Systematic Literature Review of Domain-oriented Specification Techniques

Robert Deckers[a,b], Patricia Lago[a,c]

[a] Vrije Universiteit Amsterdam, The Netherlands
[b] Atom Free IT, Heeswijk-Dinther, The Netherlands
[c] Chalmers University of Technology, Sweden

ARTICLE INFO

Keywords:
Domain-Specific Language
Domain Model
Systematic Literature Review
Method Comparison
Specification Method
Modeling Language

ABSTRACT

Context. The popularity of domain-specific languages and model driven development has made the tacit use of domain knowledge in system development more tangible. Our vision is a development process where a (software) system specification is based on multiple domain models, and where the specification method is built from cognitive concepts, presumably derived from natural language.
Goal. To realize this vision, we evaluate and reflect upon the existing literature in domain-oriented specification techniques.
Method. We designed and conducted a systematic literature review on domain-oriented specification techniques.
Results. We identified 53 primary studies, populated the classification framework for each study, and summarized our findings per classification aspect. We found many approaches for creating domain models or domain-specific languages. Observations include: (i) most methods are defined incompletely; (ii) none offers methodical support for the use of domain models or domain-specific languages to create other specifications; (iii) there are specification techniques to integrate models in general, but no study offers methodical support for multiple domain models.
Conclusion. The results indicate which topics need further research and which can instead be reused to realize our vision on system development.

1. Introduction

Since the 1980s, several approaches for domain modeling have been developed and published [43, 45, 29]. Domain modeling is a technique to capture knowledge from domain experts and domain literature into a model. A domain model can be used in various ways, *e.g.,* as a basis for requirements specification [64], as a basis for software design [17], or as a language for functional specifications [29].

The purpose of this systematic literature review (SLR) is to investigate which specification techniques exist in the context of domain modeling. These are both techniques to *create* domain models, and techniques to *use* a domain model *as a language* for other specifications, *e.g.,* the specification of a feature, application, or system aspect. In order to create system specifications, we are also interested in techniques to *integrate* domain models and to integrate specifications expressed in terms of domain models. We call all these techniques together *domain-oriented (DO) specification techniques*. The outcome of this study is used as input for MuDForM (Multi Domain Formalization Method), which is the domain-oriented specification method that we are working on.

Domain models are intended to capture knowledge about the application domains of systems [74, 63]. But we are also interested in applying domain modeling to other domains, and quality domains in particular. Most methods related to domain modeling focus on the structural (state) properties of a domain, *e.g.,* [47, 20, 4, 53, 36, 25, 24, 9, 8, 27, 19]. Though, we are especially interested in methods that facilitate the specification of behavioral (dynamic) properties, and in such a way that they are well integrated with the specification of structural properties.

This SLR is organized around three main research questions (RQs) that investigate what specification techniques exist to (**RQ1**) make specifications of a domain (called *domain specifications*), (**RQ2**) make other specifications in terms of a domain specification (called *domain-based specifications*), and (**RQ3**) integrate several domain specifications and domain-based specifications in one specification (called *integration specifications*). These different types of specification are explained further in Section 3.1. Per identified technique, we answer several questions. First, to which domains is

ORCID(s): 0000-0002-3020-7550 (R. Deckers); 0000-0002-2234-0845 (P. Lago)

it applicable, and is it applicable to quality domains? Second, how methodical is the technique? Third, how does it address a number of aspects that are specific for domain-oriented modeling?

We have identified three **contributions of this SLR** along with the **related target audiences**. The first contribution is an overview of the state-of-the-art in specification techniques to create domain models (DMs) and domain-specific languages (DSLs), and their use in the creation of other (domain-based) specifications. This SLR discusses how well those techniques are engineered, by analyzing the conciseness and clarity of the specification language and specification process. We also discuss how well the existing techniques cover the aspects that are derived from the specific objectives (introduced in Section 2.3) that we envision for our own research, *i.e.,* for the definition of MuDForM. Potential users, developers, and researchers of domain-oriented specification techniques, can use the overview to select techniques in order to apply them in their own context.

The second contribution is the identification of shortcomings in the existing literature on domain-oriented specification techniques. We identified topics that need to be researched in the future, such as the support for working with multiple domains, the integration of modeling concepts for structural and behavioral properties, and having fine-grained guidance for modeling decisions. Another gap in the literature is the incompleteness of most method descriptions. This is not a new research topic, but rather a lack in method engineering of those specification techniques. Researchers and developers of domain-oriented specification techniques may use the identified topics as a starting point for their research and method engineering activities.

The third contribution is that we have defined a reusable approach for comparing methods, which is an extension to the guidelines described by Kitchenham [73]. First, we have made a conceptual model of the domain of method engineering. This model is reusable for other method comparisons. Second, we have created a conceptual model of the application domain of the targeted methods, *i.e., domain-oriented specifications*. The concepts from both models are used to define the research questions, the search queries, and the classification framework. The classification framework consists of three parts, which can be applied to any method comparison. Furthermore, the use of concept models of the method engineering domain, and of the application domain of the targeted methods, leads to a more consistent study design and execution. Researchers who also want to compare methods, possibly via a literature review, can benefit from the approach that we followed.

The remainder of this paper is structured as follows. Section 2 introduces some background knowledge. Section 3 describes the study design and execution. Section 4 presents the study results, *i.e.,* the extracted data from the primary studies that we included. Section 5 discusses the results from the perspective of the research questions, while Section 6 discusses related works. Section 7 addresses the threats to the validity of this study. Section 8 concludes this article, and identifies topics for future research.

2. Background: MuDForM and Domain Modeling

This section provides the background information of this SLR. This study is carried out as the starting point of our research, in which we work on a integral method for system specification via multiple domain models, *i.e.,* MuDForM[1]. We mention our MuDForM research program in this SLR, because its objectives are the main reason for the RQs and the aspects in the classification framework.

Accordingly, the following explains our perspective on domain modeling, and the objectives we aim to achieve with MuDForM. These objectives will be used for the definition of the classification framework in Section 3.5, and as a yardstick in the discussion (Section 5) of the data that is extracted from the selected primary studies.

2.1. What is a domain model?

We found two different notions of DM in the literature. The notion that we use is that of a specification space, analogous to a domain in the mathematical sense. The term "domain" refers to an area of knowledge or activity and a DM describes what can happen (behavior) and what can exist (state and structure) in a domain, or in other words, what can be controlled and managed in a domain. A DM is the foundation for a shared lexicon in communication between stakeholders, and can serve directly as a structured vocabulary for making other specifications, or form the underlying model of a DSL. For example, a model of the banking domain expressed in a UML class diagram, can be used directly in other UML diagrams, or can serve as the abstract syntax of a DSL. A DM is not intended to express what should happen, does happen, is likely to happen, or has always happened in the domain, because we assign those aspects to different types of specifications, like a system, application, or feature specification. The knowledge captured in a DM is

[1] A MuDForM is used to shape tacit and "muddy" data into knowledge building blocks.

not limited to a specific way of working in the domain nor to a specific system that operates in the domain. Approaches for DSLs like [2, 18, 28], comply with this notion of DM.

The other notion in the literature is that a domain is a collection of related systems. Accordingly, a DM defines a set of (system) features that are common in the domain. This notion is used for example by FODA [72]. According to this notion, a DM can only be made with a set of systems or features in mind, while in the notion that we adhere to, one can talk about the concepts in a domain independently from any feature or system.

2.2. The MuDForM vision

We envision software development as a process in which the involved people make decisions in their own area of knowledge, *i.e., domain*. Those decisions must be integrated, and finally result in a machine-readable specification. That is why our research focuses on an integral method for creating DMs, for using DMs as a language to create other (domain-based) specifications, and for integrating multiple DMs and domain-based specifications. It is the ultimate intention for a system to be completely defined in domain-oriented specifications, and that if other kinds of specifications are used, then they are also explicitly integrated.

We envision that a major difference between MuDForM and most other methods is that, in addition to modeling the objects in a domain, MuDForM also considers domain actions to be first-class domain concepts, and MuDForM integrates objects and actions. Domain actions describe the atomic changes in their domain. They are elements for the creation of composite behavioral specifications, *e.g.,* processes, scenarios, and system functions. Another difference comes from our notion of DM: DMs are descriptive and the result of analysis, and system specifications, *e.g.,* feature models, are design artefacts and prescriptive. We foresee that the third major difference is the extensive use of natural language processing in the modeling process.

2.3. MuDForM objectives

Based on our vision and experience with domain modeling, architecture, and model driven development, we have defined a set of objectives for the development of MuDForM. We introduce them shortly to justify the design of the classification framework described in Section 3.5. The objectives are:

O1 In any system development process, there are people that have concerns about different aspects and that take decisions about different aspects. The **distinction of multiple domains**, and their specification in DMs or DSLs, is the basis for dealing with the multitude of aspects in a development process. Moreover, a specification method should offer multiple mechanisms, *e.g.,* composition, consistency, transformation, or weaving, to integrate DMs or DSLs, and domain-based specifications.

O2 There is no limitation to what kind of aspects can be relevant in system development. A specification method should therefore be **independent from any domain** or system, and a method user (modeler) should not need any prior knowledge about the domain or system that is being specified. In practice, domain modeling is mostly used for the application domains of targeted systems, or for design aspects of software. We think domain modeling should also be applicable to quality domains, such as reliability, security, and usability.

O3 The knowledge of people about particular aspects can be seen independently from any specific (software) system, and is potentially usable in multiple systems. A specification method should reflect this and support **self-contained specifications** that are independent from their application in a specific system specification. To use specifications in different contexts, *i.e.,* to build other specifications, they should be composable, interpretable, and translatable.

O4 In our notion of domain, a DM captures what can happen and what can exist in a domain, and a system specification or feature specification are more about what shall happen and what shall exist in a system and its context. A method should support the separation of what can happen from what shall happen, *i.e.,* **distinguish descriptive domain specifications from prescriptive domain-based specifications**.

O5 Most domains and systems are not only about entities with a state, but also about change. A method should therefor support both the **the specification of state** of a domain at a certain moment **and the specification of change** of state over time. In other words, specifications should address things that exist, things that happen, and how these things are related. This perspective is similar to the notion of structural (static) properties and behavioral (dynamic) properties in UML [79].

O6 Almost all people, including domain experts, use natural language to convey their knowledge and decisions. A specification method should **support** the transformation of knowledge stated in **natural language** into specifications in an unambiguous specification language. Preferably, such specifications should themselves also be translatable into natural language. The purpose of this support is to minimize loss of semantics and better mutual understanding in the communication between modelers and domain experts. The MuDForM vision is to have method concepts that are close to human cognition. Natural language is a starting point for the method concepts, because it has evolved over thousands of years to support communication between people.

O7 A method should be **engineered**, which means it has a clear underlying model (often called meta model) with clear semantics, a defined notation (viewpoints and syntax), defined process steps (method flow), and guidance for the steps and viewpoints. Furthermore, a modeling process should help in eliciting input, help in achieving completeness and consistency, and enable the traceability of modeling decisions. We elaborate on these characteristics in Section 3.5.

O8 The purpose of a specification is mostly to realize a system (or part of of a system). So, the transition from a set of (domain-based) specifications to a working system should be feasible. In other words, the relation between specification method and **architecture** should be clear.

In summary, the main motivation for this SLR is to identify and characterize what solutions the existing literature provides for these objectives, in order to use them in the development of MuDForM. We, the authors of this SLR, have a background in software architecture, domain modeling and model driven development. When we started to work on MuDForM, we already knew of specification techniques from several books on domain modeling and domain-specific languages [17, 29, 28, 51, 33, 18]. We observed that those books did not address all the MuDForM objectives. Especially, the use of natural language processing and dealing with multiple models are topics that are hardly addressed. We did a preliminary informal literature scan on these topics and found some useful studies [8, 31, 27, 40, 19, 16, 52, 1], but they were mostly not containing relevant content for answering our research questions. Hence this SLR.

