Further investigations of potential existing tools to be extended are necessary.

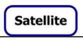
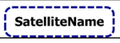
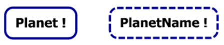
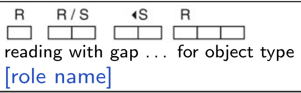
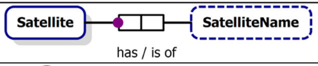
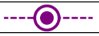
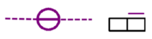
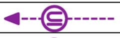
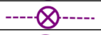
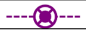
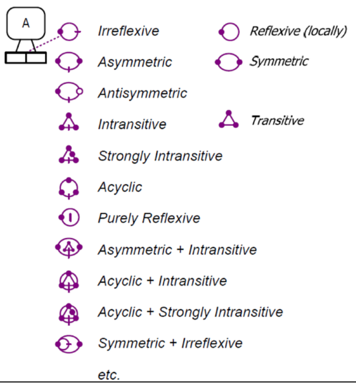
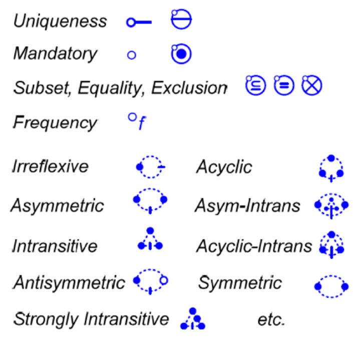
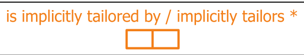
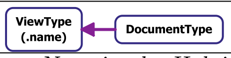
The theoretical foundations shall contribute to initiatives to conceptualize and standardize these trace dependencies. The provided reference model extends similar reference models for high-end traceability as provided by Ramesh and Jarke [5] to enable shared semantics within the target domain. The overall goal is to enable semantic interoperability between different tools used through the overall space system development and operations life-cycle, between ESA, the space related national agencies and industry, as envisioned, e.g., by Winkler and Pilgrim [69] and tackled by ESA's OSMoSE initiative. Furthermore, advanced requirements management systems supporting requirements reuse through pattern catalogues, such as the ECSS E-RMS [83], need such traceability models to support conformance analysis and enhance the catalogue by usage feedback.

We expect our contributions to be relevant beyond the space engineering context. The interoperability difficulties encountered in the European space business are not bound to this domain. They exist in all large projects involving many tools and many teams, distributed geographically and in time. Similarly, the reuse through standards and related dependencies and conformance analysis issues are also found in other safety critical, standardized, embedded systems domains, as aeronautics, defense, automotive, or health care.

Acknowledgements We gratefully acknowledge financial support by European Space Agency's (ESA) NPI program under NPI No. 4000118174/16/NL/MH/GM. Furthermore, we are thankful for the fruitful collaboration with ECSS and ESA's ESTEC, Noordwijk. In particular, the contributions of Serge Valera, Andreas Jung, and the whole TEC/SW team. Our best thanks for constructive discussions and feedback also to Marsha Chechik, Daniel Strüber, Veronika Vasileva, and Jakob Großer, as well as the anonymous reviewers, who helped to improve the quality of the paper. Modeling of ORM performed with NORMA [122] and UML with Astah [123].

Funding Open Access funding enabled and organized by Projekt DEAL.

A Basic ORM graphic notation

Construct	Example
Entity Type	
Value Type	
Independent Object Type	
Predicate (un-, bin-, ternary, . . .)	 R R / S S R reading with gap . . . for object type [role name]
Simple *Mandatory Role Constraint*	
Inclusive-Or	
Uniqueness (external or internal)	
Subset Constraint	
Exclusion	
Exclusive-Or	
Ring Constraint	 A Irreflexive Reflexive (locally) Asymmetric Symmetric Antisymmetric Intransitive Transitive Strongly Intransitive Acyclic Purely Reflexive Asymmetric + Intransitive Acyclic + Intransitive Acyclic + Strongly Intransitive Symmetric + Irreflexive etc.
Deontic Constraints	 Uniqueness Mandatory Subset, Equality, Exclusion Frequency f Irreflexive Acyclic Asymmetric Asym-Intrans Intransitive Acyclic-Intrans Antisymmetric Symmetric Strongly Intransitive etc.
Semi- (+) / Derived (*) Fact Type	
Sub-typing	
	Notation by Halpin c.f. [52–55]

