A Systematic Mapping Study on Domain-Specific Languages

Leandro Marques do Nascimento[1,2], Daniel Leite Viana[1], Paulo A. M. Silveira Neto[1,3], Dhiego A. O. Martins[1], Vinicius Cardoso Garcia[1], Silvio R. L. Meira[1]

[1] Informatics Center, Federal University of Pernambuco (UFPE)
Recife, Brazil

[2] Department of Informatics, Federal Rural University of Pernambuco (UFRPE)
Recife, Brazil

[3] Department of Informatics, Federal Rural University of Pernambuco (UFRPE)
Serra Talhada, Brazil

{lmn2, dlv2, pamsn, daom, vcg, srlm}@cin.ufpe.br

***Abstract*—Domain-Specific Languages (DSLs) offer substantial gains in expressiveness and ease of use compared with general purpose languages. This way, DSLs have gained significant attention in industry and academy, as can be seen by the increased number of related publications in key conferences and journals. This paper aims to provide a broad view of the DSL research field by performing a Systematic Mapping Study. Adopting a detailed search strategy, 4450 studies were initially identified, and, after filtering, 1440 primary studies were selected and categorized using a particular classification scheme. So, this work presents the most popular application domains where DSLs have been applied, identifies different tools for handling DSLs, including language workbenches, and enumerates several techniques, methods and/or processes for dealing with DSLs.**

Keywords: Domain-specific languages; systematic mapping study; programming languages; mini languages; little languages.

I. INTRODUCTION

Software systems are built upon computer languages or better called programming languages. A programming language is a notation for expressing computations (algorithms) in both machine and human readable form. Appropriate programming languages and tools may drastically reduce the cost of building new applications as well as maintaining existing ones [1]. For humans, it would be easier to write computer programs if a natural language could be used, such as English or Portuguese, for instance. However, computer languages must follow a rigid predefined structure, with a specific grammar and syntax, and to learn this structure is not so easy for many people, taking significant time for someone to be "fluent" in that kind of language.

In the context of programming languages, a Domain-Specific Language (DSL) is a language that provides constructs and notations tailored toward a particular application domain [2]. Usually, DSLs are small, more declarative than imperative, and more attractive than General-Purpose Languages (GPL) for their particular application domain due to easier program understanding, reduced semantic distance between the problem and the program, and enhanced productivity. Some well-known examples of DSLs are BNF (syntax definition), HTML (hypertext markup), SQL (database queries), and VHDL (hardware design).

DSLs trade generality for expressiveness in a limited domain, and this can bring several benefits to software engineering. However, these benefits do not come for free. The cost of DSL design, development and maintenance has to be taken into account. Without appropriate methodologies and/or tools these costs can be higher than savings. Although DSLs have been developed from the beginning of computer science (an early example is APT, a DSL for numerical control of machine tools developed back in the 1950s at MIT [3]), many unanswered questions remain regarding when and how to develop a DSL.

Therefore, this paper presents a systematic mapping study in order to better understand the DSL research field, through synthesizing evidence to suggest important implications for practice, as well as identifying research trends, open issues, and areas for improvement. A Mapping Study (henceforth abbreviated to 'MS') [4] is an evidence-based approach, applied in order to provide an overview of a research area, and to identify the quantity and type of research and results available within it. Hence, the goal of this investigation is to identify, evaluate, and synthesize state-of-the-art domain-specific programming practices in gathering evidence of what has been achieved so far in this discipline. We are also interested in cataloging which are the domains that have taken advantage of using DSLs. This way, researchers and/or practitioners may know which DSLs have been applied to a particular domain and then reuse or adapt it for any other specific needs. This systematic mapping process was conducted from November, 2011 to April, 2012.

The remainder of this paper is organized as follows: Section 2 presents the related work. In Section 3, the research methodology used in this paper is described including the research questions, the search strategy and the classification scheme. Section 4 reports the main findings. In Section 5, the threats to validity are shown, and at last, Section 6 draws some conclusions and provides recommendations for further research on this topic.

II. RELATED WORK

The literature on DSLs provides a large number of studies, regarding both general and specific issues, as will be discussed later in this paper. However, a general search for (*"mapping study"* OR *"systematic literature review"*) AND *"domain-specific languages"* in well-known search engines have shown that no publication have tried to address the issues of this research field using specifically the MS approach. Actually, many papers presented the state-of-the-art in this field using other approaches than a MS and they are next described as related work.

