2017 43rd Euromicro Conference on Software Engineering and Advanced Applications

A Systematic Mapping Study on DSL Evolution

Jürgen Thanhofer-Pilisch*, Alexander Lang*, Michael Vierhauser† and Rick Rabiser*
*Christian Doppler Lab MEVSS, ISSE, Johannes Kepler University Linz, Austria
Email: rick.rabiser@jku.at
†Computer Science and Engineering, University of Notre Dame, IN, USA
Email: mvierhau@nd.edu

***Abstract*—Domain-specific languages (DSLs) are frequently used in software engineering. In contrast to general-purpose languages, DSLs are designed for a special purpose in a particular domain. Due to volatile user requirements and new technologies DSLs, similar to the software systems they describe or produce, are subject to continuous evolution. This work explores existing research on DSL evolution to summarize, structure and analyze this area of research, and to identify trends and open issues. We conducted a systematic mapping study and identified 98 papers as potentially relevant for our study. By applying inclusion and exclusion criteria we selected a set of 34 papers relevant for DSL evolution. We classified and analyzed these papers to create a map of the research field. We conclude that DSL evolution is a topic of increasing relevancy. However, research on language evolution so far did not focus much on the characteristics DSLs exhibit. Also, there are not many cross-references between our primary studies meaning researchers are often not aware of potentially useful work. Our study results help researchers and practitioners working on DSL-based approaches to get an overview of existing research on DSL evolution and open challenges.**

***Keywords*-domain-specific languages; systematic mapping study; DSL evolution.**

I. Introduction

Domain-specific languages (DSLs) [1] are used to develop software solutions tailored to a specific problem or domain. The probably most well-known examples of DSLs are HTML and SQL. When compared to general-purpose languages, some of the main advantages of DSLs are their concise focus and their expressiveness regarding the investigated problem as well as their understandability due to the smaller logical gap between the DSLs' grammar and the problem domain.

In contrast to general-purpose programming languages, DSLs are generally not Turing-complete as they are tailored and limited to a certain task and lack expressiveness for other tasks. For instance, some DSLs are developed according to the "You ain't gonna need it" (YAGNI) principle [2]. This principle suggests to keep a DSL and its functionality as narrow as possible, i.e., to the minimum that is required to cover all relevant use cases. Keeping this principle in mind, a DSL is very likely to evolve to support future scenarios, e.g., to address new requirements. Evolving a DSL often implies not only grammatical changes but also requires modifications to the surrounding software infrastructure it is embedded in. Therefore, when developing a DSL, one must be aware of the challenges and impacts of future DSL evolution.

In our own work, for instance, we have been developing a DSL that supports defining constraints in a monitoring framework to represent requirements to be checked at runtime [2]. We have developed this language based on needs expressed to us by industrial engineers and by following the YAGNI principle. However, through its application to different systems in the industrial domain [3], new requirements motivated the extension of the DSL. While extending and refactoring our DSL we started studying research and experiences on DSL evolution already reported in the literature.

Through an ad-hoc literature review we quickly found systematic mapping studies and surveys of domain-specific languages in general [1], [4] [S4], studies discussing when and how to develop DSLs [S33], and studies providing an overview of implementation approaches for DSLs [5]. However, while papers on DSLs and general language evolution (e.g., Grifoni et al. [S26]) are easy to find with an ad-hoc literature search, it is hard to find and distinguish the papers that focus on the evolution of DSLs.

This paper thus focuses on structuring the research field of DSL evolution using a systematic mapping study [6], [7] to get a comprehensive overview of the approaches existing and the problems discussed in this field and to identify topics motivating further research. Our study results are intended for both researchers and practitioners working on DSL-based approaches, helping them to get an overview of existing research on DSL evolution and open challenges and topics for further research.

II. Systematic Mapping Study

Initial literature searches we conducted showed that a number of papers in several research areas discuss DSL development, potentially including approaches to DSL evolution. We thus decided to perform a systematic mapping study (SMS) to get an overview of the existing work in this field and to identify promising research on DSL evolution. Systematic mapping studies have been introduced to software engineering by Petersen et al. [6], [7]. Similarly to systematic literature reviews (SLRs) [8] – a method that has gained a lot of attention in software engineering research [9] – SMSs aim at identifying all literature relevant for a certain research question through a systematic, documented, and repeatable process reducing researcher bias as far as possible. In contrast to SLRs, however, SMSs provide a coarse-grained overview of the identified literature. Specifically, SMSs do not aim to analyze the identified research in detail but to provide a structure and categorization of the type of research reports

978-1-5386-2141-7/17 $31.00 © 2017 IEEE
DOI 10.1109/SEAA.2017.25

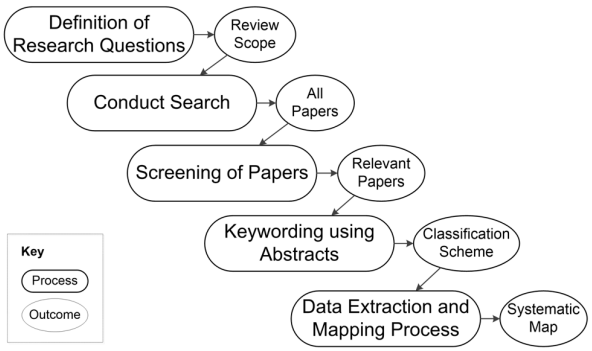

Fig. 1: The process of a systematic mapping study [6].

and results that have been published and a map giving a visual or tabular summary. SLRs typically focus on identifying best practices based on empirical evidence, while SMSs focus on classification, conducting thematic analysis, and identifying publication fora [6]. Both, SLRs and SMSs, support identifying research gaps, however, in SLRs this is more a side-effect of the in-depth analysis of the identified literature, while in SMSs revealing topic areas with a lack of publications is one of the key goals. Fig. 1 depicts the process of a SMS we also use to structure the rest of this section.