References

1. Dick, J.: Rich traceability. In: 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'02), pp. 18–23 (2002)
2. Spanoudakis, G., Zisman, A.: Software traceability: a roadmap. In: Handbook Of Software Engineering And Knowledge Engineering: Vol 3: Recent Advances. World Scientific, pp. 395–428 (2005)
3. Goodrum, M., et al.: What requirements knowledge do developers need to manage change in safety-critical systems? In: 25th IEEE International Requirements Engineering Conference (RE'1). Sept. (2017), pp. 90–99. https://doi.org/10.1109/RE.2017.65
4. Gaspard-Boulinc, H., Conversy, S.: Usability Insights for Requirements Engineering Tools: A User Study with Practitioners in Aeronautics. In: 25th IEEE International Requirements Engineering Conference (RE'17), pp. 223–232 (2017)
5. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. In: IEEE Transactions on Software Engineering 27.1, pp. 58–93 (2001). https://doi.org/10.1109/32.895989
6. Gotel, O.C.Z., Finkelstein, A.C.W.: Contribution structures [Requirements artifacts]. In: 2nd IEEE International Symposium on Requirements Engineering, pp. 100–107 (1995)
7. da Silva, L.F., Leite, J.C.S.d.P.: Generating requirements views: a transformation-driven approach. In: Electronic Communications of the EASST 3 (2006): 3rd Workshop on Software Evolution Through Transformations: Embracing the Change (2006). https://doi.org/10.14279/tuj.eceasst.3.39.21
8. Leite, J.C.S.d.P., Oliveira, A.d.P.A.: A client oriented requirements baseline. In: IEEE International Symposium on Requirements Engineering (RE'95), pp. 108–115 (1995). https://doi.org/10.1109/ISRE.1995.512551
9. Leite, J.C.S.d.P., et al.: Enhancing a requirements baseline with scenarios. In: Requirements Engineering 2.4, pp. 184–198 (1997)
10. Siegemund, K., et al.: Towards ontology-driven requirements engineering. In: Workshop Semantic Web Enabled Software Engineering at 10th International Semantic Web Conference (ISWC) (2011)
11. van Lamsweerde, A.: Requirements Engineering. From System Goals to UML Models to Software Specifications. John Wiley & Sons Inc (2009). (ISBN: 978-0-470-01270-3)
12. Schwarz, H., et al.: Graph-based traceability: a comprehensive approach. In: Software and Systems Modeling (SoSym) 9.4, pp. 473–492 (2010)
13. Maletic, J.I., et al.: Using a hypertext model for traceability link conformance analysis. In: International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 47–54 (2003)
14. Espinoza, A., et al.: Analyzing and systematizing current traceability schemas. In: 30th An nual IEEE/NASA Software Engineering Workshop (SEW), pp. 21–32 (2006). https://doi.org/10.1109/SEW.2006.12
15. Rupp, C., Joppich, R.: Anforderungsschablonen - der MASTER-Plan für gute Anforderungen. German. In: C. Rupp and die SOPHISTen. Requirements-Engineering und -Management - Aus der Praxis von klassisch bis agil. 6th ed. Carl Hanser Verlag München, pp. 215–246 (2014). ISBN: 978-3-446-43893-4
16. Rupp, C., Günther, A.: Das SOPHIST-REgelwerk - Psychotherapie für Anforderungen. German. In: C. Rupp and die SOPHISTen. Requirements-Engineering und -Management - Aus der Praxis von klassisch bis agil. 6th ed. Carl Hanser Verlag München, pp. 123–164 (2014). ISBN: 978-3-446-43893-4
17. Mavin, A., et al.: Easy approach to requirements syntax (EARS). In: 17th IEEE International Requirements Engineering Conference (RE'09), pp. 317–322 (2009). https://doi.org/10.1109/RE.2009.9
18. RequirementsWorking Group. Guide for Writing Requirements. Tech. rep. INCOSE-TP-2010-006- 03. Version 3. International Council on Systems Engineering (INCOSE) (2019)