3. Study Design and Execution

This section describes the design and execution of our SLR. We follow the guidelines described by Kitchenham [73]. The purpose of this SLR is to investigate what specification techniques exist in the context of domain modeling, and compare them on their applicability, degree of method engineering, and how well they support the aspects that are derived from the MuDForM objectives. We are especially interested in techniques for analysis of natural language, techniques for handling multiple domains, and guidelines for modeling decisions. Another goal is to identify research topics, based on shortcomings and gaps that we detected in the existing literature with respect to our objectives.

Section 3.1 explains the research questions. From those questions and the inclusion/exclusion criteria (Section 3.3), we derive the search queries (Section 3.2) and the classification framework (Section 3.5). Section 3.4 describes the search process, *i.e.,* study execution. Based on the data extracted from the search results (described in Section 4), Section 5 discusses the answers to our research questions.

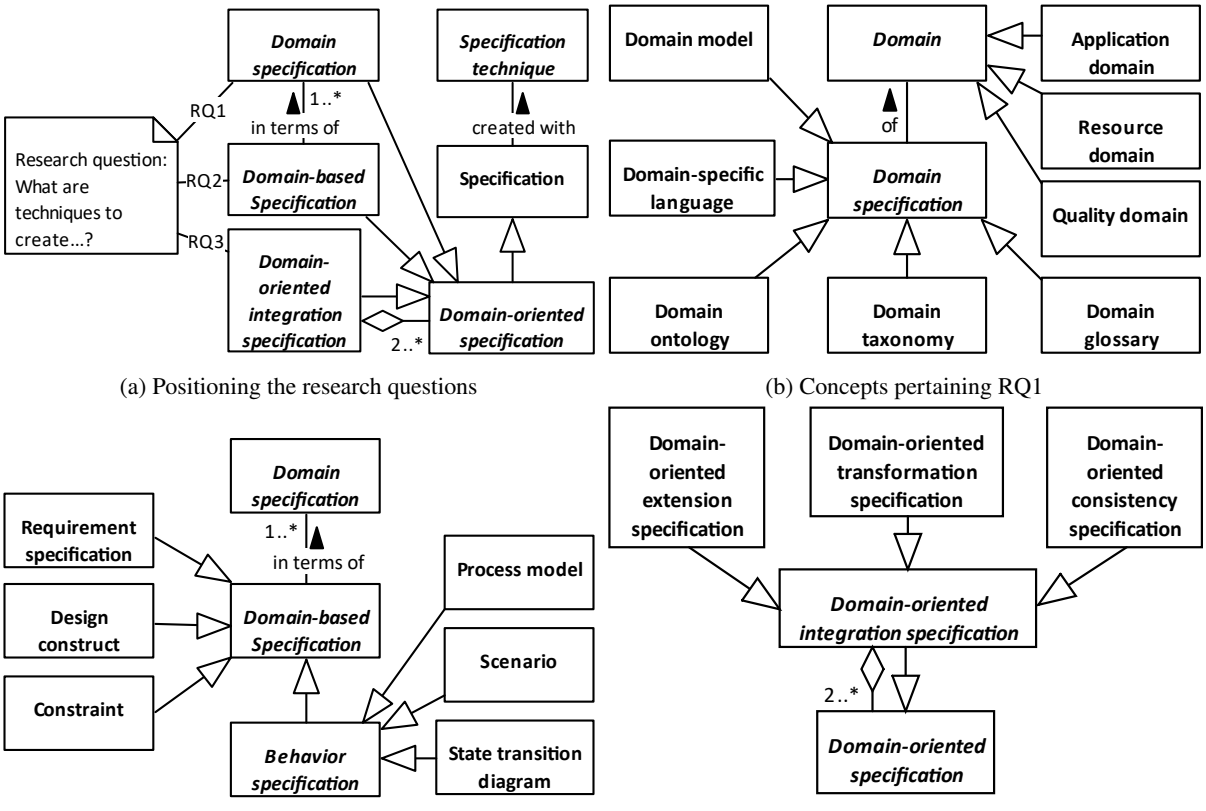
Of course, the results of this SLR, and the references to the found specification techniques in particular, can also be used by researchers and practitioners that investigate and develop domain-oriented specification methods.

3.1. Research Questions

This section elaborates on the research questions (see Section 1) of this SLR. In order to formulate RQ1-RQ3, the derived search queries, and the classification framework, in a coherent and unambiguous way, we created a conceptual model of the research domain of this SLR, *i.e.,* the domain of domain-oriented specification techniques. This model is presented in four class diagrams throughout this section.

We are interested in three categories of *specification techniques*[2]: *domain specifications*, *domain-based specifications*, and *domain-oriented integration specifications*. To be clear, this doesn't mean that the specification techniques themselves have to be explicitly domain-oriented. We are interested in all specification techniques that can be used to create domain-oriented specifications. Figure 1a depicts that each category corresponds to a research question that starts with the phrase "What are specification techniques to create ...?". We will now explain each research question.

[2]The italic words in the text refer to elements in the models.



(a) Positioning the research questions

(b) Concepts pertaining RQ1

(c) Concepts pertaining RQ2

(d) Concepts pertaining RQ3

Figure 1: Research questions: positioning and related concepts

Domain specifications (RQ1): What are techniques to create specifications of a domain?

We are interested in structured specifications of a *domain*, such as *domain models*, *domain-specific languages*, and *domain ontologies*. We found several types of domain specifications (see Figure 1b). But we are not looking for techniques that result in just an enumeration of the concepts in a domain, like *domain glossaries* or vocabularies, because they have no explicit structure. *Taxonomies* sometimes have a hierarchical structure, but they typically do not provide insight in how the concepts at one hierarchy level relate to each other.

The most common use of domain modeling techniques is for *application domains*. But we are also interested in techniques other types of domains, in particular *quality domains*. There are many quality domains, denoted by quality attributes such as security, usability, or maintainability. We are not looking for specifications of these quality domains, but for techniques to create their specification.

Software design and programming can also be seen as a domain, *i.e.,* the *resource domain*. This large domain can be divided into sub domains (often called design aspects), like user interaction, logging, persistence, rule checking, error handling, encryption, component deployment, load balancing, system decomposition, data communication, resource usage, and so on. We are not looking for specifications of these sub domains, but for techniques to create their specification.

Domain-based specifications (RQ2): What are techniques to create specifications in terms of domain specifications?

We are looking for techniques to create *domain-based specifications*, *i.e.,* specifications in terms of a *domain specification* [3]. Figure 1c shows some examples of types of specifications that could be domain-based: *(quality) requirements*, *constraints*, *design constructs*, or *behavior specifications* like *process models*, *scenarios*, or *state transition diagrams*. Besides using a DS as terminology, they are also written in terms of a language that is

[3] From hereon we will use DS (domain specification) instead of 'DM and/or DSL'.

specific for their type of specifications, such as a requirements language, constraint language, or process modeling language. Preferably, such a language is a DSL by itself, or at least specified via a DM.

To clarify how a domain-independent method could support the creation of a specification in terms of a DS, we describe three examples of specification techniques that address this RQ. First, if the domain elements that describe the behavior in a domain are used to specify processes steps, *i.e.,* they are the types of process steps, then it is possible to detect overlap between processes regarding sequences of steps that occur in multiple processes. In such case, a method guideline can be given to identify sub processes in a set of process models, *e.g.,* "Define a separate process for those sequences of process steps that occur in multiple processes". Second, if requirements or constraints are specified in terms of a DS, then method guidelines can be given to detect inconsistencies between requirements, *e.g.,* "Check requirements that use the same domain class". Third, if domain classes are used to specify the object structures in a system, then guidelines can be given for how to do this top-down, *e.g.,* "Start specifying functions for domain classes that are not a part of a composition or aggregation". More examples can be found in [61] – a MuDForM based publication about specifying features in terms of a domain model.

Specification integrations (RQ3): What are techniques to create specifications of integrations between domain specifications, and between domain-based specifications?

We are looking for specification techniques to create *DO integration specifications* of two or more *DO specifications* (*domain specifications* or *domain-based specifications*) via explicit integration methods, languages, models, or other mechanisms. We distinguish at least techniques for *transformation*, *extension*, and *consistency* between DO specifications (see Figure 1d). We see correspondence between specifications, as for example in the IEEE42010 standard for architectural descriptions [68], as a form of *consistency specification*. We see merge and composition, as for example in [13], weaving, as for example in [44], and other ways to combine two or more specifications into a new specification, as a form of *extension specifications*.

3.2. Search Queries

This section explains the creation of the search queries. The term specification techniques, which is used in all three research questions, is not commonly used in the literature as a denominator. Therefore, using this term will not give adequate results. That is why we first scanned through the literature we were familiar with, and made a model of the used terminology. Figure 2 shows different types of *specification techniques* that we found. It is not a complete lexicon, but a summary of the most common categories. The most occurring techniques are Specification (or Modeling) language, and Specification (or Modeling) method. Specification languages can be based on a Meta model, (which could be defined in a meta modeling language). A Meta model can also be the underlying model of a Specification method. Method definitions may also contain Method steps and Guidelines, and distinguish different Method viewpoints which use a Specification language as their notation. As such, Meta model, Method step, Guideline, and Method viewpoint can be seen as partial Specification techniques, and are of interest for this study.

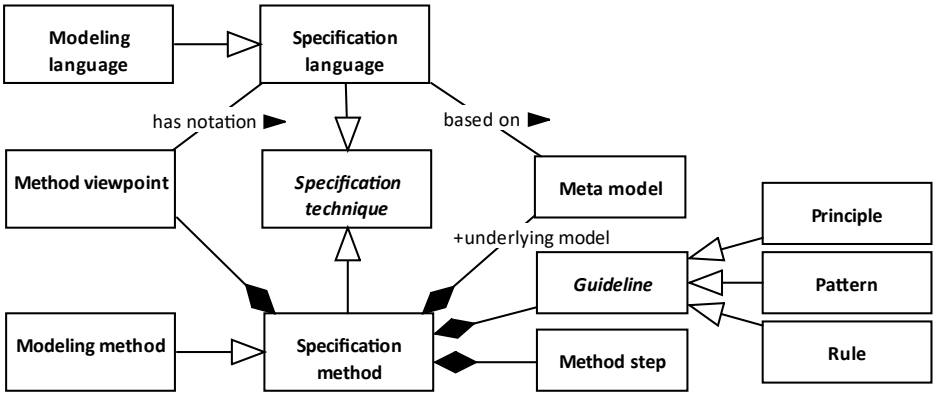


Figure 2: Examples of different specification techniques and their aspects.

It is not useful to just search for all types of techniques that we identified in the model, because this yields an unwieldy amount of results (more than 500,000 hits on Google scholar). The search string was thus narrowed down in

several steps to come to a manageable set of results:

1. We omitted *guidelines* (*principles*, *patterns*, *rules*), *method steps*, *viewpoint*, and *meta model* because they should always be defined in the context of a method. We will still use these terms as a denominator in the data extraction.

2. We omitted *ontology*, because ontology languages are in general less expressive than domain modeling languages. Namely, they are mostly limited to just capturing terms from the domain and relations between the terms. Some ontology languages go a bit further and distinguish classes, attributes, and different types of relations between classes. Though, almost all domain modeling languages also have those concepts. Though, if a study uses ontologies as an ingredient of a domain modeling approach, then we will include it if it matches the criteria.