A. Research Questions

The first step in a SMS is to define the research questions. As described above our main goals are to structure the existing work on DSL evolution, to gain a comprehensive overview, and to identify challenges and topics that may require further research.

RQ1. What approaches for DSL evolution have been published? This question aims at identifying approaches supporting DSL evolution that have been proposed by existing research. The focus is on approaches that are general enough to be applied to more than only one DSL.

RQ2. What do publications report about DSL evolution in practice? Using DSLs is quite common in software engineering research and practice. To get an idea of how DSLs are developed and used and particularly how they evolve, we want to summarize information and experiences published on this topic.

RQ3. What approaches for general-purpose programming language evolution were adopted for DSL evolution? General-purpose programming languages and DSLs have many things in common. As the evolution of general-purpose programming languages is a more widely explored research field, we expect that there exists work that applies the findings of this field to the related field of DSL evolution.

B. Conduct Search for Primary Studies

For this SMS we want to consider any paper discussing DSL evolution regardless of the application area, thus we avoided limiting our search to a certain subtopic. We defined common synonyms for the field of domain-specific languages (DSL, Domain specific language, and Domain-specific language) and for the field of evolution (evolution, maintenance, refactoring, and reengineering) to avoid missing possibly relevant papers.

TABLE I: Digital libraries and search terms.

Library	Search Term
ACM DL	acmdlTitle:(+(DSL ”Domain specific language” language grammatical) +(evolution maintenance evolve evolving reuse refactoring refactor)) AND recordAbstract:(+(”Domain specific language” DSL) +(evolution evolve evolving refactoring maintenance))
IEEEXplore	(((p_Abstract:DSL) OR (p_Abstract:DSLs) OR (p_Abstract:”Domain specific language”) OR (p_Abstract:”Domain specific languages”)) AND ((p_Abstract:evolving) OR (p_Abstract:evolve) OR (p_Abstract:evolution) OR (p_Abstract:maintenance) OR (p_Abstract:refactoring) OR (p_Abstract:reengineering)) AND (p_Title:language))
ScienceDirect	TITLE-ABSTR-KEY((DSL OR ”Domain specific language”) AND (evolution OR evolve OR evolving OR maintenance OR refactor OR reengineer))
SpringerLink	TITLE((DSL OR ”language”) AND (evolution OR evolve OR evolving OR maintenance OR refactor OR reengineer OR engineer))

We decided to use four digital libraries – ACM DL, IEEEXplore, ScienceDirect, and SpringerLink – as these are the most common ones in the field of software engineering. We conducted the SMS from March 2016 to June 2016 and searched for all publications published before that time period.

We iteratively refined our search terms and evaluated the search results based on their number and quality. Specifically, we conducted several manual searches for relevant papers and checked if these were also found by our search terms to ensure that the terms cover the vast majority of relevant papers in the field. Each of our search terms combines the identified synonyms using Boolean AND and OR operators and appropriate filters to search in titles and abstracts only wherever possible. We mostly limited the search to titles and abstracts only as we learned during the pilot search phase that this delivers the most promising results: titles often contained too little information to determine whether the paper is relevant for our work whereas searching in the whole paper often leads to a large number of false positives.

Additionally, for ScienceDirect we limited the search to computer-science related publications. Also, as SpringerLink did not allow a very detailed search term configuration, we initially started with a very broad search for possibly relevant paper titles and subsequently refined these search results by automatically filtering them with a custom-developed script to allow papers with the appropriate keywords in the title only. The final search strings for each of the four digital libraries are listed in Table I. After removing duplicates our search resulted in 98 distinct papers.

TABLE II: Inclusion and exclusion criteria.

Inclusion Criteria	Exclusion Criteria
Papers presenting or discussing a general approach for DSL evolution.	Papers using the abbreviation "DSL" in another context (e.g., "Digital Subscriber Line").
Papers reporting practical experiences made when evolving a DSL.	Papers that mention DSLs and evolution in a different context, e.g., software evolution supported by a DSL.
Papers discussing relations between DSLs and general-purpose language evolution.	Papers not discussing DSL evolution at all.
Papers dealing with any other topic related with DSL evolution, e.g., DSL refactoring.	Grey literature such as editorials and keynote summaries.

C. Screening of Papers for Inclusion and Exclusion

For the screening (inclusion and exclusion) process we reviewed and checked the titles, abstracts and keywords of the 98 found papers. However, in some cases it was also necessary to skim through the content of the paper to get a good understanding of what the paper is about. We derived a set of inclusion and exclusion criteria (cf. Table II) from our research questions and goals to decide whether a paper is relevant for our work or not. At least one inclusion criterion had to be fulfilled for a paper to be included, i.e., all papers that at least address one of our research questions or discuss DSL evolution in general were added to our results list. We excluded all papers for which at least one exclusion criterion was fulfilled. For instance, we excluded all papers that were completely out of scope or dealing with a DSL and software evolution but not addressing the field of (domain-specific) language evolution. Also, we excluded grey literature such as keynote summaries and editorials.