19. ISO/IEC/IEEE 29148: Systems and software engineering—Life cycle processes—Requirements engineering. ISO/IEC/IEEE 29148:2018(E) (ISO/- IEC/IEEE). Nov. (2018)
20. Castañeda, V., et al.: The use of ontologies in requirements engineering. In: Global Journal of Researches in Engineering 10.6, pp. 2–8 (2010). ISSN: 2249-4596
21. Krueger, C.W.: Software reuse. In: ACM Computing Surveys 24.2, pp. 131–183 (1992). ISSN: 0360-0300. https://doi.org/10.1145/130844.130856
22. Palomares, C., et al.: Requirements reuse and requirement patterns: a state of the practice survey. Empir. Softw. Eng. **22**, 2719–2762 (2017). https://doi.org/10.1007/s10664-016-9485-x
23. van Lamsweerde, A.: Requirements engineering in the year 00: a research perspective. In: 22nd International Conference on Software Engineering (SE). ACM, pp. 5–19 (2000)
24. Robinson, W.N., et al.: Requirements interaction management. In: ACM Computing Surveys 35.2, pp. 132–190 (2003). ISSN: 0360-0300. https://doi.org/10.1145/857076.857079
25. Carlshamre, P., et al.: An industrial survey of requirements interdependencies in software product release planning. In: 5th IEEE International Symposium on Requirements Engineering, pp. 84–91 (2001). https://doi.org/10.1109/ISRE.2001.948547
26. Mavin, A., et al.: Does goal-oriented requirements engineering achieve its goal? In: 25th IEEE International Requirements Engineering Conference (RE), pp. 174–183 (2017)
27. Pinheiro, F.A.C.: Requirements traceability. In: do Prado Leite, J.C.S., Doorn, J.H. (eds.) Perspectives on Software Requirements. Springer, pp. 91–113 (2004). ISBN: 978-1-4615- 0465-8. https://doi.org/10.1007/978-1-4615-0465-8_5
28. Goknil, A., et al.: Change impact analysis based on formalization of trace relations for requirements. In: Oldevik, J., et al. (eds.) ECMDA Traceability Workshop (EC- MDA-TW), Vol. 274, pp. 59–75 (2008). SINTEF Report. ISBN: 978-82-14-04396-9
29. Goknil, A., et al.: Semantics of trace relations in requirements models for consistency checking and inferencing. In: Software and Systems Modeling (SoSym) 10.1 (2011), pp. 31-54
30. Samer, R., et al.: New approaches to the identification of dependencies between requirements. In: 31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI'19), pp. 1265–1270 (2019). https://doi.org/10.1109/ICTAI.2019.00-91
31. Abbas, M.: Variability aware requirements reuse analysis. In: 42nd ACM/IEEE International Conference on Software Engineering (ICSE): Companion Proceedings, pp. 190–193 (2020). https://doi.org/10.1145/3377812.3381399
32. Reinhartz-Berger, I., Kemelman, M.: Extracting core requirements for software product lines. In: Requirements Engineering 25.1, pp. 47–65 (2020). https://doi.org/10.1007/s00766-018-0307-0
33. Goldin, L., et al.: Reuse of requirements reduces time to market. In: IEEE International Conference on Software Science, Technology Engineering, pp. 55–60 (2010). https://doi.org/10.1109/SwSTE.2010.17
34. Naish, J., Zhao, L.: Towards a generalised framework for classifying and retrieving requirements patterns. In: 1st International Workshop On Requirements Patterns, pp. 42–51 (2011). https://doi.org/10.1109/RePa.2011.6046721
35. Siena, A., et al.: From laws to requirements. In: 1st International Workshop on Requirements Engineering and Law (RELAW), pp. 6–10 (2008). https://doi.org/10.1109/RELAW.2008.6
36. Siena, A., et al.: A meta-model for modelling law-compliant requirements. In: 2nd International Workshop on Requirements Engineering and Law (RELAW), pp. 450–451 (2009). https://doi.org/10.1109/RELAW.2009.1