3. Some authors use the term *approach*, mostly because they find the term method too specific. We do not want to ignore those studies. Thus, we include term approach as a possible generalized and more informal term for method.

4. We use the term *language* instead of *modeling language* or *specification language*, because we always search for it in combination with specification or model.

Given RQ1 and the explanations of Figure 1b, and after alternative terms that express the same semantics, we obtain the following search string:

```
((method) OR (approach) OR (methodology) OR (methods) OR (approaches) OR (methodologies))
AND ((domain-model) OR (domain-specific-language) OR (domain-models) OR (domain-modeling)
                    OR (domain-modelling) OR (domain-specific-languages))
```

This still led to more than 17,000 hits on Google Scholar. Therefore, we decided to limit the search string to the title of the publication and compensate this limitation with snowballing.

Further, because RQ2 and RQ3 include specification techniques as well as domain specifications, we reckon that the defined search string also covers these questions. So, RQ2 and RQ3 are not framed in two distinct search strings. Instead, we address them in the classification framework with their own classification aspect, as explained in Section 3.5.

3.3. Selection Criteria

This section addresses the inclusion/exclusion criteria that are used to select the primary studies.

Inclusion criteria. We run our search queries on Google Scholar, because it covers all well-known scientific publication sources, like ACM, Springer, and IEEE. Our inclusion criteria are:

(I1) Research publications subject to scientific peer review. Studies that were not submitted to scientific peer review might have claims that are not objectively verified on credibility. So, journal papers, PhD theses, and papers in conference or workshop proceedings, are considered. Also books and technical reports issued by respected institutes or authors are taken into account. But white papers, or articles in commercial magazines, are discarded.

(I2) Studies written in English.

(I3) Studies available online as full text. Exceptions can be made for well-known books on the subject.

Exclusion criteria. The exclusion criteria are:

(E1) Studies that do not contribute any specification technique for DO specifications, which includes DMs, DSLs, domain-based specifications, and DO integration specifications. For example, we exclude studies in which specifications, like requirements or DSL definitions, are only used as an example, while they do not explain how to make them.

(E2) Studies that focus on techniques for testing, reviewing, or checking specifications. We are looking for techniques in the context of system development, i.e., the creation and maintenance of domain-oriented specifications.

(E3) Studies that focus on techniques for human behavior or on how to organize the specification process. For example, SCRUM prescribes the specification of all work items, but it does not address how to specify them or how to apply them correctly.

(E4) Secondary and tertiary studies (e.g., systematic literature reviews, surveys, etc.). It is important to note that, though secondary studies are excluded, we may use them for precisely scoping the contribution of this SLR and for checking the completeness of the set of selected primary studies.

(E5) Studies that describe an approach for creating a DS and that do not comply with our notion of DM as explained in Section 2.1. As such we exclude studies that consider a domain as a set of systems or applications. We are interested in studies that see a domain specification as a language, and not as a framework to specify applications and systems. Of course, we will include a study if parts of it offer specification techniques that do comply with our notion of DM.

(E6) Studies that mainly focus on implementing DSs in a target environment without using an explicit DS of that environment. Transformation specifications from a source DS to a target DS are in scope. But transformations from a source domain to program code, without an explicit DS of the targeted software environment, are excluded.

3.4. Study Execution

As depicted in Figure 3, we followed the guidelines described by Kitchenham [73], leading to the following steps and search results:

1. The *initial search* took place on June 15, 2020 and led to 602 unique studies.
2. Then we *applied the criteria* in three exclusion stages: based on the title, based on the abstract, and based on the full text. This resulted in the inclusion of 20 primary studies. Besides those, we also kept 9 of the excluded studies for snowballing, because we found relevant citations during reading them.
3. We applied *snowballing* (as described by Jalali and Wohlin [71]) based on the citations in the already included studies, and in the studies we kept for references. This led to the selection of 125 extra references.
4. By *applying the criteria* to those, we selected 19 extra studies, bringing the total to 39 studies.
5. As indicated in Section 2.3, we added several relevant *books and studies in a informal search*, namely [8, 31, 27, 40, 19, 16, 52, 1, 17, 29, 28, 51, 18, 33], and double-checked them against the selection criteria.
6. Finally, we *extracted the data* and performed the analysis on a total of 53 primary studies, of which 7 are books and the rest are articles in journals, conferences, workshops, and reports published by well known academic institutes.

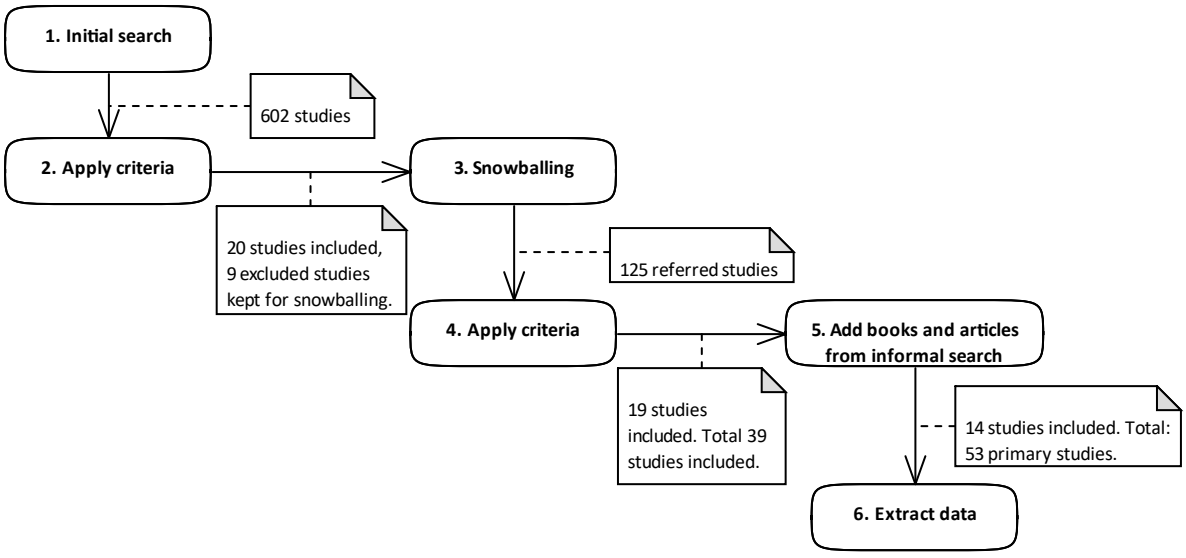


Figure 3: Study execution

We care to note that most papers were excluded because they did not contribute a specification technique (E1). The majority of them were about a specific DS and its usage, and did not offer an explanation of the used specification technique to create or use that DS.

Table 1
Relation between classification framework, research questions, and MuDForM objectives

CLASSIFICATION CATEGORY	CLASSIFICATION ASPECT	HELPS TO ANSWER			SERVES OBJECTIVE							
		RQ1	RQ2	RQ3	O1	O2	O3	O4	O5	O6	O7	O8
Application scope	Domain dependence	■	■	■	□	■	■	□	□	□	□	□
	Quality domains	■	■	■	□	■	□	□	□	□	□	□
	Architecture	■	■	■	■	□	■	□	□	□	□	■
Method engineering	Assuring consistency	■	■	■	□	□	□	□	□	□	■	□
	Provide traceability	■	■	■	□	□	□	□	□	□	■	□
	Detect incompleteness	■	■	■	□	□	□	□	□	□	■	□
	Definition completeness											
	Underlying model	■	■	■	□	□	□	□	□	□	■	□
	Notation	■	■	■	□	□	□	□	□	□	■	□
	Method steps	■	■	■	□	□	□	□	□	□	■	□
	Guidance	■	■	■	□	□	□	□	□	□	■	□
	Formalness	■	■	■	□	□	□	□	□	□	■	□
MuDForM specific	Domain-based	□	■	□	□	□	□	■	□	□	□	□
	Structural and behavioral	■	■	■	□	□	□	□	■	□	□	□
	Multiple domains	□	□	■	■	□	□	□	□	□	□	□
	Natural language											
	As input	■	■	■	□	□	□	□	□	■	□	□
	Translatable back into text	■	■	■	□	□	□	□	□	■	□	□

Many other studies were excluded because they were about code generation without considering the target environment as a domain, and thus not treating code generation as a form of domain integration (E6).

We found a few PhD theses on the topic of domain-oriented specifications, but we did not find articles that were part of or derived from those theses, and none of theses actually explained the specification techniques used to create or to use the DSs. But we kept theses in the process for snowballing their references, because they had relevant citations, as mentioned in step 2 above. The details of the study execution are available in the replication package [60].

3.5. Classification Framework

This section discusses the aspects that we use to analyze and compare the primary studies. We distinguish the aspects in three classification categories (see Table 1): application scope of the technique, method engineering level, and contribution to the MuDForM objectives.

Each aspect is explained below, and an indication of the possible values is given. All aspects have a default value of "not addressed" which means that the study does not cover the aspect at all. Another possible value is "mentioned", which means that the aspect is recognized and possibly discussed, but that no clear contribution or solution is given.

Besides explaining all classification aspects, the next sections also state for each aspect (i) which research questions it helps answering and (ii) which MuDForM objectives it serves. A summary overview is also given in Table 1.

Application scope This category considers the context in which the specification technique is applicable, and helps to answer all three RQs. We classify the techniques on:

1. **Domain dependence**: We want to see if there are limitations to the domains to which the technique is applicable. This classification aspect is added to see how well existing techniques serve MuDForM objectives O2 and O3. Possible values: no specific domain, the name of a specific domain, or characteristics of the targeted domains. Although a technique might not be specific for a domain, we will also extract the domains of the examples in the study.

2. Their suitability for **quality domains**. We want to see if literature exists that shows how to apply domain modeling techniques to the domain of quality, and if this requires specific modeling concepts or modeling steps. This serves objective O2. Keep in mind that we are not looking for concepts that enable dealing with quality as a topic in the development process. For example, the distinction between functional requirements and non-functional requirements enables to deal with them separately, but just the distinction does not help to specify them differently. The literature might offer solutions for particular quality attributes or other classifications of non-functional requirements. These must be considered if they provide a specification technique.
   Possible values: any quality, a specific quality domain (e.g., as given by ISO/IEC25010 [69]), explicitly mentioned characteristics of dealing with quality (e.g., quality attribute scenarios as explained by Bass et al. in [57]).

3. Their usefulness in the definition of the **architecture** of a system. How does the technique fit in the context of architecture activities and architecture artefacts? We are specifically interested in how DO specifications are used to create a system in the targeted software environment, i.e., the software technologies and platforms that the system is supposed to operate on and connect to. Of course, this aspect helps to cover MuDForM objective O8. But it also serves O1 and O3.
   Possible values: an explicit architecture approach, a specific (possibly partial) match with ISO/IEEE42010, specific matches with architecture elements.

Method engineering We also classify studies on how well their approach or method is described and on how systematic it is. The aspects below are not specific for domain-oriented methods, but are relevant for all specification methods. This classification category serves MuDForM objective O7 and is relevant for answering all three RQs. We classify techniques on:

1. Support for **assuring the consistency** of a DO specification, or between DO specifications. This means, not just testing if a set of specifications is consistent, but defining specifications such that their consistency level is known at any moment and it is clear what must be done to achieve consistency. Mechanisms to prevent inconsistency are also contributing to this goal.
   Possible values: a specific mechanism to assure consistency (*e.g.,* fully based on a DSL or DM, or detection of elements that are used in several specifications).

2. How well they **provide traceability** from (intermediate) specifications back to the input. It must be possible to trace the decisions that led to a specification.
   Possible values: on model/document level, on smallest specification element level, an indication of somewhere in between, or a specific mechanism to provide traceability.