Inclusion and exclusion criteria were applied by the first two authors of this paper separately. For each case in which authors disagreed, a discussion took place until consensus was reached. The authors agreed on 15 papers in the first round and were able to increase this number to 32 papers through discussion in the second round.

We also used 'snowballing' to identify further relevant papers. Snowballing [10] relies on investigating the reference lists of all papers selected as primary studies (backward snowballing) and the papers that reference one of our primary studies (forward snowballing; evaluated using Google Scholar) to identify possibly yet undiscovered papers that are also relevant for our context. Using snowballing we eventually identified two more relevant papers leading to a final set of 34 primary studies.

D. Classification of Relevant Papers

We identified keywords (cf. Table III) to classify the papers to gain a better overview of all relevant papers. We iteratively developed the keywords by summarizing and grouping the main contributions of the relevant papers during screening. We did not rely on the primary studies' keywords as these often were too general or unspecific to derive a satisfying classification scheme.

TABLE III: Keywords for classification.

Keywords	Description
DSL creation	Papers proposing solutions for creating a DSL already considering their future evolution.
DSL grammar	Papers writing about the evolving grammar or syntax of a DSL.
Challenges in DSL evolution	Papers that identify or point out challenges that occur during evolution of a DSL.
Co-evolution of domain vs. DSL	Papers considering the problem of the tight coupling of a DSL to the domain which it was developed for (changes in domain lead to DSL evolution).
Tool for DSL evolution	Papers presenting and discussing a tool to support DSL evolution.
General approach for DSL evolution	Papers that propose solutions and approaches for DSL evolution that can be applied to more than one DSL. (cf. RQ1)
Practical example of DSL evolution	Papers containing reports and experiences of evolving a DSL in practice. (cf. RQ2)
References to general-purpose languages	Papers pointing out commonalities between general-purpose programming languages and their evolution and domain-specific languages (and their evolution). (cf. RQ3)

As recommended by Peterson et al. [6], in addition to the scheme described in Table III, we also used the paper type classification scheme proposed by Wieringa et al. [11].

E. Data Extraction

During the data extraction process we again screened all primary studies and assigned them to our classification terms (cf. Table III) and evaluated their research types [11]. Again the first two authors inspected and classified the papers. In case of disagreement authors discussed the proper classification. In addition to the classification of our primary studies, another result of this step was the identification of the primary studies that were relevant to answer our research questions. As we derived three of our classification keywords directly from our research questions, we can consider the papers classified to them as the sets of papers that are relevant for the corresponding research question.

III. Results

Our search for primary studies led to 98 papers from four different literature search engines. Fig. 2 shows both the number of papers found per year and the number of papers per year we selected as primary studies. Furthermore, we analyzed the publication fora of our primary studies to identify venues that are particularly relevant for DSL evolution. Table IV shows these publication venues. Most venues have only one single

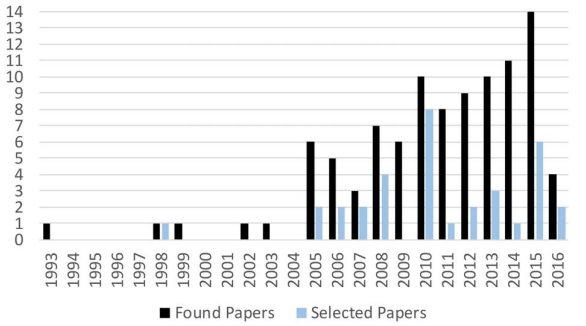

Fig. 2: Found/selected publications per year (2016 only includes papers published before June of that year).

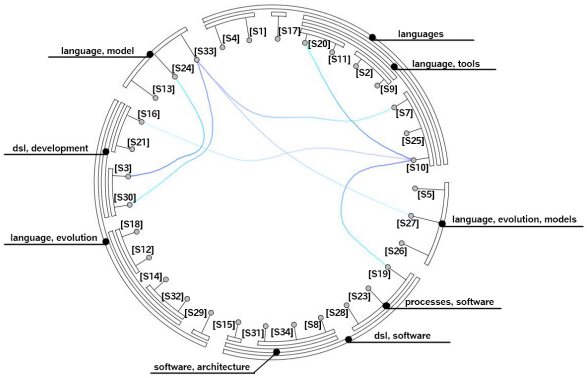

Fig. 3: ReVis citations and clustering visualization.

paper. Exceptions are SLE (3 papers), GTTSE (2 papers) and SCP (2 papers).

In a second step we used the ReVis [12] tool to visualize and analyze our primary studies regarding the common topics they address and regarding the frequency of citations among studies. ReVis supports the process of conducting a systematic literature review or mapping study. We used ReVis' capabilities for clustering publications and for reference visualization. Clustering is done in ReVis by analyzing the titles and abstracts of all primary studies. Clusters are calculated using keywords that appear across different papers. While the automatically identified clusters do not match perfectly with our manual classification, it is still close to what we developed.

Clusters are visualized by the arcs around the circle of publications (cf. Fig. 3). ReVis also visualizes the internal reference structure of the publications as edges between the publications. For the visualization of the references between our primary studies, we developed a script that analyzes our primary studies and prepares this information in an extended BibTex format that can be used as input for the ReVis tool.

We then produced a mapping of our classification terms to the corresponding primary studies as shown in Table V. Table VI shows the classification of the research types following Wieringa et al.'s definition [11]. To better analyze our classification of the papers, we also created a bubble chart representing the number of papers classified for each pair of classification terms and research types (cf. Fig. 4) in a more graphical way.

