The Journal of Systems and Software 113 (2016) 140–172

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Extracting reusable design decisions for UML-based domain-specific languages: A multi-method study

Stefan Sobernig[a,*], Bernhard Hoisl[a], Mark Strembeck[a,b]

[a] Vienna University of Economics and Business (WU), Welthandelsplatz 1, 1020 Vienna, Austria
[b] Secure Business Austria (SBA) Research gGmbH, Favoritenstraße 16, 1040 Vienna, Austria

ARTICLE INFO

Article history:
Received 11 April 2015
Revised 5 October 2015
Accepted 18 November 2015
Available online 26 November 2015

Keywords:
Domain-specific language
Unified modeling language
Design decision
Design rationale
Domain-specific modeling
Model-driven development

ABSTRACT

When developing domain-specific modeling languages (DSMLs), software engineers have to make a number of important design decisions on the DSML itself, or on the software-development process that is applied to develop the DSML. Thus, making *well-informed* design decisions is a critical factor in developing DSMLs. To support this decision-making process, the model-driven development community has started to collect established design practices in terms of patterns, guidelines, story-telling, and procedural models. However, most of these documentation practices do not capture the details necessary to reuse the rationale behind these decisions in other DSML projects. In this paper, we report on a three-year research effort to compile and to empirically validate a catalog of structured decision descriptions (*decision records*) for UML-based DSMLs. This catalog is based on design decisions extracted from 90 DSML projects. These projects were identified—among others—via an extensive systematic literature review (SLR) for the years 2005–2012. Based on more than 8,000 candidate publications, we finally selected 84 publications for extracting design-decision data. The extracted data were evaluated quantitatively using a frequent-item-set analysis to obtain characteristic combinations of design decisions and qualitatively to document recurring documentation issues for UML-based DSMLs. We revised the collected decision records based on this evidence and made the decision-record catalog for developing UML-based DSMLs publicly available. Furthermore, our study offers insights into UML usage (e.g. diagram types) and into the adoption of UML extension techniques (e.g. metamodel extensions, profiles).

1. Introduction

In model-driven development (MDD), a domain-specific modeling language (DSML) is a specialized modeling language tailored for graphical modeling tasks in a particular application domain, e.g. for business domains such as health-care and banking, or technical domains such as access control and system backup (Spinellis, 2001; Mernik et al., 2005; Kelly and Tolvanen, 2008; Zdun and Strembeck, 2009). MDD uses models at different abstraction levels as central development artifacts. Executable models are then derived from higher-level models through model transformations (Selic, 2003; Sendall and Kozaczynski, 2003; Mens and Gorp, 2006). By raising the abstraction level in the software-development process, DSMLs aim at increasing the productivity of developers and at reducing maintenance costs (Langlois et al., 2007; Mernik et al., 2005; Stahl and Völter, 2006).

Developing DSMLs based on the Unified Modeling Language (UML Object Management Group, 2015b) and/or on the Meta Object Facility (MOF Object Management Group, 2015a) has become a popular option in the MDD context (Hutchinson et al., 2014; Nascimento et al., 2012; Hutchinson et al., 2011; Pardillo and Cachero, 2010; Staron and Wohlin, 2006). This is because the UML/MOF infrastructure provides built-in implementation techniques for DSMLs, which allow for reusing and for extending the UML directly. As another advantage, the UML can leverage industry-grade tool support (e.g. Sparx Systems Enterprise Architect, IBM Rational Software Architect, Eclipse Model Development Tools). UML has been subjected to scientific evaluations of its semantic foundations (see, e.g., Mallet and Andre, 2009; Selic, 2004; Broy and Cengarle, 2011) and comes with standardized modeling extensions (see, e.g., Object Management Group, 2012a; 2012b). Moreover, the UML benefits from the systematic and continuous maintenance through the Object Management Group (OMG). Thereby, the UML and the MOF provide a rich DSML development toolkit for DSML design and implementation.

As in most software-development activities, experiences and lessons learned from developing DSMLs based on MOF/UML in a

* Corresponding author. Tel.: +43 1313364878.
E-mail address: stefan.sobernig@wu.ac.at (S. Sobernig).

http://dx.doi.org/10.1016/j.jss.2015.11.037
0164-1212/

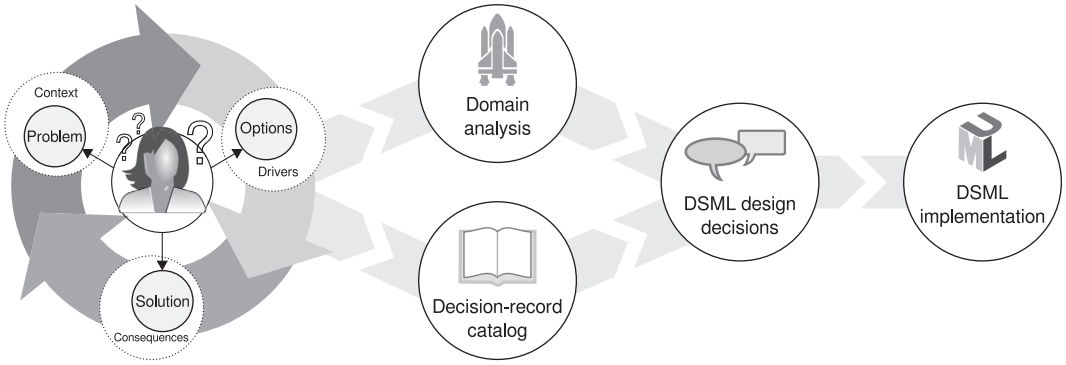


Fig. 1. High-level overview of design-decision making for developing domain-specific modeling languages (DSMLs) using the Unified Modeling Language (UML).

disciplined manner are particularly valuable, yet barely documented (Strembeck and Zdun, 2009; Zdun and Strembeck, 2009). Design-decision making in and on DSML development usually includes design decisions on language-model definition, constraint specification, concrete-syntax design, and platform integration (Strembeck and Zdun, 2009). Each of these concerns potentially involves multiple, interrelated design decisions.

Critical details of design decisions are rarely documented explicitly—a fact also referred to as the *capture problem* of design-rationale documentation (Burge et al., 2008; Dutoit et al., 2006). Decision details include different solutions considered before arriving at a final design decision (i.e. decision options), the decision-makers' positive and negative assessments of the considered options given a set of requirements on the DSML's design (decision drivers), and the positive and negative effects on subsequent design-decision making observed when having adopted one or several options (decision consequences).

An important barrier to documenting design rationale in necessary detail, in DSML development and in software development in more general, is the considerable overhead that results from creating and maintaining design-rationale documentation. Other problems explored in the research on documenting design-rationale include the intrusiveness of documentation techniques, lack of incentives, and cognitive barriers in software-design processes (see, e.g., Dutoit et al., 2006; Falessi et al., 2013; Horner and Atwood, 2006). As a consequence, new DSML development projects cannot profit from experiences gained in prior DSML projects. Moreover, without such a body of knowledge on DSML development and evolution, existing DSMLs can become difficult to maintain.

Important goals in many fields of software-engineering (SE) research have been to limit the effort for documenting design decisions for DSML projects and to increase the quality of the documented rationale. This research has explored ways of gathering, validating, and referencing collections of *reusable generic design rationale*—such as software patterns (Harrison et al., 2007)—to achieve these two goals. In our work, we build on the software-pattern tradition for capturing and for reusing SE knowledge. A software pattern documents a proven solution to a recurring engineering problem in or across different application domains, including e.g. software architecture, distributed systems, and application integration (see, e.g., Buschmann et al., 2000; Völter et al., 2005; Hohpe and Woolf, 2004). While patterns are already useful when used in isolation, they unfold their whole potential as part of *pattern languages* (Coplien, 1996; Buschmann et al., 2007). A pattern language documents the relations between different patterns and describes how interrelated SE problems can be solved in a proven way. In particular, a pattern language gives structure to a pattern catalog and guides the software developer in choosing a suitable selection of patterns to address a given problem.

In this spirit, in a three-year research effort, we documented recurring design decisions extracted from 90 UML-based DSMLs in terms of a decision-record catalog (Hoisl et al., 2014a). Similar to pattern descriptions, we document reusable DSML design decisions via a structured text format (*decision record*). Each decision record documents detailed information about a given decision-making situation (*decision point*). The catalog of decision records also describes typical combinations of decisions on developing a DSML. This way, the catalog serves as a source of reusable generic design rationale when developing DSMLs using UML.

Fig. 1 shows a high-level overview of design-decision making for developing a UML-based DSML. In a decision-making situation, a decision is triggered by a decision *problem*. An exemplary problem is about how to express design constraints of a DSML. Each problem occurs in a particular *context*. For instance, one must define constraints on the DSML's language elements which cannot be expressed via graphical symbols. Decision making is affected by different *drivers* that influence the choice of a particular solution to the problem: When and where do constraints have to be checked? Are model-level checks via some graphical DSML editor needed, or instance-level checks in a particular target platform? For each problem, typically different solutions (*options*) are available; for example, defining constraints via a formal constraint language or using informal textual annotations. Each option comes with characteristic *consequences*. A consequence of formal constraints, for instance, is that they can be checked via corresponding software tools. At the same time, formal constraints are harder to change and to maintain than informal annotations.

A decision record in our catalog reflects this structure of decision-making situations by describing a recurring problem in a given context, common solution options as well as characteristic drivers and consequences. The decision-record catalog and the results of a domain analysis (see, e.g., Evans, 2004) for the DSML's application domain are then used by DSML developers to make the actual design decisions for a particular DSML. Finally, the design decisions guide the actual implementation of the DSML using UML (see Fig. 1).

In this paper, we report on applying a systematic literature review (SLR) and a content analysis as methods to rigorously collect, document, and validate design decisions on UML-based DSMLs. We provide examples on how the respective design decisions are documented and how they can be applied when defining a new UML-based DSML. We do not, however, discuss all design decisions and corresponding decision options identified through this study. They are documented in a companion catalog in full detail (Hoisl et al., 2014a).[1] In other words, this paper emphasizes the research method

[1] The design-decision catalog is available at http://epub.wu.ac.at/4466/.

and documents how design decisions were identified, extracted, and validated. The key contributions of this paper are:

- *Research design*: Our multi-stage and multi-method research effort took place over three years. The stages of our work include preparatory design reviews, early manual literature searches (Hoisl et al., 2012b), a pilot study (Filtz, 2013), construction of a quasi-gold-standard (QGS) paper corpus, an extensive automated search, and a citation-driven manual search (*snowballing*; Jalali and Wohlin, 2012; Webster and Watson, 2002). In a sequential multi-method (qualitative-quantitative) research design, we conducted an extensive systematic literature review (SLR), as well as content analysis and data mining to extract details on reusable, generic design decisions from the SLR.
- *Literature review*: The SLR was performed by adopting established guidelines (Jalali and Wohlin, 2012; Kitchenham, 2004; Zhang et al., 2011). Starting from more than 8,000 publications retrieved by automated and snowballing searches, in a two-extractor process, we selected 84 publications representing 80 UML-based DSMLs. A detailed SLR protocol is available (Sobernig et al., 2014).[2]
- *Content analysis*: Design decisions were extracted from these 84 publications using a deductive content analysis (Saldaña, 2013; Schreier, 2013). This content analysis involved a coding schema (including indicators and decision rules for coders), several inter-subject iterations over the paper material (segmentation, coding), and an inter-rater reliability analysis.
- *Data mining*: We performed a frequent-item-set analysis (Borgelt, 2012) to mine the design-decision data collected from the SLR for recurring decision-making patterns.

This paper is accompanied by three companion documents. First, and in line with established SLR practice, there is a detailed review protocol (Sobernig et al., 2014) of all steps performed to make our SLR transparent and reproducible. Second, there is a *pre-study revision* of the decision-record catalog (Hoisl et al., 2012a) which served as input for sub-steps of the SLR (e.g. construction of a quasi-gold standard) and of the content analysis (e.g. construction of a coding schema). Third, there is a *post-study revision* (Hoisl et al., 2014a) of this catalog which incorporates the results of our SLR as an increment to the pre-study revision.

The remainder is structured as follows: In Section 2, we justify the choice of capturing DSML design rationale via an SLR and the choice of representing design rationale in terms of structured decision records. We also motivate our research by giving two illustrative, concrete examples of using the decision-record catalog in DSML development: design-space analyses and design-process documentation. Section 3 enumerates the research questions driving the SLR study. In Section 4, we present the details of conducting and the intermediate results of the various stages of our study. The design-decision data gathered from the corresponding identified DSML designs, as well as study limitations, are described in Section 5. Important implications are discussed in Section 6. Related work on DSL development and empirical research on UML is reviewed in Section 7. Section 8 concludes the paper by reiterating over key contributions and by pointing to future work.

2. Design rationale on DSML development

A domain-specific language (DSL) is a software language which is specialized for addressing a given class of engineering problems, which are characteristic for an application domain. A DSL is based on abstractions aligned to this domain and provides a concrete syntax suitable for employing its abstractions effectively. A domain-specific *modeling* language (DSML) is a DSL with a graphical concrete syntax for the primary purpose of diagrammatic modeling in a particular application domain (Zdun and Strembeck, 2009; Strembeck and Zdun, 2009). A DSML focuses on providing modeling abstractions for the problem concerns in the application domain which are independent from a given software platform, rather than on issues of implementing the domain (see, e.g., Atkinson and Kühne, 2007). The collection of specification artifacts, which contribute to defining a DSML, is referred to as the DSML's language model in the broader sense (Strembeck and Zdun, 2009). This language model contains a *core language-model* which describes the structural elements and their relationships using metamodeling techniques (e.g. metamodels and/or profiles in the MOF/UML). This core language-model is also known as a DSML's abstract syntax, especially when specified using grammars. Hereafter, for brevity, we refer to a DSML's *language model* to denote its core language-model (abstract syntax) only. As another characteristic, formal specification techniques are used in DSML development to express the structural and behavioral semantics of the DSML and its instance models (Jackson and Sztipanovits, 2009). A DSML is then commonly deployed as part of a model-driven development toolkit (e.g. as part of the Eclipse Modeling Framework).

For the scope of our study, we look at DSMLs which are *internal* to or *embedded* into the Unified Modeling Language version 2.x (UML 2.x; Pardillo and Cachero, 2010; Giachetti et al., 2009; Lagarde et al., 2008). The language models of embedded DSMLs are defined on top of the UML 2.x or by extending the UML 2.x language model (Object Management Group, 2015b). UML 2.x was rapidly adopted by DSML developers as the host modeling-language for their UML-based DSMLs. For one, this was facilitated by important modeling toolkits already providing UML 2.x support even before its final release in 2005 (e.g. Sparx Systems Enterprise Architect, Eclipse Modeling Tools). In some cases, this went hand in hand with dropping support for UML 1.x entirely. Besides, UML 2.x provides a larger base of predefined structural and behavioral diagram types (14 vs. 8 in UML 1.x), on which a DSML can be built. In addition, UML 2.x and its language architecture (Cook, 2012) provide for three implementation techniques for embedded DSMLs: language-model extension, language-model piggybacking, and language-model specialization. Note that a DSML development project might require a combination of these three implementation techniques.

To begin with, a DSML can be defined as a *language-model extension* (Spinellis, 2001) of UML 2.x. In this scenario, the UML 2.x metamodel is extended and modified in an additive manner. This can be achieved by first introducing new metamodel packages, which contain new or redefining metaclass definitions. These metamodel packages are defined using the Meta Object Facility (MOF) and they may reference predefined UML 2.x metamodel packages. To create an actual UML 2.x metamodel derivative, they are then merged into the UML 2.x metamodel packages (Dingel et al., 2008; Burgués et al., 2008).

A second option is to have a DSML use UML 2.x as its base and add DSML-specific elements without changing the underlying structural and behavioral UML 2.x semantics (*language-model piggybacking*; Spinellis, 2001). This can be achieved using UML 2.x profiles (Pardillo and Cachero, 2010; Giachetti et al., 2009), which decorate models defined using the UML 2.x metamodel or a UML 2.x metamodel derivative.

A third DSML implementation technique applicable to UML 2.x is a variant of *language-model specialization* (Spinellis, 2001): metamodel pruning. In metamodel pruning, an effective metamodel (i.e. the DSML language model) is extracted from an original metamodel (e.g. UML 2.x metamodel) in an automated manner (Sen et al., 2009; Blouin et al., 2015). The extraction procedure establishes a generalization relationship between the effective (super-)metamodel and the original (sub-)metamodel. This way, DSML models defined over

[2] The SLR protocol is available at http://epub.wu.ac.at/4467/.

the smaller, effective metamodel also conform to the larger, original metamodel.

The above DSML implementation techniques for UML-based DSMLs go beyond the ones available in UML 1.x. Moreover, UML 2.x avoids known semantics pitfalls relating to these implementation techniques in UML 1.x (e.g. profiles; Cook, 2012; Henderson-Sellers and Gonzalez-Perez, 2006). This also determined our choice of UML 2.x.

Design rationale (DR; Burge et al., 2008; Dutoit et al., 2006) on DSML development is the reasoning and justification of decisions made when designing, creating, and using the core artifacts of a DSML (e.g. abstract and concrete syntax, behavior specification, metamodeling infrastructure, MDD tool chain). Documenting design rationale explicitly aims at helping design-decision makers by providing and explaining past decisions (e.g. in a design-space analysis) and by improving the understanding of a DSML design during development and maintenance (e.g. as a kind of design-process documentation). Most importantly, documenting design rationale on UML-based DSML development must go beyond a mere enumeration of advantages and disadvantages specific to the three DSML implementation techniques in UML 2.x (see above). On the one hand, the choice of implementation techniques is only one decision dimension. On the other hand, the rationale behind such a choice can be specific to every DSML development project.

We distinguish between two kinds of DSML design rationale (Harrison et al., 2007): DSML-*specific* design rationale reflects reasoning required and collected explicitly during a particular design process for a single DSML. Examples of explicitly documented, specific design rationale in software-language engineering may be found in artifacts created in source-configuration management tools and development-issue trackers and open-standards artifacts. Examples include Java Community Process documents and the recorded issue votes during the ANSI/CLISP X3J13 specification process as collected in Steele (1990). DSML-*generic* design rationale is reasoning knowledge obtained through developing multiple DSMLs, for one or several application domains. Generic design rationale is commonly found only as the implicit knowledge of experienced DSML engineers. Software patterns have been used in software-language engineering to document generic design rationale explicitly (see, e.g., Spinellis, 2001; Günther, 2011).

An approach for managing design rationale in UML-based DSML development involves a procedure to represent design rationale, to capture it, and to put the documented design rationale to use in other DSML development projects. Representing and organizing a body of design rationale chunks deals with structuring and creating documentation items. In this paper, our emphasis is on creating documentation items on generic DSML design rationale (*decision records*) as explained in Section 2.1. Two intended scenarios for using the documented generic DSML design rationale (i.e. design-space analysis, process documentation) are presented in Section 2.2. Our approach of capturing and of transforming specific DSML rationale into generic design rationale is elaborated on in Section 4.

2.1. Representing design rationale

The process of developing a DSML involves different, characteristic development activities (Strembeck and Zdun, 2009; Clark et al., 2008). From a decision-making perspective, each development activity also marks a *decision point*, i.e. a point in time at which particular design-decision problems must be addressed. In doing so, different design solutions based on their assumed or known properties for the DSML design as well as their effects on any subsequent design decisions are assessed. From the angle of design-rationale documentation, a decision point is a point in time for recording an on-going decision-making process.

In our approach, design rationale on a given decision point is captured from multiple DSML projects and represented as a reusable resource for decision making at this decision point in developing a new DSML. We refer to this reusable resource as a *decision record* (see Fig. 3).[3] In particular, we consider six decision points in UML-based DSML development and, therefore, six decision records (D1–D6, hereafter).

Language-model definition (D1) involves identifying the respective domain abstractions that must be represented in our DSML, after a systematic analysis and a structuring of the respective application domain. The main challenge at this decision point is how to identify and to describe these domain abstractions in order to arrive at a comprehensive and comprehensible basis for developing the DSML. *Language-model formalization* (D2) is about how the domain definition created in D1 can be turned into a formal model. By formal model, we refer to a description that can be checked for conformance against previously defined well-formedness rules. Moreover, a formal model is amenable to processing and to manipulating by automated tools (Paige et al., 2014). For the DSMLs studied in this paper, a formal language model is realized using well-defined metamodeling languages (MOF/UML). A metamodeling language is itself based on a well-defined and well-documented language model. The latter is referred to as the meta-metamodel, such as the CMOF for the UML metamodel. In addition, a metamodeling language provides at least one well-defined and well-documented concrete syntax to define the DSML's language model. The DSMLs of interest in this paper can, for example, employ the CMOF diagram syntax to specify a UML metamodel extension. To remove ambiguity from a formal language model, a language model is extended to include additional *language-model constraints* (D3) to express constraints on domain abstractions, such as invariants for domain concepts, pre- and post-conditions, as well as guards. *Concrete-syntax definition* (D4) is about deciding on the "user interface" of UML-based DSMLs, looking at several options (e.g. model annotations, reuse or extend a diagrammatic syntax, mix foreign syntaxes with the UML syntax). The *behavior specification* (D5) of a DSML defines behaviors of one or more DSML language element(s). It determines how the language elements of the DSML interact to produce the behavior intended by the DSML engineers using, e.g., UML M1 behavior models or formal textual specifications. To produce executable, platform-specific model artifacts from DSML models, all DSML artifacts need to be mapped to a software platform using techniques of *platform integration* (D6; e.g. model transformations, code generators).

The order in which these development activities, and the corresponding decision points, are considered can characterize different DSML development styles (Strembeck and Zdun, 2009). Depending on the DSML development scenario some decision points may also be skipped (depending on, e.g., application domain, usage intention, and development style).

2.1.1. Decision records

A decision record provides two or more descriptions of proven solutions to a generic and recurring problem in developing a DSML. The problem described by a decision record must not only recur, that is, be observable for many DSML development projects, but it must also have the quality of requiring an act of design-decision making. For structuring and presenting the recurring DSML design decisions as decision records, we developed and refined a document template (Hoisl et al., 2014a; 2012b; 2012a). Each decision record is structured into seven sections. The most important sections are identified in

[3] Throughout the paper, we apply some notation conventions to refer to decision records and their content items such as decision options. D*i* denotes a decision record corresponding to decision point *i*; O*i*.*j* refers to decision option *j* at decision point *i*. In set notation, the *O*-prefix for decision options as set elements is omitted for brevity.

D4 Concrete-Syntax Definition

Problem. *In which representation should the domain modeler create models using the DSML?*

Context. The concrete syntax serves as the DSML's interface. Different syntax types can be defined and tailored to the need of the modeler . . .

. . .

Options.

O4.1 Model annotation: Attach UML comments as concrete-syntax cues to a UML model, containing complementary domain information such as keywords, narrative statements, or formal definitions (see, e.g., [3]). . . .

O4.2 Diagrammatic syntax extension: . . .

. . .

Decision drivers. An overview of positive and negative links between decision drivers and available options is shown in the table below. . . .

Non-diagrammatic UML notation requirements: Textual notations [1] for the UML are auxiliary representations and act as frontend syntaxes (O4.4).

Driver/Option	O4.1	O4.2	O4.3	O4.4	O4.5	O4.6	O4.7
Non-diagrammatic UML notation requirements	o	o	−	−	−	o	o
Degree of cognitive expressiveness	−	+	+/−	+/−	+/−	−	o
Disruptiveness	−	+	+	+	++	−	+/−
Degree of required modeling-tool support	++	−	+/−	+	−−	++	o

Consequences.

Usability evaluation: The DSML syntax is especially important from the DSML user perspective. If a DSML is mainly used by non-programmers, a special focus on usability aspects is needed. . . .

. . .

Application. In our case studies we provide a couple of different concrete syntax definitions such as UML stereotype-specific annotations for reusing symbols (P1, P3, P7, P9, P10). . . .

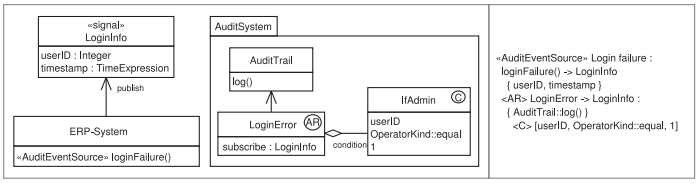


Figure 1: Exemplary graphical and textual concrete syntax [2].

Sketch. Figure 1 shows an example of two concrete syntax definitions . . .

Fig. 2. Excerpt from the actual decision record on concrete-syntax definition (D4) in Hoisl et al. (2014a), highlighting the document sections corresponding to the key concepts in Fig. 3.

Fig. 3, an excerpt from the actual decision record on concrete-syntax definition (D4) is depicted in Fig. 2.

A decision record first describes a recurring design-*decision problem* that has been repeatedly observed for several DSML development projects. The exemplary decision record in Fig. 2 gives a problem statement frequently observed when deciding on the concrete-syntax style for a DSML: "In which representation should the domain modeler create models using the DSML?". This problem applies to a specific *decision context*. The decision context is primarily set by one of the decision points characteristic for DSML development (D1–D6, above). In addition, a particular metamodeling toolkit (e.g. MOF/UML), the application domain modeled by a DSML, and the target software platform can contribute to establishing the decision context. To give an example: The problem statement in Fig. 2 would not apply for a DSML project which is not about providing a particularly tailored or any concrete syntax to modelers. This could be because modelers are expected to work on an abstract-syntax representation only.

As its core content, a decision record lists *decision options* which describe solutions to the initial stated decision problem. The excerpt in Fig. 2 shows the option MODEL ANNOTATION (O4.1) which is about realizing a tailored concrete syntax by means of model annotations. Next, a decision record documents means to select an option (or a combination of options) in terms of *decision drivers*. An exemplary driver in Fig. 2 is the cognitive expressiveness of concrete-syntax styles. These drivers are likely to steer the DSML designer towards a particular option or option combination. This selection decision affects the solution spaces of subsequent decisions. For example, they can set a new decision context. To scaffold follow-up decision making, a decision record makes the DSML designer aware of known *decision consequences*. Consequences can include the need to evaluate other decision options within the same decision record or in related

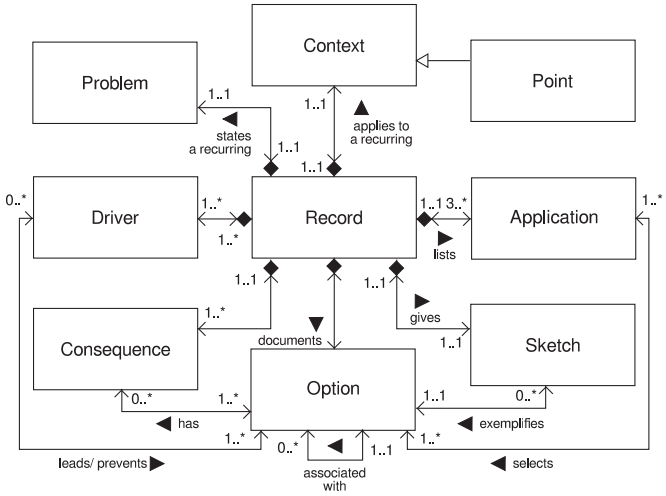


Fig. 3. An overview of the 9 key concepts and their relationships: decision point, decision context, decision problem, decision record, decision option, decision driver, decision consequence, decision application, and decision sketch.

decision records. However, consequences can also point to follow-up decision problems not covered by the decision-record catalog alone. Such a consequence is the need for usability evaluations in Fig. 2.