3. How well it helps to **detect incompleteness** in the targeted specification, and in the used input. A method should offer guidance in gathering knowledge about the (to be) modeled entity, for example by the use of standard types of questions for the involved (domain) experts, or questions that are a entry point for the analysis of input documents.
   Possible values: specific guidelines or steps for detecting and acquiring missing input information.

4. The **definition completeness** of the specification technique. According to Kronlöf [75] a method definition should provide:

   (a) An **underlying model**, *e.g.,* meta model, core model, or abstract syntax, of the specification technique, which forms the foundation for the semantics of a specification.
   Possible values: a specific meta model, set of concepts of the underlying model, a specific (meta) modeling language.

   (b) An explicit **notation**, possibly used in different viewpoints. All the viewpoints of the method should be defined in terms of the concepts of the underlying model.
   Possible values: specific viewpoints, notation descriptions, a specific language (like UML)

   (c) Explicit **method steps** that go through the viewpoints and that have clear entry criteria and exit criteria.
   Possible values: list or model of steps, comments about the relation to the viewpoints and/or about the granularity of the steps.

   (d) **Guidance** for taking steps and making specification decisions.
   Possible values: (reference to) a set of guidelines. These may be specific for each step or viewpoint.

5. **Formalness**: The degree to which the technique delivers formal specifications, and how it combines formal and informal specifications, *i.e.,* semi-formal specifications. A formal language has formal semantics and can potentially be processed in an automated way.
   Possible values: not formal, explicit formalism, via formal meta model, indication of hybridity, via model consistency rules, via unambiguous semantics.

MuDForM specific The MuDForM objectives defined in Section 2.3 could potentially be met by existing specification techniques. We discuss how well the identified studies serve the objectives and classify them on the following aspects:

1. **Domain-based**: The degree to which a specification technique uses a domain specification to define specifications in terms of that domain specification. DMs and DSLs are both considered as domain specifications that can be used to make other specifications. This aspect is added to the framework to serve objective O4 and to answer RQ2.
   Possible values: specification uses DS as terminology, specification is instance of DS, specific mechanisms to integrate specifications written in the same DSL or DM.

2. The degree to which the specification technique supports the specification of **structural** (static) properties, **behavior** (dynamic) properties, and their relation. Most techniques just cover either structural properties or behavioral properties. So, this classification aspect is particularly interesting when a study actually covers the integration of structural and behavioral properties. This aspect is added to evaluate objective O5.
   Possible values: static, dynamic, dynamics of statics, statics of dynamics, dynamics structures, possibly with additionally mentioned specification concepts for those. For example:

   (a) Static: classes, objects, entities, attributes, class associations, specializations.

   (b) Dynamic: activities, events, use cases, functions.

   (c) Dynamics of statics: operations of classes, activities per class, functions of a system.

   (d) Statics of dynamics: activity parameters, classes per activity, classes per use case, parameters of functions.

   (e) Dynamics structures: flows, process models, activity diagrams, state transition diagrams, petri nets.

3. Suitability for working with **multiple domains**. This aspect is added to the framework to address RQ3 and serves the evaluation of objective O1.
   Possible values: specific mechanisms for dealing with multiple domains. For example for specifying transformation/synchronization/consistency between domains, or for structuring domains into new domains, *e.g.,* via extension, merge, composition, or decomposition.

4. Support for **natural language**. This aspect is added to serve objective O6.

   (a) The degree to which they support texts in natural language **as input** for the specification process.
   Possible values: specific mechanisms to deal with concepts found in natural language texts. For example:

      i. Setting context for (domain) terminology, like books or reports in a series, articles, chapters, sections, and paragraphs. These are potential namespaces for the elements in specifications.

      ii. The processing of grammatical concepts like Subject, Noun, Predicate, Possessive case, Preposition, Phrase, Object, Number (amounts, singular, plural), Direct object, Gerund, Indirect object, Case, Collective noun, Comparative, Conjunctive, Infinitive, Imperative mood, Ordering events (in time), Adjectives, Adverbs, Appositive, Modifier, Classification, etc..

   (b) The degree to which DO specifications are **translatable back into text** in natural language and how well that text is still consistent with to the original input text.
   Possible values: specification mechanisms for translation of specification elements into text, indication of the degree to which semantics are lost in the translation.

4. Study Results

This section uses the classification framework described in Section 3.5 to organize and present our major observations. To this aim, we first extracted the data from each of the 53 primary studies through the perspective of each classification aspect. Then, we made a summary per aspect, as reported in Sections 4.2–4.4. We collected all the extracted data in one spreadsheet, which is part of the replication package [60]. For easy reference, at the end of this paper we have provided the List of Primary Studies with reference numbers [1] through [53]. Hence in the following, the first 53 references indicate primary studies. First, we discuss our observation regarding the publication trends.

Table 2, which has the same structure as Table 1, shows which primary studies address each aspect.

Table 2
Primary Studies per Classification Aspect

CLASSIFICATION CATEGORY	CLASSIFICATION ASPECT	Addressed by PRIMARY STUDIES
Application scope	Domain dependence	-
	Quality domains	-
	Architecture	[47, 5, 53, 17, 46, 45, 48, 28, 29, 41, 22, 32, 34, 7, 31, 18, 51, 33]
Method engineering	Assuring consistency	[29, 9, 40, 16]
	Provide traceability	[5, 4, 36]
	Detect incompleteness	[37, 38, 29]
	Definition completeness	
	Underlying model	[42, 2, 10, 20, 37, 39, 38, 5, 50, 4, 53, 36, 25, 3, 21, 48, 28, 9, 11, 34, 30, 8, 40, 19, 51]
	Notation	[20, 37, 38, 5, 50, 4, 23, 53, 36, 3, 17, 24, 21, 48, 43, 28, 29, 9, 12, 41, 22, 11, 8, 31, 27, 52, 19, 18, 51, 16]
	Method steps	[42, 47, 20, 39, 5, 35, 50, 23, 53, 36, 6, 25, 24, 43, 28, 26, 29, 9, 12, 41, 22, 32, 7, 27, 40, 1, 19]
	Guidance	[10, 39, 5, 50, 4, 36, 17, 46, 43, 28, 29, 12, 41, 15, 31, 27, 52, 1, 18, 51, 16]
	Formalness	[44, 25, 21, 26, 41, 11, 14, 8, 33]
MuDForM specific	Domain-based	[37, 39, 38, 25, 48, 29, 33]
	Structural and behavioral	[38, 17, 21, 43, 29, 12, 41, 52, 1, 33]
	Multiple domains	[2, 10, 44, 39, 36, 6, 3, 46, 48, 26, 9, 22, 32, 11, 13, 34, 14, 30, 49, 15, 8, 31, 40, 19, 51, 16]
	Natural language	
	As input	[35, 4, 29, 12, 41, 27, 52, 1]
	Translatable back into text	[35, 29, 52]

4.1. Publication Trends

Figure 4 shows the total number of included studies per publication type (in the y-axis) and their distribution over time according to their year of publication (in the x-axis). We observe an increase after 2002, with peaks in 2004 and 2009. Studies before 2000 are about DM approaches and not about DSLs. An explanation is suggested by Czech *et al.* [59], because they state that the term DSL did not exist before 2000. However, Kosar *et al.* say that there was a Usenix conference in 1997 on DSLs [74]. Prieto-Díaz states in his 1990 paper [81] that a domain language, preferably formal, is one of the outputs of domain analysis. Nascimento *et al.* [77] say that the idea of DSLs was already published in 1965, but that the term domain or DSL was not used. After 2000, the studies cover the whole DSL development process, in which the creation of a DM is positioned as one of the DSL development phases. Consequently, DM creation receives less attention than before 2000. Though, Chaudhuri *et al.* [7] state in their 2019 paper that there is still not much literature about creating the abstract syntax of a DSL. In the last decade, the included studies' topics have also shifted towards issues related to multiple domains, and to multiple models in general.

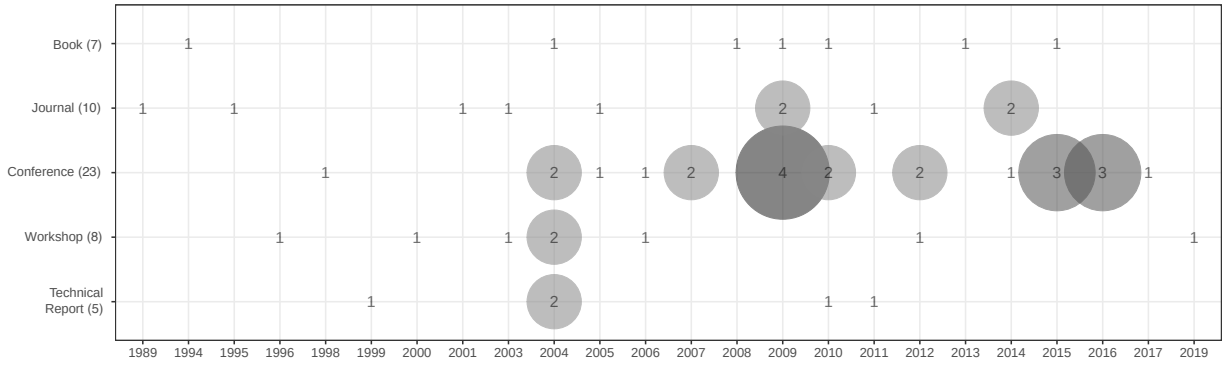


Figure 4: Publication Trends - Venues of the Years

Concerning the types of publications, Figure 4 shows that most studies are peer-reviewed scientific works (41/53 are conference-, workshop- or journal papers). A significant number of books (7) and technical reports (5) were providing useful insights.

We also looked (in Figure 5) at the publication trends with respect to the coverage of the aspects over the years. The Figure emphasizes three clusters around 2004, 2009 and 2015. All are centered around aspects of method completeness and multiple domains; with a growing attention for guidance. We notice that the topics of notation, guidance, and

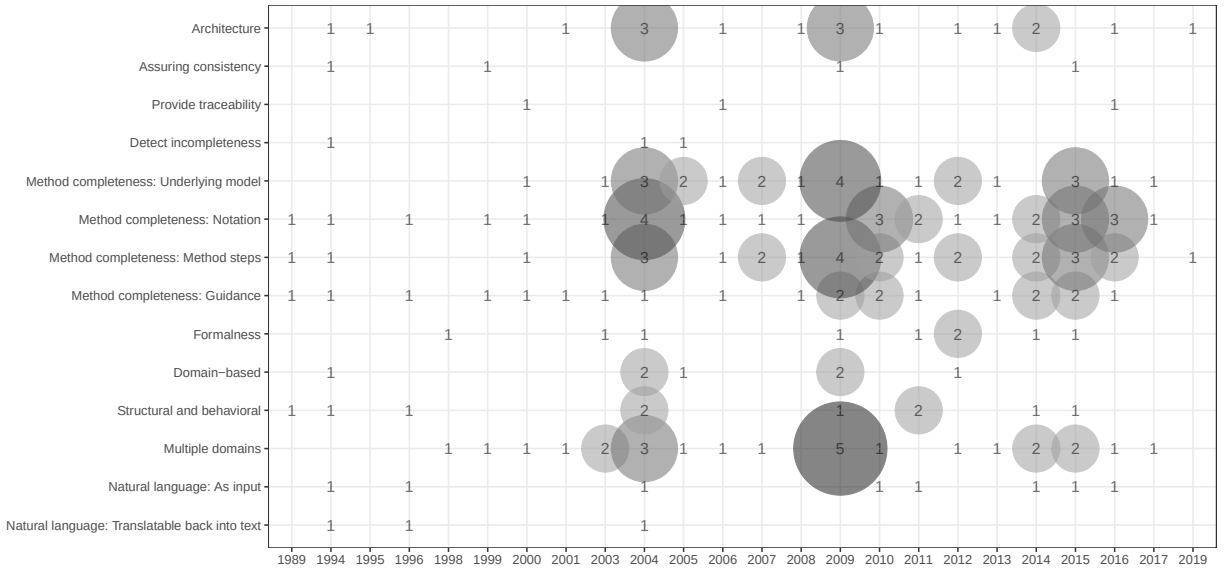


Figure 5: Publication Trends – Aspects over the Years

multiple domains are addressed throughout the whole time span.