Fig. 5 depicts the distribution of primary studies across the three research questions. Specifically, 21 publications address RQ1, 14 address RQ2, and 4 address RQ3. The studies [S17] and [S21] are the only studies not part of this figure (nor of the Tables VII–IX below) as they are not related with a particular

TABLE IV: Publication fora.

Acronym	Primary Studies	Publication Name
Conferences		
CSD	[S22]	International Conference on Complex Systems Design
ECMFA	[S19]	European Conference on Modelling Foundations and Applications
ECOOP	[S21]	European Conference on Object-Oriented Technology
ECSAW	[S15]	European Conference on Software Architecture Workshops
ICSE	[S27]	International Conference on Software Engineering
MoDELS	[S23]	International Conference on Model Driven Engineering Languages and Systems
SLE	[S20] [S25] [S18]	International Conference on Software Language Engineering
WCRE	[S31]	Working Conference on Reverse Engineering
Journals		
AIR	[S26]	Journal of Artificial Intelligence Review
CSUR	[S33]	ACM Computing Surveys Journal
ENTCS	[S3]	Journal of Electronic Notes in Theoretical Computer Science
IET-SEN	[S28]	IET Software Journal
IST	[S4]	Journal of Information and Software Technology
JSS	[S1]	Journal of Systems and Software
PCS	[S2]	Journal of Procedia Computer Science
SCP	[S5] [S6]	Journal of Science of Computer Programming
SM	[S10]	Journal of Software Maintenance
Workshops		
DSM	[S11]	Workshop on Domain-Specific Modeling
IWPSE-EVOL	[S8]	Workshop on Software Evolution
LDTA	[S13]	Workshop on Language Descriptions, Tools, and Applications
MISE	[S30]	International Workshop on Modeling in Software Engineering
RAM-SE	[S12]	Workshop on Reflection, AOP and Meta-Data for Software Evolution
WRT	[S9]	Workshop on Refactoring Tools
Others		
AWP	[S34]	Book Publisher: Addison Wesley Pub
GTTSE	[S14] [S16]	International Summer School of Generative and Transformational Techniques in Software Engineering
IDS	[S17]	International Dagstuhl Seminar
OOPSLA	[S7]	Symposium on Object-oriented Programming Systems, Languages, and Applications
TASE	[S29]	Symposium on Theoretical Aspects of Software Engineering
SDL	[S24]	International SDL Forum on Model-Driven Dependability Engineering
SERVICES	[S32]	World Congress on Services

TABLE V: Publications per classification keyword.

Keywords	Primary Studies
DSL creation	[S3] [S6] [S8] [S11] [S12] [S16] [S19] [S20] [S29] [S30] [S32] [S33] [S34]
DSL grammar	[S1] [S6] [S8] [S11] [S12] [S13] [S14] [S21] [S24] [S26] [S27] [S28] [S30] [S32] [S33] [S34]
Challenges in DSL evolution	[S1] [S2] [S9] [S10] [S17] [S20] [S21] [S22] [S25] [S26] [S29]
Co-evolution of domain vs. DSL	[S8] [S18]
Tool for DSL evolution	[S3] [S5] [S11] [S12] [S19] [S29] [S30] [S31] [S32] [S34]
General approach for DSL evolution (RQ1)	[S1] [S2] [S3] [S5] [S6] [S9] [S11] [S12] [S13] [S18] [S19] [S20] [S22] [S23] [S24] [S29] [S30] [S31] [S32] [S33] [S34]
Practical example of DSL evolution (RQ2)	[S7] [S8] [S10] [S14] [S15] [S16] [S20] [S25] [S27] [S28] [S29] [S31] [S33] [S34]
References to general-purpose languages (RQ3)	[S4] [S6] [S25] [S26]

TABLE VI: Publications per research type [11].

Res. Type	Primary Studies
Validation research	[S5] [S18] [S22]
Evaluation research	[S23] [S31]
Solution proposal	[S2] [S3] [S6] [S9] [S11] [S12] [S13] [S15] [S19] [S20] [S24] [S28] [S29] [S30] [S32]
Philosophical paper	[S4] [S17] [S21] [S26]
Opinion paper	–
Experience paper	[S1] [S7] [S8] [S10] [S14] [S16] [S25] [S27] [S33] [S34]

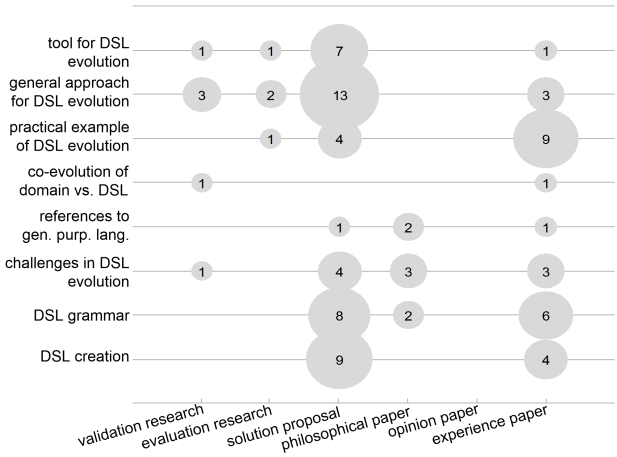

Fig. 4: Mapping of research types and classification terms.