To provide evidence that the listed decision options are taken from observed practice, a decision record comprises DSML projects as examples of *application* of the individual options or option combinations. In Fig. 2, one can see that a number of DSML projects also recorded as part of our decision-record catalog are referenced as applications. A decision record is then closed by replicating one concrete realization *sketch* of one decision option, taken from one DSML application. In Fig. 2, an example of providing a combined graphical and textual concrete-syntax is given.

This document template has been derived—over multiple research stages (see Section 4)—from prior work on documenting design rationale in software engineering. In particular, it is inspired by software-pattern descriptions (Buschmann et al., 2007) and architectural design decisions (Shahin et al., 2009). For the content of a decision record, in particular the decision options, to qualify as generic design rationale, it must reflect *recurring* practice. To have recurrence, we require a decision option to be encountered in a certain number of DSML applications. To become included into the draft, pre-study revision of the decision records, decision options had to be accounted for by two distinct sources. For example, an option could be supported by one primary study documenting a DSML and one secondary study on designing UML-based DSMLs. Based on the study's results, in the post-study revision, we verified which design option has at least applications in *three* different third-party DSMLs, that is, DSMLs not developed by the authors. This way, we adopt a commonly followed rule of thumb in the software-pattern community. This rule of thumb mandates that a software-pattern description must provide at least three known uses of the pattern in existing software systems (see, e.g., Coplien, 1996; Buschmann et al., 2007).[4]

The catalog revised based on the study results provides six decision records—one for each decision point (D1–D6)—containing 27 decision options to describe a DSML (Hoisl et al., 2014a).[5] Tables 1 and 2 provide a preview of the study results, including thumbnail descriptions of nine frequently adopted decision options which help characterize a majority of the DSML designs identified in our study (see also Section 6).

2.1.2. Decision-option sets

Making a *decision* when developing a specific UML-based DSML is about evaluating and finally adopting one or several decision options listed by a decision record in our decision-record catalog (Hoisl et al., 2014a). This way, a decision links to and conforms with a decision record (see also Fig. 3). Design-decision making on a given DSML involves decisions at several, but not necessarily all decision points (D1–D6). Adding up all decisions results in a set of adopted decision options (an *option set*, hereafter) which represents the DSML design as product of this decision making.[6]

Consider the example of UML4SOA (Mayer et al., 2008), one of the 80 third-party UML-based DSMLs reviewed in our study (see Section 5 and also the Appendix). UML4SOA refines the UML activity, class, and component diagrams to model service-oriented architectures (SOA). The language model of UML4SOA is defined textually (O1.1) and integrates with the UML via a UML metamodel extension (O2.3) as well as equivalent UML profile definitions for tool adoption (O2.2; see Table 1). In addition, the metamodel extension and profile definitions are accompanied by OCL constraint definitions (O3.1). The metamodel extension comes with new and resampled diagram symbols (O4.2), the profiles imply model annotations (e.g. comments containing tags; O4.1) and symbol reuse (O4.6; see Table 2). As for platform integration, UML4SOA employs intermediate model representations (O6.1) to transform extended UML activities in several steps (O6.5) into web-service orchestration specifications (BPEL) using API-based generators (e.g. the Eclipse/EMF Java API; O6.3). Based on our catalog of decision records, UML4SOA can be described as an option set containing ten options from our decision-record catalog (Hoisl et al., 2014a): {1.1, 2.2, 2.3, 3.1, 4.1, 4.2, 4.6, 6.1, 6.3, 6.5}.

In addition, option subsets are eligible to represent *decision-option associations*. An association between two or multiple decision options represent a possible, intentional co-occurrence of two (or more) corresponding decision options. An association denotes that two or more decision options must be considered together, without implying any particular (e.g. temporal) order of adoption. Decision-option associations can represent decision drivers and decision consequences which directly relate to decision options. Consider the proper subset {2.2, 4.1, 4.6} contained by UML4SOA's option set above. It denotes that—as a consequence of adopting UML profiles (O2.2)—UML4SOA reuses diagram symbols of the activity diagram notation (O4.6) which become extended by model annotations (O4.1; e.g. stereotype tags, tagged values). As we will learn in Sections 5 & 6, this option subset can be frequently observed in DSMLs. Our catalog currently documents 21 such decision-option associations (Hoisl et al., 2014a).

2.2. DR reuse: design-space analysis

In this section, we exemplify the use of our decision-record catalog. The decision records facilitate re-constructing the decision space of an existing DSML using Questions-Options-Criteria (QOC) diagrams. Note that we do not consider our catalog to be the only and authoritative source of input in this usage scenario. Rather, they complement existing material, such as pattern collections on software-language development (see, e.g., Spinellis, 2001).

An important area for using documented design rationale is to facilitate the creation of a systematic description of an existing design which reflects the reasoning of the designers. Here, the objective is an explicit description which is sufficiently detailed to assist in decision making and stakeholder communication during ongoing phases

[4] Note, however, that the number of occurrences observed for important decision options (and decision-option sets) turned out much higher than three (see Sections 5 and 6).

[5] Note that there are actually 31 decision codes/numbers. Four of those codes/numbers serve for coding pseudo-decision options; e.g., not taking any decision. Depending on the analysis requirements, they are either ignored or included as dedicated no-option codes.

[6] Option sets are also an outcome of the coding step during content analysis (coding form; see Section 5.1) and a prerequisite for later data mining (see Section 3).

Table 1
Thumbnail descriptions of 12 (out of 27 total) decision options specific to the decision points 1–3 (D1–D3). 6 decision options (O1.1, O1.4, O2.2, O2.3, O3.1, O3.4; in bold font face) characterize a critical number of DSML designs, a key finding of our SLR study (see Section 6).

	Problem statement	Options	Drivers
D1	How should the domain (or domain fragment) be described?	**O1.1 INFORMAL TEXTUAL DESCRIPTION** Use informal text to identify and to describe domain abstractions and their relationships (e.g. domain-vision statements, domain-distillation lists). O1.2 FORMAL TEXTUAL DESCRIPTION Use formal text to identify and to describe domain abstractions and their relationships (e.g. a grammar, a universal algebra). O1.3 INFORMAL DIAGRAMMATIC MODEL Use ad hoc diagrams to identify and to describe domain abstractions and their relationships (e.g. early feature diagrams, pseudo class diagrams). **O1.4 FORMAL DIAGRAMMATIC MODEL** Use formally defined diagrams to identify and to describe domain abstractions and their relationships (e.g. MOF and UML class diagrams, STATEMATE statecharts).	Availability of existing diagrammatic domain descriptions, intended target audience, correspondence mismatches with UML semantics, consistency preservation effort, cognitive effectiveness of a representational format
D2	In which MOF/UML-compliant way should the domain concepts be formalized?	O2.1 M1 STRUCTURAL MODEL Implement the language model using structural UML models at level M1 (e.g. class or composite-structure diagrams). **O2.2 PROFILE RE-/DEFINITION** Implement the language model by creating (or by adapting existing) UML profiles (i.e. <<`profile`>> packages containing stereotype definitions). **O2.3 METAMODEL EXTENSION** Implement the language model by creating one or several metamodel extensions (i.e. <<`metamodel`>> packages containing new metaclasses and associations). O2.4 METAMODEL MODIFICATION Implement the language model by creating one or several metamodel extensions (i.e. <<`metamodel`>> packages containing redefining metaclasses and associations).	Overlap of DSML and UML domain spaces, degree of DSML expressiveness, portability and evolution requirements, compatibility with existing artifacts
D3	Do we have to define constraints over the language model(s)? If so, how should these constraints be expressed?	**O3.1 CONSTRAINT-LANGUAGE EXPRESSION** Make language-model constraints explicit using a constraint-expression language (e.g. OCL, EVL). O3.2 CODE ANNOTATION Make language-model constraints explicit using expressions in (or a specialized sub-language embedded within) a general-purpose programming language (GPL; e.g. Java). O3.3 CONSTRAINING MODEL TRANSFORMATION Express language-model constraints as part of existing model (model-to-text/model-to-model; M2T/M2M) transformations, or through dedicated ones (e.g. OCL model-navigation expressions or conditional statements in ETL/EOL templates). **O3.4 INFORMAL TEXTUAL ANNOTATION** Use informal and unstructured text annotations to capture constraint descriptions in the language model (e.g. prose text in UML comments).	Constraint formalization requirements, language-model checking time, integrated language-model constraint requirements, maintainability effort, portability requirements, language model and constraints conformance

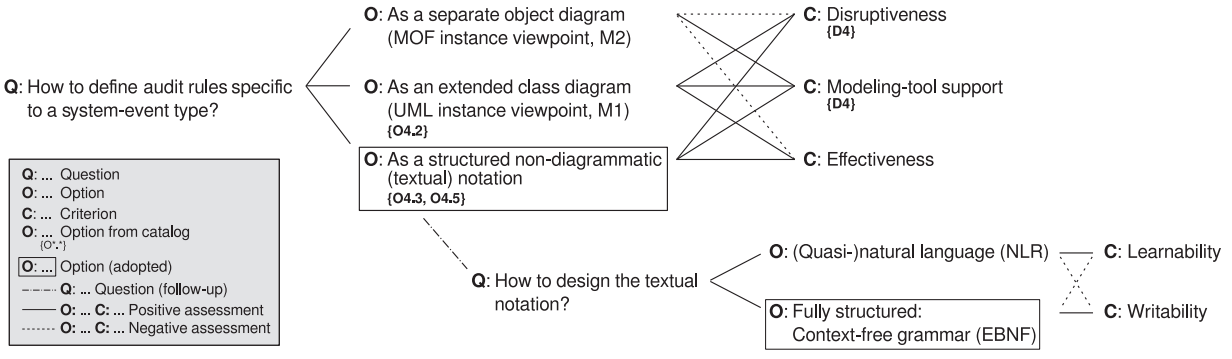


Fig. 4. A partial QOC representation of a selected design-space fragment for the DSML SecurityAudit. The boxed options are the decisions made in the design of the DSML. This example follows the original QOC notation introduced in MacLean et al. (1996); advanced QOC notation elements such as arguments are omitted for brevity.

of an artifact's lifecycle. Relevant phases include test creation, corrective maintenance, and authoring documentation. An established semi-formal, diagrammatic approach for describing a space of design arguments is the Questions, Options, and Criteria (QOC) notation (Dutoit et al., 2006; MacLean et al., 1996).

Questions deal with features of the designed artifact. In the example from Fig. 4, such a feature is the representation of audit rules in the technical domain of system auditing. An audit rule determines which system events should be audited by an auditor component or subsystem. Consider file-access events by selected system users as an example. The DSML SecurityAudit provides UML-based support to model such audit rules as part of an auditing system. The DSML representation should allow domain modelers (a security auditor) to author audit rules. In QOC, *options* model possible answers to the questions. The exemplary design space shown in Fig. 4 of the SecurityAudit DSML consist of three options. Clearly, the actual design space of the SecurityAudit DSML was larger. Two QOC options have correspondences to decision options from our catalog (D4:

Table 2
Thumbnail descriptions of 15 (out of 27 total) decision options specific to the decision points 4–6 (D4–D6). 3 decision options (O4.1, O4.6, O6.2; in bold font face) characterize a critical number of DSML designs, a key finding of our SLR study (see Section 6).

	Problem statement	Options	Drivers
D4	In which representation should the domain modeler create models using the DSML?	**O4.1 MODEL ANNOTATION** Attach UML comments as concrete-syntax cues to a UML model, containing complementary domain information such as keywords and narrative statements. O4.2 DIAGRAMMATIC SYNTAX EXTENSION Extend one or multiple UML diagram types by creating novel symbols adding to the basic UML symbol set. O4.3 MIXED SYNTAX Create your DSML's concrete syntax either as a non-diagrammatic syntax (textual or tabular) or as a diagrammatic syntax not integrated with the UML's. O4.4 FRONTEND-SYNTAX EXTENSION Create your DSML's concrete syntax as a non-diagrammatic (textual or tabular) one which extends a non-diagrammatic frontend syntax to the UML (e.g. HUTN). O4.5 ALTERNATIVE SYNTAX Create a diagrammatic syntax extension to the UML (O4.2) and provide one or more alternative syntaxes (see O4.3 and O4.4). **O4.6 DIAGRAM SYMBOL REUSE** Reuse built-in UML diagram symbols without modification.	Non-diagrammatic UML notation requirements, degree of cognitive expressiveness, disruptiveness, degree of required modeling-tool support
D5	Do we have to define (additional) behavioral semantics for the DSML? If so, how should the additional behavior of DSML elements be defined?	O5.1 M1 BEHAVIORAL MODEL Specify additional behavior of language-model elements using UML behavioral models at level M1 (e.g. state machine or activity diagram). O5.2 FORMAL TEXTUAL SPECIFICATION Specify the additional DSML behavior using a textual formalism (e.g. algebraic expressions or Z notation). O5.3 INFORMAL TEXTUAL SPECIFICATION Specify the additional DSML behavior using an informal textual description. O5.4 CONSTRAINING MODEL EXECUTION Implement behavioral constraints in a (partial) execution engine for DSML models (e.g. xMOF).	Model-consistency preservation, domain characteristics, limited expressiveness, behavior verification, visualization preferences
D6	How should the DSML artifacts be mapped to (and/or integrated with) a software platform?	O6.1 INTERMEDIATE MODEL REPRESENTATION Provide for generating a second and intermediate model based on a DSML model using M2M transformations. **O6.2 GENERATOR TEMPLATE** Create transformation templates, defined in a M2T transformation language, which turn DSML models into platform-specific textual specifications. O6.3 API-BASED GENERATOR Realize the platform-specific model transformation (e.g. code generation) by instrumenting a programmatic representation of DSML models. O6.4 (DIRECT) MODEL EXECUTION Use a (partial) model-execution engine to generate platform-specific instructions directly from DSML models. O6.5 M2M TRANSFORMATION Perform platform integration via (multiple) endogenous M2M transformations specified via M2M transformation languages (e.g. ATL, ETL).	Targeting multiple platforms, maintainability effort of static code fragments, non-executable models

Table 3
Exemplary overview of positive and negative links between criteria (or drivers as found in our catalog) and available options depicted as a rationale table. They form the basis of the assessments shown in the QOC representation in Fig. 4. (+)+: (very) positive influence; o: no influence; (−)−: (very) negative influence. An option having either a (very) positive or a (very) negative influence—depending on the intended DSML's application domain, professional background as well as prior knowledge and experience of users etc.—is denoted by (+)+/(−)−.

Driver/Option	O4.2	O4.3	O4.5
Disruptiveness	+	+	++
Modeling-tool support	−	+/−	−−
Cognitive expressiveness	+	+/−	+/−
Non-diagrammatic UML notation	o	−	−

O4.2, O4.3, O4.5; see Fig. 4). Depending on the design context, a QOC option might map to several decision options in our catalog or vice versa. *Criteria* indicate properties or effects of adopting given options, e.g., to satisfy certain requirements. Our catalog describes decision drivers for adopting or discarding certain decision options. The three relevant decision options (O4.2, O4.3, O4.5) describe, for example, four drivers which are reproduced in Table 3 (see also decision record D4 in Hoisl et al., 2014a). The decision records enumerate positive and negative links between drivers and options. As such, our catalog offers candidate criteria to be adopted in a QOC analysis. In Fig. 4, two drivers are referenced as QOC criteria (disruptiveness, modeling-tool support). The positive and negative links form the basis for the *assessment* of the QOC options to answer the QOC question. Finally, a QOC *decision* denotes the act of marking QOC options as adopted (see the solid rectangles in Fig. 4). Each decision may be followed by another QOC question. In our example, the choice of providing a structured, textual notation is succeeded by a question on the concrete-syntax style to be used.

Systematically linking QOC diagrams and structured documentation of design decisions to perform design-space analyses has already been proposed. Zdun (2007) integrates QOC diagrams and software patterns for systematic pattern selection.

3. Research questions

By describing and documenting DSML designs through the lenses of decision points and decision records, we take a decision-makers' perspective on a DSML design. On the one hand, a decision-making perspective targets the stakeholder roles relevant for DSML design-decision making (e.g. business designer, software architect, domain engineer, DSML implementer, domain modeler). On the other hand, such a perspective facilitates recording the results of decision making.

A decision-making perspective complements development-process perspectives on DSML development, such as the one proposed by Strembeck and Zdun (2009). A process perspective puts emphasis on the actual development activities and development artifacts in DSML development. However, a decision-making perspective and the resulting design-decision documentation shifts focus on DSML development at a finer grained level of abstraction and makes decision interdependencies explicit—independent of the actual development-process style (Strembeck and Zdun, 2009).

Recording and presenting repeatedly observed design decisions has the potential of facilitating the documentation of design decisions in other development contexts. Such contexts can be set by a new DSML project or by a maintenance task on an existing DSML. In Section 2.2, we make the case for a systematic reuse of previously gained process and decision knowledge (e.g. options, drivers, consequences) in design-space analyses. To this end, our study was guided by the following research question:

Research question 1: *What are the design-decision options for UML-based DSML designs reported in scientific literature?*

To answer this question, we conducted a systematic literature review (SLR) and a systematic content analysis. In its preparation, execution, and reporting, we apply established guidelines (Jalali and Wohlin, 2012; Webster and Watson, 2002; Kitchenham, 2004; Zhang et al., 2011). Section 4 summarizes the SLR procedure and its results. An overview of the content analysis of the SLR paper corpus is given in Section 5. The combined SLR and content-analysis protocol can be found in Sobernig et al. (2014).

Each UML-based DSML design can be described by an option set that contains a particular combination of the options described by the decision-record catalog.[7] Therefore, at first glance, our catalog of 27 decision options defines an extensive design space of decision-option sets (see Tables 1 and 2): In a convenience view, one can generate $2^{27} - 1$ unique combinations of these 27 decision options to characterize DSML designs. This convenience view neglects any combinatorial constraints, such as the ones imposed by documented associations between decision options.

We require that a DSML design must at least report one option on language-model definition (D1) and another one on language-model formalization (D2). This requirement follows from the following three assumptions: First, we only consider DSMLs that were designed in a language-model-driven development style (Strembeck and Zdun, 2009), which implies at least one decision on D1. Second, at least one of the corresponding options for the extension of the UML must be chosen (D2) in order to qualify as a UML-based DSML. Third, backed by a documentation analysis based on scientific publications, we expect that these two design dimensions are mandatory in scientific reports on DSML projects. This might even be the case when they were not explicitly addressed in the actual design-decision making of the DSML designers.

Even when considering the above presence conditions on D1 and D2, our catalog still allows for expressing a vast space of distinct options sets for DSMLs: 117,964,800 unique option combinations! This number computes as follows starting from the 27 options in our catalog: $2^4 - 1$ (D1) times $2^4 - 1$ times (D2) times 2^{27-4-4} (D3–D6). In practice, however, we expect a reduced observable design-decision space across existing DSMLs. In particular, there is partial evidence suggesting that certain UML extension techniques are more commonly adopted than others (e.g. profiles; Nascimento et al., 2012; Pardillo, 2010). This leads to our second research question:

Research question 2: *What are frequently observed decision options and frequently observed combinations of decision options (option sets) in and across existing UML-based DSML designs?*

Collecting option sets that represent existing DSML designs allows for important insights beyond their mere observability. If several existing DSML designs exhibit identical option sets, then such *recurring* option sets indicate repeated decision-making practice in designing DSMLs. At the same time, repeatedly observed option sets can serve for grouping different DSML designs into families that share characteristic combinations of design options.

To address our second research question, we are thus interested in frequency patterns that occur for the collected option sets. Such frequency patterns can help in characterizing an empirically *observable* subset of the theoretically possible design space that is described by our catalog. Detecting and interpreting such decision patterns (e.g. hub decisions) has been reported as an important use case for design-rationale documentation (Kruchten et al., 2006).

In order to characterize the observable design-decision space for UML-based DSMLs, we mine for frequent option (sub-)sets using an analysis that is based on *frequent item sets* (Borgelt, 2012; Hahsler et al., 2005). We are interested in commonly recurring combinations of decision options which are (proper) subsets of observed option sets. Technically, we want to extract option (sub-)sets that adhere to certain constraints (i.e. minimum support, closedness, freeness, maximality; see Borgelt, 2012; Hahsler et al., 2005). Sobernig et al. (2014) provide a technical background on those concepts in the context of our study.

Such frequent option (sub-)sets can express characteristic fragments of a DSML design as well as complete DSML designs (also called "prototype option-sets"). These option sets can differ in terms of the number of options (or, proper option subsets) that they contain (size), in terms of their relatively higher or lower levels of support, and whether they are contained as-is in the base of observed option sets or not; i.e., whether they totally describe at least one DSML design alone, rather than a fragment of it. Table 4 summarizes the three kinds of characteristic option (sub-)sets that we consider in answering our second research question. Section 5 elaborates on our findings.

4. A review-driven approach to capturing generic DSML design rationale

To answer our research questions, we set out to distill *generic* DSML design rationale from a maximal number of DSML design documents. This way, the extracted design rationale is more likely valid beyond single DSML designs. To the best of our knowledge, no such source of generic rationale on UML-based DSML designs existed prior to our decision-record catalog. The secondary studies on DSML development, which we identified as related work, do not reflect actual design-decision making in DSML development projects; specifically not for UML-based DSMLs (see Section 7). Therefore, our emphasis was on gathering *primary* studies on UML-based DSMLs. A primary study is a piece of design documentation authored by the respective DSML developers themselves. In scientific literature, primary studies come as DSML solution proposals and/or personal experience reports (Wieringa et al., 2005).

Capturing design rationale (DR) behind a UML-based DSML design can be achieved in different, systematic ways (Dutoit et al., 2006). First, one can recover design decisions by reviewing DSML artifacts after the fact (e.g. abstract-syntax or concrete-syntax specifications). This can either be done by the DSML developers themselves or by third-party experts in DSML development and in DR documentation (see also Section 2.2). Second, DSML designers might record their rationale themselves as a byproduct of the decision-making process. A third source are records of communication

[7] Option sets are also an outcome of the coding step during content analysis (coding form; see Section 5.1) and a prerequisite for data mining (see RQ2).

Table 4
Overview of the option-set constructs considered for analyzing frequency patterns in the selected DSMLs. Each construct is defined as a set of data-mining restrictions (e.g. closedness, maximality, freeness; Borgelt, 2012) over the space of (frequent) option sets representing the reviewed DSMLs. Details on these underlying data-mining restrictions are provided in Sobernig et al. (2014).

Kind of option (sub-)set	Description
Smallest common option subset	A frequent option subset which is also a smallest (i.e. of minimal size) recurring *proper* option subset contained by observed DSML designs and/or by observed design fragments. We distinguish between two kinds of smallest common option subset: (1) option subsets specific to one decision record (D1–D6); (2) option subsets specific to two or more decision records (D1–D6).
Prototype option-set with frequent extensions	A frequent option set which represents a largest option subset (design fragment) which was also frequently found to represent complete DSML designs. This prototype option-set is frequently found extended by adding other (frequently observed) options. In this sense, it represents an *evolutionary* prototype to derive extended DSML designs.
Prototype option-set with infrequent extensions	A frequent option set which represents a largest option subset (design fragment) which was also frequently found to represent complete DSML designs. Extensions that add options to this (*evolutionary*) prototype for deriving other option sets (DSML designs) are infrequent.

created by DSML designers, for example, language-user documentation, change/maintenance documentation, and scientific publications. Finally, if available, design-support software can be used for documenting design-decisions (e.g. IDEs including support for design-knowledge management; Tang et al., 2010). To the best of our knowledge, however, contemporary design-support software for DSMLs does not provide DR capturing facilities.

As in other fields of software development, the *capture problem* (Burge et al., 2008; Dutoit et al., 2006) often prevents DSML design rationale from becoming documented explicitly by DSML developers. In earlier, preparatory studies including manual design reviews, a snowballing study, and a pilot SLR, we found, however, that *scientific publications* of DSML developers are important primary studies for documented DSML design rationale (Hoisl et al., 2012b; Filtz, 2013). Scientific publications must not necessarily document the rationale for design decisions directly. Often, they provide a systematic overview of DSML design artifacts and references to them (e.g. abstract-syntax or concrete-syntax definitions). Moreover, DSML developers submitting to relevant scientific publication venues (e.g. SoSyM journal, MoDELS conference) are more likely to report UML-specific design decisions explicitly. They adopt best-practice examples for design documentation from OMG standard documents directly, from already published papers in the respective venues, and from secondary studies on UML-based DSML development (e.g. Paige et al., 2000; Grant et al., 2004; Robert et al., 2009; Selic, 2007). Finally, for some DSMLs being research prototypes only, scientific reports are often the only source of documented design decisions.

For these reasons, we opted for a systematic literature review (SLR). The main goal of this SLR was to identify a maximum number of scientific publications which document design rationale on UML-based DSMLs as primary studies. The targeted scientific publications were required to be marked by a sufficient documentation quality. By documentation quality, we refer to the correctness and the completeness of the design documentation.

4.1. Planning & conducting the review

The SLR was performed in three steps (see Fig. 5). First, to provide a basis for evaluation of the search procedure, we established a corpus of reference publications (viz., a quasi-gold standard, QGS; see Section 4.1.1). Based on this reference corpus, we identified the search engines for an automated publication search. We refined the corresponding search terms and the queries in several iterations. Second, we performed the actual engine-based publication search (see Section 4.1.2). Based on the bibliographical records extracted publications selected up to this point, we then performed a backward-snowballing search (see Section 4.1.3). Backward snowballing is the practice of manually identifying additional publications for selection from the reference lists (citations) of a given set of publications (Jalali and Wohlin, 2012; Webster and Watson, 2002).