We have also looked at the application domains of the examples or case studies in the studies. We found that only six studies [5, 3, 9, 11, 51, 28] have real case studies or examples. None of the studies mentions the business or organization in which the techniques are used. Several studies [24, 26, 22, 52, 35, 46, 45, 14, 49, 27] have no demonstrative example of how their specification techniques are used. The rest has either small illustrative examples or a running example throughout the study.

We found that a banking example is used the most (six times) in the 53 studies. But the examples are all slightly different. It might be a good idea having a reference (banking) case description that can be used by over and over in different studies. This would save time in case development and in understanding the application of whatever concept is the topic of research. Examples about processes for reserving, ordering, paying, and delivering products or services are also used regularly. Another category of examples concerns the software domain itself, like the example about components and deployment in [13], or the transformation from Petri Nets to Statecharts in [30] and [31]. The examples in [38, 7, 43] and some examples in [28] are more about, or closely related to, embedded software.

We also looked for correlations between the domains and the other aspects of the classification framework, but we found no significant ones.

4.2. Application scope

Domain dependence. All approaches and methods of the selected studies are domain independent. Though, some studies [45, 9, 44, 48, 43, 7] explicitly limit themselves to the specification of software systems, by seeing a DSL as a system specification language, or as a programming language [18]. All studies aim at specifying the application domain or at the functionality that the system should provide for the application domain.

None of the studies addresses resource domains or quality domains. Though, some have examples covering the resource domain, like the example about component deployment and hardware in [13].

Suitability for quality domains. There is no method that is specifically targeted at the specification of quality. All studies demonstrate their technique via specifications of the application domain, *e.g.,* banking, or the system domain, *e.g.,* components and interfaces. It seems that the explicit specification of quality is simply ignored or avoided. Kelly and Tolvanen [28] mention that domain-specific modeling is leading to higher quality, but they do not explain how or provide specification techniques for quality. We have found some examples of DMs for a specific quality, like the security DM by Firesmith [64], which is used to specify security requirements. However, those examples are excluded, as they do not contribute any specification technique.

Relation to architecture. Most studies do not address the relation to (software) architecture, and none of them deals with the architecture of systems that are specified via multiple DSs. Some studies, *e.g.,* [47], distinguish an explicit step from a DS to its implementation in a targeted software environment, but they do not explain how to specify the transformation, what detailed steps to take, or what guidelines to follow.

Some studies use explicit mechanisms to transform DO specifications into a specification that can be executed in the target (software) environment. We found the following types of mechanisms:

- Some approaches, *e.g.,* [42, 39], are UML based and, as such, can build upon the use of UML for modeling a software design. These approaches use classes as the main modeling concept, and suggest that a system design is based on the modeled class structure.

- Some approaches, *e.g.,* [17], model the application domain and prescribe that the derived software system has elements that correspond with elements of the created DS. Sagar and Abirami [41] describe a specification technique for functional requirements, and they suggest that those requirements correspond with functions of the system. This kind of use of a DS can be seen as an architectural style or design pattern.

- Some approaches, *e.g.,* [29, 28], give examples of transformations from a DS to a target environment, such as a relational database or user interface library. The study of Zhang et al. [53] follows the structure of model driven architecture [78], which has a step for the transformation of a platform independent (domain) model to a platform specific model.

- Some approaches, *e.g.,* [48, 7], focus on DSLs that specify parts of a system design. A clear example is the DSL for component and interface specification in [46]. In these cases, the DSL itself can be seen as a design pattern for a software system, because the system structure follows the structure of the DSL.

In general, methods with an underlying (meta) model with clear (formal) semantics, *e.g.,* [19, 20] are easier to embed in an architecture (pattern), because they enable a formal transformation of the DS into software.

4.3. Method engineering

Support for assuring consistency. Some studies, *e.g.,* [9, 28, 32], mention consistency, but do not provide mechanisms to achieve it. Evans [16] provides different approaches for consistency across domains, which can serve as techniques to make DO integration specifications on top of DSs. Romero et. al. [40] describe an approach for achieving consistency across viewpoints, which is based on the use of correspondences, similar to the concept of correspondence in the ISO/IEC 42010 standard [68].

The KISS method [29] proactively supports consistency via method steps and guidelines that iterate over the different views on one model. The guidelines prescribe how existing views are used as a starting point for creating a new view, and how changes in one view impact other views, without providing an explicit metamodel.

Traceability (to input). None of the studies addresses traceability explicitly. But studies that provide fine-grained method steps, an underlying (meta) model, or guidelines for modeling decisions, offer support implicitly. Namely, to achieve a traceable process, changes to the specification must be logged, preferably with a rationale. First, if the method provides method steps at the level of specification changes, then these steps can be used as the type of the changes. Second, if a meta model is given, then changes can be defined in terms of create, update, delete actions of instances of the meta model. Third, guidelines can serve as rationales for logged specification decisions.

Detect incompleteness. None of the studies explicitly addresses the detection of incompleteness in the used input. The approaches that see a DM as an abstraction of a set of systems, *e.g.,* [38], help in achieving completeness by explicitly checking if all DM elements are used in a specific application model. Some studies prescribe the presence of multiple viewpoints in one model, *e.g.,* [29]. This might result in the detection of missing information in the input of the modeling process, which may lead to requests for extra input from the domain experts, and as such contributes to the completeness of a specification.

Method completeness: underlying model. Most studies use UML or MOF as their underlying model. Some have their own meta model like KerMeta [19] or MEMO ML [20, 21]. The major observation here is that most meta models limit themselves to structural concepts like classes, attributes, and relations between classes. ORM [35], Normalized Systems [33], and the KISS method [29] offer also behavioral concepts, but do not provide a meta model.

Method completeness: notation (syntax and viewpoints). In the selected studies, UML is used most often as a graphical notation. Its usage is mostly limited to a class diagram, sometimes extended with OCL for specifying rules on

top of the classes. Some studies, *e.g.,* [8, 10, 16, 15], use packages and package diagrams to model relations between domains.

Only graphical DSLs sometimes offer more than one viewpoint. In case of a textual DSL, only one viewpoint is presented. This view is the complete textual specification of a model in terms of the DSL. For example, they do not even distinguish header files and implementation files as different viewpoints.

Most studies show a notation in their examples, or prescribe a step for explicitly defining a notation. Chaudhuri *et al.* [7] do not address notation, because they explicitly restrict themselves to the abstract syntax or meta model.

Method completeness: method steps. Many studies provide a step for making a DS. The steps of most approaches are course grained, and reflect phases or stages in the specification process. The only study that offers fine-grained steps for making a model is from Ibrahim and Ahmad [27]. We did not find any other approach that has steps at the fine-grained level of modeling concepts like class or attribute. The KISS method [29] provides fine-grained steps for grammatical analysis on an input text to come to an initial model. But after grammatical analysis, the model engineering phase is defined by steps at the level of viewpoints. As mentioned before in the observations about traceability, a meta model offers implicit modeling steps via the creation, update, deletion of modeling concept instances.

Some DSL approaches, *e.g.,* [42, 47, 20, 39, 50, 24, 43, 28, 7], have a step for creating a domain model, meta model, or abstract syntax. But they do not go into detail on how to do that, or how to derive a DSL from the created model.

Method completeness: guidance. The studies about patterns for DSL design [46], for DSL implementation [18], for DM integration [16], and for model transformations [31] all provide guidelines for choosing between patterns. Remarkable is that Frank *et al.* [20] state that the creation of a domain language is a demanding task, mostly performed by highly specialized experts, and good guidance is currently missing. This contradicts slightly with the elaborate guidelines given in the studies that are about the processing of natural language to make a (domain) specification [1, 4, 27, 41, 29, 12, 52].

None of the studies offers guidelines for all modeling steps or for all prescribed views. This means that the predictability of a process that follows such an approach is low because it strongly depends on the expertise and domain knowledge of the modeler.

Formalness. Most studies do not mention how formal their specification techniques are. The approaches that use UML as their underlying model can be seen as partially formal, depending on the part of UML that is used.

Simos and Anthony [44] and Frank [21] present languages with a formal meta model. Kelly and Tolvanen [28] state that all DSLs must be formal, so they can be parsed and used for generating code. Of course, all DSs that are used to generate software, must be unambiguously parsable.

Golra *et al.* [23] explicitly choose informal modeling in their approach for developing DSLs. Though, they state that a DSL itself implicitly has formal semantics via its implementation in software.

4.4. MuDForM specific

Domain-based specifications. Most studies that discuss the creation of domain-based specifications, consider an application model to be an instance of a DS. We are not interested in these types of approaches because of the same reasons as given in exclusion criterion E5. None of the studies that see a DS as a language to define other specifications, provides steps and guidance to do so. The KISS method [29] and the Normalized systems approach [33] use the DM to specify functions, processes, or workflows. They both provide examples, but no explicit steps and guidelines are given.

Some approaches [4, 51] show examples of requirements specifications in terms of a DSL, but they do not provide steps and guidelines for creating them.

Integrated structural and behavioral properties. Most approaches only cover structural properties (classes, attributes, relations between classes). Some approaches [38, 17, 21, 43, 12, 41, 1] also provide behavioral concepts of structural elements (operations of classes). Reinhartz-Berger *et al.* [37] focus purely on behavior, via activity diagrams. Some studies [5, 53, 25, 17, 24] offer behavioral concepts and structural concepts, but they do not explain their coherence. These are mostly UML based studies, which use classes for structural properties, and use cases or activities for behavioral properties, but do not explain how to relate them to each other.

Only the KISS method [29] and the Normalized systems approach [33] offer autonomous modeling concepts, method steps, and guidance for specifying structural properties, behavioral properties, and the relation between them.

Suitability for working with multiple domains. We did not find a study that offers methodical support for working with multiple domains. Many studies [46, 32, 2, 10, 36, 6, 3, 26, 22, 11, 13, 34, 14, 30, 49, 15, 8, 19] offer mechanisms for dealing with multiple models and their integration, but those mechanisms are not explicitly merged with the specification technique that was used to create the original models. A method to create a DS or domain-based

specification could provide certain model properties which make the models easier to integrate, *e.g.,* being in the third normal form, or being context free. The found studies mostly offer techniques based on relations between packages, like merge, refine, reference, specialization, assembly, instantiation, and unification.

We did not find studies that use consistency rules or correspondence rules as a technique to specify domain-oriented integrations. All studies either combine two DSs into a new DS, or define transformations from a source DS to a target DS.

Natural language as input. Several studies [29, 41, 1, 52, 27, 12] offer explicit steps and guidelines for transforming natural language text into model elements, but most studies do not. Some studies mention the involvement of domain experts and that their "words" become terms in a model, *e.g.,* [35, 50, 17, 28], but they do not explain how to do this systematically, *i.e.,* with clearly defined method steps and guidelines for eliciting knowledge from domain experts.

Translatable to natural language. Most studies do not address the translation of specification into natural language. The study by Hoppenbrouwers *et al.* [26] offers the paraphrasing of all model parts in natural language. Of course, any specification written in a defined language can be spoken in natural language by simply reading the specification in terms of the specification language literals. Specification languages that have concepts that are close to natural language, like in the KISS method [29] and the Normalized System approach [33], are more suitable, because they offer an easier transition from text to model and back.