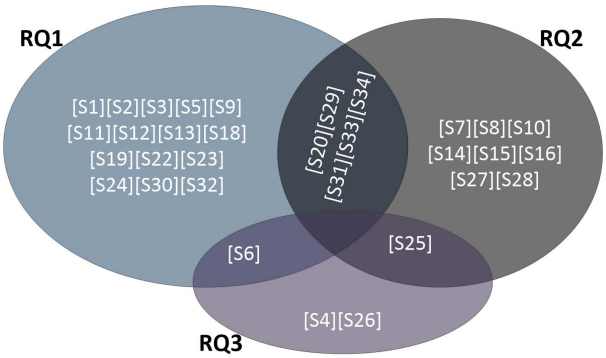

Fig. 5: Venn diagram of publications per research question.

research question. We have selected them because they discuss DSL evolution in general (cf. our fourth inclusion criterion). Table V shows which primary studies cover which keywords and research question(s).

IV. Discussion of Results

We briefly describe the main contribution of every primary study in Tables VII, VIII and IX.

Since 2005 we can observe an *increase of papers focusing on DSLs*, showing the growing relevance of this topic over the last years (cf. Fig. 2). Analyzing the publication venues we learned that the International Conference on Software Language Engineering (SLE) is the forum with the most publications regarding DSL evolution. As the three papers published in the proceedings of this conference do address all of our three research questions (cf. Tables IV and V), we can safely state that this conference can be considered as particularly relevant for DSL evolution. The papers published at the International Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE) address RQ1 and RQ2. Both papers published in the Journal of Science of Computer Programming (SCP) solely address RQ2. All other publications are from different venues, which shows that *publications focusing on DSL evolution are quite scattered across publication fora*.

Regarding the classification of the papers we observed that *papers are well spread over all classification terms, but cluster in terms of research types*. The most significant ones are solution proposals and experience papers. This might be caused by the still young topic of DSL evolution, i.e., in early phases many new solutions and corresponding experiences are typically published, while validation and evaluation of already existing solutions are less frequent. However, the fact that *only few evaluation research exists so far* is still an indicator that DSL evolution has not been thoroughly explored and more studies should be conducted.

Analyzing the number of references between our primary studies we observed that *references between publications, both within and between clusters are very sparse*. This further motivates our overall study, which we hope supports researchers in identifying relevant related work inside and outside their research community.

The number of general DSL evolution approaches is still quite limited. Nevertheless we could identify several ap-

TABLE VII: Results for RQ1 – What approaches for DSL evolution have been published?

Source	Contribution
[S1]	States that the usability of a DSL is crucial for its maintainability and adaptability. The authors present a framework to evaluate the usability of DSLs to support creating maintainable DSLs.
[S2]	Discusses the challenge of integrating several existing DSLs into one single DSL and presents an automated solution to migrate models of an old version of a DSL to its current version.
[S3]	Presents a tool to support evolutionary changes to a DSL by automating the adaptation of the DSL parser and the migration of existing words to the new DSL version.
[S5]	Discusses different scenarios in DSL evolution and presents a structured approach for automated DSL evolution.
[S6]	Describes the Rascal meta-programming language approach to develop and evolve DSLs.
[S9]	Discusses the problem of references between artifacts written in different (domain-specific) languages. Specifically, if an artifact is refactored, its language-external references have to be updated too. Therefore, the authors present an approach and a tool using cross-language linking.
[S11]	Presents a language workbench called Melange, which supports importing, assembling, and customizing pieces of existing DSLs like syntax or semantics to create a new DSL through reuse.
[S12]	Introduces a framework called Neverlang, which supports the user in creating new DSLs with a specific focus on reusability of the language definition to tackle the co-evolution problem of DSLs and their underlying domain.
[S13]	Uses language-generic techniques to preserve static name bindings and to detect name binding violations, following the goal of reducing the effort for refactoring DSLs.
[S18]	Proposes a solution for interpreting a DSL by referring to dynamically updated language configurations to cope with changing DSL contexts.
[S19]	Introduces an experimental approach to evaluate the evolutionary capabilities of a DSL by applying, classifying, and ranking changes according to their maintenance difficulty.
[S20]	Presents a domain-specific meta-modeling language to express commonalities and variations of a set of related or similar DSLs.
[S22]	Discusses and explores the challenges in designing and evolving DSLs. Proposes treating DSLs as software comprising different artifacts and providing support for adding, removing or modifying these artifacts.
[S23]	Introduces an approach to automate the co-evolution of meta-models and models in arbitrary software domains like DSLs.
[S24]	Focuses on refactoring of DSLs using model transformation to automate adaptation of language-dependent components.
[S29]	Discusses automating multi-level language evolution and proposes concepts for step-wise DSL development and evolution.
[S30]	Proposes an infrastructure for collaborative DSL development and evolution, involving both, technical and non-technical stakeholders.
[S31]	Introduces an approach to automate converting existing DSL models to the current version of the DSL after the language was changed.
[S32]	Uses a service-oriented architecture approach to define the semantics of a DSL, to analyze its syntax and to specify DSL programs to overcome the weaknesses of existing DSL implementation techniques like lacking interoperability capabilities and limited tool support.
[S33]	Presents design patterns for all different phases of DSL implementation and development.
[S34]	Describes the costs and benefits of developing a DSL and argues for sophisticated tool support, i.e., in Microsoft Visual Studio, especially to support DSL design and evolution.

TABLE VIII: Results for RQ2 – What do publications report about DSL evolution in practice?