One of the first published papers to coin the concept of a DSL is from 1965 [5]. It presents a family of unimplemented computing languages that is intended to span differences of a given application area by a unified framework.

In the 1980s, Bentley [6] tried to summarize the concept of the so called *Little Languages*. The paper describes examples of small languages that could be developed with the technology available back there, e.g. COBOL and FORTRAN.

In a paper from 2000, Deursen et al. [7] list a selection of 75 key publications in the area. It discusses terminology, risks and benefits, examples of domain-specific languages, design methodologies, and implementation techniques.

In a more recent work from 2005, Mernik et al. [2] try to answer the question "*When and How to Develop Domain-Specific Languages?*". The paper brings a list of DSLs developed until then for different domains. The work identified five DSL development phases: decision, analysis, design, implementation, and deployment, and then relates the listed DSLs with their development phases. At last, the work enumerates domain analysis tools and language development systems, giving a full view of the open issues in the area.

One of the most recent related work that could be identified is [8], from 2011. It compares four different approaches for DSL implementation: ANTLR, Ruby, Stratego and Converge. From their comparative study, it was observed that each approach has its merits and demerits and there is no single approach that would apply to all scenarios. The work does not mention directly the use of language workbenches.

Indeed, we believe our study states current and relevant information on research topics that can complement others previously published. By current, we mean that, as the number of studies published has increased rapidly, as shown in Figure 2, it justifies the need of more up to date empirical research in this area to contribute to the community investigations. Moreover, applying a MS approach to map out the research area of DSL gives us a full overview of what is being done and what is lacking attention from academia/ industry, as well as allow future extensions and replications.

III. RESEARCH METHODOLOGY

The experimental software engineering community is working towards the definition of a standard processes for conducting literature reviews. There are mainly two different approaches to be cited: Systematic Literature Reviews (SR) and Systematic Mapping Studies (MS) [9]. While a SR is a mean of identifying, evaluating, interpreting and comparing all available research relevant to a particular question [9], a MS intends to "map out" the research undertaken rather than to answer detailed research questions [4]. A MS comprises the analysis of primary studies that investigate aspects related to predefined research questions, aiming at integrating and synthesizing evidence to support or refute particular research hypotheses.

In this study, we merged ideas from Petersen et al. [4] with some good practices defined in the guidelines proposed by Kitchenham and Charters [9], such as the protocol definition. Therefore, we could apply a process for a mapping study, including best practices for conducting systematic reviews, making the best use of both techniques.

A MS is basically performed in three phases. All phases are detailed in following sections: *1)* Definition of the protocol, which comprises the research questions and the search strategy. This phase is commonly used in systematic reviews. *2)* Conducting the study with screening of relevant papers. During this phase, a classification scheme is used. *3)* Keywording relevant topics, data extraction and systematic mapping.

A. Research Questions

This mapping study intends to identify relevant publications about Domain-Specific Languages, understanding how they can be created and which ones have been created so far. In addition, this study tries to enumerate the domains in which DSLs have been applied, which knowledge is necessary from the domain experts to start using the language, and so forth which are the open issues of the whole research field.

In summary the main research question of this study is: ***In which manner are Domain Specific Languages (DSLs) being created, used and maintained?***

B. Research Sub-questions

Moreover, in order to make the mapping study main objective more clear and repeatable, some research sub-questions are defined, as following:

- **Q1.** *Which techniques, methods and/or processes are used while working with DSLs, i.e. creation, application, evolution and extension of DSLs?*
- **Q2.** *Which DSLs have been created and are available for use or are described in some type of publication?*
- **Q3.** *In which domains are these DSLs being used?*
- **Q4.** *Which tools are used for the development and usage of DSLs and how such tools support those activities?*

C. Search Strategy, Data Sources and Studies Selection

According to our research questions and in order to increase the coverage of our search, we decided to use the following search string, which brings only general terms grouped by an **OR** clause:

"*domain-specific language*" **OR** "*domain-specific modeling language*" **OR** "*generative programming*"

Therefore, instead of restricting our search items with other keywords, we understand that any work that mentions one of the three items listed is going to be returned by the search engine anyway. Although the number of manuscripts returned could increase considerably, few or even no relevant studies would be left over. Indeed, experts in the

DSL research field may say there are other related terms, such as "*little/small language*" or "*Architecture Description Language*" (ADL). Despite of including those terms in our automatic search, we decided to look for those terms in the manual search and snow-balling process (which follows up the reference list of each selected manuscript), since papers that have those terms and do not have the term "*domain-specific language*" are quite rare and can be easily found during a fine-grained and non-automatic search process.