We selected publications for inclusion and assessed their quality based on predefined criteria at each stage. The selection and quality-assessment decisions involved multiple raters per publication (i.e. the authors). Therefore, we report the inter-rater reliability (IRR) for the data extracted from the selected publications. IRR measures document patterns of agreement and disagreement between two and more raters in their assessments of details extracted from the selected publications (Gwet, 2012). This section summarizes the adopted procedures at each stage (working tasks, criteria selection), the intermediate results (QGS, review, and snowballing corpora), and the intermediate evaluations (validity, reliability measurement).

4.1.1. Quasi-gold standard

To guide publication search, and to report the search validity, we developed a quasi-gold standard (QGS; Zhang et al., 2011; Kitchenham and Brereton, 2013). A quasi-gold standard is a collection of manually selected papers from a number of venues and outlets (e.g. journals, conference series), which are known and recognized for publishing work on the topic areas under review by a relevant community; in our case: model-driven development, UML, and DSMLs. The results of the automated literature search must include the quasi-gold standard. Otherwise, the search strategy must be adapted.

As opposed to a "gold standard" collection of papers for evaluating literature searches, a *quasi*-gold standard is specific to a set of venues/outlets over a specified time span (Zhang et al., 2011). We used the QGS to guide the main, semi-automated literature search (i.e., to select the search engines and to guide the selection of search terms) and, finally, to validate its quality attributes (i.e. sensitivity and precision).

Constructing our own QGS corpus became necessary because, at the time of designing the SLR, there was no adequate third-party publication corpus available. In 2013, the only candidate was a prior mapping study by Nascimento et al. (2012) on application domains of DSLs, DSL development tooling, and research agendas on DSL engineering. However, our study aimed at a specific subset of DSLs (viz. UML-based DSMLs). Nascimento et al. (2012) reviewed 170 publications on DSMLs and only 21 of these were specific to the UML. Finally, while covering an extensive publication period (1966–2011), the mapping study by Nascimento et al. (2012) does not extend to 2012 as required by our study. To the best of our knowledge, there is still no adequate third-party corpus available.

The publication collections obtained from our earlier research steps (Hoisl et al., 2012b; Filtz, 2013) did not qualify as a reference corpus either. This was because of their strong bias towards our research projects and our own experiences. Furthermore, the publications from our prior research (Hoisl et al., 2012b) were also limited to DSMLs for security-related application domains. The third-party publications collected during our pilot study were not considered as QGS candidates because of flaws in our pilot SLR design (Filtz, 2013).

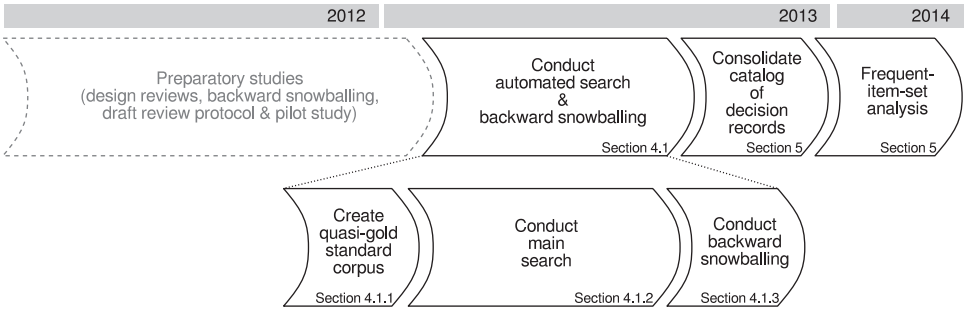


Fig. 5. A timeline overview of the research stages and the corresponding sections in this paper.

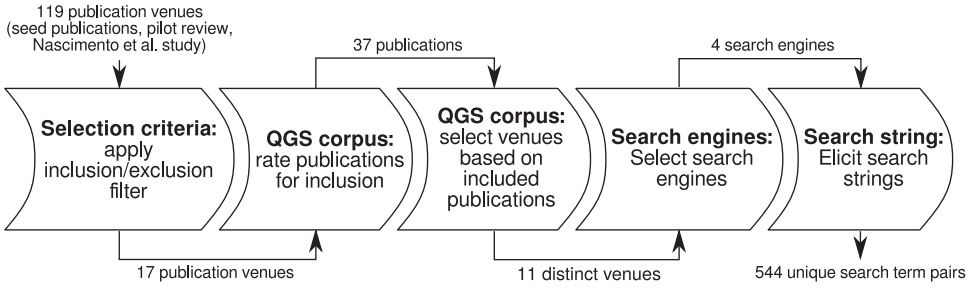


Fig. 6. Sub-steps of establishing a QGS corpus and eliciting search strings for the main search.

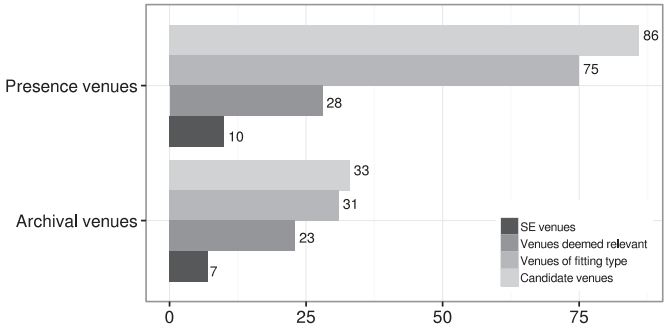


Fig. 7. Overview of the stepwise selection of publication venues for establishing the quasi-gold standard (QGS) corpus.

Fig. 6 visualizes the five sub-steps of QGS construction. As a starting point, we considered all publications collected from the three sources above. These included our own DSML publications (Hoisl et al., 2012b), the third-party publications from our pilot review (Filtz, 2013), and the third-party publications returned by the mapping study (Nascimento et al., 2012). The three sources accounted for 159 papers and 119 publication venues in total. The publication venues included 86 presence venues (conferences, symposia, and workshops) and 33 archival venues (journals, monographs; see also Fig. 7).

Selection criteria. In a pair session, the 119 publication venues were filtered by two authors (Hoisl, Sobernig). Fig. 7 summarizes the intermediate results of filtering. Finally, 17 venues were selected. The seven journals included top-tier SE venues such as IEEE TSE, ACM TOSEM, and SoSyM. The ten selected conference venues comprised i.a. ICSE, OOPSLA, MoDELS, and ASE.[8] The selected venues complied with four criteria:

1. *Time coverage*: A publication venue covers the years between and including 2005 and 2012. In 2005, the UML 2.0 specification was published. 2012 marks the year before performing the engine searches in January and February 2013.
2. *Community relevance*: An archival venue is deemed relevant by a scientific audience if it is listed with the ERA 2012 journal list (Australian Research Council, 2012). The ERA list resulted from a public and international consultation process among scientists. 23 of the 31 journals were listed with ERA 2012. For a presence venue, we examined whether the venue has a regular publication history depending on the venue format (e.g. yearly, bi-yearly) during the review period. This was the case for 28 of the 75 conferences.
3. *SE focus*: The venue has a dedicated software-engineering (SE) focus. The two authors judged a venue based on publicly available SE venue lists, including SCImago Journal Rank (Scimago Lab, 2013) and Microsoft Academic Search (Microsoft Corporation, 2013). 17 out of the remaining 51 venues fulfilled this condition.
4. *Content maturity*: The venue is committed to publishing mature and scientifically rigorous content. In general, we verified whether there was a peer-review procedure in place. As for presence venues, 11 workshops were excluded. Two archival venues other than journals were discarded (i.e. a festschrift and a project-report monograph).

Quasi-gold standard corpus. Two authors manually screened the selected 17 venues. This involved 418 journal volumes and 80 proceeding issues published between 2005–2012. In this initial iteration, candidate papers were selected by reviewing title, author-provided keywords, and abstracts (in this order). The reviewers used centrally maintained publication-history records of each venue (e.g. DBLP[9]) and the publication bases of the venue publishers (i.e. IEEE, ACM, Springer, Elsevier) to additional important metadata (e.g. abstracts).

The screening result was a collection of 83 articles (52 journal and 31 proceedings articles). Each article was rated for inclusion into the QGS corpus independently by two authors. 37 publications were positively rated by both authors and formed the final QGS corpus (24 journal and 13 proceedings articles). The two independently rating authors arrived at the same selection decision (inclusion or exclusion)

[8] See Appendix A in Sobernig et al. (2014) for the venues' full names.

[9] See http://dblp.uni-trier.de/ , last accessed: Feb 2, 2015.

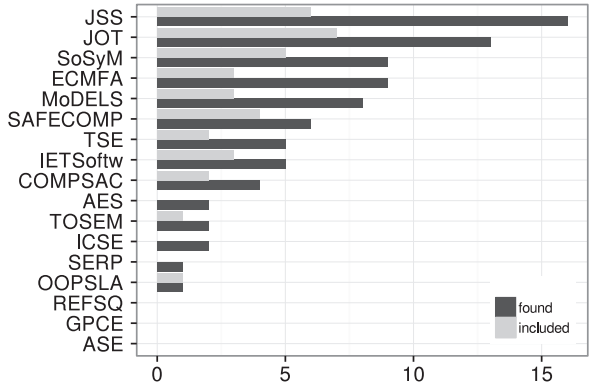


Fig. 8. Publications found during manual screening of the 17 QGS venues and publications finally included into the QGS corpus (per venue; ordered by the decreasing number of found papers). See Appendix A in Sobernig et al. (2014) for the venues' full names.

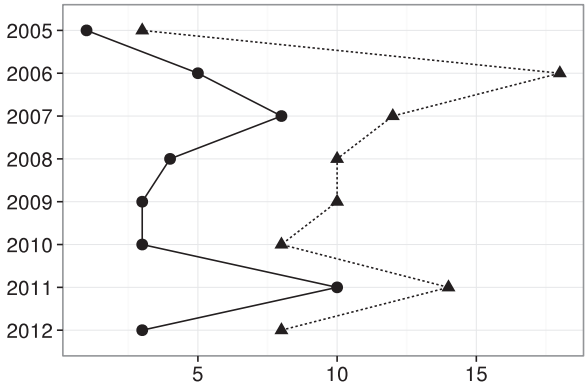


Fig. 9. The numbers of publications *found* (dashed line) and *included* (solid line) into the QGS corpus, per publication year (2005–2012).

for 75.9% of the 83 publications without negotiation. The conflicting selection decisions were first revisited in a joint session between the two rating authors. Then, if necessary, the third, non-involved author adjudicated on an article at issue.

Any inter-subjective rating process bears the risk of random ratings and personal bias, so that the percent agreement of 75.9% is likely to be overstated (Gwet, 2012). Applying standard chance-correction (Cohen's Kappa), we established that the two rating authors achieved actual decision agreement beyond chance (inclusion/exclusion) in approx. 51% of the total expect cases beyond chance. In technical terms, Cohen's Kappa coefficient $\widehat{\kappa_C}$ on the 83 publications and the corresponding 166 ratings amounted to 0.51 $\pm$ 0.009.[10] Kappa's chance correction results from a worst-case assumption (random assignment), the "true", but unknown agreement level lies between 0.51 and 0.759. According to standard Kappa benchmarks (Fleiss, 1981), this indicates an intermediate to good chance-corrected (worst-case) reliability level. Therefore, we considered the underlying selection procedure (criteria) reliable. Any decision conflicts were resolved as outlined above.

Comparatively equal shares of candidate articles published in journals and conferences, respectively, made it into the QGS corpus. Approximately 46% of the journal articles (24/52) and 42% of the proceedings articles (13/31) entered the corpus. Fig. 8 depicts the number of finally included publications per venue. JOT, JSS, SoSyM, and SAFECOMP were the top four venues contributing publications to the QGS corpus. These top four accounted for ≈60% (22/37) of the included publications. From six venues, not a single publication was selected. The distribution over time shows that ≈49% of the included publications (18/37) were published in two peak years: 2007 and 2011 (see Fig. 9).

Based on the 37 QGS publications, two subsequent steps were performed. On the one hand, the relevant search engines for the automated search were identified. On the other hand, a search string for the automated search was constructed from the QGS corpus.

Search engines. The 37 QGS publications were published in eleven distinct venues, which are issued by five different publishers: Springer (15 papers), AITO (7), IEEE (7), Elsevier (6), and ACM (2). For these publishers, we identified four search engines: SpringerLink, IEEE Xplore, Scopus, and ACM Digital Library. These met previously defined requirements, such as full time-coverage, access to bibliographical metadata, and minimal content overlap (Sobernig et al., 2014).

Search string. We extracted the search string from the QGS publication corpus in a systematic and subjective manner (Zhang et al., 2011). The extracted 49 search terms were grouped into two term sets: 17 search terms specific to a (meta-)modeling technology (e.g. "uml", "mof") as well as 32 search terms specific to DSL and DSML development (e.g. "profile", "metamodel"). The resulting search string, from which the four engine-specific search expressions were derived, is documented in Sobernig et al. (2014). This final search string was developed in several iterations to maximize search sensitivity. Search sensitivity reflects the number of QGS publications retrieved by a given search string (Zhang et al., 2011). The iterations are reported in full detail in Sobernig et al. (2014).

The final search string represents pairs of search terms, with each pair drawing one search term from each term set. More precisely, the terms within each set were considered alternatives (exclusive-or; e.g. "uml" *xor* "mof"). Then, to arrive at term pairs, the two sets were merged element-wise using conjunction. The final query expression, therefore, represented 544 unique term pairs (e.g. "uml" *and* "profile"). This tactic met a triple objective: First, we arrived at more precise search terms regarding our study context. Second, this tactic minimized any semantic overlap between terms related to technologies and DSML engineering, respectively. Third, all four engines supported this structured search string. We could enter the search string as one concatenated boolean expression in a disjunctive-normal form (leaving aside minor concrete-syntax differences), rather than in its complex expansion into unique term pairs.

4.1.2. Main search

In this step, we conducted the automated publication search using the search string developed in the previous step on the four selected search engines: ACM Digital Library, IEEE Xplore, Scopus, and SpringerLink. Fig. 10 provides an overview of the sub-steps of this automated search.

Engine-based search. This step involved four activities: search execution, duplicate cleansing, validity computation, and QGS-based capping. *Search execution* yielded 5,778 search hits split into four result sets, one for each of the four search engines. When extracting the results, we applied and preserved the relevance-based sorting of search hits (as provided by each engine).

From the ACM Digital Library, 933 hits were retrieved. IEEE Xplore returned 1,845 hits. Searching Scopus and SpringerLink yielded 2,000 and 1,000 hits, respectively (see Fig. 11). IEEE Xplore, Scopus, and SpringerLink cap the result sets when exporting them into a processable format. For IEEE Xplore, which enforces a capping at 2,000 hits,

[10] $\pm$ signals the leave-one-out Jackknife variance estimate $Var(\widehat{\kappa_C})$ of Cohen's Kappa statistic $\widehat{\kappa_C}$ to quantify the degree of statistical insecurity inherent in the data-generating process.

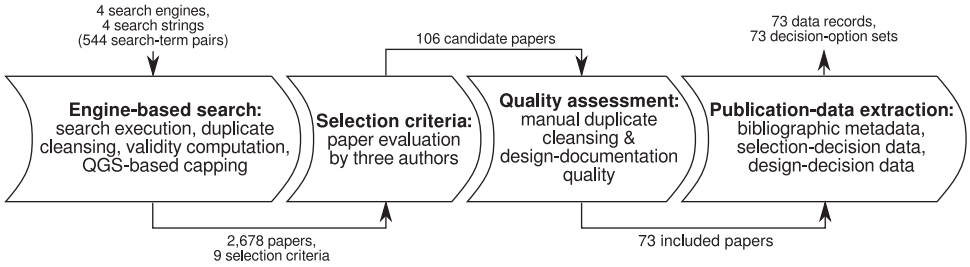


Fig. 10. Sub-steps of conducting the main search.

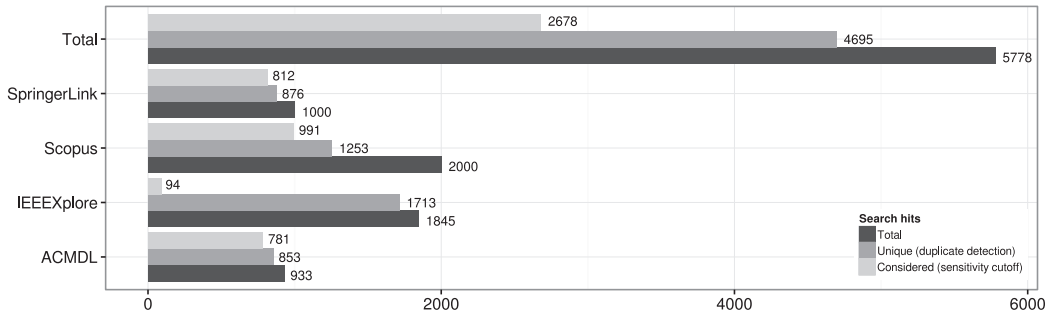


Fig. 11. Overview of cleansing (duplicate detection) and reduction steps (sensitivity cutoff) for each result set, yielding in total 2,678 papers entering the selection procedure.

the restriction did not apply given a smaller number of actual hits. As for Scopus and SpringerLink, the numbers reported above correspond to the capped result sets. The actual result sets of Scopus and SpringerLink amounted to more than 16,500 and more than 8,500 hits, respectively.

The four engines demanded their own variants of the complex search string (see Section 4.1.1). This was mainly due to different concrete syntaxes of the query processors. In addition, all provided different means to limit the searches to our required range of publication years (2005–2012) and to publications having a full-text body written in English language. The resulting four search strings are documented in the SLR protocol (Sobernig et al., 2014).

Duplicate detection was performed for each result set and across the four result sets. In total, we removed 1,083 hits as unwanted duplicates (see Fig. 11). These duplicated hits were almost entirely caused by redundant hits between two and more result sets, rather than replicated entries within one result set alone. As a minor exception, however, Scopus contained ten duplicated hits within its result set. Duplicate detection was performed in three passes: First, we used the Document Object Identifiers (DOI) provided by the searched publication data bases for their hits. 5,232 of the 5,778 hits were equipped with a DOI. Second, for the publications lacking a DOI, we matched their publication titles character-wise and in a case-insensitive manner. Third, to overcome possible barriers of the conservative, exact matching strategy, we matched hits based on their tokenized titles and a Jaccard similarity function (Naumann and Herschel, 2010). Barriers included the heterogeneity in the titles in the presence of punctuation, encoding artifacts, and typos. Duplicate removal was performed in a manner preserving the relevance sorting of the four result sets.

Validity computation was performed on the remaining, unique 4,695 hits (see Fig. 11). For this, we computed the quasi-sensitivity of the engine-based search with respect to our QGS corpus. The overall objective was to obtain a search sensitivity at a level between 70 and 80% (Zhang et al., 2011). In other words, between 70 and 80% of the QGS publications should be contained in the collated result set. We arrived at an overall quasi-sensitivity of ≈75.7% for the main search: 28 of 37 QGS publications were successfully retrieved across all four search engines. Scopus contributed 14, SpringerLink seven, ACM Digital Library four, and IEEE Xplore three QGS publications. Note that this validity computation was repeatedly performed, actually, over several search iterations to maximize the sensitivity.

Finally, we applied a *capping* of the search hits based on the retrieved QGS publications. We defined a cutoff at the position of the last QGS publication found in each result set. For example, the last QGS publication in the set of 853 unique ACM Digital Library hits was listed at position 781. Thus, position 781 became the cutoff position for result set of ACM Digital Library (see Fig. 11). This capping strategy was justified because of the substantial quasi-sensitivity level achieved with the engine searches (see above). After capping, 2,678 hits or 46.4% of the total search hits entered the manual selection procedure.

Selection criteria. The 2,678 papers in the cleansed, reduced, and collated result set were evaluated by two authors according to the nine selection criteria (i.e. four venue-specific and five publication-specific ones), yielding a total of 5,709 inclusion and exclusion decisions, respectively. After this selection step, 106 of the 2,678 papers (≈4%) were included. The nine selection criteria accurately reflect the study's objective. To give one example, the fact that we were interested in DSML designs based on the UML 2.0 led us to impose 2005 as the starting year, given that the UML 2.0 specification was formally released in July 2005.

The first four criteria were specific to the publication venue of a given search hit. These four criteria corresponded to the ones already applied for selecting the QGS venues: time coverage (2005–2012), community relevance, SE focus, and content maturity (see Section 4.1.1). Testing the venue-specific criteria first allowed us to make a selection decision on papers sharing a venue at once. This helped us considerably to cope with the extensive result set.

Once satisfying all venue-specific criteria, a paper was checked for five publication-specific criteria. Most importantly, it was established whether the paper's full-text was accessible to us, e.g. whether our institutional subscriptions to the publisher's digital libraries covered these items, or whether an author copy could be retrieved otherwise. Only 47 papers could not be accessed at all. Based on the full-text, it was then decided whether a paper reports an actual DSML design. This decision involved several checks regarding the paper type (e.g.

excluding editorial introductions or position papers) as well as the paper's scope as either primary or secondary studies on DSMLs.

To determine whether a candidate paper and the corresponding DSML design were based on the UML 2.x, we first checked the paper's references list for citations of the corresponding OMG UML specification documents. If missing, we reviewed the full-text for clarifying statements and the various design artifacts. If available, we also attempted to infer the UML version dependencies from the concrete-syntax elements used in diagrams. In parallel, we verified whether all necessary details of the DSML design are reported by the candidate paper. For example, we verified whether a valid specification of a UML profile and its elements is provided.

In total, approximately a quarter of the papers (i.e. 704 out of 2,678) were excluded because they did not comply with the report-type criterion. Papers not reporting on UML-based DSML designs or a secondary study on DSMLs fell into this group. 152 of the excluded publications were checked and agreed on by two authors independently from each other.[11] Approximately, 70% of the papers (1,868/2,678) did not satisfy all of the eight remaining venue- or publication-specific criteria.

Quality assessment. The 106 papers that were included at this point were then further assessed for two quality properties: (1) duplicates and (2) erroneous DSML designs or design reports. In total, 33 papers did not pass this assessment step. Given the inherent limitations of the semi-automated duplicate detection (see above), quality assessment involved manual checking for duplicates when working on the full-text bodies and the design details. This way, we found another eight duplicates (see `duplicated` in Fig. 12). For example, one hit turned out having two replicated entries between two or more result sets which did not share the same DOI. This way, they had escaped the semi-automated duplicate detection before.

The extent, to which we could extract design rationale from the included papers, was directly dependent on the completeness, the expressiveness, and the correctness of the design documentation by the publication authors. This challenge is a concrete instance of inappropriate rationale representation in design-rationale capturing in more general (Dutoit et al., 2006). We checked the documented DSML designs for a number of well-discussed, primarily syntactic issues when it comes to designing UML-based DSMLs (see, e.g., Henderson-Sellers and Gonzalez-Perez, 2006; Pardillo, 2010; Atkinson et al., 2003). To the extent concrete-syntax issues also signal semantics violations, we also considered certain classes of semantics defects. This way, we identified 25 papers whose DSML designs were reported in an inappropriate, non-extractable manner and were therefore excluded from data extraction. The quality issues are reported in full detail in the SLR protocol (Sobernig et al., 2014).

Extraction of publication data. After having completed the quality assessment, 73 papers representing 2.7% of the original search hits remained (see Fig. 12). For this final publication set, we extracted or completed the publication-specific data. Besides, we coded the DSML design decision data according to our decision-record catalog (see Section 5.3).

[11] Validation by the second extractor for false negatives was based on a 20% sample of all 704 papers processed by the first extractor and considered as either wrong report type or as containing erroneous DSML documentation artifacts. Due to this sampling, in a probabilistic projection, we risking having missed out between 0 and up to 23 publications. With one false negative found in this sample, in the worst case, we risk having missed up to 33 false negatives in the total set of 704 publications (binominal confidence interval (0, 34]; 99% confidence level). Note, however, that any false negative at this point might still have been ruled out as a true negative due to four subsequently evaluated criteria (e.g. a wrong UML version) and during quality assessment (i.e. duplicate removal). With only 73 out of 106 publications (i.e. ≈68.9%) having entered the final publication corpus (see Fig. 10), we arrive at the number of up to 23 potentially missed publications in the paper corpus as false negatives under the above empirical probability: 33∗0.689 ≈22.7

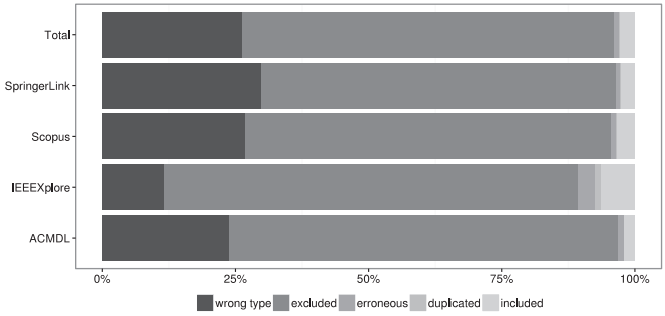


Fig. 12. Overview of important selection decisions on the 2,678 papers entering the selection procedure, with 106 entering quality assessment, and 73 finally becoming included.

Overall, we recorded 15 metadata items for each included paper. These items included bibliographical entries such as the publication year, paper-specific keywords, and the venue. In addition, the selection decision was noted for each hit. The included papers were further described by eight decision-mining entries: Beyond the DSML project name (if available), the DSML projects were classified according to their application domain(s) and the relevant UML diagram types. Next, the respective decision options identified for a given DSML design were recorded with respect to each of the six decision points from our decision-record catalog. Each DSML design became represented as one decision-option set (see Section 2.1.2). The final paper corpus including the 73 papers identified via the engine-based search is characterized according to the extracted publication data in Section 4.2. In Section 5.3, we report on the extracted design-decision data in more detail.

Inter-rater reliability. 2,400 of the 2,678 search hits, which remained after applying the cutoff points, were reviewed by one author. 278 were rated and assessed for inclusion or for exclusion by two authors. This co-rated subset comprised all publications included by one author, all publications considered erroneous, and a randomized 20% sample of the papers excluded by one author.

For this co-rated subset, the respective rating authors achieved a percent agreement of 88.5%. That is, for more than 245 of the 278 publications two authors arrived at the same selection decision (i.e. included or excluded) independently from each other and without negotiations. Chance-corrected (worst-case) inter-rater agreement $\widehat{\kappa_C}$ amounted to $\approx 0.875 \pm 0.0004$.[12] This represents a *very good* (worst-case) agreement level according to standard benchmarks (Gwet, 2012; Fleiss, 1981). Therefore, we considered the underlying selection procedure (criteria) reliable. Any conflicting selection decisions were first revisited in a joint session between the two rating authors. Then, if necessary, the third, non-involved author adjudicated on a search hit at issue.