Source	Contribution
[S7]	Reports experiences and lessons learned from evolving a DSL over several generations of a test framework.
[S8]	Describes influences that cause a DSL to evolve and reports on the iterative evolution of a DSL.
[S10]	Discusses the role of DSLs in software maintenance including challenges for DSL evolution based on experiences from industrial practice.
[S14]	Presents a case study on evolving the syntax, semantics and robustness of a real-world DSL.
[S15]	Describes the evolution of a DSL-based architecture abstraction approach.
[S16]	Reports a case study on the incremental development and evolution process of a DSL for Web applications.
[S20]	Introduces a domain-specific meta-modeling language to support iterative development of DSLs and discusses the challenges of this approach.
[S25]	Investigates several research questions on language evolution using case studies of four evolving modeling DSLs.
[S27]	Examines the evolution of a modeling DSL and its programming interface. Also presents lessons learned from cooperating with industry.
[S28]	Discusses the use of the semantic Web in the creation and evolution of a DSL for software processes.
[S29]	Explains why automated language evolution is necessary, discusses challenges and proposes a tool to support incremental co-evolution of languages and their program generation architecture.
[S31]	Presents an approach for generating version converters for DSLs to support DSL evolution. This approach is evaluated using three different case studies.
[S33]	Presents design patterns to support DSL developers in developing and maintaining a DSL.
[S34]	Describes how to develop DSLs with Microsoft Visual Studio including support for their design and evolution.

TABLE IX: Results for RQ3 – What approaches for general-purpose programming language evolution were adopted for DSL evolution?

Source	Contribution
[S4]	Presents a systematic mapping study on domain-specific languages including a short discussion on evolution in DSLs vs. general-purpose programming languages.
[S6]	Describes how modular language design can be applied to DSLs using the Rascal meta-programming language.
[S25]	Investigates several research questions on language evolution using case studies of four evolving modeling DSLs.
[S26]	Presents an overview of computational methods and grammars in language evolution and discusses current approaches in language evolution research.

proaches which are applicable not only for one particular DSL or domain. Table VII shows all primary studies related to RQ1.

Through our analysis of the primary studies we could identify three DSL evolution trends, i.e., seven approaches focus on *(i) automating DSL evolution* (often with a special focus on automating the migration of old DSL models to the current grammar) [S2], [S3], [S5], [S23], [S24], [S29], [S31], four approaches put their focus on *(ii) efficient DSL creation and maintainability* [S1], [S12], [S19], [S30], and two approaches emphasize the *(iii) reuse of (parts of) existing DSLs* [S11], [S20]. Other studies present tools or approaches covering different parts of the DSL evolution process.

Regarding RQ2 – practical experiences – we consider the number and the diversity of publications as quite good. However, we still miss more lessons learned from reports on practical DSL evolution, particularly *lessons that are applicable outside the particular application context*. Furthermore, we would expect *papers summarizing and discussing the existing*

reports on practical DSL evolution as quite beneficial to get a better overview about DSL evolution in practice.

Furthermore, as we only found four publications regarding RQ3, *we would deem additional work on the commonalities between general-purpose language evolution and DSL evolution as very promising*, since we suppose that there are additional approaches for evolving general-purpose programming languages that could also be used for DSL evolution. Most publications on this topic did not examine DSL evolution in more detail.

As there is rarely any work on co-evolution of domains and DSLs contained in our primary studies, we expect that researchers interested in this topic could benefit from taking a look at the general work on co-evolution of meta-models and models such as the work by Khelladi et al. [13]. We would expect work that examines the commonalities and differences between these two fields to be quite interesting.

Follow-up (systematic) studies could continue our work and investigate in more detail, for example, what general software evolution techniques – e.g., iterative development, automated testing, refactoring – are applicable to DSL evolution.

V. Threats to Validity

A threat to the validity of our study, like in most SMSs, is the selection of the used digital libraries and the search terms. We addressed this threat by using four of the most important digital libraries in computer science that allowed us to define accurate search terms by providing a versatile set of possible search query constructs. In addition, we iteratively refined our search terms to ensure a sufficient quality of the search results. To avoid inappropriate inclusion and exclusion of papers, we omitted unnecessary complexity in the inclusion and exclusion criteria and compensated this with a thorough screening of the papers to decide on inclusion or exclusion as well as through discussion among multiple authors.

Another threat to validity of the chosen set of papers is that we might have missed papers actually containing reports about DSL evolution in practice (RQ2), which, however, do neither mention this in their title nor abstract. We tried to mitigate this threat using snowballing but can of course never know we have really found all relevant papers.

The most general threat to the validity of our results is of course that many practical experiences with DSLs and their evolution do not get published in scientific papers, either because they are made by practitioners, who do not write scientific papers, or simply because researchers do not write them up. While analyzing grey literature (such as blog posts and online articles) would help with this threat, this requires a different research approach than a systematic literature study.

VI. Conclusion

In this paper we presented the results of a systematic mapping study on DSL evolution. We identified 34 papers addressing this topic and answered three research questions regarding general approaches on DSL evolution, reports of practical DSL evolution, and general-purpose programming language evolution. We demonstrated that DSL evolution is a topic of increasing relevancy and gave an overview of the different main contributions of our primary studies. During screening and analyzing of the found publications we came up with a set of promising research fields and also could identify trends, such as automated language evolution and reuse of (parts of) DSLs. Our analyses also revealed that references between publications are very sparse. We hope our study supports researchers in identifying relevant related work inside and outside their research community and practitioners in getting an overview of the research field. In future work we aim to build on the results of this SMS and report our own experiences on a DSL we have evolved.