37. Zeni, N., et al.: Applying GaiusT for extracting requirements from legal documents. In: 6th In- ternational Workshop on Requirements Engineering and Law (RELAW), pp. 65–68 (2013). https://doi.org/10.1109/RELAW.2013.6671349
38. Guo, J., et al.: Tackling the term-mismatch problem in automated trace retrieval. In: Empirical Software Engineering 22.3, pp. 1103–1142 (2017)
39. Wang, W., et al.: Detecting software security vulnerabilities via requirements dependency analysis. In: IEEE Transactions on Software Engineering (2020). https://doi.org/10.1109/TSE.2020.3030745. Early Access
40. Renault, S., et al.: A pattern-based method for building requirements documents in call-for-tender processes. In: International Journal of Computer Science & Applications (IJCSA) 6.5 (2009): Special Issue on Advanced Solutions for Information Systems Engineering. Ed. by A. Flory and M. Collard, pp. 175–202. ISSN: 0972-9038. http://www.tmrfindia.org/ijcsa/v6i57.pdf (visited on 11/09/2020)
41. Ramadan, Q., et al.: A semi-automated BPMNbased framework for detecting conflicts between security, data-minimization and fairness requirements. In: Software and Systems Modeling (SoSyM) 19.5, pp. 1191–1227 (2020). https://doi.org/10.1007/s10270-020-00781-x
42. Alvarez, J.L., et al.: Model-based system engineering approach for the Euclid mission to manage scientific and technical complexity. In: Modeling, Systems Engineering, and Project Management for Astronomy VI. SPIE Astronomical Telescopes + Instrumentation. Vol. 9911. International Society for Optics and Photonics. Aug. 18, (2016), p. 99110C. https://doi.org/10.1117/12.2231373
43. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. ECSS system—Description, implementation and general requirements. ECSS-S-ST-00C (ECSS). July 31 (2008)
44. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space project management—Project planning and implementation. ECSS-MST- 10C (ECSS). Mar. 6 (2009)
45. Bartodziej, C.J.: The concept industry 4.0. In: The Concept Industry 4.0: An Empirical Analysis of Technologies and Applications in Production Logistics. Springer, pp. 27–50 (2017). ISBN: 978-3-658-16502-4. https://doi.org/10.1007/978-3-658-16502-4_3
46. Edwards, P.R., et al.: International requirements for payload multi-platform reuse methodology. In: ESA Workshop on Aerospace EMC (Aerospace EMC), pp. 1–7 (2019). https://doi.org/10.23919/AeroEMC.2019.8788967
47. Bos, V., et al.: Time and space partitioning the eagleeye reference misson. In: Data Systems in Aerospace (DASIA). Vol. 720. ESA Special Publication. Aug. (2013), pp. 22–29. https://ui.adsabs.harvard.edu/abs/2013ESASP.720E..22B/abstract (visited on 11/11/2020)
48. Bos, V., et al.: Time and space partitioning using on-board software reference architecture. In: IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). Oct. (2016), pp. 17-20. https://doi.org/10.1109/ISSREW.2016.49
49. Ayuso, A.: EagleEye—VIRTUAL SPACECRAFT MISSION REQUIREMENTS. Tech. rep. TOSEMS-VSRF-SPC-0001. Version 5. SENER, Oct. 6 (2004)
50. Pace, F.: EARTH OBSERVATION REFERENCE MISSION—SYSTEM SPECIFICATION. Tech. rep. ATB-RAC-D5. ESA - ESTEC (2009)
51. Pace, F., et al.: ATB SOFTWARE REQUIRE- MENTS SPECIFICATION. Tech. rep. ATB-RACD8. Version 2.1. ESA-ESTEC (2010)
52. Pace, F., Barrena, V.: EARTH OBSERVA- TION REFERENCE MISSION—SW SPECIFI- CATIONS. Tech. rep. ATB-RAC-D8-D. ESA - ESTEC (2010)
53. Srungavruksham, D.T.: CSW V6 Requirements Document. EagleEye TSP porting to HWIL configuration (RTB). Tech. rep. EERTB-SSF-SRS- 005. Version 5.0. SSF (2017)

54. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering—Control engineering. ECSS-E-60-A (ECSS). Sept. 14 (2004)
55. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering—Control performance. ECSS-E-ST-60-10C (ECSS). Nov. 15 (2008)
56. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering—Stars sensors terminology and performance specification. ECSS-E-ST-60-20C (ECSS). Nov. 15 (2008)
57. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering—Software—Part 1: Principles and requirements. ECSSE- ST-40Part1B (ECSS). Nov. 28 (2003)
58. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering—Software—Part 2: Document requirements definitions (DRDs). ECSS-E-ST-40Part2B (ECSS). Mar. 31 (2005)
59. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering—Interface and communication protocol for MIL-STD- 1553B data bus onboard spacecraft. ECSS-E-ST- 50-13C (ECSS). Nov. 15 (2008)
60. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering—Space data links—Telecommand protocols, synchronization and channel coding. ECSS-E-50-04A (ECSS). Nov. 14 (2007)
61. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering—Ground systems and operations—Telemetry and telecommand packet utilization. ECSS-E-70-41A (ECSS). Jan. 30 (2003)
62. Gotel, O.C.Z., Finkelstein, A.C.W.: An analysis of the requirements traceability problem. In: IEEE International Conference on Re- quirements Engineering (RE), pp. 94–101 (1994). https://doi.org/10.1109/ICRE.1994.292398
63. ISO/IEC/IEEE 24765: Systems and software engineering—Vocabulary. ISO/IEC/IEEE 24765: 2010(E) (ISO/IEC/IEEE). Dec. 15 (2010)
64. Gotel, O., Finkelstein, A.: Extended requirements traceability: results of an industrial case study. In: 3rd IEEE International Symposium on Requirements Engineering (ISRE), pp. 169–178 (1997). https://doi.org/10.1109/ISRE.1997.566866
65. Darimont, R., et al.: GRAIL/KAOS: an environment for goal-driven requirements engineering. In: 19th International Conference on Software Engineering (ICSE'97), pp. 612–613 (1997)
66. Letelier, P.: A framework for requirements traceabilityin UML-based projects. In: 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE), pp. 173–183 (2002)
67. Haidrar, S., et al.: On the use of model transformation for requirements trace models generation. In: International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS). IEEE, pp. 1–6 (2017)
68. Espinoza Limón, A., Garbajosa Sopeña, J.: The need for a unifying traceability scheme. In: ECMDA Traceability Workshop. Oct. 19 (2005)
69. Winkler, S., Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. In: Software and Systems Modeling (SoSyM) 9.4, pp. 529–565 (2010)
70. Salay, R., et al.: Language independent refinement using partial modeling. In: de Lara, J., Zisman, A. (eds.) Fundamental Approaches to Software Engineering, pp. 224–239. Springer, Berlin Heidelberg (2012)
71. Pinheiro, F.A.C., Goguen, J.A.: An object oriented tool for tracing requirements. In: IEEE Software 13.2, pp. 5264 (1996). https://doi.org/10.1109/52.506462
72. Glinz, M.: A Glossary of Requirements Engineering Terminology. Tech. rep. Version 1.6. International Requirements Engineering Board IREB e.V. (2014)
73. Bruneliere, H., et al.: A feature-based survey of model view approaches. In: Software & Systems Modeling (SoSym), pp. 1–22 (2017). https://doi.org/10.1007/s10270-017-0622-9
74. Lexico. Meaning of document in English. https://www.lexico.com/definition/document (visited on 05/10/2021)
75. ISO/IEC/IEEE 42010: Systems and software engineering—Architecture description. ISO/IEC/-IEEE 42010:2011 (ISO/IEC/IEEE). Dec. 1 (2011)
76. Palm, S.U.: ATB CONSOLIDATION—Space/- Ground Interface Control Document. Tech. rep. TER-ATBC-TS-ICD-004. Version 1.1. Terma (2013)
77. Li, F.-L., et al.: From stakeholder requirements to formal specifications through refinement. In: Fricker, S.A., Schneider, K. (eds.) Requirements Engineering: Foundation for Software Quality. Springer International Publishing, pp. 164–180 (2015). ISBN: 978-3-319-16101-3. https://doi.org/10.1007/978-3-319-16101-3_11
78. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering - Technical requirements specification. ECSS-E-ST-10- 06C (ECSS). Mar. 6 (2009)
79. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. ECSS system—Glossary of terms. ECSS-S-ST-00-01C (ECSS). Oct. 1 (2012)
80. Kang, K.C., et al.: Feature-oriented product line engineering. In: IEEE Software 19.4, pp. 58–65 (2002). https://doi.org/10.1109/MS.2002.1020288
81. Gervasi, V.: Keynote: requirements philology. In: 2nd Workshop on Natural Language Processing for Requirements Engineering (NLP4RE'19) (2019)
82. Lions, J.-L.: ARIANE 5 Flight 501 Failure. Report by the Inquiry Board. (1996). http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html (visited on 02/28/2018)
83. Valera, S.: ECSS Requirements Management System—[E-RMS]. SOW—Statement of Work ESATECSWM-SOW-009970. ESA/ESTEC (2018)
84. SAVOIR. SAVOIR Documentation Tree. Tech. rep. SAVOIR-TN-000. ESA ESTEC (2016)
85. Halpin, T.: OBJECT-ROLE MODELING FUNDAMENTALS. Technics Publications, A Practical Guide to Data Modeling with ORM (2015). ISBN:978-1-63462-074-1
86. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd ed. Morgan Kaufmann (2010). ISBN:978-0-12-373568-3
87. Halpin, T.: Object-role modeling. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems. Springer, pp. 1941–1946 (2009). ISBN: 978-0-387-35544-3. https://doi.org/10.1007/978-0-387-39940-9_251
88. Schwarz, H., et al.: Using expressive traceability relationships for ensuring consistent process model refinement. In: 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 183–192 (2010). https://doi.org/10.1109/ICECCS.2010.66
89. ISO/IEC 9834-8: Information technology—Procedures for the operation of object identifier registration authorities—Part 8: generation of universally unique identifiers (UUIDs) and their use in object identifiers. ISO/IEC 9834-8:2014 (ISO/IEC). Aug. (2014)
90. Mouratidis, H., Jürjens, J.: From goal-driven security requirements engineering to secure design. In: International Journal of Intelligent Systems 25.8, pp. 813–840 (2010)
91. Ahmadian, A.S., et al.: Model-based privacy and security analysis with CARiSMA. In: 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'17), pp. 989–993 (2017). https://doi.org/10.1145/3106237.3122823