5. Discussion

In the following we discuss our observations in relation to the three research questions (Sections 5.1–5.3) followed by additional observations that emerged from the results (Sections 5.4–5.6).

5.1. RQ1: No fully engineered Method

We did not find any method that was engineered in full, *i.e.,* with a good answer to all the method engineering aspects of the classification framework, or even for just the aspects of method completeness.

Most of the studies do not address consistency. As such, one cannot assume anything about the well-formedness and consistency of a made specification. This could be acceptable when specifications are only used in an informal way. But if specifications are used to create other specifications, then consistency rules and how well a specification meets them, are important. None of the studies provides explicit guidance for detecting incompleteness of specifications. The effect is that the completeness of a specification depends on either the proactive attitude of the involved (domain) experts to bring in missing information, or that the modeler herself has complementary domain knowledge that allows her to ask the right questions. Method steps could provide guidance for asking the right questions to the involved experts or for searching the input documents for a specific type of information.

None of the found methods has a complete definition (*i.e.,* covering underlying model, notation, steps, and guidance). Some approaches provide a meta model and a notation, and others provide high level steps, sometimes guidance, and sometimes the use of an existing language like UML. The lack of an underlying model makes it hard to have an unambiguous well-defined interpretation of models, and difficult to separate the semantics from the syntax. If there is a language defined, but no steps and guidance, then modelers must be very experienced, and the specification process becomes unpredictable. Moreover, it is impossible to create a tool that guides the modeler through the modeling process. And, if such a tool is made, then the tool builder has (implicitly) decided about the steps and guidance, which might result in a sub-optimal specification process.

5.2. RQ2: No methodical support for applying a created DSL or DM

The approaches in the included studies that are about DO specification techniques, are mostly limited to creating a DS. They typically do not incorporate steps and guidance for applying a created DS. So, none of the studies proposes an approach that prescribes how to use a DS that is created with that approach. In the literature there are publications about the usage of a specific DS, *e.g.,* [64] and the many examples from [62]; these, however, are specific for the DS at hand, and as such are excluded as primary studies, because they do not contribute a specification technique.

We think that an approach for creating a DS should also take into account that the DS is applied in a proper way. We looked separately for literature about the application of a created DS and found some studies on evaluating the usability of a created DSL from Barivšić *et al.* [55, 56, 54]. They state that evaluation of DSLs does not happen often. Gabriel *et al.* [65] state that the community in software language engineering does not systematically report on the experimental validation of the languages it builds. Barišić *et al.* also state that DSL usability testing can be costly,

but that poor usability is more costly in the long run. Rodrigues *et al.* [82] conclude in their SLR that there is a lack of systematic approaches to evaluate DSLs, and notice that most authors do not report problems with the language they created. Gray *et al.* [66] write that "poor documentation and training [of a DSL]" is one of the 10 reasons why DSLs are not used more in industry. Völter [85] states that it is important to communicate to users how the DSL works and documentation should be example-driven or task-based. He even observes that in non-scientific domains, domain experts are not expected to be the ones that specify systems with the DSL. Instead, the domain experts pairs up with the DSL developer to apply the DSL, or the DSL developer does all the specification based on discussions with the domain expert.

The lack of methodical guidance for applying a created DS is risky. The creators of a DS might know precisely its semantics and also how to use it. But targeted users of the DS probably do not, and should be instructed. Although not specifically mentioned in the found literature, we have seen this many times in industry projects. A DSL is defined, but only the DSL creators understand how to make models with it, or understand the exact semantics. The effect is that the DSL is not easy to learn by the targeted users, and that they have to learn it via examples or personal guidance from the creators. Even if such support is present, then the available examples might not exploit all the features of the DSL and the creators might not be available for guidance, or assign different semantics to the DS. The effect is that the DS is not optimally used, and often leads to the perception that investments in domain-specific modeling do not pay off. We think this lack of methodological guidance is one of the most important shortcomings in the literature on domain-oriented methods, which is subscribed by Gray *et al.* [66] and Barivšić *et al.* [54].

The potential advantage of having DS creation and DS usage in the same method is that the method part about usage can use the method engineering properties of a created DS. For example, knowing that a DSL has a context-free grammar makes the parsing of models in terms of that DSL predictable. Or, if the behavior in a domain is specified via atomic actions that form the only way to change objects, then those actions may form building blocks for specifying the steps in a process model. In the design of MuDForM, we could benefit from the studies about specifying requirements in terms of a given DS [4, 29, 51, 33].

5.3. RQ3: No integral Support for Multiple Domains

In our view, a (software) system specification can be seen as an integration of multiple domain specifications and domain-based specifications. A method to realize this view, should guide the creation of those specifications as well as their integration. We did not find such a method. We found studies that offer a single technique to integrate two specifications, *e.g.,* [19, 8], but there is no study that offers techniques that aim at integrating the specifications of more than two domains. Dynamic mechanisms, like transformation and runtime weaving, can only work in such a context, if they are based on specifications of consistency between domain specifications and domain-based specifications. A number of methods mention aspect weaving as a mechanism to integrate specifications, *e.g.,* [44, 11]. Weaving could be used to integrate functionality with other quality attributes, and thus can be seen as one of the options to integrate domains. However, none of the studies explains how to do it. The studies that offer techniques for integrating models in general, and not DMs specifically, *e.g.,* [16, 34] do not benefit from the properties that a well-defined language and method can offer, because they cannot assume anything about the method engineering properties of the models they integrate. For example, if two DMs are normalized in the 3^{rd} normal form, then their integration is easier, and the resulting model is also easier to normalize.

5.4. Behavior is mostly ignored and poorly integrated with structure

Although all studies are domain independent, they do not offer the modeling concepts for the same aspects of a domain, and vary in how well and how clear those concepts are integrated with each other. Most studies offer concepts for structural properties (classes, attributes, associations, specializations); some are centered around behavior via concepts like processes, functions, or features. This means that the suitability of such methods highly depends on the aspects that are relevant in the modeled domain. If a domain is mainly data-centric, then a method that has mainly concepts for structural properties will probably suffice. Similarly, if a domain is mainly characterized by behavioral aspects, then processes or functions could suffice as the central modeling concept. But if a domain has both structural and behavioral properties, then most methods will not suffice. To our knowledge, only the KISS method [29] offers a language that enables treating both structural and behavioral properties, and offers modeling concepts, steps, and guidelines to relate those properties in a consistent way.

5.5. Minimal Interface with Natural Language

Some methods provide guidelines for the transformation of natural language text into a model in a specification language. Because natural language has evolved over thousands of years, it contains concepts and grammar that suit the way humans want to communicate about the world (domain) around them. According to Chomsky [58] and elaborated by Pinker [80], humans have an innate capability to process natural language.

In general, a modeling language and natural language are not isomorphic, leading to differences in their expressiveness. The effect is that the verbalization of model parts does not resemble the input text that was used to make the model. Therefore, the translation of a specification into natural language text suffers from loss of semantics with respect to the original input text. The most occurring symptom of this shortcoming is the use of entities or classes in the model to represent verbs from the text, like in [41, 12]. In such a model it is not clear which classes correspond with a noun and which classes correspond with a verb. So, you cannot generate a sentence that expresses the original meaning, in the case that a class was derived from a verb.

We think that a specification language for creating DO specifications should have concepts that are close to human reasoning, observation, and communication. This might be offered partially by natural language. But, more research is needed to decide which natural language concepts can be used and which additional concepts are needed to define the specification language for a method like MuDForM.

5.6. The Terminology around Domain Models is not unified

During our work for this SLR, we discovered the use of different terminologies related to domain modeling. In the context of DSLs, a DM is comparable to the abstract syntax of a DSL. Both can be seen as a graph with domain concepts. In the studies that discuss the design of a DSL, a DM is often mentioned as the starting point for defining an abstract syntax. However, we did not find any study that offers a completely defined method for the transformation of domain knowledge into an abstract syntax.

We learned that the term *domain analysis* refers to the activity that leads to a DM. As such, publications about domain analysis might contribute more in-depth insights related to RQ1.

As already discussed in Section 3.1, where we defined RQ1, a DM and a domain ontology are comparable. If one would create both for a specific domain, then one would end up with two structures that have a high overlap in the used terms.

To be complete, also the terms *underlying model* (as we use it in our classification framework in Section 3.5) and *meta model* are sometimes used when a DS is used to create another specification, *e.g.,* in code generation. We think it is a good idea to create an overview, possibly via a domain model, of all these related terms and their meaning, to be able to understand the existing literature better and to decrease ambiguity. To this aim, we find it valuable to carry out a follow-up study specifically on the literature in domain analysis and ontology engineering.

6. Related Work

One of the most cited papers on DSLs is "When and How to Develop Domain-specific Languages" of Mernik *et al.* [76]. This paper defines 5 phases of DSL development: decision, analysis, design, implementation, and deployment. In this SLR, we specifically focus on analysis and design. The implementation phase is addressed in this SLR via the integration of multiple domains. Namely, we consider the implementation of a DSL or DM if it is realized as an DO integration specification between a source domain and target domain. We also consider the phase of usage, *i.e.,* to apply a DS in the specification of a system (or parts or aspects of it). We did not find DSL-related literature about this phase.

The study from Nascimento *et al.* [77] classifies DSLs into domains. Their domain classification contains mainly different types of software systems, e.g., control systems, web applications, embedded systems, and parallel computing. This is logical if a DSL is seen as a system specification language, but not if a DSL is seen as a language for making statements about a domain, like we do. Kosar *et al.* [74] also present a systematic mapping study on DSLs. They observe that 59% of their included primary studies mention domain analysis as a step in the creation of a DSL. Of those studies, only 6% used a formal analysis approach. They conclude that domain analysis is mostly done in an informal way and incomplete. They mention that one of the reasons for this weakness is that domain analysis is too complex and outside software engineers' capabilities. This might be because that study seems to see a DSL as software creation language, instead of a domain knowledge capturing technique, or as a specification language in general. We think that domain analysis is not a software engineer's task. With MuDForM, we explicitly aim at making the domain analysis

phase a systematic activity, with explicit steps and guidance, and a formal underlying model which makes it more predictable, less complex, and easier to learn.

Czech *et al.* [59] gathered best practices for domain-specific modeling (DSM). They state explicitly that domain-specific modeling aims at generating code and raising the level of abstraction beyond programming. They distilled 130 best practices from 19 studies. They grouped the best practices in different classes that each indicate an aspect of a DSM solution: domain model, language design and concepts, generators, DSL-tooling, meta-model tooling, and practices that concern an entire DSM-solution. Only 3 best practices are about the domain model. These practices are not about modeling itself, but about the context of a DM. We went through all best practices that are about the language concepts. We observe that they did not find and distill best practices for applying a DSL, nor for dealing with multiple domains, which corresponds with our discussion in Sections 5.4 and 5.5. Also, this study has a programming-oriented perspective on modeling and DSLs, whereas our perspective is more focused on expressing, formalizing, and applying domain knowledge and inter-domain knowledge, in several types of DO specifications.

Iung *et al.* [70] published a systematic mapping study (SMS) on DSL development tools. We think that that SMS is complementary to this SLR, because they both investigate literature on DO specifications, but each from a different perspective. There is only one overlap in the classification frameworks, namely with respect to "Notation". Iung *et al.* consider if there is support for graphical or textual specification, while we consider the actual notation of the specification language itself.