Acknowledgments

This work has been supported by the Christian Doppler Forschungsgesellschaft Austria and Primetals Technologies. We want to thank Roberto Lopez-Herrejon for feedback on earlier versions of this paper.

References

[1] A. Van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography." *Sigplan Notices*, vol. 35, no. 6, pp. 26–36, 2000.

[2] M. Vierhauser, R. Rabiser, P. Grünbacher, and A. Egyed, "Developing a DSL-based approach for event-based monitoring of systems of systems: Experiences and lessons learned," in *Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering*. ACM, 2015, pp. 715–725.

[3] R. Rabiser, M. Vierhauser, and P. Grünbacher, "Assessing the usefulness of a requirements monitoring tool: A study involving industrial software engineers," in *38th International Conference on Software Engineering (ICSE 2016), ICSE Companion*. Austin, TX, USA: ACM, 2016, pp. 122–131.

[4] N. Oliveira, M. J. Pereira, P. Henriques, and D. Cruz, "Domain specific languages: A theoretical survey," *INForum'09-Simpósio de Informática*, 2009.

[5] T. Kosar, P. E. Martı, P. A. Barrientos, M. Mernik *et al.*, "A preliminary study on various implementation approaches of domain-specific language," *Information and Software Technology*, vol. 50, no. 5, pp. 390–405, 2008.

[6] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proc. of the 12th Int'l Conf. on Evaluation and Assessment in Software Engineering*. Keele University, 2008, pp. 1–10.

[7] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

[8] B. A. Kitchenham, "Guidelines for performing systematic literature reviews in software engineering," EBSE Technical Report EBSE-2007-01, Keele University, Durham, UK, Tech. Rep., 2007.

[9] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Information and Software Technology*, vol. 55, no. 12, pp. 2049–2075, 2013.

[10] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, "Using mapping studies in software engineering," in *Proc. of PPIG*, vol. 8. Lancaster University, 2008, pp. 195–204.

[11] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requirements Engineering*, vol. 11, no. 1, pp. 102–107, 2006.

[12] K. R. Felizardo, N. Salleh, R. M. Martins, E. Mendes, S. G. MacDonell, and J. C. Maldonado, "Using visual text mining to support the study selection activity in systematic literature reviews," in *Proc. of the Int'l Symp. on Empirical Software Engineering and Measurement*. IEEE, 2011, pp. 77–86.

[13] D. E. Khelladi, R. Hebig, R. Bendraou, J. Robin, and M.-P. Gervais, "Detecting complex changes and refactorings during (meta)model evolution," *Information Systems*, vol. 62, pp. 220–241, 2016.

APPENDIX: PRIMARY STUDIES

[S1] Diego Albuquerque, Bruno Cafeo, Alessandro Garcia, Simone Barbosa, Silvia Abrahão, and António Ribeiro. Quantifying usability of domain-specific languages: An empirical study on software maintenance. *Journal of Systems and Software*, 101:245–259, 2015.

[S2] Nikolay Nikolov, Alessandro Rossini, and Kyriakos Kritikos. Integration of DSLs and Migration of Models: A Case Study in the Cloud Computing Domain. *Procedia Computer Science*, 68:53–66, 2015. 1st International Conference on Cloud Forward: From Distributed to Complete Computing.

[S3] Elmar Juergens and Markus Pizka. The Language Evolver Lever — Tool Demonstration. *Electronic Notes in Theoretical Computer Science*, 164(2):55–60, 2006.

[S4] Tomaž Kosar, Sudev Bohra, and Marjan Mernik. Domain-Specific Languages: A Systematic Mapping Study. *Information and Software Technology*, 71:77–91, 2016.

[S5] Bart Meyers and Hans Vangheluwe. A framework for evolution of modelling languages. *Science of Computer Programming*, 76(12):1223 – 1246, 2011. Special Issue on Software Evolution, Adaptability and Variability.

[S6] Bas Basten, Jeroen van den Bos, Mark Hills, Paul Klint, Arnold Lankamp, Bert Lisser, Atze van der Ploeg, Tijs van der Storm, and Jurgen Vinju. Modular language implementation in Rascal – experience report. *Science of Computer Programming*, 114:7–19, 2015.

[S7] Steve Freeman and Nat Pryce. Evolving an Embedded Domain-specific Language in Java. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, pages 855–865. ACM, 2006.

[S8] Marcel van Amstel, Mark van den Brand, and Luc Engelen. An Exercise in Iterative Domain-specific Language Design. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pages 48–57. ACM, 2010.

[S9] Philip Mayer and Andreas Schroeder. Towards Automated Cross-language Refactorings Between Java and DSLs Used by Java Frameworks. In *Proceedings of the 2013 ACM Workshop on Refactoring Tools*, WRT '13, pages 5–8. ACM, 2013.

[S10] Arie van Deursen and Paul Klint. Little Languages: Little Maintenance. *Journal of Software Maintenance*, 10(2):75–92, March 1998.

[S11] Thomas Degueule, Benoit Combemale, Arnaud Blouin, and Olivier Barais. Reusing Legacy DSLs with Melange. In *Proceedings of the Workshop on Domain-Specific Modeling*, pages 45–46. ACM, 2015.

[S12] Walter Cazzola and Davide Poletti. DSL Evolution Through Composition. In *Proceedings of the 7th Workshop on Reflection, AOP and Meta-Data for Software Evolution*, pages 6:1–6:6. ACM, 2010.