We ended up adding "*domain-specific modeling language*" and "*generative programming*" because we noticed that these terms are extremely related to the research field just by checking at the most relevant papers according to the search engines relevance ordering.

The study was conducted using automatic and manual search. We did not establish any inferior year-limit. For automatic search, six search engines and digital databases of scientific sources were used: *ACM Digital Library*, *IEEEXplore*, *SpringerLink*, *Science Direct*, *Scopus* and *Engineering Village* (also known as *El Compendex*). Besides, the manual search includes the most important international, peer-reviewed journals published by Elsevier, IEEE, ACM and Springer, and 26 different conferences.

After performing the automatic and manual search, a total of 4450 papers were identified, 93 of them from manual search. During manual search, a snow-balling process was done. Next, the studies were submitted to the inclusion and exclusion criteria, as we detail in the following section.

The studies selection involved a screening process composed of three filters, in order to select the most suitable results, since the likelihood of retrieving not adequate studies might be high. Figure 1 details each filter.

Regarding the **inclusion criteria**, the studies were submitted to the following conditions:

- Books, papers, technical reports and 'grey literature' regarding Domain Specific Languages, Domain Specific Modeling Languages and/or Generative Programming. No date filtering was applied.
- While verifying if a given article may be included in our study, we can check if it is possible to answer 'yes' for at least one of the following questions:
  - Is it a DSL or DSML?
  - Is it a technique, method or process for handling DSLs/DSMLs?
  - Is it a tool (language workbench) for handling DSLs/DSMLs?
  - Is it any type of philosophical paper that discusses concepts of DSLs, DSMLs and/or any related generative programming technique?

Considering the **exclusion criteria**, the studies were submitted to the following conditions:

- Articles not written in English.
- Literature that was only available in the form of abstracts or Powerpoint presentations. Posters, short papers (less than 2 pages) and invited conference talks with no relevant results can be excluded.
- Articles in press, journals and conferences editorials/reviews can also be excluded.
- Duplicated and/or incomplete studies.

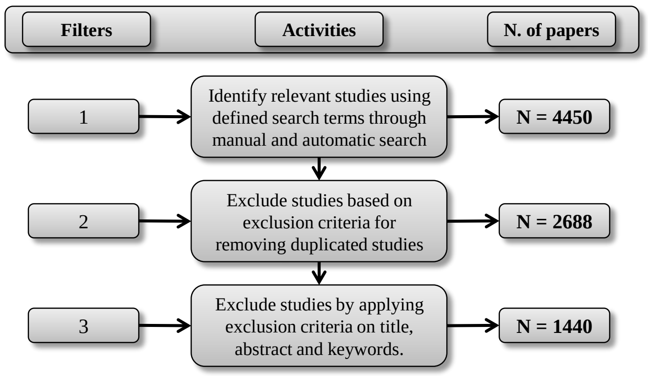
Filters
Activities
N. of papers
1
Identify relevant studies using defined search terms through manual and automatic search
N = 4450
2
Exclude studies based on exclusion criteria for removing duplicated studies
N = 2688
3
Exclude studies by applying exclusion criteria on title, abstract and keywords.
N = 1440

Figure 1. Stages of the selection process and the corresponding number of papers.

After performing the selection process, some results can be seen in Figure 2 which shows the distribution of the primary studies, considering the publication year. The Figure 2 clearly gives us the impression that many correlated areas in software engineering and computer science in general are taking more and more advantage of DSLs in practice, as we can check by looking at the growth curve.

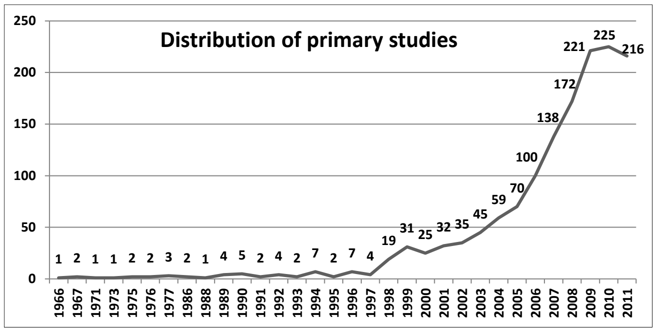

Figure 2. Distribution of studies by their publication years after 3rd filter.