4.1.3. Backward snowballing

To incorporate prior work considered relevant by the authors of the 73 included papers, we performed a manual, citation-based search using the bibliographical references taken from the 73 included papers. We followed a backward-snowballing procedure as documented in Jalali and Wohlin (2012) and Webster and Watson (2002). The procedural steps are summarized in Fig. 13.

Manual search. The snowballing procedure ended after having completed three search iterations. The initial iteration used the 73 papers included from the main search hits as a start set (see Section 4.1.2).

[12] Missing data in terms of hits only rated by one author have been considered for computing the marginal probabilities in deriving $\widehat{\kappa_C}$. $\pm$ signals the leave-one-out Jackknife variance estimate $Var(\widehat{\kappa_C})$ of Cohen's Kappa statistic.

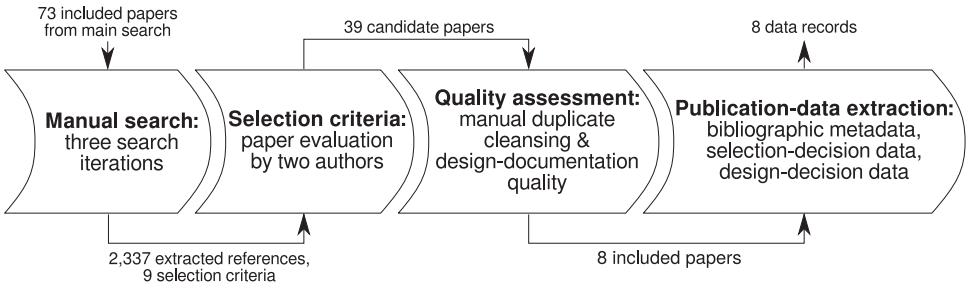


Fig. 13. Sub-steps of conducting the citation-based search (backward snowballing).

The subsequent two iterations were triggered by newly included papers from a previous iteration. After having completed the third iteration, no new candidate papers were found. In each iteration, we first extracted the references lists from the start-set publications. In the initial iteration, we obtained 2,116 references from 73 papers. In the second iteration, we worked on 200 references taken from seven papers included during the first iteration. The third and last iteration used 21 references from one additional paper that was found during the second iteration. Across the three iterations, we reviewed a total of 2,337 references.

All 2,337 extracted references were checked by at least one reviewer, duplicate detection was performed manually. This was a consequence of the unavailability of structured and processable reference lists from all four data sources (ACM Digital Library, IEEE Xplore, Scopus, and SpringerLink). On top, the bibliographical reference data was widely heterogeneous (mixed bibliography styles, missing bibliographical entries such as DOIs).

Selection criteria. We made the selection decision on the snowballing hits by applying the nine venue- and publication-specific criteria as for the main, engine-based search (see Section 4.1.2). This way, we selected another 39 publications as inclusion candidates. In addition, snowballing yielded one more QGS publication adding to the 28 QGS publications from the main search (see Section 4.1.2). We, thus, arrived at a final quasi-sensitivity of ≈78.4% (29/37) for our review.

Approximately 7.3% of the extracted references (170/2,337) did not relate to reports on UML-based DSML designs as primary studies, but rather to other (non-UML) DSMLs, to secondary studies on DSML designs, or they were not related to DSMLs at all. 37 of these off-topic references were verified by two authors independently from each other.[13] The majority of 91% (2,128/2,337) did not satisfy the eight remaining venue- and publication-specific criteria.

Quality assessment. The 39 candidate publications were then assessed for duplicates and issues of design-documentation quality. In total, by comparing the snowballing hits to the cleaned automated search hits, we identified and removed 25 unwanted duplicates. For six of the remaining 14 publications, the documentation-quality assessment revealed quality issues similar to the ones during the main search. The issues included ambiguous UML metamodel extensions, faults in defining and applying the UML profiles, and syntax errors (see the SLR protocol; Sobernig et al., 2014). Eight publications entered the paper corpus (see Fig. 13).

Extraction of publication data. From the eight additional publications, we extracted bibliographical metadata and coded their design-decision data (see Sections 4.1.2 and 5.3, respectively). Missing (e.g. publication years) and inconsistent metadata (e.g. title formats) prevented us from extracting descriptive statistics over all backward snowballing items.

Inter-rater reliability. During selection and quality assessment, 2,268 of the 2,337 snowballing hits were reviewed by exactly one author. 69 hits were assessed for inclusion and for exclusion by two authors. The co-rating authors agreed on more than 80% of the rated references in each of the iterations. The corresponding chance-corrected (worst-case) agreement levels $\widehat{\kappa_C}$ for the each iteration were ≈0.967 ± 0.0005, ≈0.833 ± 0.0345, and 1, respectively.[14] Therefore, we achieved a *very good* to almost *perfect* level according to standard Kappa benchmarks (Gwet, 2012; Fleiss, 1981) for each iteration. Therefore, we considered the underlying selection procedure (criteria) reliable. Any conflicting selection decisions were first revisited in a joint session between the two rating authors. Then, if necessary, the third, non-involved author adjudicated on a search hit at issue.

4.2. Paper corpus

To summarize, the main engine-based search and the snowballing searches resulted in retrieving and reviewing 5,015 publications. The main search accounted for 2,678 (53.4%), snowballing for another 2,337 publications (46.6%). From these 5,015 publications, we considered a total of 81 articles as relevant: 73 from main search plus 8 from snowballing. Recall that from 37 QGS publications, 29 had been returned by the main and snowballing searches. To complete the paper corpus, we re-considered the missing eight QGS publications for inclusion based on all 9 selection criteria. This way, we classified two QGS journal articles and one QGS conference article as relevant. We so arrived at a paper corpus of 84 publications out of a total of 5,023 reviewed publications (the complete list of all 84 publications can be found in the Appendix). The corpus was composed of 54 conference articles (64%) and 30 journal articles (36%).

Fig. 14 shows the timeline distribution of the 84 included articles, discriminating between archival and presence venues. The release years of important UML revisions (2.0, 2.1.2, 2.2, 2.3, and 2.4.1) in this time window are plotted as vertical lines. In general, the total number of publications on UML-based DSMLs, which are considered relevant for the scope of our study, appears to be in decline. Conference articles show a clear decrease and journal articles stagnate. 2006 and 2011 represent intermittent peaks. This general observation of a

[13] Validation by the second extractor for false negatives was based on a 20% sample of all 170 papers processed by the first extractor and considered as wrong report type. Due to this sampling, in a probabilistic projection, we risk having missed out between zero and up to seven publications. With one false negative found in this sample, in the worst case, we risk having missed up to 31 false negatives in the total set of 170 publications (binominal confidence interval (0, 32]; 99% confidence level). Any false negative found at this point might still have been ruled out as a true negative due to four subsequently evaluated criteria (e.g. a wrong UML version) and during quality assessment (i.e. duplicate removal). With only eight out of 39 publications (i.e. ≈20.5%) having entered the final publication corpus (see Fig. 13), we arrive at the number of up to seven potentially missed publications in the paper corpus as false negatives under the above empirical probability: 31∗0.205 ≈6.4.

[14] Missing data in terms of hits only rated by one author have been considered for computing the marginal probabilities in deriving $\widehat{\kappa_C}$. ± signals the leave-one-out Jackknife variance estimate $Var(\widehat{\kappa_C})$ of Cohen's Kappa statistic.

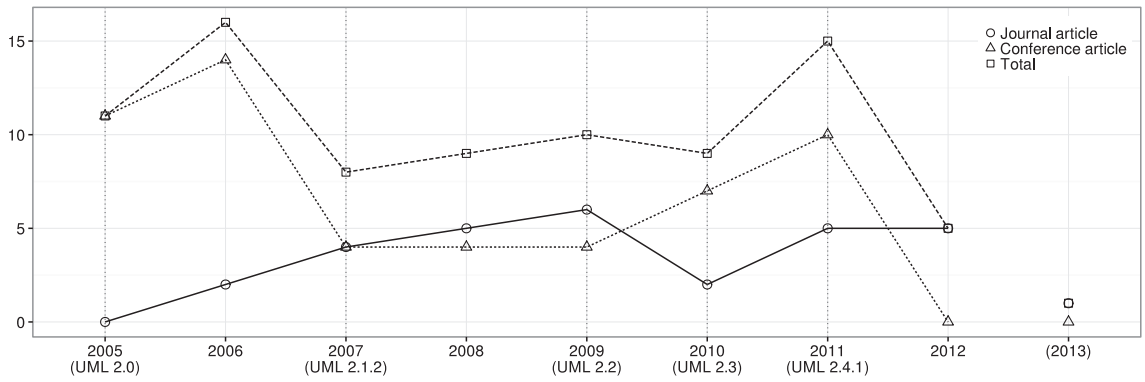


Fig. 14. Finally selected articles per venue type (journal, conference, total) and per publication year (2005–2012). The release years of important UML revisions are plotted as vertical lines. Note that the paper corpus extends to 2013. This is due to running the search in Spring 2013 permitting results from 2013 to handle the ambiguity of "publication year" as either venue or paper-publishing year.

Table 5
Included articles per publication venue. Journals are flagged with an asterisk (∗). See Appendix A in Sobernig et al. (2014) for the venues' full names.

Frequency	Publication venue
10	MoDELS, SoSyM∗
9	ECMFA
4	IST∗, JOT∗, SAC
3	SAFECOMP
2	COMPSAC, CSI∗, EDOC, ICWE, IETSoftw∗
1	CSMR, EMSOFT, ER, ESEC, FASE, HASE, ICSEA, ICSOC, ICWS, IJICIC∗, IJSEKE∗, Informatica∗, ISeB∗, ISSRE, JRPIT∗, JSW∗, OOPSLA, OTM, QSIC, SAM, SCC, SCCC, SEAA, SEFM, SEKE, SFM, SOCA∗, SoMeT, SP&E∗, SPLC

Table 6
Frequency of occurrence of ACM 2012 CCS categories for the corpus of 84 selected articles.

Frequency	ACM 2012 CCS categories
11	Service-oriented architectures
10	Web services
7	Embedded systems, Model verification and validation, Software development techniques
6	Security requirements
5	Model checking, Requirements analysis, Web applications
4	Data warehouses, Graphical user interfaces, Real-time systems, Software development process management, Software testing and debugging
3	Avionics, Business process modeling, Safety critical systems, Software architectures, Software evolution, Software security engineering, System on a chip, Web interfaces
2	Design patterns, Fault tree analysis, Measurement, Metrics, Orchestration languages, Reusability, Software product lines, Software safety, Transportation
1	Access control, Architecture description languages, Availability, Collaborative and social computing, Database design and models, Data mining, Distributed architectures, Electronic commerce, Engineering, Enterprise data management, Enterprise information systems, Error detection and error correction, Estimation, Hardware description languages and compilation, Hypertext languages, Operating systems security, Performance, Robustness, Scenario-based design, Semantic web description languages, Software design engineering, Software fault tolerance, Software performance, Software reliability, Systems analysis and design, Telecommunications, Trust frameworks, Ubiquitous and mobile computing, Use cases, Version control

paper drop is in line with findings on UML profiles prior to 2010 by Pardillo (2010).

In total, the 84 included articles were published in 29 different conference proceedings and in 13 different journals (see Table 5). The top four conference venues (MoDELS, ECMFA, SAC, and SAFECOMP; 14% of all conference venues) account for 26 publications (48% of the total 54 conference publications). The top four journal venues (SoSyM, IST, JOT, and CSI; 31% of all journal venues) account for 20 articles (67% of the total 30 journal articles). In total, more than half of all included publications (46 publications, 55%) were published in eight different venues. There are two conference outliers (MoDELS and ECMFA; 10 and 9 publications, respectively) and one journal outlier (SoSyM; 10 articles) contributing an over-proportionally high number of included articles.

To map the domain coverage of the selected DSMLs, we classified every DSML project of the included publications according to the 2012 ACM Computing Classification System (CCS)[15]. For example, GWfM-Sec is a UML-based DSML for the modeling of security-critical, inter-organizational workflows and provide a mapping to the Web Services Choreography Description Language (WS-CDL). We assigned this article the following CCS categories: *software security engineering, web services, orchestration languages*.

In total, we applied 155 category assignments for all 84 publications (the assignments per publication can be found in the Appendix). This corresponds to a mean of 1.85 categories per publication. The 155 assignments were specific to 61 unique CCS categories. Table 6 reports the resulting frequency distribution of assigned categories. A notable number of DSMLs fall into the areas of service-oriented and embedded software systems, model verification and validation, as well as requirements analysis and security requirements. However, the frequency distribution clearly demonstrates that the paper corpus covers a very broad and diverse range of DSML application domains. This is an important achievement and prerequisite for answering our first research question (see Section 6).

5. Results: Captured design decisions

The paper corpus of 84 publications described the designs of 80 unique DSMLs. This is because, at closer inspection, eight of the 84 publications turned out to cover complementary details of four distinct DSML designs. In the following, we shift perspective from the 84 publications to these 80 unique DSMLs to report on the extracted design-decision data. A list of all DSMLs is provided in the Appendix.

[15] See http://www.acm.org/about/class ; last accessed: Sep 9, 2015.

Table 7
Correspondences between content items of the decision-record catalog and the CA coding schema, which was built according to the guidelines in Schreier (2013).

Decision-record catalog	Coding schema
Decision record	Main category
Decision option	Subcategory
Record name, option name	Category name
Record description, option description	Category description
Applications, sketch, option descriptions	Indicators, positive examples
Option descriptions, decision consequences	Decision rules

5.1. Content analysis

We performed a content analysis (CA) on all corpus papers using the 80 DSMLs as units of analysis. The content analysis involved human coding of the papers' full-text according to a previously defined coding schema. This coding schema was systematically derived from the decision-record catalog by the authors. The CA purpose was a threefold test for this coding schema:

1. Test for saturation: Is each decision category reflected at least once in the corpus?
2. Test for exhaustiveness: Can each unit of coding (e.g., text fragment, specification artifact) be assigned to an already defined category?
3. Test for generalizability: Are the decision categories *general* in terms of being found for at least three units of analysis (DSMLs)?[16]

Hence, we applied a *directed* (deductive) content analysis using *hypothesis coding* (Saldaña, 2013). Note that these three tests are specific to the content-analysis step of our multi-method study. Our two main research questions are exploratory (see Section 3). Running the three tests sets the context for systematically answering the research questions (esp. RQ2; see Section 6).

Coding schema. The coding schema comprised category definitions, category indicators, category examples, and decision rules. The structure and content of the (pre-study) decision-record catalog provided a framework for building the coding schema as a two-level category hierarchy. At the top level, the decision records formed main categories (D1–D*m*). Decision options became subcategories. The subcategory codes (O1.1–O*m.n*) were then used for material coding and coding analysis (see below). The decision records already provided all description items for the coding categories and instruction items for coders (see Table 7).

Indicators complement category descriptions and can signal the presence of a category of interest (decision option) in the material to the coder. Important indicators in our coding schema were *artifact types* characteristic for DSMLs and *technology projections*. The decision record on "language-model constraints" (D3; Hoisl et al., 2014a), for instance, identifies kinds (and examples) of code listings. As for technology projections, this decision-record description refers to constraint-expression languages such as OCL and EVL as examples.

To illustrate the categories during coding, we included positive examples of what the category is meant to cover. Positive examples included concrete instances of artifact types (e.g. concrete code listings) and technology projections provided by the catalog (sketches). Sketches were extracted from DSMLs or secondary studies on extending the UML and on building UML-based DSMLs. The following is an exemplary sketch taken from the decision record on "language-model constraints" (D3; Hoisl et al., 2014a):

> "**Sketch**. Consider the following excerpt from P8: For a UML activity, each action can be guarded by a constraint whose conditions refer to a set of operands and checking operations. At runtime (level M0), the operations are called to evaluate whether an action should be entered, depending upon some contextual state. Constraint 1 shows a constraint-language expression (OCL) accompanied by a complementary textual annotation. […]
>
> *Constraint* 1 : The operands specified in a ContextCondition are either ContextAttributes or ConstantValues :
>
> ```
> context ContextCondition inv :
>   self.expression.operand.oclAsType(OperandType)->
>       forAll(o|
>       o.oclIsKindOf(ContextAttribute)or
>       o.oclIsKindOf(ConstantValue))
> ```
>
> "
>
> (Hoisl et al., 2014a, pp. 22–23)

Some categories exhibited an overlap between concepts and between indicators (examples). Consider, for example, the subtle differences between the concrete-syntax decision options (categories) DIAGRAMMATIC SYNTAX EXTENSION (O4.2) versus MIXED SYNTAX (O4.3). To guide coding under ambiguity, the coding schema included decision helpers ("decision rules") taken from the decision-record catalog. To draw a definitional line between the two, the decision-record description contains guidelines such as "[…] in contrast to O4.2, this option [O4.3] would define a new and domain-specific diagram type" (Hoisl et al., 2014a, p. 24).

Another set of decision rules was derived from the description of decision associations (see Section 2.1.2). Given a unit of coding, coders are so pointed to related categories in other main categories for further consideration. For instance, once having assigned a category PROFILE RE-/DEFINITION (O2.2) to a given unit of coding, the same unit of coding is likely to contain details on specific concrete-syntax choices. Therefore, a decision rule points the coder to potentially relevant concrete-syntax categories (e.g., DIAGRAM SYMBOL REUSE; O4.6). The decision association "native stereotype definition" exemplifies such a decision rule:

> "A UML profile definition (O2.2) for the language-model formalization was observed in combination with a concrete syntax specification via annotating model elements (O4.1) and reusing diagram symbols (O4.6; see, e.g., P22 or P54). […] A stereotype inherits all semantics (abstract syntax) and the notation (concrete syntax) from its extended UML base class." (Hoisl et al., 2014a, p. 38)

Segmentation and coding were all performed using PDF copies of the selected materials, PDF reader software, and portable PDF annotations.

Segmentation. In this step, we marked relevant parts (themes) and units of coding. From preparatory work, we were aware that statements and content fragments documenting design decision do not follow the internal structure of the scientific publication (e.g., dedicated sections per design aspect). Rather, we had found them spread across the document, including appendix material and externally referenced companion material. In addition, we had decided to consider all auxiliary design-documentation artifacts contained (e.g. diagrams) or referenced by the publication at hand, if fully accessible. Important artifacts included package diagrams as well as implementation artifacts such as metamodel, profile, and concrete-syntax specifications. Therefore, we first marked statements and content items (e.g. tables, figures, listings) having a common point of reference (themes).

We used color highlighting to identify six themes in a document ("domain analysis", "UML2", etc.; see Fig. 15). The available themes are defined by the decision-context descriptions of the decision records. For example, the theme "domain analysis" results from the

[16] The indicative threshold of three is borrowed from the software-pattern community, see Section 2.1.1 for a justification.

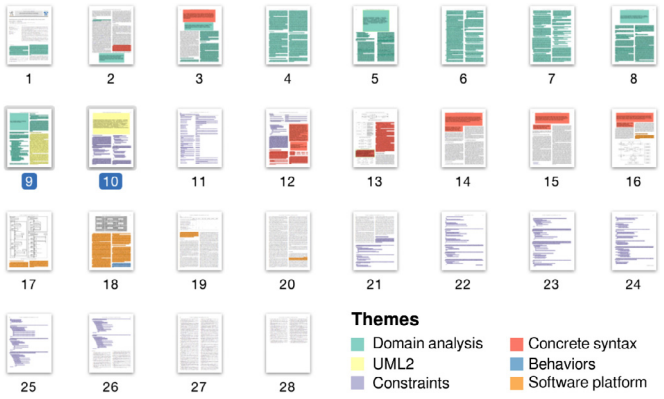


Fig. 15. Thumbnail overview of the corpus paper on the DSML BusinessActivities after segmentation. Five (out of six possible) themes are identified as colored segments. The ninth and tenth pages (465, 466) are highlighted for the coding details in Fig. 16.

decision record on "language-model definition" (D1) which describes the broader decision context as part of a domain-analysis step:

> "**Decision context**. A prerequisite for DSML design is a systematic analysis and the structuring of the language domain. By applying a domain analysis method, such as domain-driven design [...], information about the selected domain is collected and evaluated (e.g. based on literature reviews, scenario analyzes, and collected expert knowledge). [...]" (Hoisl et al., 2014a, p. 12)

Those parts of a document containing details on domain-analysis procedures, techniques, and analysis findings were marked using the theme's color code (see the color legend in Fig. 15). In Fig. 15, five theme segments are shown color-highlighted (out of six possible ones) for the paper on the DSML BusinessActivities. As an example, the "domain analysis" segment in the themed document groups those document items which deal with analyzing the application domain (RBAC for business processes) and which define key domain abstractions and their relationships (e.g. separation and binding of duty). This way, all corpus material was split up into up to six different thematic segments (per DSML).

Within each thematic block, we then identified smaller parts (units of coding) to be assigned codes in a separate step (see paragraph on "Main coding"). Units of coding were complete phrases, phrase blocks, and content items such as tables, figures, listings, and formula blocks. Relevant phrases were identified using underline marks, other content items were marked using margin bars. See Fig. 16 for various examples of marks and units of coding. For instance, on p. 465 of the excerpt, three listings depicting constraint expressions (OCL) were marked using vertical margin bars. Each of these listings formed a separate unit to be later assigned a code. Note that the identified segments were not broken up into units of coding exhaustively. Phrases and content items deemed irrelevant for the three above tests were not considered as units of coding. The two paragraphs discussing rationale for making or limitations of given formal propositions on p. 464 in Fig. 16 are examples of such excluded content.

Segmentation as a separate working step reduced the risk of accidentally skipping relevant content during actual coding, yet it helped break down the large paper corpus into a manageable material collection for main coding. Note that segmentation step was carried as an integrated part of SLR data extraction (see Section 4.1). During data extraction, the corpus papers were divided up between the authors for extraction and segmentation. This helped avoid excessive and time-consuming iterations over the entire corpus. To establish that all authors perform segmentation in a consistent manner, segmentation was performed collaboratively in a joint coding session for papers collected during preparatory studies (Hoisl et al., 2012b; Filtz, 2013).

Table 8
Coding consistency per main coding category (D1–D6); Kupper–Hafner (KH) Index; $\hat{\pi}^*$: Percent agreement (incl. missing ratings); $\hat{C}^*$: Chance-corrected KH indices (incl. missing ratings); $Var(\ldots)$: leave-one-out Jackknife variance estimates.

	D1	D2	D3	D4	D5	D6
$\hat{\pi}^*$	0.85	0.96	0.84	0.94	0.98	0.87
$Var(\hat{\pi}^*)$	*0.0004*	*0.0002*	*0.0005*	*0.0002*	*0.0001*	*0.0004*
$\hat{C}^*$	0.77	0.95	0.80	0.92	0.98	0.84
$Var(\hat{C}^*)$	*0.0012*	*0.0003*	*0.0008*	*0.0003*	*0.0002*	*0.0007*

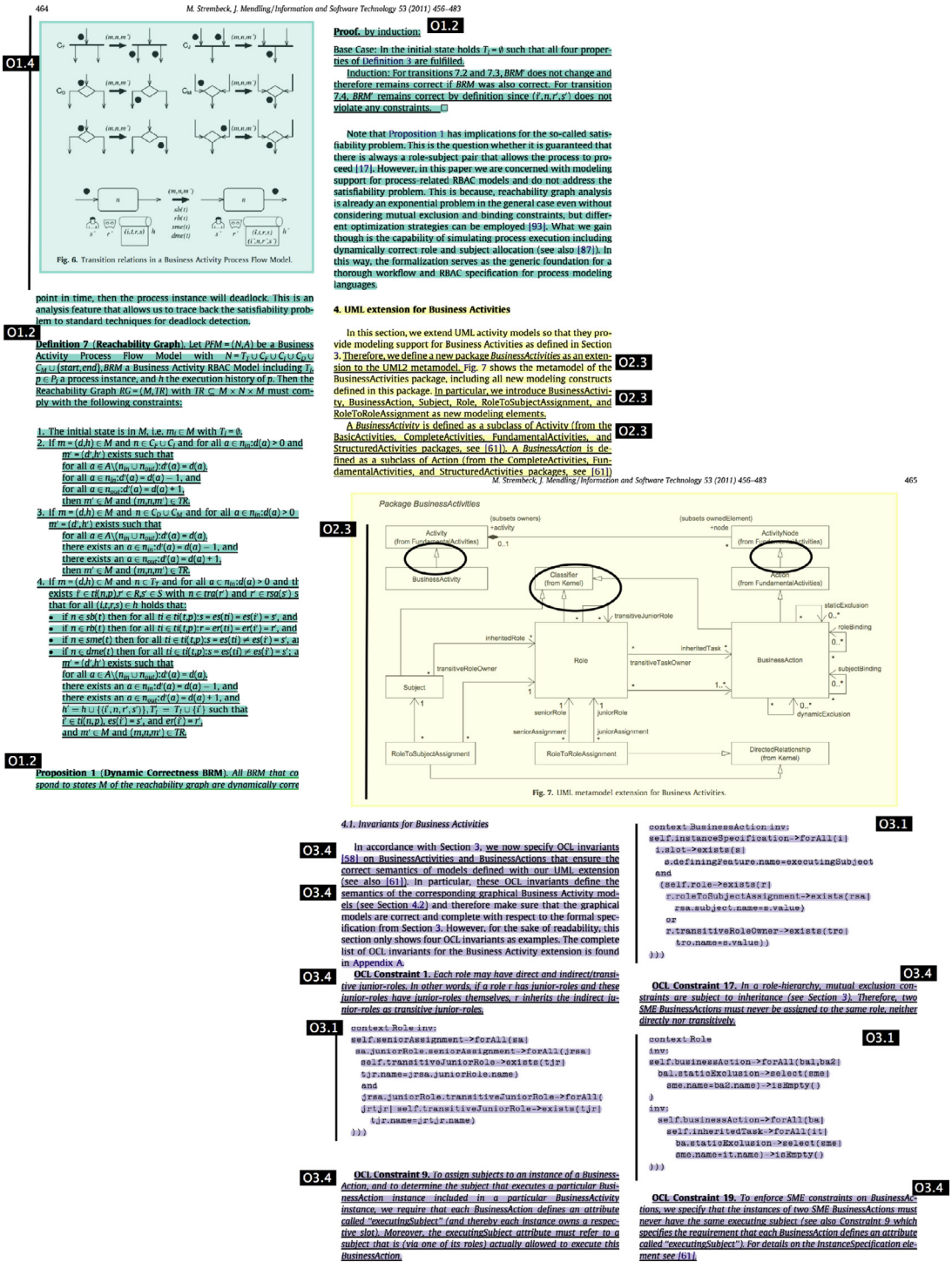
Main coding. In the main coding phase, each unit of coding became assigned to one of the categories (decision options) of the coding schema. Category assignment was performed in line with the overall SLR extraction procedure (see Section 5). For the majority of the corpus papers, coding was conducted independently by two authors, each author being "blinded" for the assignments of his alter. A smaller share was coded/recoded by a single author. This was due to one author having initially excluded a DSML or the underlying publication(s), for example. As a result, the overall paper corpus was effectively split into two parts: For 62 of the 80 DSMLs, coding was performed twice and independently by two authors. This part was also checked for coding consistency (see below). For the remaining 18 papers, there were assignments by a single author.