92. Peldszus, S., et al.: Secure data-flow compliance checks between models and code based on automated mappings. In: 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS'19), pp. 23–33 (2019). https://doi.org/10.1109/MODELS.2019.00-18
93. Gamma, E., et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series, Pearson Education (1994). ISBN: 9780321700698
94. Tarr, P., et al.: N degrees of separation: multidimensional separation of concerns. In: IEEE International Conference on Software Engineering (ICSE), pp. 107–119 (1999)
95. OMG® Unified Modeling Language® (OMG UML®) Version 2.5.1. formal/2017-12-05 (Object Management Group (OMG)). Dec. (2017)
96. Favaro, J., et al.: Next generation requirements engineering. In: 22nd Annual INCOSE International Symposium (2012)
97. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. Space engineering—Software. ECSS-E-ST-40C (ECSS). Mar. 6 (2009)
98. ECSS Secretariat and ESA-ESTEC Requirements & Standards Division. ECSS—Draft rules and template for ECSS Standards. ECSS-D-00-01C (ECSS). May 20 (2014)
99. ESA TEC-QR. ESA standardization manual. 1st ed. ESSB-D-000. ESA. Mar. (2014)
100. Lauesen, S.: Software Requirements: Styles and Techniques. Addison-Wesley (2002)
101. Ayala-Rivera, V., Pasquale, L.: The grace period has ended: an approach to operationalize GDPR requirements. In: 26th IEEE International Requirements Engineering Conference (RE), pp. 136–146 (2018). https://doi.org/10.1109/RE.2018. 00023
102. Wiegers, K.E.: Software Requirements: Practical Techniques for Gathering and Managing Requirement Through the Product Development Cycle, 2nd edn. Microsoft Press (2003)
103. Business Process Model and Notation (BPMN) Version 2.0.2. formal/2013-12-09 (Object Management Group (OMG)). Dec. (2013)
104. Palm, S.U.: ATB Consolidation Central Software Requirements. Tech. rep. TER-ATBC-TS-REQ- 001. TERMA (2014)
105. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. In: Empirical Software Engineering 14.2, pp. 131–164 (2009). https://doi.org/10.1007/s10664-008-9102-8
106. Kallio, H., et al.: Systematic methodological review: developing a framework for a qualitative semi-structured interview guide. In: Journal of Advanced Nursing (JAN) 72.12, pp. 2954–2965 (2016). https://doi.org/10.1111/jan.13031
107. Opdenakker, R.: Advantages and Disadvantages of Four Interview Techniques in Qualitative Research. In: Forum Qualitative Sozialforschung / Forum: Qualitative Social Research 7.4 (Sept. 30, 2006). https://doi.org/10.17169/fqs-7.4.175
108. Schneider, K., et al.: Enhancing security requirements engineering by organizational learning. In: Requirements Engineering 17.1, pp. 35–56 (2012). https://doi.org/10.1007/s00766-011-0141-0
109. Rath, M., et al.: Are Graph Query Languages Applicable for Requirements Traceability Analysis? In: (REFSQ) Workshops (2017)
110. Braimniotis, M.: A Transformation from ORMConceptual Models to Neo4j GraphDatabase. MA thesis. Radboud University Nijmegen (2017)
111. Owen, A.: Mapping ORM to TGraph. MA thesis. University of Koblenz-Landau (2017)
112. Großer, K., et al.: Requirements document relations—a reuse perspective on traceability. Supplementary models and files. https://uni-ko-ld.de/requirementrelations (visited on 10/07/2021)
113. Zogaan, W., et al.: Datasets from fifteen years of automated requirements traceability research: current state, characteristics, and quality. In: 25th IEEE International Requirements Engineering Conference (RE'17), pp. 110–121 (2017)
114. Ebert, J., Franzke, A.: A declarative approach to graph based modeling. In: Mayr, E.W., et al. (eds.) Graph-Theoretic Concepts in Computer Science. Springer Berlin Heidelberg, pp. 380–50 (1995). ISBN: 978-3-540-49183-5
115. Tekinerdogan, B., Erata, F.: Modeling traceability in system of systems. In: Symposium on Applied Computing. ACM, pp. 1799–1802 (2017)
116. Kowsari, K., et al.: Text classification algorithms: a survey. In: Information 10.4 (2019). ISSN: 2078-2489. https://doi.org/10.3390/info10040150
117. Kadhim, A.I.: Survey on supervised machine learning techniques for automatic text classification. In: Artificial Intelligence Review 52.1, pp. 273–292 (2019). https://doi.org/10.1007/s10462-018-09677-1
118. Kurtanović, Z., Maalej, W.: Mining user rationale from software reviews. In: 25th IEEE International Requirements Engineering Conference (RE), pp. 61–70 (2017). https://doi.org/10.1109/RE.2017.86
119. Houmb, S.H., et al.: Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and UMLsec. In: Requirements Engineering 15.1, pp. 63–93 (2010). https://doi.org/10.1007/s00766-009-0093-9
120. Gärtner, S., et al.: Maintaining requirements for long-living software systems by incorporating security knowledge. In: 22nd IEEE International Requirements Engineering Conference (RE'14), pp. 103–112 (2014). https://doi.org/10.1109/RE.2014.6912252
121. Halpin, T.: ORM 2 Graphical Notation. Dec. (2011). http://www.orm.net/pdf/ORM2GraphicalNotation.pdf
122. Natural ORM Architect for Visual Studio. ORM Solutions and Neumont University. https://github.com/ormsolutions/NORMA (visited on 02/07/2020)
123. Astah Professional. ChangeVision, Inc. https://astah.net/products/astah-professional/ (visited on 11/18/2020)