Torres *et al.* [84] published an SLR on cross-domain model consistency checking by model management tools. They use a totally different notion of domain as the one discussed in Section 2.1. Their notion of domain relates to the engineering discipline of the modeler, such as electrical, mechanical, or software engineering. Their notion of consistency is not about different related aspects in one or more models, but about how models from different engineering backgrounds actually fit together in the structure and behavior that they describe. This notion of consistency between models resembles interface matching between components. As such, this type of consistency addresses questions like: are the parameters the same? do they have the same types? does the behavior match? Differently, we consider consistency between models as a property that is specified by consistency rules that can address any specification aspect.

7. Threats to Validity

This SLR has similar threats to validity as other SLRs that followed the guidelines of Kitchenham [73] and Wohlin [86, 71]. We took a number of actions to make sure we selected a proper set of primary studies and achived valid results.

To assure that we defined the right search strings, we did an initial scan of the literature that we knew, we made concepts models of the found terminology in that literature, as presented in Sections 3.1 and 3.2. For feasibility reasons, initially we only searched in the titles of the studies; therefore we cannot rule out that we missed out studies that report contributions to specification techniques. To mitigate this threat, we applied snowballing, in which we did not pose the limit to publications with the terms domain model or domain-specific language (and derivatives) in the title. Moreover, through the inclusion of relevant books, and the inclusion of studies from the initial informal search, we are confident that we have covered a significant knowledge base.

The exclusion of relevant studies during the selection process is another potential threat to validity. We mitigated this threat by defining clear exclusion criteria and a multistage screening process. When we doubted the inclusion of a study after reading the abstract, introduction, and conclusion, we first included it and then read it fully. We excluded it only when it became clear, during the data extraction, that such a study did not provide relevant information. If not, both authors read the paper, and decided together. Though, for classification aspects that are mostly not addressed explicitly in the included studies, like *assuring consistency*, *providing traceability*, or *detecting incompleteness*, we analyzed if a study contributed indirectly, and reported that analysis. Finally, the objectives defined in Section 2.3 were used as a yardstick in the discussion in Section 5 to come to an analysis that is relevant for the domain-oriented development method that we envision, *i.e.,* MuDForM.

8. Conclusions and Future Work

This paper describes a systematic review of the literature on domain-oriented specification techniques, through the perspective of our vision on system specification. It provides an overview of the state-of-the-art in specification techniques to create domain models (DMs) and domain-specific languages (DSLs), and their use in the creation of other specifications. We were interested in studying research questions on specification techniques to: *create specifications of*

domains (**RQ1**), *create specifications in terms of domain specifications* (**RQ2**), and *create specifications of integrations between domain specifications, and between domain-based specifications* (**RQ3**). This SLR also identifies shortcomings in the existing literature on domain-oriented specification techniques. We found that no method covers all the method engineering aspects framed in our classification framework. Also, most studies focus on creating DSs, and none provides guidance to actually use the created DS. Similarly, no method for creating DSs has a method part that addresses how to deal with multiple domains. This implies that dealing with the application of the DS, and the integration of multiple DSs, is left to be addressed in a specific development context.

These results provide an important foundation for our future work on MuDForM. As described in the beginning of this paper and framed in the classification framework used to review the existing literature, we aim to design a method that is well engineered, covers the creation of DMs, the use of DMs to create other specifications, the integration of multiple DMs and domain-based specifications, and that is closely connected to natural language.

In particular, following the results of this SLR we envisage the following research topics and activities:

- The development of an underlying model (meta model), method steps, and guidelines:
  - We think that the starting point can be the KISS method for Object Orientation [29], because it is the only method that serves objectives O3, O4, O5, and O6.
  - The method part for DO specification integration has to be designed from scratch, because there is not one candidate study that offers a good foundation. But we think that the UML package concept, combined with integration specification options to serve objectives O1 and O2, is a good starting point. The studies mentioned under "Suitability for working with multiple domains" in Section 4.4 offer several options for DO integration specifications.
  - We can also use the existing guidelines for processing natural language, mentioned under "Natural language as input" in Section 4.4. This serves objective O6.
  - We will validate MuDForM in a number of industry cases.
- We think a DO specification method should be aligned with human perception, thinking, and reasoning, which means it should be based on primitives that are close to human cognition. Natural language fulfills this vision to a certain extent, but not completely. We plan to investigate the literature from cognitive sciences to analyze what those cognitive concepts could be, identify which ones do not have a clear natural language counterpart, and extent MuDForM accordingly. For example, Hofstadter and Sander [67] explain and reason that analogies lie at the core of human thinking. We will investigate how this concept can be supported in a specification method.
- As mentioned in Section 5.6, we will further investigate the literature on domain analysis to understand if we can use method ingredients from existing methods. If needed, we will start an SLR on this topic, to benefit from existing methods and to justify MuDForM design decisions. We already looked at the approaches from Shlaer and Mellor [43], FODA [72], and ODM [45], which are mentioned as the most common domain engineering methods by van Deursen *et al.* [62]. DSSA [83] is also often mentioned in literature, but it does not offer a specific solution for making specifications, and as such did not qualify as a primary study for this SLR.
- We also mentioned in Section 5.6 that it would help researchers and domain engineers if a clear overview of domain-related terminology would be created. This involves at least the terms domain model, ontology, conceptual model, feature model, DSL, abstract syntax, meta model, and underlying model.

Finally, we have defined a reusable approach for comparing methods in an SLR. First, we clarified our overall (MuDForM) research objectives, and we made models of the method domain and the compared methods' application domain. After that, we used the objectives and models to define the research questions, the search queries, and the classification framework. Our approach is an extension to the well-established standard for literature reviews described by Kitchenham [73], and can be used by others who want to perform an SLR or method comparison.

Acknowledgements

We are grateful to the colleagues and peers that reviewed the various versions of this paper and provided feedback on the work (in alphabetical order): Bonne van Dijk, Wan Fokkink, Marc Hamilton, Arjan van Krimpen, Ivano Malavolta, Daan Pasmans, Jan Schoonderbeek, and Eric Suijs.

List of Primary Studies

[1] Abirami, S., Shankari, G., Akshaya, S., Sithika, M., 2015. Conceptual modeling of non-functional requirements from natural language text, in: Jain, L.C., Behera, H.S., Mandal, J.K., Mohapatra, D.P. (Eds.), Computational Intelligence in Data Mining - Volume 3, Springer India, New Delhi. pp. 1–11.

[2] Abouzahra, A., Bézivin, J., Didonet, M., Fabro, D., Jouault, F., 2005. A practical approach to bridging domain specific languages with uml profiles, in: Workshop on Best Practices for Model Driven Software Development, OOPSLA.

[3] Abouzahra, A., Sabraoui, A., Afdel, K., 2017. A metamodel composition driven approach to design new domain specific modeling languages, in: 2017 European Conference on Electrical Engineering and Computer Science (EECS), IEEE. pp. 112–118.

[4] Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., 2016. Extracting domain models from natural-language requirements: Approach and industrial evaluation, in: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Association for Computing Machinery. pp. 250–260.

[5] Asnina, E., 2006. The formal approach to problem domain modelling within model driven architecture, in: 9th International Conference on Information Systems Implementation and Modelling, pp. 97–104.

[6] Campos, E., Kulesza, U., Freire, M., Aranha, E., 2014. A generative development method with multiple domain-specific languages, in: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., Raatikainen, M. (Eds.), Product-Focused Software Process Improvement, Springer International Publishing, Cham. pp. 178–193.

[7] Chaudhuri, R., S., Natarajan, S., Banerjee, A., Choppella, V., 2019. Methodology to develop domain specific modeling languages, in: Proceedings of the 17th ACM SIGPLAN International Workshop on Domain-Specific Modeling, pp. 1–10.

[8] Clark, T., Evans, A., Kent, S., 2003. Aspect-oriented metamodelling. The Computer Journal 46, 566–577.

[9] Clark, T., Sammut, P., Willans, J.S., 2015. Applied metamodelling: A foundation for language driven development (third edition). CoRR abs/1505.00149. URL: http://arxiv.org/abs/1505.00149, arXiv:1505.00149.

[10] Cuadrado, J.S., Molina, J.G., 2009. A model-based approach to families of embedded domain-specific languages. IEEE Transactions on Software Engineering 35, 825–840.

[11] Degueule, T., Combemale, B., Blouin, A., Barais, O., Jézéquel, J.M., 2015. Melange: A meta-language for modular and reusable development of dsls, in: Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering, ACM. pp. 25–36.

[12] Elbendak, M., Vickers, P., Rossiter, N., 2011. Parsed use case descriptions as a basis for object-oriented class model generation. Journal of Systems and Software 87, 1209–1223.

[13] Emerson, M., Sztipanovits, J., October 2006. Techniques for metamodel composition, in: OOPSLA, 6th workshop on domain specific modeling, ACM Press. pp. 122–139.

[14] Erdweg, S., Giarrusso, P.G., Rendel, T., 2012. Language composition untangled, in: Proceedings of the Twelfth Workshop on Language Descriptions, Tools, and Applications, pp. 1–8.

[15] Evans, A., Maskeri, G., Sammut, P., Willans, J.S., 2003. Building families of languages for model-driven system development, in: Workshop in Software Model Engineering, San Francisco, CA, Citeseer. pp. 1–9.

[16] Evans, E., 1999. Deconstructing the Domain: A Pattern Language for Handling Large Object Models. Technical Report. citeseer.

[17] Evans, E., 2004. Domain-Driven Design: Tackling Complexity in the Heart of software. Addison-Wesley.

[18] Fowler, M., 2010. Domain specific languages. Addison-Wesley Professional.

[19] France, R., Fleurey, F., Reddy, R., Baudry, B., Ghosh, S., 2007. Providing support for model composition in metamodels, in: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), IEEE. pp. 253–253.

[20] Frank, U., 2010. Outline of a method for designing domain-specific modelling languages. Technical Report. Universität Duisburg-Essen, Institut für Informatik und Wirtschaftsinformatik (ICB).

[21] Frank, U., 2011. The MEMO Meta Modelling Language (MML) and Language Architecture. Technical Report. Universität Duisburg-Essen, Institut für Informatik und Wirtschaftsinformatik (ICB).

[22] Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C., 2016a. Addressing modularity for heterogeneous multi-model systems using model federation, in: In Companion Proceedings of the 15th International Conference on Modularity, ACM. pp. 206–211.

[23] Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C., 2016b. Using free modeling as an agile method for developing domain specific modeling languages, in: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 24–34.

[24] Grant, E., Narayanan, K., Reza, H., 2004. Rigorously defined domain modeling languages, in: Proc. of the 4th OOPSLA Workshop on Domain-Specific Modeling, pp. 1–8.

[25] Grant, E.S., 2012. A meta-model approach to defining uml-based domain-specific modeling language, in: Proceedings of the international multiconference of engineers and computer scientists 2012, pp. 780–785.

[26] Hoppenbrouwers, S., Bleeker, A., Proper, H., 2004. Modeling linguistically complex business domains. Technical Report. Nijmegen Institute for Information and Computing Sciences, University of Nijmegen.

[27] Ibrahim, M., Ahmad, R., 2010. Class diagram extraction from textual requirements using natural language processing (nlp) techniques, in: 2nd International Conference on Computer Research and Development (ICCRD'10), IEEE. pp. 200–204.

[28] Kelly, S., Tolvanen, J.P., 2008. Domain-Specific Modeling. IEEE Computer Society.

[29] Kristen, G., 1994. Object Orientation, The KISS Method, From Information Architecture to Information System. Addison Wesley.

[30] Kühne, T., Mezei, G., Syriani, E., Vangheluwe, H., Wimmer, M., 2009. Explicit transformation modeling, in: International Conference on Model Driven Engineering Languages and Systems, Springer. pp. 240–255.