The choice of a particular category assignment was first recorded by each author for each unit of coding as a text mark. The mark's text identified the category assigned to a given unit of coding. For example, "Fig. 7" on page 465 in Fig. 16 was handled as a single unit of coding and text-marked by "O2.3". This text mark indicates that the package diagram documents a UML metamodel extension in a formal and diagrammatic way (METAMODEL EXTENSION). The UML metamodel is extended by adding six sub-metaclasses (`BusinessAction`, `Subject` etc.). Note that this coding step resulted in repeatedly assigning categories to several different coding units per unit of analysis (DSML): Consider the three OCL listings in Fig. 16, each of them being marked "O3.1" (CONSTRAINT-LANGUAGE EXPRESSION). Each unit of coding could also be assigned to two or more categories. Each text mark could be accompanied with text comments, e.g., for providing some rationale for the assignment.

In a second step, the category assignments were prepared for further analysis using a summarizing coding form. The coding form is documented in the SLR protocol (Sobernig et al., 2014). For every DSML, each coding author recorded whether a given category has been assigned at least once, i.e. there is at least one unit of coding having assigned a certain category (decision option). Recording and further analyzing the state of presence (absence) of one category per DSML was sufficient to answer the three tests and the overall research questions. Each row in these coding forms translates directly into the decision-option sets used for further analysis (see Section 2.1.2).

Coding consistency. To compare the coding performed by the pairs of authors, we used summary measures of inter-rater agreement. Inter-rater agreement signals how consistent the different coders were in assigning categories (codes) of the coding schema to units of analysis (DSMLs) in the material. That is, to which extent they are in agreement on codes to assign and/or on codes to discard independently from each other, based on the coding schema, and without negotiating a joint decision. Any conflicting coding was first revisited in a joint session between the two coding authors. Then, if necessary, the third, non-involved author adjudicated on a DSML at issue.

Table 8 summarizes the percent agreement (upper bound) and the extended chance-corrected Kupper–Hafner indices (lower bound, worst case) for the six main coding categories. The two independently coding authors arrived at the same assignment of categories to units of analysis (DSMLs) for 84% (D3) through 98% (D5) of the 62 co-rated DSML designs. Applying chance correction (extended KH index $\hat{C}^*$) under worst-case assumptions (random assignment), we established



464 M. Strembeck, J. Mendling / Information and Software Technology 53 (2011) 456–483

Fig. 6. Transition relations in a Business Activity Process Flow Model.

point in time, then the process instance will deadlock. This is an analysis feature that allows us to trace back the satisfiability problem to standard techniques for deadlock detection.

Definition 7 (Reachability Graph). Let $PFM = (N, A)$ be a Business Activity Process Flow Model with $N = T_I \cup C_I \cup C_i \cup C_O \cup C_M \cup \{start, end\}$, BRM a Business Activity RBAC Model including T_O, $p \in P_I$ a process instance, and h the execution history of p. Then the Reachability Graph $RG = (M, TR)$ with $TR \subseteq M \times N \times M$ must comply with the following constraints:

1. The initial state is in M, i.e. $m_I \in M$ with $T_I = \emptyset$.
2. If $m = (d,h) \in M$ and $n \in C_O \cup C_i$ and for all $a \in n_{in}$: $d(a) > 0$ and $m' = (d',h')$ exists such that
   for all $a \in A \setminus (n_{in} \cup n_{out})$: $d'(a) = d(a)$,
   for all $a \in n_{in}$: $d'(a) = d(a) - 1$, and
   for all $a \in n_{out}$: $d'(a) = d(a) + 1$,
   then $m' \in M$ and $(m,n,m') \in TR$.
3. If $m = (d,h) \in M$ and $n \in C_O \cup C_M$ and for all $a \in n_{in}$: $d(a) > 0$ $m' = (d',h')$ exists such that
   for all $a \in A \setminus (n_{in} \cup n_{out})$: $d'(a) = d(a)$,
   there exists an $a \in n_{in}$: $d'(a) = d(a) - 1$, and
   there exists an $a \in n_{out}$: $d'(a) = d(a) + 1$,
   then $m' \in M$ and $(m,n,m') \in TR$.
4. If $m = (d,h) \in M$ and $n \in T_F$ and for all $a \in n_{in}$: $d(a) > 0$ and th exists $t' \in ti(n,p)$, $r \in Rs$, $s \in S$ with $n \in trg(r')$ and $r' \in rsa(s')$ s that for all $(i,t,r,s) \in h$ holds that:
   - if $n \in sb(t)$ then for all $ti \in ti(t,p)$: $s = es(ti) = es(t') = s'$, and
   - if $n \in rb(t)$ then for all $ti \in ti(t,p)$: $r = er(ti) = er(t') = r'$, and
   - if $n \in sme(t)$ then for all $ti \in ti(t,p)$: $s = es(ti) \neq es(t') = s'$, a
   - if $n \in dme(t)$ then for all $ti \in ti(t,p)$: $s = es(ti) \neq es(t') = s'$; a

   $m' = (d',h')$ exists such that
   for all $a \in A \setminus (n_{in} \cup n_{out})$: $d'(a) = d(a)$,
   there exists an $a \in n_{in}$: $d'(a) = d(a) - 1$, and
   there exists an $a \in n_{out}$: $d'(a) = d(a) + 1$, and
   $h' = h \cup \{(i', n, r', s')\}$, $T_I' = T_I \cup \{i'\}$ such that $i' \in ti(n,p)$, $es(i') = s'$, and $er(i') = r'$, and $m' \in M$ and $(m,n,m') \in TR$.

Proposition 1 (Dynamic Correctness BRM). *All BRM that co spond to states M of the reachability graph are dynamically corre*

Proof. by induction:

Base Case: In the initial state holds $T_I = \emptyset$ such that all four properties of Definition 3 are fulfilled.

Induction: For transitions 7.2 and 7.3, BRM' does not change and therefore remains correct if BRM was also correct. For transition 7.4, BRM' remains correct by definition since (i', n, r', s') does not violate any constraints. □

Note that Proposition 1 has implications for the so-called satisfiability problem. This is the question whether it is guaranteed that there is always a role-subject pair that allows the process to proceed [17]. However, in this paper we are concerned with modeling support for process-related RBAC models and do not address the satisfiability problem. This is because, reachability graph analysis is already an exponential problem in the general case even without considering mutual exclusion and binding constraints, but different optimization strategies can be employed [93]. What we gain though is the capability of simulating process execution including dynamically correct role and subject allocation (see also [87]). In this way, the formalization serves as the generic foundation for a thorough workflow and RBAC specification for process modeling languages.

4. UML extension for Business Activities

In this section, we extend UML activity models so that they provide modeling support for Business Activities as defined in Section 3. Therefore, we define a new package *BusinessActivities* as an extension to the UML2 metamodel. Fig. 7 shows the metamodel of the BusinessActivities package, including all new modeling constructs defined in this package. In particular, we introduce BusinessActivity, BusinessAction, Subject, Role, RoleToSubjectAssignment, and RoleToRoleAssignment as new modeling elements.

A *BusinessActivity* is defined as a subclass of Activity (from the BasicActivities, CompleteActivities, FundamentalActivities, and StructuredActivities packages, see [61]). A *BusinessAction* is defined as a subclass of Action (from the CompleteActivities, FundamentalActivities, and StructuredActivities packages, see [61])

465 M. Strembeck, J. Mendling / Information and Software Technology 53 (2011) 456–483

Fig. 7. UML metamodel extension for Business Activities.

4.1. Invariants for Business Activities

In accordance with Section 3, we now specify OCL invariants [58] on BusinessActivities and BusinessActions that ensure the correct semantics of models defined with our UML extension (see also [61]). In particular, these OCL invariants define the semantics of the corresponding graphical Business Activity models (see Section 4.2) and therefore make sure that the graphical models are correct and complete with respect to the formal specification from Section 3. However, for the sake of readability, this section only shows four OCL invariants as examples. The complete list of OCL invariants for the Business Activity extension is found in Appendix A.

OCL Constraint 1. *Each role may have direct and indirect/transitive junior-roles. In other words, if a role r has junior-roles and these junior-roles have junior-roles themselves, r inherits the indirect junior-roles as transitive junior-roles.*

```
context Role inv:
self.seniorAssignment->forAll(sa|
 sa.juniorRole.seniorAssignment->forAll(jrsa|
  self.transitiveJuniorRole->exists(tjr|
  tjr.name=jrsa.juniorRole.name)
  and
  jrsa.juniorRole.transitiveJuniorRole->forAll(
  jrt|jr| self.transitiveJuniorRole->exists(tjr|
    tjr.name=jrt|jr.name)
))))
```

OCL Constraint 9. *To assign subjects to an instance of a BusinessAction, and to determine the subject that executes a particular BusinessAction instance included in a particular BusinessActivity instance, we require that each BusinessAction defines an attribute called "executingSubject" (and thereby each instance owns a respective slot). Moreover, the executingSubject attribute must refer to a subject that is (via one of its roles) actually allowed to execute this BusinessAction.*

```
context BusinessAction inv:
self.instanceSpecification->forAll(i|
 i.slot->exists(s|
  s.definingFeature.name=executingSubject
 and
  (self.role->exists(r|
   r.roleToSubjectAssignment->exists(rsa|
    rsa.subject.name=s.value)
   or
   r.transitiveRoleOwner->exists(tro|
    tro.name=s.value))
)))
```

OCL Constraint 17. *In a role-hierarchy, mutual exclusion constraints are subject to inheritance (see Section 3). Therefore, two SME BusinessActions must never be assigned to the same role, neither directly nor transitively.*

```
context Role
inv:
self.businessAction->forAll(ba1,ba2|
 ba1.staticExclusion->select(sme|
  sme.name=ba2.name)->isEmpty()
)
inv:
 self.businessAction->forAll(ba|
  self.inheritedTask->forAll(it|
   ba.staticExclusion->select(sme|
    sme.name=it.name)->isEmpty()
)))
```

OCL Constraint 19. *To enforce SME constraints on BusinessActions, we specify that the instances of two SME BusinessActions must never have the same executing subject (see also Constraint 9 which specifies the requirement that each BusinessAction defines an attribute called "executingSubject"). For details on the InstanceSpecification element see [61].*

Fig. 16. Excerpt (two pages) from the segmented and coded corpus article on DSML BusinessActivities.

that the two rating authors achieved actual coding agreement beyond chance in approx. 77% (D1) through 98% (D5) of the total expected cases beyond chance. In technical terms, coding consistency amounted to an extended KH index $\hat{C}^*$ equal to and greater than 0.77 $\pm$ 0.0012[17] for all six main coding categories (see Table 8). This includes agreement in terms of categories not assigned to DSMLs by both coders. Therefore, for the 62 co-rated DSML designs, the co-rating authors were in medium to good chance-corrected (worst-case) agreement according to standard benchmarks (Gwet, 2012; Fleiss, 1981). Hence, we considered the coding guided by the coding schema consistent. Any coding conflicts were resolved as outlined above.

Test results. Based on the coded decision data, the three tests guiding content analysis were answered as follows:

1. Saturation: Is each decision category reflected at least once in the corpus?

[17] $\pm$ signals the leave-one-out Jackknife variance estimate $Var(\hat{C}^*)$ of the *extended* chance-corrected Kupper–Hafner Index $\hat{C}^*$.

Table 9
Overview of the structural and behavioral UML diagram-types according to Appendix A in Object Management Group (2015b), as adopted and/or tailored by 77 out of 80 DSMLs identified using the SLR. Three DSMLs, which target all or are unspecific about the UML diagram types, are omitted.

DSMLs (cnt.)	Diagram type (ranked by decreasing cnt.)
59	Structure diagrams
50	Class
14	Component
14	Package
7	Composite structure
5	Object
3	Deployment
0	Profile
39	Behavior diagrams
20	Activity
11	State machine
8	Use case
4	Sequence
1	Interaction overview
0	Communication
0	Timing

No: Three categories out of the 31 available categories were not assigned to any unit of coding, and therefore, not reflected in any DSML.
Therefore: We annotated the corresponding decision options in the decision-record catalog to reflect this missing evidence from our content analysis. See Sections 5.5 and 6 for a discussion.

2. Exhaustiveness: Can each unit of coding (text fragment, specification artifact) be assigned to an already defined category?
No: The paper corpus revealed units of coding which could not be assigned to any of 26 categories of the initial coding schema.
Therefore: The coding schema was extended accordingly, by introducing five new categories on options for behavior specification and platform integration. See Sections 5.5 and 6 for a discussion.
3. Generalizability: Are the decision categories general in terms of being found for at least three units of analysis (DSMLs)?
No: Seven of the finally available 31 available categories were not assigned to at least three DSMLs.
Therefore: We proceeded with a frequency-pattern analysis for the 24 generalizable categories (see Section 5). The presentation of the decision options corresponding to the seven non-generalizable categories was revised, for example, by marking them as *candidate* options. See Sections 5.5 and 6 for a discussion.

The detailed numbers (frequencies) underlying these tests are reported in Section 5.3.

5.2. Descriptive data

Before reviewing the design-decision data, we characterize the 80 DSMLs regarding the adopted UML diagram types and their specification size.

Diagram types. The majority of the 59 DSMLs tailor a structural UML diagram type; 38 DSMLs adopted and tailored one or more structural and no behavioral diagram types (see Table 9; the diagram types tailored by each DSML are listed in the Appendix). Class diagrams are adopted by 50 DSMLs, followed by component and package diagrams (14 DSMLs each). This observation completes the empirical picture of a preponderance of class diagrams in broader UML usage (see, e.g., Hutchinson et al., 2014; Budgen et al., 2011). Profile diagrams have not been found adopted by any of the DSMLs.

With respect to behavioral diagrams, 39 DSMLs tailor at least one behavioral diagram type; 18 build on behavioral diagrams only, without building on any UML structural diagram type. Activity (20), state machine (11), and use case diagrams (8) are most frequently used, while communication and timing diagrams are not used by any DSML.

21 DSMLs provide both tailored structural and behavioral diagrams. Similar distributions between structural and behavioral diagram types were found in an earlier secondary study on UML profile usage (Pardillo, 2010).

Specification size. We quantified the core language-model sizes of the 80 DSMLs. Our findings below indicate that, on the one hand, our pool of DSMLs compares with prior reports on UML extension sizes. On the other hand, our DSML pool covers a greater size variety than prior work (Staron and Wohlin, 2006; Pardillo, 2010). Depending on the different, underlying UML implementation techniques (O2.1–O2.4), the specification size was established differently.

For 61 DSMLs defining their language models using UML profiles (O2.2), we counted the stereotype definitions and the corresponding, distinct base UML metaclasses. In this group, we find a median of 13 $\pm$ 8.9[18] stereotype definitions per DSML. A typical profile extends a median of 5 $\pm$ 3 distinct base metaclasses per DSML. As outliers, three DSMLs defined 40 (AspectSM), 48 (WebML), and 116 stereotypes (IEC61508), respectively. Two DSMLs extend 14 base metaclasses (UML4SOA, SafeUML). The specification sizes of these 61 DSMLs slightly differ from those reported by related work on UML profiles, but fall into a closely comparable size range. Pardillo (2010) studied 39 UML profiles, with a median of 9 $\pm$ 5.9 stereotypes per profile and a median of 4 $\pm$ 1.5 extended base metaclasses per profile. Staron and Wohlin (2006) cover three UML profiles containing between seven and 13 stereotype definitions.

17 DSMLs used a UML metamodel extension and/or modification (O2.3, O2.4). For these, we collected the number of newly introduced and redefined UML metaclasses. A typical DSML adds and redefines a median of 12 $\pm$ 11.9 UML metaclasses (O2.3, O2.4). One outlier includes 51 UML metaclasses (DMM/UCMM). Existing empirical work reports on UML metamodels of between 20 and 30 metaclasses (Staron and Wohlin, 2006).

For three DSMLs defining their language model using a UML class model at level M1 (O2.1), we attempted to count the number of UML classes. Given their incomplete design documentation, we failed in this attempt for two DSMLs (EM, UML-GUI). For the one remaining DSML (SECTET), we counted 20 UML classes.

5.3. Extracted decision data

Content analysis yielded one decision-option set per DSML (see Section 2.1.2). For brevity, we refer to decisions option by their option code (e.g. O1.4, O3.1) in the following, rather than by name (e.g. FORMAL DIAGRAMMATIC MODEL, CONSTRAINT-LANGUAGE EXPRESSION). For a complete reference on the relevant decision options, see Tables 1 and 2 in Section 2.1.

Decision points. Each of the 80 DSMLs covered the points of language-model definition (D1) and language-model formalization (D2), respectively. This was also a minimum requirement for a DSML design to become included in our study (see Section 4.1.2). For the remaining four decision points (D3–D6), we also recorded whether or not any decision could be recovered.

Table 10 shows the frequencies per decision-option code observed for the 80 coded DSMLs. The majority of DSMLs adopted one or several decision options for D3 (language-model constraints) and D4 (concrete syntax): 48 DSMLs explicitly documented language-model

[18] We report the variance in terms of the *median absolute deviation from the median* (MADM) using the $\pm$ notation along with the median value.

Table 10
The number of occurrences (abs. frequency n_{opt}, rel. frequency f_{opt}) of each of the 27 (31) decision-option codes available from the decision-record catalog (Hoisl et al., 2014a) in the 80 reviewed DSML designs. The 27 actual decision options are described in Tables 1 and 2. Four decision-option codes represent the absence of a decision (i.e. no-option codes: O3.5, O4.7, O5.5, and O6.6).

Decision option (code)	n_{opt} (f_{opt}, %)	Decision option (code)	n_{opt} (f_{opt}, %)
Language-model definition (D1)		Concrete-syntax definition (D4)	
INFORMAL TEXTUAL DESCRIPTION (O1.1)	80 (100)	DIAGRAM SYMBOL REUSE (O4.6)	69 (86.3)
FORMAL DIAGRAMMATIC MODEL (O1.4)	23 (29.8)	MODEL ANNOTATION (O4.1)	62 (77.5)
FORMAL TEXTUAL DESCRIPTION (O1.2)	5 (7.3)	DIAGRAMMATIC SYNTAX EXTENSION (O4.2)	14 (17.5)
INFORMAL DIAGRAMMATIC MODEL (O1.3)	3 (3.8)	*No decision* (O4.7)	7 (8.8)
Language-model formalization (D2)		MIXED SYNTAX (O4.3)	3 (3.8)
PROFILE RE-/DEFINITION (O2.2)	62 (77.5)	FRONTEND-SYNTAX EXTENSION (O4.4)	1 (1.3)
METAMODEL EXTENSION (O2.3)	17 (21.3)	ALTERNATIVE SYNTAX (O4.5)	1 (1.3)
M1 STRUCTURAL MODEL (O2.1)	4 (5)	Behavior specification (D5)	
METAMODEL MODIFICATION (O2.4)	2 (2.5)	*No decision* (O5.5)	77 (96.3)
Language-model constraints (D3)		INFORMAL TEXTUAL SPECIFICATION (O5.3)	2 (2.5)
INFORMAL TEXTUAL ANNOTATION (O3.4)	35 (43.8)	M1 BEHAVIORAL MODEL (O5.1)	1 (1.3)
No decision (O3.5)	32 (40)	FORMAL TEXTUAL SPECIFICATION (O5.2)	1 (1.3)
CONSTRAINT-LANGUAGE EXPRESSION (O3.1)	31 (38.8)	CONSTRAINING MODEL EXECUTION (O5.4)	0 (0)
CODE ANNOTATION (O3.2)	0 (0)	Platform integration (D6)	
CONSTRAINING MODEL TRANSFORMATION (O3.3)	0 (0)	*No decision* (O6.6)	54 (67.5)
		GENERATOR TEMPLATE (O6.2)	16 (20)
		M2M TRANSFORMATION (O6.5)	9 (11.3)
		API-BASED GENERATOR (O6.3)	7 (8.8)
		INTERMEDIATE MODEL REPRESENTATION (O6.1)	4 (5)
		(DIRECT) MODEL EXECUTION (O6.4)	1 (1.3)

constraints (D3; e.g. OCL expressions) beyond the constraints expressed directly via their formalized language models. 73 DSMLs included at least one decision on the concrete syntax (D4) of the DSML. On the contrary, only three DSMLs reported on decisions related to additional behavioral specifications (D5), and only 26 DSML designs comprised decisions related to platform integration (D6). The latter is reflected by the comparatively large numbers of no-option codes displayed in Table 10 (O5.5 and O6.6, respectively).

Language-model definition (D1). For identifying and describing relevant domain abstractions and their relationships, the decision-record catalog documents four decision options (O1.1–1.4). Each of the four available option codes was identified at least once. 52 DSMLs applied a single option only (see Table 10). This decision was to use a textual, natural-language representation (O1.1). The 28 remaining DSMLs are characterized by a combination of two or three D1 options. Not a single design used all four options. 23 of 28 DSMLs provide a formal diagrammatic definition (O1.4) of their language model, prior to actually formalizing (implementing) the language model on top of UML. In all 23 cases, this decision option is accompanied by a textual, natural-language definition of the language model (O1.1).

Language-model formalization (D2). To realize the DSML by reusing the UML, concrete UML techniques for implementing the language DSML as a UML extension, as a UML specialization, and/or by piggybacking on UML must be considered. The decision-record catalog describes four combinable options (O2.1–O2.4). Each available option code was marked at least for one DSML design (see Table 10). 75 of the 80 DSMLs applied exactly one option, the remaining five projects adopted two options. 62 of the DSML projects (≈78%) involved UML profiles (O2.2) to realize the language model, and ≈21% of the DSMLs (17/80) extend the UML metamodel without modifying it (O2.3). Only three DSMLs applied both, UML metamodel extensions (O2.3) and UML profiles (O2.2), 14 DSMLs used UML metamodel extensions only. Just two DSMLs explicitly modify the UML metamodel (O2.4) when extending it (O2.3).

Language-model constraints (D3). Once a language model has been formalized (D2), additional structural conditions can be placed on DSML models at the level of the language model (e.g. consistency conditions on model elements). The decision-record catalog lists four options of defining language-model constraints (O3.1–O3.4). Two out of the four available options (see also Table 10) were applied in the 80 DSMLs: constraint-language expressions (O3.1 in 31 DSMLs) and textual annotations (O3.4 in 35 DSMLs). We did not find evidence for code annotations (O3.2) and translational constraining (O3.3). In 30 DSMLs, only one of these options was adopted: O3.1 13 times and O3.4 17 times. In 18 DSMLs, O3.1 and O3.4 are adopted both.

Concrete-syntax definition (D4). Defining the DSML's concrete syntax requires deciding on the style of the primary modeling interface presented to the domain modeler. In addition, it must be decided whether or not, and, if yes, how to integrate the DSML concrete syntax with the diagrammatic one of UML. Our decision-record catalog documents six options (O4.1–O4.6). All six options have been found applied at least once (see also Table 10). 73 of the 80 DSMLs contained at least one decision on their concrete syntax. Many DSML projects (49) took a combination of two decision options on their concrete-syntax style; 17 DSMLs applied one and 14 DSMLs three options. The two most frequently found decision options are model annotations (O4.1: 62 DSMLs) and the unmodified reuse of existing UML diagram symbols (O4.6: 69). 48 of the 49 DSML designs, which adopt exactly two options, reflect these two options (O4.1 and O4.6). A UML diagram-syntax extension is adopted in 14 DSMLs (O4.2; e.g. by introducing new symbols or modifying existing ones).

Behavior specification (D5). Behavior specifications stipulate how language elements of the DSML interact to produce the intended system behavior in a platform-independent manner. Only three of the 80 DSML design involve dedicated behavioral specification artifacts. These three DSMLs document these refinements of behavioral-semantics by adopting a UML M1 model representation (O5.1), a formal textual specification (O5.2), or in an informal textual way (O5.3). Only one of these three DSMLs applies two options in a complimentary manner (O5.1 and O5.3). The remaining two picked a single option only (O5.2, O5.3). Constraining model execution (O5.4) was not found applied in any DSML.

Platform integration (D6). Platform integration comprises support for mapping DSML models into specification artifacts, which are processable and possibly executable by a targeted software platform, in

Table 11
Overview of the seven prototype option-sets (ordered by decreasing absolute support). See Tables 1 and 2 in Section 2.1 for descriptions of the option codes. Details on the exemplary DSMLs including citations are documented in the Appendix and in Hoisl et al. (2014a).

Prototype	Option set	Support (abs.)	Frequency (abs.)	DSMLs (ex.)
UML piggybacking plus informal constraints	{1.1, 2.2, 3.4, 4.1, 4.6}	30	5	UML-AOF, PredefinedConstraints, UML-PMS
UML piggybacking plus formal constraints	{1.1, 2.2, 3.1, 4.1, 4.6}	26	4	REMP, CUP, UML4PF
Two-level UML piggybacking	{1.1, 1.4, 2.2, 4.1, 4.6}	22	5	SPArch, MoDePeMART, RichService
UML piggybacking for domain-specific M2T system	{1.1, 2.2, 4.1, 4.6, 6.2}	15	3	DPL, WCAAUML, WS-CM
UML piggybacking plus mixed constraints	{1.1, 2.2, 3.1, 3.4, 4.1, 4.6}	13	3	ArchitecturalPrimitives, SHP, C2style
UML metamodel ("middleweight") extension	{1.1, 2.3, 4.6}	10	4	UML2Ext, UML4SPM, MDATC
Two-level UML piggybacking plus mixed constraints	{1.1, 1.4, 2.2, 3.1, 3.4, 4.1, 4.6}	5	3	UACL, SafeUML, and IEC61508

a fully or semi-automated manner. Our decision-record catalog describes five support techniques for platform integration (O6.1–O6.5). 26 DSMLs applied at least one of these five techniques and each platform-integration technique was applied at least once (see also Table 10). From the 26 DSMLs, 19 DSMLs applied a single platform-integration option only rather than a combination of two or more options. The most frequently adopted option is O6.2 (generator templates) in 20% of the DSMLs (16/80). Generator templates are also the option, which is most frequently used in isolation (with 12 of the 19 single-option DSMLs). Generation templates are followed by model-to-model (M2M) transformations (O6.5: 9 DSMLs) and API-based generators for platform-specific models (O6.3: 7 DSMLs). M2M transformations (O6.5) are found mostly in combination with at least one other D6 option (in six out of nine DSMLs). The remaining 68% of the DSMLs do not consider or at least did not document any platform-integration techniques (O6.6: 54 DSMLs).