Katharina Großer is PhD candidate and member of the Institute for Software Technology (IST) within the Faculty for Computer Science at the University of Koblenz-Landau. Her PhD project on ontology supported requirements engineering is part of the NPI Programme of the European Space Agency (ESA), where she prior completed an internship. She graduated with a Master of Science (M.Sc.) in Computational Visualistics from University of Koblenz-Landau, where her Master's thesis was awarded the AFCEA price. She is alumnus of the Evangelisches Studienwerk Villigst.

Volker Riediger is Senior Researcher at the Institute for Software Technology (IST) at the University of Koblenz-Landau. He has more than 20 years of experience in research, technology transfer and teaching in software engineering, including research topics such as software maintenance, software evolution, program understanding and model-based development. He has a PhD in Computer Science from the University of Koblenz-Landau. Dr. Riediger has been head of the GI (Gesellschaft für Informatik) section "SRE Software Reengineering" from 2012 to 2016. Before his engagement in the university, he worked for more than 10 years as professional software developer in several application domains.

Jan Jürjens is a Professor, leading the Research Group for Software Engineering at the Institute for Software Technology (IST) within the Faculty for Computer Science of the University Koblenz-Landau. He is also Director Research Projects at the Fraunhofer Institute for Software and Systems Engineering ISST in Dortmund. Previous positions include a Professorship for Software Engineering at TU Dortmund, a Royal Society Industrial Fellowship at Microsoft Research Cambridge, a non-stipendiary Research Fellowship at Robinson College (Univ. Cambridge), where in 2009 he was appointed as Senior Member, and a Postdoc position at TU München. Jan holds a Doctor of Philosophy in Computing from University of Oxford and is author of "Secure Systems Development with UML" (Springer, 2005; Chinese translation 2009) and other publications mostly on software engineering and IT security. More information: http://jan.jurjens.de.