[31] Lano, K., Kolahdouz-Rahimi, S., 2014. Model-transformation design patterns. IEEE Transactions on Software Engineering 40, 1224–1259.

[32] Lochmann, H., Hessellund, A., 2009. An integrated view on modeling with multiple domain-specific languages, in: Proceedings of the IASTED International Conference Software Engineering SE 2009, pp. 1–10.

[33] Mannaerts, H., Verelst, J., 2009. Normalized systems. Koppa BvBa.

[34] Marvie, R., 2004. A transformation composition framework for model driven engineering. Technical Report. Laboratoire d'Informatique Fondamentale de Lille.
[35] Proper, H.A., Bleeker, A.I., Hoppenbrouwers, S.J.B.A., 2004. Object–role modelling as a domain modelling approach, in: Proceedings of the Workshop on Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD'04), pp. 317–328.
[36] Purao, S., Storey, V., Sengupta, A., Moore, M., 2000. Reconciling and cleansing: an approach to inducing domain models, in: International Workshop on Information Systems and Technologies (WITS), pp. 61–66.
[37] Reinhartz-Berger, I., Soffer, P., Sturm, A., 2005. A domain engineering approach to specifying and applying reference models. Enterprise modelling and information systems architectures .
[38] Reinhartz-Berger, I., Sturm, A., 2004. Behavioral domain analysis — the application-based domain modeling approach, in: 7th International Conference on The Unified Modeling Language. Modeling Languages and Applications. UML 2004. Lecture Notes in Computer Science, vol 3273, Springer-Verlag. pp. 410–424.
[39] Robert, S., Gérard, S., Terrier, F., Lagarde, F., 2009. A lightweight approach for domain-specific modeling languages design, in: 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, IEEE. pp. 155–161.
[40] Romero, J.R., Jaén, J.I., Vallecillo, A., 2009. Realizing correspondences in multi-viewpoint specifications, in: IEEE International Enterprise Distributed Object Computing Conference, pp. 163–172.
[41] Sagar, V.B.V., Abirami, S., 2014. Conceptual modeling of natural language functional requirements. Journal of System and Software 88.
[42] Selic, B., 2007. A systematic approach to domain-specific language design using uml, in: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), pp. 2–9.
[43] Shlaer, S., Mellor, S.J., 1989. An object-oriented approach to domain analysis. SIGSOFT Softw. Eng. Notes 14, 66–77. doi:10.1145/71633.71639.
[44] Simos, M., Anthony, J., 1998. Weaving the model web: A multi-modeling approach to concepts and features in domain engineering, in: Proceedings. Fifth International Conference on Software Reuse, pp. 94–102.
[45] Simos, M.A., Aug 1995. Organization domain modeling (odm): Formalizing the core domain modeling life cycle, in: SIGSOFT Software Engineering Notes, Special Issue on the 1995 Symposium on Software Reusability, pp. 196–205.
[46] Spinellis, D., 2001. Notable design patterns for domain-specific languages. Journal of Systems and Software 56(1), 91–99.
[47] Strembeck, M., Zdun, U., 2009. An approach for the systematic development of domain-specific languages. Softw. Pract. Exper. 39, 1253–1292.
[48] Sturm, A., I, Reinhartz-Berger, 2004. Applying the application-based domain modeling approach to uml structural views, in: the 23rd International Conference on Conceptual Modeling (ER'2004), Lecture Notes in Computer Science 3288, pp. 766–799.
[49] Vallecillo, A., June 2010. On the combination of domain specific modeling languages, in: European Conference on Modelling Foundations and Applications, pp. 305–320.
[50] Visic, N., Fill, H., Buchmann, R.A., Karagiannis, D., 2015. A domain-specific language for modeling method definition: From requirements to grammar, in: IEEE 9th International Conference on Research Challenges in Information Science (RCIS), pp. 286–297.
[51] Voelter, M., 2013. DSL Engineering Designing, Implementing and Using Domain-Specific Languages. Createspace Independent Publishing Platform. URL: http://dslbook.org.
[52] van der Vos, B., Hoppenbrouwers., J., Hoppenbrouwers, S., 1996. Nl structures and conceptual modelling: the kiss case, in: Applications of Natural Language to Information Systems: Proceedings of the Second International Workshop, p. 197.
[53] Zhang, Y., Liu, X., Wang, Z., Chen, L., 2012. A model-driven method for service-oriented modeling and design based on domain ontology, in: Computer, Informatics, Cybernetics and Applications. Lecture Notes in Electrical Engineering, vol 107., Springer. pp. 991–998.

References

[54] Barišić, A., Amaral, V., Goulão, M., 2018. Usability driven dsl development with use-me. Computer Languages, Systems & Structures 51, 118–157.
[55] Barišic, A., Amaral, V., Goulao, M., Barroca, B., 2011. Quality in use of dsls: Current evaluation methods. Proceedings of the 3rd INForum-Simpósio de Informática (INForum2011) .
[56] Barišic, A., Amaral, V., Goulão, M., Barroca, B., 2014. Evaluating the usability of domain-specific languages, in: Software Design and Development: Concepts, Methodologies, Tools, and Applications. IGI Global, pp. 2120–2141.
[57] Bass, L., Clements, P., Kazamn, R., 2003. Software Architecture in Practice, Second Edition. Addison-Wesley.
[58] Chomsky, N., 1998. On Language: Chomsky's Classic Works Language and Responsibility and Reflections on Language. New Press.
[59] Czech, G., Moser, M., Pichler, J., 2019. A systematic mapping study on best practices for domain-specific modeling. Software Quality Journal , 1–30.
[60] Deckers, R., Lago, P., 2022. [dataset] slr dost replication package. https://doi.org/10.5281/zenodo.6635964.
[61] Deckers, R., van den Brand, D., Lago, P., under submission. Modeling features in terms of domain models: MuDForM method definition and case study. https://research.vu.nl/en/publications/modeling-features-in-terms-of-domain-models-mudform-method-defini.
[62] Deursen, A.V., Klint, P., Visser, J., 2000. Domain-specific languages: An annotated bibliography. ACM Sigplan Notices 35, 26–36.
[63] Falbo, R.d.A., Guizzardi, G., Duarte, K.C., 2002. An ontological approach to domain engineering, in: Proceedings of the 14th international conference on Software engineering and knowledge engineering, pp. 351–358.
[64] Firesmith, D., 2004. Specifying reusable security requirements. Journal of Object Technology 3, 61–75.
[65] Gabriel, P., Goulao, M., Amaral, V., 2011. Do software languages engineers evaluate their languages? arXiv preprint arXiv:1109.6794 .
[66] Gray, J., Fisher, K., Consel, C., Karsai, G., Mernik, M., Tolvanen, J.P., 2008. Dsls: The good, the bad, and the ugly, in: Companion to the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications, Association for Computing Machinery, New York, NY, USA. p. 791–794. URL: https://doi-org.vu-nl.idm.oclc.org/10.1145/1449814.1449863, doi:10.1145/1449814.1449863.

[67] Hofstadter, D.R., Sander, E., 2013. Surfaces and Essences: Analogy as the Fuel and Fire of Thinking. Basic Books.
[68] ISO/IEC, 2007. Systems and software engineering – Recommended practice for architectural descriptions of software intensive systems, ISO/IEC 42010:2007. Technical Report. ISO/IEC/IEEE.
[69] ISO/IEC, 2011. Iso/iec 25010:2011: Systems and software engineering - systems and software quality requirements and evaluation (square) - system and software quality models.
[70] Iung, A., Carbonell, J., Marchezan, L., Rodrigues, E., Bernardino, M., Basso, F.P., Medeiros, B., 2020. Systematic mapping study on domain-specific language development tools. Empirical Software Engineering 25, 4205–4249.
[71] Jalali, S., Wohlin, C., 2012. Systematic literature studies: database searches vs. backward snowballing, in: Proceedings of the 2012 ACM-IEEE international symposium on empirical software engineering and measurement, IEEE. pp. 29–38.
[72] Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-oriented domain analysis (FODA) feasibility study. Technical Report. Software Engineering Institute, Carnegie Mellon University.
[73] Kitchenham, B., 2004. Procedures for performing systematic reviews. Keele, UK, Keele University 33, 1–26.
[74] Kosar, T., Bohra, S., Mernik, M., 2016. Domain-specific languages: A systematic mapping study. Information and Software Technology 71, 77–91.
[75] Kronlöf, K., 1993. Method integration, concepts and case studies. John Wiley and Sons.
[76] Mernik, M., Heering, J., Sloane, A.M., 2005. When and how to develop domain-specific languages. ACM computing surveys (CSUR) 37, 316–344.
[77] do Nascimento, L.M., Viana, D.L., Neto, P., Martins, D., Garcia, V.C., Meira, S., 2012. A systematic mapping study on domain-specific languages, in: The Seventh International Conference on Software Engineering Advances (ICSEA 2012), pp. 179–187.
[78] OMG, 2003. MDA guide version 1.0. Technical Report. OMG. URL: http://www.omg.org/docs/omg/03-06-01.pdf.
[79] OMG, 2017. Unified Modeling Language Version 2.5.1. Technical Report. OMG. URL: https://www.omg.org/spec/UML/2.5.1/pdf.
[80] Pinker, S., 1994. The language instinct: how the mind creates language. William Morrow and Company.
[81] Prieto-Díaz, R., 1990. Domain analysis: An introduction. SIGSOFT Softw. Eng. Notes 15, 47–54. URL: https://doi-org.vu-nl.idm.oclc.org/10.1145/382296.382703, doi:10.1145/382296.382703.
[82] Rodrigues, I.P., de Borba Campos, M., Zorzo, A.F., 2017. Usability evaluation of domain-specific languages: a systematic literature review, in: International Conference on Human-Computer Interaction, Springer. pp. 522–534.
[83] Taylor, R.N., Tracz, W., Coglianese, L., 1995. Software development using domain-specific software architectures. ACM SIGSOFT Software Engineering Notes 20, 27–37.
[84] Torres, W., Van den Brand, M.G., Serebrenik, A., 2020. A systematic literature review of cross-domain model consistency checking by model management tools. Software and Systems Modeling , 1–20.
[85] Völter, M., 2009. Best practices for dsls and model-driven development. Journal of Object Technology 8, 79–102.
[86] Wohlin, C., 2016. Second-generation systematic literature studies using snowballing, in: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–6.

Robert Deckers is an external PhD candidate at the Vrije Universiteit Amsterdam, the Netherlands. He has been working on code generation, model interpreters and specification methods since 1991. He has developed and applied specification languages, modelling methods and supporting tools. He has been active as an architect and consultant in architecture, domain modeling, and requirements at all organizational levels. Robert is author of the book "DYA|Software, architecture approach for mission critical applications". Robert teaches software architecture and domain modeling at universities and companies. In 2013, he has started his own company to devote himself to MDD, because "the world must understand that software development is about integrating knowledge and not about realizing technology". More info available at www.linkedin.com/in/robertdeckers/.

Patricia Lago is Full Professor in software engineering at the Vrije Universiteit Amsterdam, the Netherlands, where she leads the Software and Sustainability (S2) research group in the Computer Science Department. Her research is in software architecture and software quality with a special emphasis on sustainability. She has a PhD in Control and Computer Engineering from Politecnico di Torino and a Master in Computer Science from the University of Pisa, both in Italy. She is initiator and coordinator of the Computer Science Master Track in Software Engineering and Green IT, and co-founder of the Green Lab, a place where researchers, students and companies collaborate to measure the energy footprint of software solutions and the impact on software quality. She is a member of VERSEN and the Steering Committees of IEEE ICSA and ECSA conference series. She is also SC Chair of ICT4S. More information is available at www.patricialago.nl.