Decision-option sets. A decision-option set adds up all decisions recorded for a given DSML as a set of option codes corresponding to the decision options compiled by the catalog (see Section 2.1.2). Therefore, we obtained a base of 80 decision-option sets from the 80 DSMLs (see also the Appendix). By running a frequent-item-set analysis on these 80 observed decision-option sets, one for each DSML, 53 of the 80 observed option sets were unique (non-duplicated). From these 53, 14 unique option sets represent two or more DSMLs. In our study, none of the option sets were shared by more than five DSMLs. The maximum number of decision options included in an option set is ten for the unique option sets (i.e. for option sets that were applied for exactly one DSML) and seven options for the 14 shared ones (i.e. for option sets describing two or more DSMLs).

Remember that we consider an option subset to be frequent when it is used by three or more DSMLs. This is also referred to as a *minimum support* of three for an option subset. This follows from a threshold on backing up a software-pattern description with at least three known uses of a given pattern in existing software systems; a threshold which is commonly applied in the software-pattern community (see, e.g., Coplien, 1996; Buschmann et al., 2007). We found 188 of such frequent options sets; that is, option sets which are contained partly or fully in more than three observed option sets.

Smallest common option subsets. In our pool of 80 DSMLs, we found two smallest common option subsets *specific to one decision point*. For language-model constraints (D3), the proper option subset {3.1, 3.4} reflects that 18 DSMLs (i.e. the option subset is said of having a support of 18) define language-model constraints using both, a constraint-expression language as well as auxiliary or complimentary textual constraint definitions in natural language. As for platform integration (D6), a second proper option subset {6.2, 6.5} (support: 3) indicates that the respective 3 DSMLs use a two-level model transformation chain (PIM-PIM-PSM): First, platform-independent models (PIM) are transformed into another PIM representation which is then transformed into a structured textual, platform-specific (PSM) representation. For example, in UML2Alloy extended UML class models (PIM) are transformed into models of an Alloy metamodel (PIM) which are finally transformed into textual Alloy definitions accepted by an Alloy model checker (PSM).

Seven smallest common option subsets contain options through two and more decision points (D1–D6). Four out of seven option subsets *specific to two or more decision points* relate language-model formalization (D2) and language-model constraining (D3). These three subsets {2.2, 3.4}, {2.2, 3.1}, and {2.2, 3.1, 3.4} show that applying one or several UML profiles is often associated with defining language-model constraints either only textually (30 DSMLs), or by using a special purpose constraint-expression language (26 DSMLs), or both (13 DSMLs). In contrast, metamodel extensions (O2.3) are found frequently combined with both constraint-definition strategies, rather than either of the two exclusively: {2.3, 3.1, 3.4}. Metamodel extensions (O2.3) are also commonly applied together with diagrammatic syntax extensions (O4.2) and M2M transformation (O6.5). Finally, 22 of the 80 DSMLs adopt UML profiles (O2.2) for realizing a language model (O1.4).

Prototype option-sets. For the 80 DSMLs, we extracted seven distinct prototype option-sets (see Table 11). A prototype option-set describes complete DSML designs, on the one hand. On the other hand, it is also (frequently) extended to DSML designs to include additional decision options (see Table 4). These option sets are characterized based on the underlying UML implementation techniques: UML extension, UML piggybacking, and UML specialization (see Section 2 for definitions based on Spinellis, 2001).

Six prototype option-sets come *with frequent extensions*. For example, the option set describing UML-PMS, a DSML for performance modeling of mobile systems building on UML activities, denotes four additional DSML designs (total frequency: 5). Besides, this option set is found as a large subset in the option sets of 25 additional DSMLs (support: 30). The majority of prototype option-sets (5) involve UML profiles only (O2.2) and therefore realize UML piggybacking. Just one prototype option-set—UML4SPM, an extension of UML activities and classes to model software-development processes—builds on metamodel extensions (O2.3) only and, therefore, represents a UML extension (a.k.a. middleweight extension; Bruck and Hussey, 2008). All six prototype option-sets, that come with frequent extensions, involve at least one concrete-syntax decision option. The only platform-integration option adopted in three prototype option-sets are M2T generator templates (O6.2). These prototypes, therefore, represent the important group of domain-specific model-to-text transformation systems using UML piggybacking.

In addition, we found a seventh prototype option-set labeled "two-level UML piggybacking plus mixed constraints" which comes *with infrequent extensions* (cf. Table 4). These DSMLs are realized at two model levels: a diagrammatic language model independent from the UML and a UML profile implementing the language in the UML. The three DSMLs for this prototype are UACL, SafeUML, and IEC61508 (see Table 11). SafeUML is a domain-specific adaptation of component and class diagrams for modeling in the avionics safety domain, to give an example. This prototype option-set is, however, only extended by less than three additional DSMLs (2). Therefore, it is considered

extended infrequently. These DSMLs provide for constraints expressed both via CONSTRAINT-LANGUAGE EXPRESSION (O3.1) and INFORMAL TEXTUAL ANNOTATION (O3.4) (hence: "plus mixed constraints").

5.4. Study limitations

We deliberately narrowed the applicability of our findings to DSMLs embedded into UML 2.x (see Section 2). Therefore, we excluded DSMLs based on UML 1.x and other, non-UML metamodeling infrastructures (e.g. Kermeta, Ecore, XMF).

While this appears, at first glance, as a barrier to generalizing the recorded design decisions, the choice of UML 2.x was necessary because important design decisions taken for the UML 2.x are substantially different from those for UML 1.x; not to mention from other metamodeling infrastructures. Regarding UML 1.x, there are important lines separating the UML 2.x and UML 1.x regarding their language architectures and the foundational semantics of the available extension techniques (e.g. profiles, package merge; see Cook, 2012; Dingel et al., 2008; Henderson-Sellers and Gonzalez-Perez, 2006). By focussing on UML 2.x, we aimed at increasing the internal validity of our SLR-based study design at the expense of its external validity. Note, however, that many recorded decisions can be adopted in a broader sense to be compatible with DSMLs based on other metamodeling infrastructures and DSLs generally (e.g. concrete-syntax decisions).

Based on our SLR, we applied a documentation analysis to extract design decisions from scientific publications and their companion material. We considered supporting material if reported by and available from the publication authors. A documentation analysis represents an indirect data-collection technique (Singer et al., 2008). Therefore, information on ordering of design decisions over time (decision sequences) often remained implicit and, therefore, unrecoverable for us. Even if documented, any indirectly observed order of decision options adopted by DSML engineers might have also followed from the presentation requirements of a scientific publication (i.e. the one reporting on a DSML); an order which does not necessarily correspond to the original one during decision making. Therefore, in our research setting, we can only study option sets in terms of decision associations. For the same reason, we focused on one process style of DSML development only (i.e. language-model-driven development; see Section 3). We might have neglected design decisions characteristic for other development styles (e.g. mockup-driven DSML development; Strembeck and Zdun, 2009).

SLR studies such as this one have the major problem of finding a representative set of relevant primary studies. In general, we closely followed established guidelines on designing and conducting SLRs available from research on evidence-based software engineering to avoid any pitfalls (Jalali and Wohlin, 2012; Kitchenham, 2004; Zhang et al., 2011). As for the search strategy, we used an extensive automated search based on four search engines by primary publishers in the software-engineering field (ACM Digital Library, IEEE Xplore, Scopus, and SpringerLink). This search yielded 5,778 hits which included the majority of reference publications (28/37) previously collected in a manual search. Backward snowballing added another reference publication. Therefore, our automated search missed eight publications that should have been found. While the missing ones were considered for inclusion in a separate step, this still indicates that we risk having missed other relevant primary studies on UML-based DSMLs, in more general. Note that we addressed this threat right from the beginning, by building our review procedure around the principle of continuous search validation and search refinement driven by a QGS as a recommended practice (Zhang et al., 2011; Kitchenham and Brereton, 2013).

We applied a review and extraction process involving two independently working data extractors on each item (search hit, publication, DSML). This was to avoid limitations of an extractor-checker procedure (Turner et al., 2008). On the flip side, this required us to control the workload per data extractor in the light of the excessive result sets when checking for false negatives. Recall that we were required to process 5,778 hits from automated search and 2,337 from snowballing. A false negative is a hit which is deemed excluded by one data extractor, in the first iteration during selection, and which becomes included after a negotiated agreement between the two extractors. This negotiated agreement is triggered by an independent inclusion decision of the second extractor. In these cases, we drew subsets of hits for the second, independent assessment (i.e. 20% random samples). As elaborated in Sections 4.1.2 and 4.1.3, due to this sampling and in a probabilistic projection (99% confidence level), we risk having missed out between zero and 22 publications as false negatives during automated search (from 704 negatives) and between zero and seven publications during snowballing (from 170 negatives), respectively.[19] This threat cannot be neglected, however, we mitigated for this risk in two ways. First, we applied sampling for false-negative checking only for decisions which were based on structured and objectively evaluable decision criteria (e.g. publication year, venue names based on predefined venue lists). This way, we hope to have minimized the risk of undisclosed disagreement on negatives between the respective two data extractors. For the actual decision-data extraction, which involved more subjective assessments, no sampling was applied. For example, to establish whether a DSML's design documentation is of unacceptable quality, each DSML design was reviewed by two independent data extractors. Second, we consider the results reported (frequency counts, prototype designs) robust against the risk of false negatives. Consider, for example, the substantial support reported for the prototype designs. Each prototype design is observed for between ten and 30 DSMLs (see Table 11). It is unlikely that false negatives in the ranges above would reduce the observed support of just one or of a few prototypes, so that they would not be observable anymore.

As a result of our quality-assessment procedure, we excluded another 31 DSML candidates due to their incomplete and/or incorrect documentation of their designs. More precisely, we watched out for critical, UML-specific specification defects (Henderson-Sellers and Gonzalez-Perez, 2006; Pardillo, 2010; Atkinson et al., 2003) which would prevent us from extracting the decision options unambiguously, based on the design artifacts available to us via the corresponding scientific publications (e.g. diagrams, structured and unstructured text on design rationale).

At the same time, however, by excluding the DSMLs, we risk having biased our quantitative results (e.g. frequency counts and prototype designs). By making this exclusion decision, however, we strengthened the qualitative validity of our results by extracting decision options and option sets from DSMLs of acceptable documentation quality only. For the reasons put forth for potentially false negatives (see previous paragraph), we are also confident that our quantitative findings are robust against the exclusion of these DSML designs.

In our search design, we also omitted certain kinds of publications explicitly. On the one hand, we excluded publications appearing in venues not considered SE venues. This was realized by filtering venues according to publicly available third-party sources enumerating SE venues. The main objective was to put focus on DSML documentation artifacts which are likely to reflect on DSML and UML design rationale; and to avoid large numbers of publications from outside the SE domain. On the other hand, we excluded certain venue formats (e.g. workshops) and grey literature (i.e. technical reports, working papers) to increase the probability of building our analysis on documentation artifacts having a certain maturity. In addition, we targeted publications having been subjected to community-driven

[19] Note that the two intervals cannot be simply summed up because of the backward dependency between snowballing on automated search.

quality assessment (e.g. peer reviews). A similar limitation is that we restricted the search hits retrieved from the automated search based on the QGS corpus publications (see Section 4.1): from 5,778 total hits, 2,678 containing all QGS publications found during the automated search were considered for selection. This was necessary to reduce the number of publications to manageable levels for reading during selection by two independent raters. While settling us at a sufficient level of search validity (75.7%; Zhang et al., 2011), it means that we may have missed relevant publications. Note, however, that we undertook backward snowballing to make sure that we did not skip work considered relevant by the authors of the primary studies obtained as search hits.

There is a bias inherent to our SLR design in that by relying on scientific publications only, the recorded design decisions on DSMLs risk being specific to DSMLs as research-driven prototypes and proof-of-concept implementations. Design decisions during DSML development in industry might not necessarily be covered by our revised catalog and the findings in this study. The industrial case in Staron and Wohlin (2006) exemplifies this. This case is based on more balanced numbers of metamodel extensions (2) and profiles (3). At a coarser-grained level, our study reports a clear preponderance of UML profiles in 80% of the reviewed DSMLs. However, we find it difficult to assess the severity of this bias. To begin with, the primary studies reviewed in this SLR did not disclose their industrial background, if any at all. Similarly, while related empirical studies on UML usage certainly document the existence of UML extensions and UML-based DSML designs (see, e.g., Hutchinson et al., 2014; Nascimento et al., 2012; see also Section 7), they do not discriminate between industry-driven and research-driven projects.

For selection and data extraction, we used a process involving two independently working data extractors on each item (search hit, publication, DSML) to avoid pitfalls of an extractor-checker process in complex extraction settings and to follow closely corresponding recommendations (Turner et al., 2008). The observed disagreement about publications and DSMLs between the extractors was rigorously documented using established IRR statistics, adding to the detailed documentation of the final extraction agreed by both extractors. To reduce some of the overhead of a two-extractor process, we had one extractor perform selection and extraction decisions on reduced data sets. This involved a first extractor reviewing all included items, but only a sample of the excluded ones by the second extractor. A final issue with respect to extraction and selection was that due to a number of constraints arising during the process, two of the three authors performed the majority of extractions. Constraints included personalized access to publisher databases and individual time constraints. This might potentially have introduced bias.

An important motivation of this SLR study—besides validation—was to remove personal bias from the decisions recorded in the initial catalog (Hoisl et al., 2012a). This personal bias was most likely introduced by looking at our own DSML projects at that time. However, even for the SLR study, we could not fully avoid personal bias because we risked collecting data from our own publications, if retrieved at the various SLR stages (QGS corpus construction, automated search, and backward snowballing). Therefore, we excluded our own DSML projects from the result set so that from the 80 DSMLs obtained from this SLR *none* was developed by the authors themselves. Furthermore, there was only one third-party DSML (i.e. UML4SOA in Mayer et al., 2008) which was more familiar to us than the others, because one of our own DSMLs is built on top of it. Nevertheless, personal bias cannot be ruled out, simply because our understanding of DSML development has entered our judgments during data extraction.

5.5. Catalog revision

An important, secondary outcome of the SLR study is a revision of the initial decision-record catalog (Hoisl et al., 2012b; 2012a). The updated and improved version included changes to the catalog's content as well as to its presentation. Table 12 gives an overview of the changes resulting in the current decision-record catalog (Hoisl et al., 2014a). As a result of the SLR and our frequent-item-set analysis (see Section 5), the content items in the decision-record catalog were revised (decision points, options, and associations; see Section 6).

Table 12
Comparative overview of the numbers of content items (e.g., decision records, decision options) included in the decision-record catalog in its pre-study version (Hoisl et al., 2012a) and in the post-study revision (Hoisl et al., 2014a). *: The numbers in round brackets indicate that four decision options serve for coding pseudo-decisions only (e.g. not choosing any option); **: These entries show the numbers for third-party studies (DSMLs) compared to *all* studies (DSMLs), including our own.

	Pre-study revision	Post-study revision
Decision records	5	6
Decision options	22 (26)*	27 (31)*
Decision drivers	22	27
Decision consequences	9	13
Decision associations	11	21
	Underlying resources	
Primary studies	6/ 19**	84/ 97**
DSMLs	6/ 16**	80/ 90**
Secondary studies	15	25

Important but comparatively few additions and modifications to the catalog were necessary. As shown in Table 12, the main addition content-wise was the inclusion of decision record D5 on decisions relating to *behavioral specification*. While it was necessary to cover this sub-space of decision options to fully characterize the DSMLs in our study, we also found that, except for very few DSMLs, explicitly documented behavioral specifications are widely missing (see Section 5.3). However, within the limits of our study, this can be explained straightforwardly because the majority of the reviewed DSMLs focus on structural viewpoints (i.e. UML classes) and the corresponding application domains did not require an additional behavior specification. In addition, some of the DSMLs reviewed for our study directly reuse or slightly adopt UML built-in behavior (e.g. provided by UML activities). Thus, DSMLs providing a dedicated behavior specification were documented as known-usage examples in the newly added decision record D5.

Apart from D5, only one new decision option was added. O6.5 for D6 (*platform integration*) allows for characterizing DSMLs which build on endogenous M2M transformations for platform integration, i.e. transformations between models defined over the same metamodel—rather than using M2M transformations only for creating different intermediate model representations within an MDD tool chain (which is covered by O6.1). In addition, we refined the descriptions of three decision options (O2.1, O2.2, O4.3) to cover aspects revealed by the reviewed DSMLs and to better discriminate between decision options. For example, we revised decision option O2.2 (formalizing a language model via UML profiles) to include scenarios of extending and/or redefining existing UML profiles.

The resulting catalog lists 27 decision options and 21 associations between these options (e.g., dependencies; see also Table 12). In addition, the catalog provides seven prototypical solutions of existing DSMLs which represent commonly adopted combinations of decision options from 80 third-party DSMLs. Note that the catalog also records ten self-developed DSMLs summing up to a total of 90 DSMLs (see also the Appendix). 40 decision drivers and corresponding decision consequences (e.g., forward dependencies on follow-up decisions) are available to assess the decision options (e.g., in terms of rationale tables). In addition, the catalog offers application examples and implementation sketches taken from the 90 DSMLs that we examined in detail. Moreover, the decision catalog provides references to 25 secondary studies on DSML development (e.g., Jackson and Sztipanovits, 2009; Moody and van Hillegersberg, 2009).

6. Discussion

In this three-year research project, we started from our own experiences gained from building ten UML-based DSMLs to define a preliminary decision-record catalog for such DSMLs. To extend and to validate this decision catalog, we performed this systematic literature review (SLR). Our multi-stage SLR initially returned 8,152 publications: 37 articles originated from a quasi-gold standard corpus, 5,778 from the automated and 2,337 from the backward-snowballing searches. 7,069 papers were unique, 5,023 were considered for selection and quality assessment. Subsequently, we filtered these publications based on multiple quality criteria. As a result, we identified 84 publications on 80 UML-based DSML projects (see Section 4 and also the Appendix). These 80 DSMLs are "third-party" artifacts, that is, we did not participate in developing them.

Research question 1: *What are the design-decision options for UML-based DSML designs reported in scientific literature?*

Based on the literature review and content analysis, we tested scientific literature for the presence or absence of 27 decision options. From these, 24 were reportedly adopted by at least one DSML project. We found that three options are not contained by any of the 80 DSMLs (see Section 5.3): CODE ANNOTATION, CONSTRAINING MODEL TRANSFORMATION, and CONSTRAINING MODEL EXECUTION (i.e. O3.2, O3.3, and O5.4). This follows directly from the *saturation* test for the coding schema reported in Section 5.1.

CODE ANNOTATION (O3.2) and CONSTRAINING MODEL TRANSFORMATION (O.3.3) describe practices to realize and to enforce language-model constraints *indirectly*. On the one hand, this involves constraints defined as host-language expressions for a certain API (e.g. Java expressions binding the Ecore Java API; O3.2). On the other hand, such constraints could be realized in terms of assertions on a DSML model that are defined as a part of corresponding model transformations on the respective DSML models (O3.3).

Similarly, enforcing behavioral constraints (D5) via partial or prohibitive model-execution engines (CONSTRAINING MODEL EXECUTION; O5.4) was not documented for any DSML. We did, however, find these options documented in secondary studies (for O5.4 see, e.g., Mayerhofer et al., 2012) and applied them in our own DSML projects (for O3.3 see DSML SOFServices). As a result, our decision-record catalog lacks known-usage examples for the three decision options mentioned above. Hence, we highlighted these decision options as *candidate* options in the post-study revision of the catalog; that is, candidates requiring further evidence.

Recall that there are three basic UML implementation approaches for DSMLs: language-model extension, language-model piggybacking, and language-model specialization (see Section 2). The first two have been found employed frequently in the 80 DSMLs, most notably language-model piggybacking (see discussion of RQ2). Language-model specialization involves a metamodel derived from the vanilla UML2 metamodel systematically, e.g., through metamodel cloning and pruning (Sen et al., 2009; Blouin et al., 2015). This approach has also been described as a "heavyweight" metamodel extension (Bruck and Hussey, 2008). Language-model specialization is driven by the need for performing a METAMODEL MODIFICATION (O2.4), that is, the need for redefining existing UML metaclasses, their features, and their associations. The specialization approach comes at certain maintenance and deployment costs (e.g., tracking changes in the UML metamodel, low tool interoperability; Bruck and Hussey, 2008).

In total, we only found two DSMLs for which a METAMODEL MODIFICATION is reported: eSPEM and DMM/UCMM. The former further extends the UML2/SPEM metamodel (activities and state machines) to model software-development processes and artifact states. The latter provides a modeling frontend for automated GUI generation. DMM/UCMM adds new properties to existing UML metaclasses (e.g. `Class` and `Property`). Besides being comparatively infrequent, the respective design reports fall short in two ways:

First, they do not clarify how they realized the language-model specialization technically, i.e., managing metamodel derivation. Second, they also lack statements which document the intention and rationale for performing a METAMODEL MODIFICATION in the first place. Often, a METAMODEL MODIFICATION appears merely *accidental*. For instance, in a third DSML UCDM, existing UML metaclasses (e.g. `UseCase`) are associated with newly introduced metaclasses (e.g. `UseCaseDescription`). The metamodel definition for UCDM is underspecified regarding the ownership of association ends and, hence, regarding the choice of a METAMODEL EXTENSION (O2.3) or a METAMODEL MODIFICATION (O2.4):

(1) Both ends could be owned by the association, leaving the UML2 metamodel unchanged (O2.3); (2) one end could be owned by the association, the other one by a metaclass (O2.3 or O2.4, depending on whether the owning class is coming from the UML metamodel); or (3) both ends could be owned by their corresponding metaclasses (O2.4).[20]

For these three reasons (infrequency, specification ambiguity, unknown metamodel management), we cannot conclude from our study that language-model specialization has been adopted.

Even when omitting the decision options not observed in the 80 third-party DSMLs, the design space described by the 24 found options amounts to a vast number of possible option combinations.[21] Therefore, the frequency patterns identified for answering RQ2 turned out to be key.

Research question 2: *What are frequently observed decision options and frequently observed combinations of decision options (option sets) in and across existing UML-based DSML designs?*

From a total of 24 options (RQ1), 16 options are frequently found in three or more projects. Among the most frequently adopted options, the following are noteworthy (ordered by frequency): DIAGRAM SYMBOL REUSE (O4.6), PROFILE RE-/DEFINITION (O2.2), MODEL ANNOTATION (O4.1), INFORMAL TEXTUAL ANNOTATION (O3.4), CONTRAINT-LANGUAGE EXPRESSION (O3.1), FORMAL DIAGRAMMATIC MODEL (O1.4), METAMODEL EXTENSION (O2.3), and GENERATOR TEMPLATE (O6.2) (see also Table 10). We discuss their relevance below in their broader context, not necessarily in the above order.

A domain-analysis step in terms of domain engineering (Czarnecki and Eisenecker, 2000) and domain-driven design (Evans, 2004) is contained by most process guidelines available for DSL and DSML development (see, e.g., Zdun and Strembeck, 2009; Strembeck and Zdun, 2009). One challenge is to identify and to define the domain abstractions and their relationships in a manner which allows for (1) a detailed specification independent from a concrete modeling language and for (2) having the abstractions enter a UML-based language-model implementation in a seamless (e.g. instantiation-based model layers) and/or guided way (e.g., through transformations). Another challenge is making domain variability explicit, for example, in terms of feature models (Czarnecki and Eisenecker, 2000). Guidelines and the 84 primary studies reviewed put emphasis on (semi-)structured textual analysis techniques, such as domain-vision (scoping) statements, domain-distillation lists, and feature tables. This is indicated by *all* 80 DSMLs adopting an

[20] Note that the UML has these degrees of freedom, however, to avoid ambiguities, association end ownership can always be made explicit using the dot-notation (Object Management Group, 2015b).

[21] See Section 3 for the rationale behind these combinatorial computations; pre-study revision (all options; 3,686,400): $2^4 - 1$ (D1) times $2^4 - 1$ times (D2) times 2^{22-4-4} (D3-D5); post-study revision (all options; 117,964,800): $2^4 - 1$ (D1) times $2^4 - 1$ times (D2) times 2^{27-4-4} (D3-D6); post-study revision (excluding unobserved options; 29,491,200): $2^4 - 1$ (D1) times $2^4 - 1$ times (D2) times 2^{25-4-4}(D3-D6).

INFORMAL TEXTUAL DESCRIPTION (O1.1) to capture their application-domain definition.

However, prior to our study, very little was known about using a FORMAL DIAGRAMMATIC MODEL (O1.4) for a domain-analysis step. A FORMAL DIAGRAMMATIC MODEL leans itself towards a seamless and guided UML implementation of the resulting language model (e.g., in the sense of model chaining through transformations). It can also serve as the definitional basis for an FORMAL TEXTUAL DESCRIPTION (O1.2; e.g., as a type graph using set theory). We found that only 23 DSMLs adopt such a FORMAL DIAGRAMMATIC MODEL (O1.4). A closer look reveals that the dominant modeling formalism in this group are E/MOF and/or UML class diagrams. Alternatives such as Ecore or entity-relationship models (while inter-changeable to some extent) are not reported. Only a single DSML design (SOA) employed a type-graph notation in the sense of a graph-transformation system.

Two follow-up observations are noteworthy: First, the usage of E/MOF and/or UML class diagrams most often prepares the definition of UML profiles, rather than alternative UML language-model implementations. This is confirmed by the two "two-level UML piggybacking" prototypes (out of 7) identified in Section 5. This is typically justified by referring to profile-specific guidelines such as Paige et al., 2000; Grant et al., 2004; Robert et al., 2009; Selic, 2007; Atkinson and Kühne, 2002. Second, although E/MOF and/or UML class diagrams are available, they are *not* formally linked to the corresponding UML implementation model e.g. by using model transformations, inter-model consistency constraints, and/or instantiation-based model chaining. This is despite the availability of suitable approaches (see, e.g., Lagarde et al., 2007). Rather, convention-based approaches dominate. A popular example is the 1:1 name mapping of language-model elements into equally named stereotypes.

The importance as well as relative pros and cons of UML profiles as implementation vehicles have been covered to some extent (Paige et al., 2000; Grant et al., 2004; Robert et al., 2009; Selic, 2007; Atkinson and Kühne, 2002). Basic evidence on the actual adoption level, however, has been missing or has remained partial. 62 DSMLs implement their language model using newly defined and/ or redefined UML profiles (PROFILE RE-/DEFINITION, O2.2). As a consequence, "UML piggybacking" characterizes 5 of the 7 prototype designs in Section 5. This is partly a confirmatory finding for observations of prior, but inherently limited empirical studies. For example, both Pardillo (2010) and Nascimento et al. (2012) characterize publication-centric trends on UML profile usage. However, they discuss UML profiles in isolation. Our findings provide empirical insights on UML profile usage in the context of implementation alternatives, which is entirely missing from previous studies: As a general observation, UML profile usage in 62 DSMLs contrasts with 17 DSMLs using UML metamodel extensions.