[S13] Maartje de Jonge and Eelco Visser. A Language Generic Solution for Name Binding Preservation in Refactorings. In *Proceedings of the Twelfth Workshop on Language Descriptions, Tools, and Applications*, pages 2:1–2:8. ACM, 2012.

[S14] Laurence Tratt. Evolving a DSL implementation. In *Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7, 2007. Revised Papers*, pages 425–441. Springer, 2008.

[S15] Srdjan Stevanetic, Thomas Haitzer, and Uwe Zdun. Supporting Software Evolution by Integrating DSL-based Architectural Abstraction and Understandability Related Metrics. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, pages 19:1–19:8. ACM, 2014.

[S16] Eelco Visser. WebDSL: A Case Study in Domain-Specific Language Engineering. In *Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7, 2007. Revised Papers*, pages 291–373. Springer, 2008.

[S17] Barrett Bryant, Jean-Marc Jézéquel, Ralf Lämmel, Marjan Mernik, Martin Schindler, Friedrich Steinmann, Juha-Pekka Tolvanen, Antonio Vallecillo, and Markus Völter. Globalized domain specific language engineering. In *Globalizing Domain-Specific Languages: International Dagstuhl Seminar, Dagstuhl Castle, Germany, October 5-10, 2014, Revised Papers*, pages 43–69. Springer, 2015.

[S18] Paul Laird and Stephen Barrett. Towards dynamic evolution of domain specific languages. In *Proceedings of the Second International Conference on Software Language Engineering*, pages 144–153. Springer, 2009.

[S19] Jeroen van den Bos and Tijs van der Storm. A Case Study in Evidence-Based DSL Evolution. In *Proceedings of the 9th European Conference on Modelling Foundations and Applications*, pages 207–219. Springer, 2013.

[S20] Steffen Zschaler, Dimitrios S. Kolovos, Nikolaos Drivalos, Richard F. Paige, and Awais Rashid. Domain-Specific Metamodelling Languages for Software Language Engineering. In *Proceedings of the Second International Conference on Software Language Engineering*, pages 334–353. Springer, 2009.

[S21] Thomas Cleenewerck, Krzysztof Czarnecki, Jörg Striegnitz, and Markus Völter. Evolution and Reuse of Language Specifications for DSLs (ERLS). In Jacques Malenfant and Bjarte M. Østvold, editors, *Object-Oriented Technology. ECOOP 2004 Workshop Reader: ECOOP 2004 Workshops, Oslo, Norway, June 14-18, 2004, Final Reports*, pages 187–201. Springer, 2005.

[S22] Jean-Marc Jézéquel, David Méndez-Acuña, Thomas Degueule, Benoit Combemale, and Olivier Barais. When Systems Engineering Meets Software Language Engineering. In *Proceedings of the Fifth International Conference on Complex Systems Design & Management*, pages 1–13. Springer International Publishing, 2014.

[S23] Sander Vermolen and Eelco Visser. Heterogeneous Coupled Evolution of Software Languages. In *Proceedings of the 11th International Conference Model Driven Engineering Languages and Systems*, pages 630–644. Springer, 2008.

[S24] Martin Schmidt, Arif Wider, Markus Scheidgen, Joachim Fischer, and Sebastian von Klinski. Refactorings in Language Development with Asymmetric Bidirectional Model Transformations. In *Proceedings of the 16th International SDL Forum on Model-Driven Dependability Engineering*, pages 222–238. Springer, 2013.

[S25] Markus Herrmannsdoerfer, Daniel Ratiu, and Guido Wachsmuth. Language Evolution in Practice: The History of GMF. In *Proceedings of the Second International Conference on Software Language Engineering*, pages 3–22. Springer, 2009.

[S26] Patrizia Grifoni, Arianna D'Ulizia, and Fernando Ferri. Computational methods and grammars in language evolution: a survey. *Artificial Intelligence Review*, 45(3):369–403, 2016.

[S27] Thomas Aschauer, Gerd Dauenhauer, and Wolfgang Pree. A modeling language's evolution driven by tight interaction between academia and industry. In *Proceedings of the 2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 2, pages 49–58, 2010.

[S28] Ricardo Martinho, Joao Varajao, and Dulce Domingos. Using the semantic web to define a language for modelling controlled flexibility in software processes. *IET Software*, 4(6):396–406, 2010.

[S29] Markus Pizka and Elmar Jurgens. Automating Language Evolution. In *Proceedings of the First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering*, pages 305–315, 2007.

[S30] Javier Luis Canovas Izquierdo and Jordi Cabot. Community-driven language development. In *Proceedings of the 2012 4th International Workshop on Modeling in Software Engineering*, pages 29–35, 2012.

[S31] Gerardo de Geest, Sander Vermolen, Arie van Deursen, and Eelco Visser. Generating Version Convertors for Domain-Specific Languages. In *Proceedings of the 2008 15th Working Conference on Reverse Engineering*, pages 197–201, 2008.

[S32] Shih-Hsi Liu, Adam Cardenas, Xang Xiong, Marjan Mernik, Barrett R. Bryant, and Jeff Gray. A SOA Approach for Domain-Specific Language Implementation. In *Proceedings of the 2010 6th World Congress on Services*, pages 535–542, 2010.

[S33] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-specific Languages. *ACM Comput. Surv.*, 37(4):316–344, December 2005.

[S34] Steve Cook, Gareth Jones, Stuart Kent, and Alan Wills. *Domain-specific Development with Visual Studio DSL Tools*. Addison-Wesley Professional, first edition, 2007.