Furthermore, we reveal coupled design decisions such as on concrete syntax. The popularity of MODEL ANNOTATION (e.g., UML comments carrying domain-specific keywords or slot specifications; O4.1) directly follows from the UML profile preponderance: The 62 profile-based DSMLs also adopted this option. The same coupling can be found between UML profiles and the concrete-syntax option DIAGRAM SYMBOL REUSE (O4.6) in 69 DSMLs. The latter coupling partly results from the default that stereotyped elements carry on the notation defined for the extended UML metaclass, if one is defined and not overruled explicitly.

Researchers have started systematizing the use and the role of model-level consistency constraints for the UML in general (Torre et al., 2014). We complement these findings for UML-based DSMLs. We established that 31 DSMLs adopted the CONSTRAINT-LANGUAGE EXPRESSION (O3.1) option. To be more precise, they all employ OCL expressions. This level of adoption puts new focus on the portability issues of such OCL consistency rules between different evaluation engines. Portability remains limited due to the OCL/UML language specifications leaving critical details to language and tool implementers (e.g., navigation semantics between extension and extended model elements; see Langer et al., 2012 for an overview).

In addition, we found that constraint expressions are predominantly defined for an intra-model scope (e.g. to resolve ambiguities in the language model). Inter-model constraint expressions, including vertical constraints between different model levels (e.g. platform-independent vs. platform-specific), are the minority. Consider as an example a DSML language model defined at level M2 which must enforce consistency rules at level M0, i.e. the occurrence (instance) level of DSML models. In fact, we found that for such scenarios, DSML developers resort to an INFORMAL TEXTUAL ANNOTATION (O3.4). The use of implementation idioms (e.g., prototypical concept pattern; Atkinson and Kühne, 2001) and alternatives to metamodeling based on shallow instantiation (e.g., potency and deep instantiation; Atkinson and Kühne, 2007) to work around or to overcome this limitation was not observed.

Generating textual artifacts (e.g. source code, documentation, deployment descriptors) from models in software development via model-to-text (M2T) transformation is a common MDD activity (Ogunyomi et al., 2015). The importance for model-driven development of M2T transformations, in general, and M2T transformation systems based on generator templates, in particular, has been reported earlier (Generative Software, 2010). For UML-based DSMLs, such evidence has been missing so far. We found that the majority of DSMLs (16) providing for platform integration (D6) adopt the GENERATOR TEMPLATE (O6.2) option. This is a mandate for DSML-specific research on M2T transformation. Examples include scalable automated change propagation from domain-specific models to text artifacts (Ogunyomi et al., 2015) and coupled evolution of generator templates under metamodel composition for DSML integration (Hoisl et al., 2013).

Frequency patterns. In total, our study identified and describes 16 frequent combinations of decision options: 9 smallest common option subsets and 7 prototype option-sets (see Section 5). The latter are summarized below. To be considered frequent, an option subset was required to occur in at least three different DSMLs (see Section 3). During this pattern analysis, we learned that the reviewed DSMLs have a maximum of ten decision options per DSML (e.g. UML4SOA). For recurring option subsets, i.e. option subsets found in more than one DSML, the maximum number of options in a particular option subset was seven (e.g. UML-PMS). At the level of individual decision points, these maxima translate into frequently recurring subsets containing options from three decision records only: language-model definition (D1), language-model formalization (D2), and concrete-syntax definition (D4).

As a key finding, we extracted seven option subsets which form so called *prototype option-sets*. These prototype option-sets are a particularly useful way for structuring the design space described by the 24 observed decision options. First, they cover a critical share of the observed DSMLs. Second, they stress commonalities and differences in terms of decision options between these highly representative option combinations. In particular, the seven prototype option-sets characterize 30% of the observed DSMLs (24/80) in their entirety. Furthermore, they are contained as large proper subsets by 25 extended option sets; therefore reaching a total coverage of approximately 61% of the DSMLs included in our study (49/80). By looking at the common and varying decision options in the seven prototype option-sets, we find that all seven are combinations of nine decision options. These nine decision options correspond to the leaf elements of the feature diagram in Fig. 17. As a result of this observation, these nine options were highlighted in our catalog. In addition, we extended the catalog to include additional reading aids based on the prototype option-sets. For example, we added a variant of the feature diagram from Fig. 17 as well as thumbnail descriptions of the nine key options (see also Tables 1 and 2 in Section 2.1). Adding

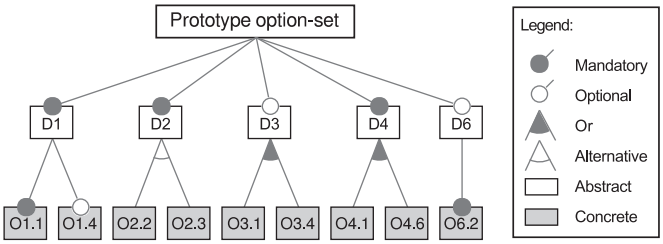


Fig. 17. A feature diagram (Apel et al., 2013) representing the seven prototype option-sets found in the pool of 80 third-party DSMLs. These seven prototype option-sets include nine different decision options. Each of the seven observed prototype option-sets listed in Table 11 is one of the possible configuration of the feature space.

these content and navigation items is a concrete means to render the catalog tailorable (Falessi et al., 2013; Kruchten et al., 2006).

Moreover, we were able to identify ten additional associations between different decision options that are not yet included in the prototype option-sets (see also Table 12). Associations were found in terms of the two smallest common option subsets (e.g. model-transformation chaining in D6; see Section 2.1) and in terms of associations derived from the prototype option-sets. From our analysis, we also learned about the frequency at which certain associations occur. This finding also allowed us to confirm already documented associations, such as the association between UML profiles and certain concrete-syntax decisions. To reflect this evidence for the reader of the catalog, we annotated the frequently observed associations accordingly (ten in total).

Revised catalog for DR reuse. As an important by-product of our study, the revised decision-record catalog serves as an evidence-based and empirically validated foundation for decision making in the context of UML-based DSML projects (see Fig. 18). For the *identification* of key decision problems, decision makers can consult our decision-record catalog, which documents recurring decision contexts and decision problems. The decision options and associations support the *inventory* of suitable design decisions for deriving a selection of candidate solutions. Finally, the *evaluation* of candidate solutions for their fit to domain-specific and domain-generic requirements can be performed using the documented decision drivers and decision consequences. In addition, the decision options are aligned with different styles of DSML development (abstract-syntax-first vs. concrete-syntax-first vs. extraction-based; see, e.g., Zdun and Strembeck, 2009; Strembeck and Zdun, 2009).

The importance of the frequency patterns summarized above stems from the fact that the documented decision options provide for a huge number of possible design option-sets. Thus, without evidence-based guidance, making an informed choice from this amount of combinations becomes impractical. Likewise, performing design-space analyses or creating design-process documentation based on the decision-record catalog can become a time-consuming and tedious task (see Section 2.2). This is because of the sheer size of the documented design-decision space. Creating and maintaining design-rationale documentation for a DSML project, however, is a substantial investment, and the effectiveness of this investment is largely affected by the tailorability of the reusable documentation of generic design rationale. A *tailorable* documentation of generic design rationale (Falessi et al., 2013) provides means for customization such that for each DSML project only relevant documentation items are provided to decision makers (e.g. domain engineers, modelers, language engineers, and software architects). Moreover, customization usually involves supporting visualizations (e.g. decision-flow and activity diagrams) for particular use cases (Kruchten et al., 2006).

7. Related work

As explained in Section 2, the practice of documenting generic design decisions via collections of structured descriptions is an established procedure in the domain of software engineering (Burge et al., 2008; Dutoit et al., 2006). In this paper, we also documented *reusable* design decisions in a structured and *tailorable* manner (Falessi et al., 2013). The way we documented the generic DSML rationale in this paper is related to similar documentation techniques applied for software-architecture design and architectural design decisions (Harrison et al., 2007; Shahin et al., 2009; Tang et al., 2010; Heesch et al., 2012).

Moreover, we identified two additional areas of related work for this paper. On the one hand, we have related work on DSL design procedures and DSL design-decision making (see Section 7.1). On the other hand, there are related contributions which have systematically gathered empirical evidence as well as generic design rationale on using the UML for domain-specific modeling (see Section 7.2).

7.1. Systematic development of domain-specific languages

Several related contributions exist that describe systematic procedures for developing DSLs. Each of these approaches is based on experiences drawn from actual DSL engineering projects, and each of the related approaches provides insights into the DSL development process, or into certain aspects of DSL design, or into DSL-related design decisions. For example, Strembeck and Zdun (2009) discuss different DSL development activities and describe how these activities can be combined to tailor a DSL engineering process.

In a complementary contribution, Zdun and Strembeck (2009) document three main decisions to be made when applying the DSL development process from Strembeck and Zdun (2009). These decisions relate to the choices of a specific type of DSL development process, of a concrete syntax style, and of developing an external vs. an embedded DSL. To render these decision descriptions reusable, Zdun and Strembeck (2009) document them in a pattern-like format. In software engineering, a pattern is a time-proven solution to a recurring design problem. A pattern description includes (at least) a “problem description”, a description of the “context” in which the respective problem occurs, and one or more (alternative) “solutions”. Typically, pattern descriptions also include different “forces” that may influence the choice of a certain solution, “consequences” that arise from a solution, as well as “known uses” of a particular solution. In this way, the description format we chose for the decision records resembles a pattern format to a certain degree. However, decision records are not identical to or variants of software patterns, since, for example, they list multiple solution propositions (decision options) rather than one.

While prior work on patterns for DSL development (Zdun and Strembeck, 2009; Strembeck and Zdun, 2009) aims at describing generic procedures and decisions for DSL development projects, our contribution in this paper provides detailed insights into design decisions for UML-based DSMLs. In this way, our work complements Zdun and Strembeck, 2009; Strembeck and Zdun, 2009, as well as other DSL development approaches such as Stahl and Völter, 2006, and Völter, 2013. This is because our work provides a systematic and in-depth documentation of the follow-up decisions that DSL engineers face after they decided to develop a UML-based DSML.[22]

A number of other patterns and pattern languages exist that can be applied in DSL development and are thereby complementary to our work. This includes patterns for the design and implementation of DSLs (Spinellis, 2001), patterns for evolving frameworks into

[22] Remember that each UML-based DSML is an embedded DSL and that UML-based DSMLs usually have a graphical concrete syntax or a mixture of graphical and textual concrete syntaxes (see also Section 2).

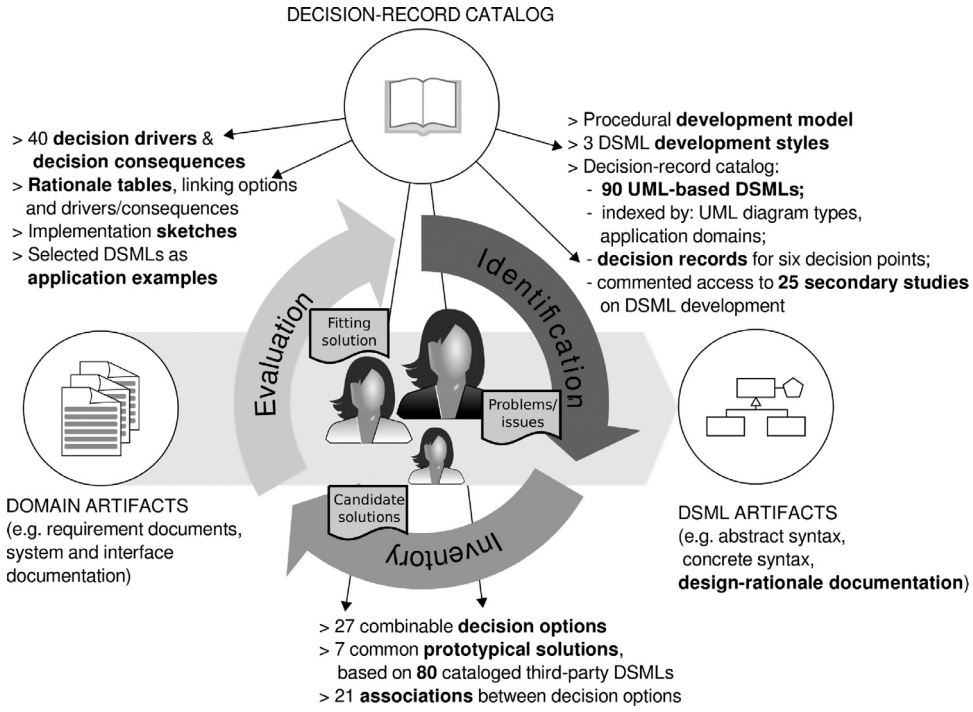


Fig. 18. Supporting decision making (identification, inventory, evaluation) on developing UML-based DSMLs via an evidence-based *decision-record catalog* providing reusable design decisions (Hoisl et al., 2014a).

DSLs (Roberts and Johnson, 1996; 1997), and approaches for pattern-based DSL development (Schäfern et al., 2011). Often, DSL-related patterns do not only describe how a DSL is developed, but also why it is developed in a specific way. In addition, pattern languages also describe potential sequences in which the patterns can be applied (Buschmann et al., 2007). Pattern sequences compare with our notion of option sets (see Section 2.1.2) in the sense that (ordered) option sets can represent sequences of adopted decision options. The research design of this study, however, does not allow us to assume a temporal ordering when extracting design decisions (see Section 5.4).

Mernik et al. (2005) used the patterns from Spinellis (2001) to conduct a survey on decision factors affecting the analysis, design, and implementation phases of DSL development. These decision factors can be considered during DSL development. For example, the decision factor *Notation* deals with the consideration whether the DSL should provide a new or an existing domain notation. For a few decision factors, Mernik et al. (2005) suggest implementation guidelines. The work of Mernik et al. (2005) is complementary to ours as it focuses on general issues of design-decision making and implementation, rather than on design decisions for a specific (host) language environment such as the UML.

Another group of related work reports observations from developing DSLs in (industrial) practice. For example, Luoma et al. (2004) conducted a study including 23 industrial projects for the definition of DSMLs. Similar to our approach, a number of DSLs are systematically compared. However, in contrast to our paper Luoma et al. (2004) provide a high-level description only and do not describe specific DSL design decisions or decision-option sets in detail. Similar to patterns, lessons learned have been used as a vehicle to preserve best practices of DSL development. For example, Wile (2004) reports on twelve lessons learned from three DSL experiments. For each lesson, he introduces a respective rule of thumb and gives an overview of the experiences that are the origin of the corresponding rule. Despite Wile's lessons learned being described at a comparatively high level of abstraction, they can, in general, also be observed in our work and are hence reflected in parts of our decision-record catalog. Kelly and Pohjonen (2009) present a report on worst practices found by reviewing 76 DSL development projects, and Karsai et al. (2009) propose 26 general guidelines for DSL development derived from their own experiences.

A UML-based DSML uses UML as its host language and extends the UML with domain-specific language elements and, therefore, qualifies as an embedded DSL (also: internal DSL). Related work on developing embedded DSLs includes the contributions by Günther et al. which describe a process and corresponding patterns for the development of internal DSLs on top of dynamic programming languages, such as Ruby or Python (Günther, 2011; Günther et al., 2010; Günther and Cleenewerck, 2010). Other related contributions describe how to build DSLs from component building blocks that can be incrementally designed and composed (see, e.g., Allen et al., 2005; Thibault et al., 1999). This idea originates from approaches such as keyword-based programming (Cleenewerck, 2003), in which so called "keywords" serve as building blocks for DSLs. In particular, a number of (universal) keywords are suggested which are then glued together to compose DSLs. This approach was first envisioned in Landin (1966) and is akin to building embedded DSLs in dynamic languages (such as Ruby, Perl, Python, or Tcl for example).

In the UML context, some authors propose approaches that define domain-specific UML extensions via UML profiles (see, e.g., Paige et al., 2000; Grant et al., 2004; Robert et al., 2009; Selic, 2007). While each of these approaches is related to our work, none of them document generic design decisions for UML-based DSMLs. Weisemöller and Schürr (2008) give an overview of standard compliant ways to define domain-specific UML extensions, while Atkinson and Kühne (2002) discuss potential issues with UML profiles and suggest a solution to address these problems. Bruck and Hussey (2008) present different techniques for tailoring the UML (e.g. lightweight profile or middleweight metamodel extension). In particular, Bruck and Hussey (2008) define a catalog of options and characterize different extension mechanisms accordingly. They also discuss pros and cons of using one approach or the other. However, Bruck and Hussey (2008) focus on UML customization techniques in general and do not

integrate design decisions in the process of DSML development (e.g. no development phases are distinguished, language-model constraints as well as platform integration are not considered).

In addition, knowledge on DSL design decisions can also be gained from analyzing toolkits for DSL development. For example, Tolvanen and Kelly (2009) present a tool for the definition and usage of integrated DSMLs. Similarly, Zdun (2010) presents a tool suite for textual DSL-based software and provides a discussion of architectural decisions for DSL development. However, most existing contributions have a strong focus on textual domain-specific programming languages. To the best of our knowledge, there is no report reflecting on design decisions embodied in toolkits for UML-based DSML development.

In summary, the related work on patterns, best practices, and lessons learned in DSL development have in common with our approach that all are based on experiences from actual DSL projects and contain some information on DSL design decisions and design rationale. Our work provides a systematic and detailed description of decision options for building UML-based DSMLs. In this way, our contribution is complementary to those other approaches and can be combined with them.

7.2. Empirical evidence on UML and MDD usage

We identified a number of systematic UML-related empirical studies closely related to our paper. The research methods employed in these studies include systematic reviews (5), surveys (2), a controlled experiment (1), and a case study (1). Our findings complement the empirical observations of a first group of related work (e.g. descriptive statistics on UML usage).

Dobing and Parsons (2006) performed a web-based survey among 171 OMG members on the perceived importance of the use-case viewpoint offered by the UML. They found that UML class and use-case diagrams are the most widely and regularly adopted diagram types (i.e. in more than two thirds of the respondents projects), with state machine and collaboration diagrams ranking lowest. With respect to UML-based DSMLs, our study shows a slightly different picture (see Section 5). While we can confirm the preponderance of class diagrams, we found that use-case diagrams are rarely customized by means of a DSML while state machine diagrams are frequently adopted in DSMLs.

Pardillo (2010) conducted a SLR for the years from 1999 to 2009 to characterize trends on UML profile usage. Most notably. The SLR documented publication trends, descriptive statistics (e.g. extended metaclasses, profile-concept usage frequencies) and definition issues of documented profile packages. The review was limited to a few handpicked, but important presence venues (ER, MoDELS/UML) including co-located workshops. An automated search in DBLP was used, yielding 63 hits. 39 of those 63 were included in the study. After a peak in 2002, Pardillo observes a decrease in publications that present a UML profile. However, the SLR study of Pardillo (2010) is not directly comparable to ours. This is because of a different review period, our study's reach beyond UML profiles, and because of our more extensive pool of venues. We cover journals and conferences beyond MoDELS/UML and ER. Yet, in our study, we also observed a falling number of yearly publications on UML-based DSMLs including UML profiles (with intermittent peaks in 2006 and 2011). Conference publications are on the decline and journal publications stagnate. We can also confirm that the MoDELS conference continues to serve as the top venue for relevant publications. It ranks first in terms of included publications in our paper corpus, while the ER conference only contributes one publication (see Section 4.2).

Nascimento et al. (2012) conducted a systematic mapping study on DSLs in more general and for an unbounded time range (1966–2011). They collected and reviewed 2,688 publications from 669 presence venues (i.e. conferences and workshops) and 180 archival venues (i.e. journals). The number of papers relating to (UML-based) DSMLs amounted to 163, 69 of which report on DSML creation. The data extracted from these DSML publications is unfortunately not discussed in detail in the research reported by Nascimento et al. (2012). Only the number of DSML publications (21) referring to the usage of UML profiles is documented. To this end, we complement the broader mapping work of Nascimento et al. (2012) for the later years in our review period by detailing the usage characteristics of profiles versus metamodel extensions in UML-based designs. The DSML-specific publications in their paper corpus served for establishing our QGS corpus as a third-party data source (see Section 4).

Garousi (2012) performed a search-driven literature review by collecting bibliographical records on UML-related monographs from the Google Books database for a time range between 2001–2009. As a major observation, the authors state a peak year in 2005, allegedly caused by authors writing new books on the UML 2.0 specification. From 2005 onwards, however, the study finds a substantial and continued reduction in new UML books being published, down to only two books in 2008. Our study draws a comparable timeline picture for scientific publications on UML 2.0 (see Section 4.2).

Nugroho and Chaudron (2008) provide empirical findings from an online questionnaire survey on the use of UML amongst 80 professional software engineers. Part of the results targeted "imperfections" in UML models; i.e. how often inconsistency, understandability, inaccuracy, and incompleteness in models led to implementation problems. The study indicates that in each of the four categories approximately 90% of the respondents reported the respective imperfection occurring sometimes, often, or very often during a project.

Hutchinson et al. (2014) and Whittle et al. (2014) report quantitative data on MDD practice in industry based on 449 responses to an online questionnaire, supplemented by qualitative data from 22 semi-structured interviews with MDD practitioners. Their report suggests that, in practice, the UML is the most commonly adopted modeling language with 85.5% of the respondents having adopted the UML. Another key finding, however, is that DSMLs have gained an important share in adoption. 39.2% of the respondents indicated that they use a custom DSML developed in house, 21.5% make use of DSMLs bundled with their modeling tool chain. Unfortunately, their report remains silent about the share of co-adoption of the standard UML and such DSMLs. Also, the authors do not discriminate between DSMLs based on the UML and those developed independently from the UML. In an earlier study, Hutchinson et al. (2011) hint at the fact that such DSMLs could qualify as DSMLs embedded within the UML (e.g. using profiles). Therefore, these studies provide motivation for our work, because they stress the need for systematic empirical research on UML-based DSML development to fill the above gap. Regarding platform integration (D6), only ≈32% of the DSML projects considered platform-integration techniques. This observation is confirmed by our study (see Section 5.3). Hutchinson et al. (2014) report a comparatively high number of projects using MDD for code generation (O6.2) as well as M2M transformation (O6.1, O6.5). Even fewer respondents indicated to rely on executable models (O6.4)—observations which are supported by our results. In addition, the reported frequency of UML diagram types matches our observations of a dominance of structural and, in particular, class diagrams (see Table 9).

Recently, Gorschek et al. (2014) reported an inquiry concerning the adoption of object-oriented design and implementation concepts (including object-oriented design models) using a web-based survey. 4,823 developers finally responded, 78.5% of which completed the survey. The only selection criterion for participants was some proficiency in an object-oriented programming language (e.g. C#, Java, Python). The authors found that approximately 50% of the respondents never (i.e. in less than 10% of the development activities) and about 70% rarely (in less than 25% of the development activities) use any diagrammatic design models. In contrast, only approximately 11%

employ models in more than 75% of their development activities. In terms of demographics, this minority of software developers has received a higher level of formal education. When using a formal diagrammatic notation, the UML or a UML-inspired notation was prominently mentioned by the respondents. This large-scale survey, while being widely unspecific of the concrete type of UML usage (e.g. diagram types), just stresses that there is only very little evidence on the use of UML-based DSMLs in software-development practice and on their design details. Our study cannot fully close this particular gap given a different intention (extraction of generic design rationale) and its data sources (i.e. mainly scientific publications by the DSML developers). However, we provide a data set of DSML projects as a starting point for follow-up work on DSML usage in industry practice.

A second group of related empirical studies also documented important generic design rationale which was included in our decision-record catalog. Below, we iterate over these secondary studies to stress how we incorporated their findings on UML-specific design rationale into our catalog.

In a follow-up and partly replicated review of Pardillo (2010) focusing on the data warehouse domain only, Pardillo and Cachero (2010) make the case of the (unwanted) coupling of abstract-syntax and concrete-syntax decisions of a DSML when adopting UML profiles. This issue has been included into our revised decision-record catalog via option associations between D2 and D4.

The SLR of Budgen et al. (2011) aims to collect and to reflect on empirical research on the UML, involving measurement, comprehension, quality, and maintenance of UML models. The review yielded 49 papers published up to 2008. The overview of papers on UML comprehension provides an important auxiliary source for our decision records on concrete-syntax design (D4). The included papers cover findings on the comprehensibility of stereotyped and non-stereotyped models for different audiences, as well as the cognitive effectiveness of different diagram types (sequence diagrams). In addition, the authors recorded the different research methods adopted. These methods included mainly small-scale controlled experiments in laboratory settings. Budgen et al. (2011) also found two empirical studies on "notation extensions" in the UML, however, the respective paper category was deliberately excluded from the analysis because the authors were only concerned with contributions using standard UML. To this end, we complement the SLR of Budgen et al. (2011) by organizing papers on UML extensions explicitly, including notation (concrete-syntax) extensions. At the same time, our SLR is a primary study on mining UML-specific design rationale.

Staron and Wohlin (2006) report on a confirmatory case study to test six alleged advantages and disadvantages of UML extension techniques (profiles and metamodel extensions) based on an industrial case: the UML-driven modeling toolkit Tau G2 by Telelogic. The authors triangulated their observations from data sources such as profile definitions and metamodel extensions implemented using Tau G2, requirements documents of Telelogic's customers, a fault database, and interviews to reflect on decision-making processes. A key observation was that software products based on metamodel extensions exhibit more faults in terms of unfulfilled or violated customer requirements. In addition, metamodel extensions were found to incur substantially more development effort (three to four weeks working time) than profiles (one week). We incorporated these evidence-based findings as arguments (drivers) into decision record D2. The observations of Staron and Wohlin (2006) rest upon metamodel-based extensions to activity diagrams as well as upon profile-based extensions to deployment, interaction overview, and component diagrams.

Staron et al. (2006) conducted a series of four experiments on the understandability of stereotyped UML class models, all replicating a single design but varying in terms of control levels and details of the treatment (e.g. order of artifact presentation). The experiments were set in an academic environment (68 student subjects) and in an industrial environment (four professional subjects). The replicated design involved twelve tasks to be solved by the subjects. The tasks comprised counting specific model elements and identifying definition defects in stereotyped and non-stereotyped models. Non-stereotyped models contained the domain semantics of the class model as unstructured UML comments attached to model elements. The authors recorded three data points per subject: total and per-task solving time as well as the number of correctly solved tasks. In this way, the authors found that the subjects solved more tasks correctly based on stereotyped models. In addition, there was a decrease in the total and relative task-solving times for stereotyped class models. We refer to the findings from Staron et al. (2006) as part of a decision driver on the understandability of concrete-syntax options (D4).

8. Conclusion

Existing approaches towards systematizing DSML development (Strembeck and Zdun, 2009; Lagarde et al., 2008) put forth a development-process perspective, treating DSML development as a complex flow of exploratory and iterative development activities. Key activities are language-model definition, constraint specification, concrete-syntax design, and platform integration (Strembeck and Zdun, 2009). A process perspective has immediate benefits. It establishes a shared process vocabulary (e.g., of different development artifacts) and represents common DSML development styles via different flows between development activities (e.g., language-model-driven or mockup-driven DSML development). On the flip side, a process perspective alone abstracts from DSML development as a result of continuous decision-making by stakeholders (e.g., domain engineers, modelers, language engineers, and software architects). Such decision making is always situated in a specific context shaped by domain requirements and architecture requirements (e.g., a certain modeling technology stack such as the UML).

Therefore, our work proposes a complementary *decision-centric perspective* on DSML development. For the scope of UML-based DSMLs, our focus is on providing *tailorable* documentation (Falessi et al., 2013) of *generic* design decisions (Harrison et al., 2007). This paper puts emphasis on the scientific approach: A three-year, multi-method empirical study based on a systematic literature review (SLR), a content analysis, and a frequent-item-set analysis was used to collect and to validate generic, reusable design decisions. Key findings are:

- The study revealed 24 design-decision options in 80 UML-based DSMLs reported between 2005 and 2012, covering design aspects from language-model implementation in the UML to platform integration (the complete list of DSMLs is provided in the Appendix).
- Language-model piggybacking (Spinellis, 2001) using UML profiles is the most frequently adopted DSML implementation technique (by 78% of the DSMLs).
- Approximately one third of the documented decision options (9/24) can be combined to characterize more than 60% of the reviewed DSMLs (49/80).
- Based on these 9 critical decision options, 7 different *prototype* DSML designs representing different variants of UML piggybacking and UML extension (Spinellis, 2001) were identified.
- The most frequently tailored UML diagram types are class diagrams (≈63% of the DSMLs) and activity diagrams (25%), respectively.
- The publication activity on UML-based DSMLs including premier journal venues (e.g. SoSyM, IST) and conference venues (e.g. MoDELS, ECMFA) is on a downward trend.

These findings were incorporated into an empirically validated and publicly available collection of reusable design decisions on UML-based DSMLs. This *decision-record catalog* (Hoisl et al., 2014a)

lists 27 *decision options* and their interdependencies for six different design-decision concerns (e.g. language-model formalization, concrete-syntax definition). We used the data on frequency as well as on commonalities and differences in DSML designs to highlight prototype designs in the catalog. This way, we provide guidance for DSML engineers and enable them to quickly find examples of decision-option combinations that were successfully applied in other DSML projects. Such guidance is particularly important because, in total, hundreds of millions of possible decision-option sets can be derived from our decision-record catalog. To the best of our knowledge, our work is the first attempt to document design rationale on UML-based DSML development driven by empirical evidence collected at a large scale.

The decision-record catalog (Hoisl et al., 2014a) and the detailed SLR protocol (Sobernig et al., 2014) are publicly available. Important by-products of this research project were multiple UML-based DSMLs (see, e.g., SOFServices), techniques for integrating (otherwise independent) DSMLs (see, e.g., Hoisl et al., 2013), testing techniques for DSML artifacts (see, e.g., Sobernig et al., 2013; Hoisl et al., 2014c), and empirical studies to evaluate the resulting research artifacts (e.g. a controlled experiment on testing notations; Hoisl et al., 2014b).

In future work, we will continue to extend and to evaluate the decision-record catalog by incorporating additional third-party DSMLs. We will replicate our SLR study for publication years starting with 2013. This way, we will also seek the missing evidence for decision options deemed candidates after our SLR study (see Section 6). In parallel, we plan to add design decisions on *DSML tooling support* as a seventh decision record (D7) to cover DSML editors and generators. Furthermore, to overcome certain study limitations (see Section 5.4), we will try to engage outside DSML experts, including the authors of the 80 DSMLs identified during our SLR study. We will employ direct inquisitive (e.g. interviews) and observational research techniques (e.g. participant observations) to collect more qualitative evidence, such as on time ordering in design-decision making.

Acknowledgments

This work has partly been funded by the Austrian Research Promotion Agency (FFG) of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) through the Competence Centers for Excellent Technologies (COMET K1) initiative.

Supplementary material

Supplementary material associated with this article (Appendix) can be found in the online version at 10.1016/j.jss.2015.11.037.

References

Allen, N., Shaffer, C., Watson, L., 2005. Building modeling tools that support verification, validation, and testing for the domain expert. In: Proceedings of the 37th Winter Simulation Conference. IEEE, pp. 419–426.
Apel, S., Batory, D.S., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines: Concepts and Implementation. Springer.
Atkinson, C., Kühne, T., 2001. Processes and products in a multi-level metamodeling architecture. Int. J. Softw. Eng. Know. 11 (6), 761–783.
Atkinson, C., Kühne, T., 2002. Profiles in a strict metamodeling framework. Sci. Comput. Program. 44 (1), 5–22.
Atkinson, C., Kühne, T., 2007. A tour of language customization concepts. Adv. Comput. 70, 105–161.
Atkinson, C., Kühne, T., Henderson-Sellers, B., 2003. Systematic stereotype usage. Softw. Syst. Model. 2 (3), 153–163.
Australian Research Council, 2012. Excellence in Research for Australia (ERA) journal list. Available at: http://www.arc.gov.au/era/era_2012/era_journal_list.htm. Last accessed: 3 October 2014.
Blouin, A., Combemale, B., Baudry, B., Beaudoux, O., 2015. Kompren: Modeling and generating model slicers. Softw. Syst. Model. 14 (1), 321–337.
Borgelt, C., 2012. Frequent item set mining. Wiley Int. Rev. Data Min. Knowl. Disc. 2 (6), 437–456.
Broy, M., Cengarle, M., 2011. UML formal semantics: Lessons learned. Softw. Syst. Model. 10 (4), 441–446.
Bruck, J., Hussey, K., 2008. Customizing UML: Which technique is right for you? Available at: http://www.eclipse.org/modeling/mdt/uml2/docs/articles/Customizing_UML2_Which_Technique_is_Right_For_You/article.html. Last accessed: 9 September 2015.
Budgen, D., Burn, A.J., Brereton, O.P., Kitchenham, B.A., Pretorius, R., 2011. Empirical evidence about the UML: A systematic literature review. Softw. Pract. Exp. 41 (4), 363–392.
Burge, J.E., Carroll, J.M., McCall, R., Mistrík, I., 2008. Rationale-Based Software Engineering. Springer.
Burgués, X., Franch, X., Ribó, J.M., 2008. Improving the accuracy of UML metamodel extensions by introducing induced associations. Softw. Syst. Model. 7 (3), 361–379.
Buschmann, F., Henney, K., Schmidt, D.C., 2007. Pattern-oriented Software Architecture – On Patterns and Pattern Languages. John Wiley & Sons.
Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., 2000. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons.
Clark, T., Sammut, P., Willans, J., 2008. Applied Metamodelling: A foundation for language-driven development, second ed. Ceteva.
Cleenewerck, T., 2003. Component-based DSL development. In: Proceedings of the 2nd International Conference on Generative Programming and Component Engineering. In: *LNCS*, 2830. Springer, pp. 245–264.
Cook, S., 2012. Looking back at UML. Softw. Syst. Model. 11 (4), 471–480.
Coplien, J., 1996. Software Patterns. SIGS Books & Multimedia.SIGS Management Briefings.
Czarnecki, K., Eisenecker, U.W., 2000. Generative Programming: Methods, Tools, and Applications. Addison-Wesley.
Dingel, J., Diskin, Z., Zito, A., 2008. Understanding and improving UML package merge. Softw. Syst. Model. 7 (4), 443–467.
Dobing, B., Parsons, J., 2006. How UML is used. Commun. ACM 49 (5), 109–113.
Dutoit, A.H., McCall, R., Mistrík, I., Paech, B., 2006. Rationale management in software engineering: concepts and techniques. In: Dutoit et al. (Eds.), Rationale Management in Software Engineering. Springer, pp. 1–48.
Evans, E., 2004. Domain-driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley.
Falessi, D., Briand, L.C., Cantone, G., Capilla, R., Kruchten, P., 2013. The value of design rationale information. ACM Trans. Softw. Eng. Methodol. 22 (3), 21:1–21:32.
Filtz, E., 2013. Systematic Literature Review and Evaluation of DSML-Design Decisions. Bachelor Thesis. WU Vienna.
Fleiss, J.L., 1981. Statistical Methods for Rates and Proportions. John Wiley & Sons.
Garousi, V., 2012. Classification and trend analysis of UML books (1997–2009). Softw. Syst. Model. 11 (2), 273–285.
Generative Software, 2010. Umfrage zu Verbreitung und Einsatz modellgetriebener Softwareentwicklung. Survey Report. Generative Software GmbH and FZI Forschungszentrum Informatik.
Giachetti, G., Marín, B., Pastor, O., 2009. Using UML as a domain-specific modeling language: A proposal for automatic generation of UML profiles. In: Proceedings of the 21st International Conference on Advanced Information Systems Engineering. In: *LNCS*, 5565. Springer, pp. 110–124.
Gorschek, T., Tempero, E., Angelis, L., 2014. On the use of software design models in software development practice: An empirical investigation. J. Syst. Softw. 95, 176–193.
Grant, E., Narayanan, K., Reza, H., 2004. Rigorously defined domain modeling languages. In: Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling, Technical Reports TR-33, University of Jyväskylä. Available at: http://www.dsmforum.org/events/dsm04/grant.pdf. Last accessed: 4 December 2015.
Günther, S., 2011. Development of internal domain-specific languages: Design principles and design patterns. In: Proceedings of the 18th Conference on Pattern Languages of Programs. ACM, pp. 1:1–1:25.
Günther, S., Cleenewerck, T., 2010. Design principles for internal domain-specific languages: A pattern catalog illustrated by Ruby. In: Proceedings of the 18th Conference on Pattern Languages of Programs. ACM, pp. 3:1–3:35.
Günther, S., Haupt, M., Splieth, M., 2010. Agile engineering of internal domain-specific languages with dynamic programming languages. In: Proceedings of the 5th International Conference on Software Engineering Advances. IEEE, pp. 162–168.
Gwet, K.L., 2012. Handbook of Inter-Rater Reliability, third ed. Advanced Analytics, LLC.
Hahsler, M., Grün, B., Hornik, K., 2005. arules—a computational environment for mining association rules and frequent item sets. J. Stat. Softw. 14 (15), 1–25.
Harrison, N., Avgeriou, P., Zdun, U., 2007. Using patterns to capture architectural decisions. IEEE Softw. 24 (4), 38–45.
Heesch, U., Avgeriou, P., Hilliard, R., 2012. A documentation framework for architecture decisions. J. Syst. Softw. 85 (4), 795–820.
Henderson-Sellers, B., Gonzalez-Perez, C., 2006. Uses and abuses of the stereotype mechanism in UML 1.x and 2.0. In: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems. In: *LNCS*, 4199. Springer, pp. 16–26.
Hohpe, G., Woolf, B., 2004. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley.
Hoisl, B., Sobernig, S., Schefer-Wenzl, S., Strembeck, M., Baumgrass, A., 2012a. A Catalog of Reusable Design Decisions for Developing UML- and MOF-Based Domain-Specific Modeling Languages. Available at: http://epub.wu.ac.at/3578/. Technical Report 2012/01, WU Vienna. Last accessed: 4 December 2015.
Hoisl, B., Sobernig, S., Schefer-Wenzl, S., Strembeck, M., Baumgrass, A., 2012b. Design decisions for UML and MOF based domain-specific language models: Some lessons learned. In: Proceedings of the 2nd Workshop on Process-based Approaches for Model-Driven Engineering. Technical University of Denmark, pp. 303–314.

Hoisl, B., Sobernig, S., Strembeck, M., 2013. Higher-order rewriting of model-to-text templates for integrating domain-specific modeling languages. In: Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development. SciTePress, pp. 49–61.

Hoisl, B., Sobernig, S., Strembeck, M., 2014a. A Catalog of Reusable Design Decisions for Developing UML/MOF-Based Domain-Specific Modeling Languages. Available at: http://epub.wu.ac.at/4466/. Technical Report 2014/03, WU Vienna. Last accessed: 4 December 2015.

Hoisl, B., Sobernig, S., Strembeck, M., 2014b. Comparing three notations for defining scenario-based model tests: A controlled experiment. In: Proceedings of the 9th International Conference on the Quality of Information and Communications Technology. IEEE, pp. 95–104.

Hoisl, B., Sobernig, S., Strembeck, M., 2014c. Natural-language scenario descriptions for testing core language models of domain-specific languages. In: Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development. SciTePress, pp. 356–367.

Horner, J., Atwood, M., 2006. Effective design rationale: Understanding the barriers. In: Dutoit et al. (Eds.), Rationale Management in Software Engineering. Springer, pp. 73–90.

Hutchinson, J., Whittle, J., Rouncefield, M., 2014. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. Sci. Comput. Program. 89, Part B, 144–161.

Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S., 2011. Empirical assessment of MDE in industry. In: Proceedings of the 33rd International Conference on Software Engineering. ACM, pp. 471–480.

Jackson, E., Sztipanovits, J., 2009. Formalizing the structural semantics of domain-specific modeling languages. Softw. Syst. Model. 8 (4), 451–478.

Jalali, S., Wohlin, C., 2012. Systematic literature studies: Database searches vs. backward snowballing. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, pp. 29–38.

Karsai, G., Pinkernell, H.K.C., Rumpe, B., Schindler, M., Völkel, S., 2009. Design guidelines for domain specific languages. In: Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling, Technical Reports B-108, HSE Print, pp. 7–13. Available at: http://epub.lib.aalto.fi/pdf/hseother/b108.pdf. Last accessed: 4 December 2015.

Kelly, S., Pohjonen, R., 2009. Worst practices for domain-specific modeling. IEEE Softw. 26 (4), 22–29.

Kelly, S., Tolvanen, J., 2008. Domain-Specific Modeling: Enabling Full Code Generation. John Wiley & Sons.

Kitchenham, B., 2004. Procedures for Performing Systematic Reviews. Joint Tech. Rep. (Keele University Tech. Rep., NICTA Tech. Rep.). Keele University & National ICT Ltd..TR/SE-0401, 0400011T.1

Kitchenham, B., Brereton, P., 2013. A systematic review of systematic review process research in software engineering. Inform. Softw. Tech. 55 (12), 2049–2075.

Kruchten, P., Lago, P., van Vliet, H., 2006. Building up and reasoning about architectural knowledge. In: Quality of Software Architecture. In: *LNCS*, 4214. Springer, pp. 43–58.

Lagarde, F., Espinoza, H., Terrier, F., Gérard, S., 2007. Improving UML profile design practices by leveraging conceptual domain models. In: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering. ACM, pp. 445–448.

Lagarde, F., Huáscar, E., Terrier, F., André, C., Gérard, S., 2008. Leveraging patterns on domain models to improve UML profile definition. In: Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering. In: *LNCS*, 4961. Springer, pp. 116–130.

Landin, P., 1966. The next 700 programming languages. Commun. ACM 9 (3), 157–166.

Langer, P., Wieland, K., Wimmer, M., Cabot, J., 2012. EMF profiles: A lightweight extension approach for EMF models. J. Object Technol. 11 (1), 1–29.

Langlois, B., Jitia, C.-E., Jouenne, E., 2007. DSL classification. In: Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling, Technical Reports TR-38, University of Jyväskylä. Available at: http://www.dsmforum.org/events/DSM07/papers/langlois_jitia.pdf. Last accessed: 4 December 2015.

Luoma, J., Kelly, S., Tolvanen, J., 2004. Defining domain-specific modeling languages: Collected experiences. In: Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling, Technical Reports TR-33, University of Jyväskylä. Available at: http://www.dsmforum.org/events/dsm04/luoma.pdf. Last accessed: 4 December 2015.

MacLean, A., Young, R.M., Bellotti, V.M.E., Moran, T.P., 1996. Questions, options, and criteria: Elements of design space analysis. In: Moran T.P. and Carroll J.M. (Eds.), Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum Associates, pp. 53–106.

Mallet, F., Andre, C., 2009. On the semantics of UML/MARTE clock constraints. In: Proceedings of the 12th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. IEEE, pp. 305–312.

Mayer, P., Schroeder, A., Koch, N., 2008. MDD4SOA: Model-driven service orchestration. In: Proceedings of the 12th International Enterprise Distributed Object Computing Conference. IEEE, pp. 203–212.

Mayerhofer, T., Langer, P., Wimmer, M., 2012. Towards xMOF: Executable DSMLs based on fUML. In: Proceedings of the 12th Workshop on Domain-Specific Modeling. ACM, pp. 1–6.

Mens, T., Gorp, P.v., 2006. A taxonomy of model transformation. Electron. Notes Theor. Comput. Sci. 152, 125–142.

Mernik, M., Heering, J., Sloane, A., 2005. When and how to develop domain-specific languages. ACM Comput. Surv. 37 (4), 316–344.

Microsoft Corporation, 2013. Microsoft academic search journal list. Available at: http://academic.research.microsoft.com/?SearchDomain=2&SubDomain=4&entitytype=4. Last accessed: 2 February 2015.

Moody, D., van Hillegersberg, J., 2009. Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams. In: Software Language Engineering. In: *LNCS*, 5452. Springer, pp. 16–34.

Nascimento, L., Viana, D.L., Neto, P.A.M.S., Martins, D.A.O., Garcia, V.C., Meira, S.R.L., 2012. A systematic mapping study on domain-specific languages. In: Proceedings of the 7th International Conference on Software Engineering Advances. IARIA XPS Press, pp. 179–187.

Naumann, F., Herschel, M., 2010. An Introduction to Duplicate Detection Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Nugroho, A., Chaudron, M.R., 2008. A survey into the rigor of UML use and its perceived impact on quality and productivity. In: Proceedings of the 2nd International Symposium on Empirical Software Engineering and Measurement. ACM, pp. 90–99.

Object Management Group, 2012a. OMG systems modeling language (OMG SysML). Available at: http://www.omg.org/spec/SysML. Version 1.3, formal/2012-06-01. Last accessed: 4 December 2015.

Object Management Group, 2012b. Service oriented architecture modeling language (SoaML) specification. Available at: http://www.omg.org/spec/SoaML. Version 1.0.1, formal/2012-05-10. Last accessed: 4 December 2015.

Object Management Group, 2015a. OMG meta object facility (MOF) core specification. Available at: http://www.omg.org/spec/MOF. Version 2.5, formal/2015-06-05. Last accessed: 4 December 2015.

Object Management Group, 2015b. OMG unified modeling language (OMG UML). Available at: http://www.omg.org/spec/UML. Version 2.5, formal/2015-03-01. Last accessed: 4 December 2015.

Ogunyomi, B., Rose, L.M., Kolovos, D.S., 2015. Property access traces for source incremental model-to-text transformation. In: Proceedings of the 11th European Conference on Modelling Foundations and Applications. In: *LNCS*, 9153. Springer, pp. 187–202.

Paige, R., Ostroff, J., Brooke, P., 2000. Principles for modeling language design. Inform. Softw. Tech. 42 (10), 665–675.

Paige, R.F., Kolovos, D.S., Polack, F.A., 2014. A tutorial on metamodelling for grammar researchers. Sci. Comput. Program. 96 (4), 396–416.

Pardillo, J., 2010. A systematic review on the definition of UML profiles. In: Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems. In: *LNCS*, 6394. Springer, pp. 407–422.

Pardillo, J., Cachero, C., 2010. Domain-specific language modelling with UML profiles by decoupling abstract and concrete syntaxes. J. Syst. Softw. 83 (12), 2591–2606.

Robert, S., Gérard, S., Terrier, F., Lagarde, F., 2009. A lightweight approach for domain-specific modeling languages design. In: Proceedings of the 35th EUROMICRO Conference on Software Engineering and Advanced Application. IEEE, pp. 155–161.

Roberts, D., Johnson, R., 1996. Evolve frameworks into domain-specific languages. In: Proceedings of the 3rd International Conference on Pattern Languages of Programs.

Roberts, D., Johnson, R., 1997. Patterns for evolving frameworks. In: Pattern Language of Program Design 3. Addison-Wesley, pp. 471–486.

Saldaña, J., 2013. The Coding Manual for Qualitative Researchers, second ed. Sage.

Schäfern, C., Kuhn, T., Trapp, M., 2011. A pattern-based approach to DSL development. In: Proceedings of the Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH). ACM, pp. 39–46.

Schreier, M., 2013. Qualitative Content Analysis in Practice, first ed. Sage.

Scimago Lab, 2013. SCImago journal & country rank. Available at: http://www.scimagojr.com/journalrank.php. Last accessed: 2 February 2015.

Selic, B., 2003. The pragmatics of model-driven development. IEEE Softw. 20 (5), 19–25.

Selic, B., 2004. On the semantic foundations of standard UML 2.0. In: Formal Methods for the Design of Real-Time System. In: *LNCS*, 3185. Springer, pp. 181–199.

Selic, B., 2007. A systematic approach to domain-specific language design using UML. In: Proceedings of the 10th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. IEEE, pp. 2–9.

Sen, S., Moha, N., Baudry, B., Jézéquel, J.-M., 2009. Meta-model pruning. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems. In: *LNCS*, 5795. Springer, pp. 32–46.

Sendall, S., Kozaczynski, W., 2003. Model transformation: The heart and soul of model-driven software development. IEEE Softw. 20 (5), 42–45.

Shahin, M., Liang, P., Khayyambashi, M.R., 2009. Architectural design decision: Existing models and tools. In: Joint Proceedings of the 3rd European Conference on Software Architecture and of the 8th Working IEEE/IFIP Conference on Software Architecture. IEEE, pp. 293–296.

Singer, J., Sim, S.E., Lethbridge, T.C., 2008. Software engineering data collection for field studies. In: Guide to Advanced Empirical Software Engineering. Springer, pp. 9–34.

Sobernig, S., Hoisl, B., Strembeck, M., 2013. Requirements-driven testing of domain-specific core language models using scenarios. In: Proceedings of the 13th International Conference on Quality Software. IEEE, pp. 163–172.

Sobernig, S., Hoisl, B., Strembeck, M., 2014. Protocol for a Systematic Literature Review on Design Decisions for UML-Based DSMLs. Available at: http://epub.wu.ac.at/4467/. Technical Report 2014/02, WU Vienna. Last accessed: 4 December 2015.

Spinellis, D., 2001. Notable design patterns for domain-specific languages. J. Syst. Softw. 56 (1), 91–99.

Stahl, T., Völter, M., 2006. Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons.

Staron, M., Kuzniarz, L., Wohlin, C., 2006. Empirical assessment of using stereotypes to improve comprehension of UML models: A set of experiments. J. Syst. Softw. 79 (5), 727–742.
Staron, M., Wohlin, C., 2006. An industrial case study on the choice between language customization mechanisms. In: Proceedings of the 7th International Conference on Product-Focused Software Process Improvement. In: *LNCS*, 4034. Springer, pp. 177–191.
Steele, G.L., 1990. Common Lisp: The Language, second ed. Digital Press.
Strembeck, M., Zdun, U., 2009. An approach for the systematic development of domain-specific languages. Softw. Pract. Exper. 39 (15), 1253–1292.
Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Babar, M.A., 2010. A comparative study of architecture knowledge management tools. J. Syst. Softw. 83 (3), 352–370.
Thibault, S., Marlet, R., Consel, C., 1999. Domain-specific languages: From design to implementation application to video device drivers generation. IEEE Trans. Softw. Eng. 25 (3), 363–377.
Tolvanen, J.-P., Kelly, S., 2009. MetaEdit+: Defining and using integrated domain-specific modeling languages. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications. ACM, pp. 819–820.
Torre, D., Labiche, Y., Genero, M., 2014. UML consistency rules: A systematic mapping study. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. ACM, pp. 6:1–6:10.
Turner, M., Kitchenham, B., Budgen, D., Brereton, P., 2008. Lessons learnt undertaking a large-scale systematic literature review. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. British Computer Society, pp. 110–118.
Völter, M., 2013. DSL Engineering – Designing, Implementing, and Using Domain-Specific Languages. Amazon Distribution.
Völter, M., Kircher, M., Zdun, U., 2005. Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware. John Wiley & Sons.
Webster, J., Watson, R.T., 2002. Analyzing the past to prepare for the future: Writing a literature review. MIS Quart. 26 (2), pp.xiii–xxiii.
Weisemöller, I., Schürr, A., 2008. A comparison of standard compliant ways to define domain specific languages. In: Workshop Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems. In: *LNCS*, 5002. Springer, pp. 47–58.
Whittle, J., Hutchinson, J., Rouncefield, M., 2014. The state of practice in model-driven engineering. IEEE Softw. 31 (3), 79–85.
Wieringa, R., Maiden, N., Mead, N., Rolland, C., 2005. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. Requir. Eng. 11 (1), 102–107.
Wile, D., 2004. Lessons learned from real DSL experiments. Sci. Comput. Program. 51 (3), 265–290.
Zdun, U., 2007. Systematic pattern selection using pattern language grammars and design space analysis. Softw. Pract. Exper. 37 (9), 983–1016.
Zdun, U., 2010. A DSL toolkit for deferring architectural decisions in DSL-based software design. Inform. Softw. Tech. 52 (9), 733–748.
Zdun, U., Strembeck, M., 2009. Reusable architectural decisions for DSL design: Foundational decisions in DSL development. In: Proceedings of the 14th European Conference on Pattern Languages of Programs. ACM, pp. 1–37.
Zhang, H., Babar, M.A., Tell, P., 2011. Identifying relevant studies in software engineering. Inform. Softw. Tech. 53 (6), 625–637.

Stefan Sobernig is a Postdoc Researcher and Lecturer at WU Vienna. In his research, he works in the fields of model-driven software development, domain-specific language engineering, feature-oriented software development, and software patterns.

Bernhard Hoisl works as a Postdoc Researcher at WU Vienna. His research interests are focused on model-driven software development, domain-specific languages, process modeling, and scenario-based testing.

Mark Strembeck is an Associate Professor of Information Systems at WU Vienna and a member of the key researcher team at Secure Business Austria (SBA). His research interests include secure business systems, model-driven software development, as well as the analysis, modeling, and management of dynamic software systems. He received his doctoral degree as well as his Habilitation degree (venia docendi) from WU Vienna.