We were able to identify the most common locals of publication. Conferences such ICSE (*International Conference on Software Engineering*), OOPSLA (*Object-Oriented Programming, Systems, Languages & Applications Conference*) and GPCE (*Generative Programming and Component Engineering Conference*) had the highest number of studies published. Similarly, the most popular journals were ACM SIGPLAN Notices, IEEE Software and ENTCS (Electronic Notes in Theoretical Computer). We catalogued manuscripts from 548 different sources (418 conferences and 130 journals).

D. *Classifying Selected Studies*

Our classification scheme assembled three facets. Facet one lists the classes of research based on [4]: *Validation Research, Evaluation Research, Solution Proposal, Philosophical Papers, Opinion Papers, Experience Papers*. Details of each class of research can be found on [4]. The two others are directly related to our research questions.

Facet 2 – DSL Research Type – considered in our study is directly related to the research sub-questions ***Q1***, ***Q2*** and ***Q4***. We tried to identify studies that report specifically the usage of a given DSL to solve a problem and also studies that report any kind of technique, method or process to handle DSLs, i.e., create, evolve, integrate, debug. What is more, we tried to enumerate what tools have been used to apply those techniques, methods and/or processes. TABLE I presents the details of Facet 2. Some concepts of this facet are based on [10].

TABLE I. FACET 2 – DSL RESEARCH TYPE.

1. ADL	Architecture Description Languages (ADLs) are aimed at the specification of high level system architectures, described in terms of components and connectors.
2. DSAL	A Domain-Specific Aspect Language combines benefits from DSLs and Aspect-Oriented Programming (AOP). It is a aspect language tailored to a specific domain.
3. DSML	A domain-specific modeling language is a special type of DSL that can be used for modeling domain-specific systems. The concept of a DSML comes originally from the adaptation of UML to specific domains.
4. External DSL	A completely separate language, for which you write a full parser, usually using a parser generator.
5. Internal DSL	An internal (or embedded) DSL is an idiomatic way of using a general-purpose language.
6. Method or Process	Any type of generic solution for a class of problems which usually involves technical and non-technical aspects. A method/process involves a set of steps to be performed in order to make it repeatable for anyone to try using it. A method/process may use a group of techniques which combined represent a generic solution for a class of problems.
7. Technique	Any type of solution for a specific problem. For example: a technique to generate Java code based on C# input; a technique for teaching how to create parsers, a technique to analyze model coupling.
8. Tools	Any type of software engineering tool used for handling DSLs.

Facet 3 addresses the domains in which DSL techniques are somehow applied and is directly related to ***Q3***. Inspired by previous publications that tried to do the same [2], [7], we identified many different domains ranging from bioinformatics to robotics and control systems, for example. We were able to enumerate 30 different domains. Among them, we selected the top 15 most referenced domains to be used as facet in this study. Since the final number of papers included in our study was quite large (1440), some domains were mentioned few times (1 or 2), then those ones are not considered to our classification. TABLE II displays Facet 3.

It is important to notice that none of the three facets are exclusive, it means, a paper may be classified in two or more categories of any of the three facets. For example, a paper may be categorized as a Solution Proposal and Validation Research, as a DSML and a Tool and also with the domains of Web and Control Systems.

TABLE II. FACET 3 – DOMAINS

1. Web	Every study that uses any type of web technology
2. Embedded Systems	Hardware and software co-design
3. Low-level Software	Low-level programming, for instance, operating systems, device drivers, etc.
4. Control Systems	Any type of control systems, for example: flight control, automation systems, etc.
5. Parallel Computing	High-performance computing, multithreaded programming
6. Simulation	Any type of simulation software
7. Data Intensive Apps	Studies that present ways of handling databases using DSL techniques
8. Real-time Systems	Systems where the time is a crucial variable
9. Security	Studies that handle security issues such as intrusion detection, access control, etc.
10. Dynamic Systems	A type of software system that can adapt to the context it is immersed
11. Visual Language	Apart from textual languages, this type of study describes a DSL with visual appealing
12. Testing	DSLs applied to the software engineering discipline of testing
13. Education	Any type of publication that mentions education as the primary goal, e.g. as in [11]
14. Network	DSLs for manipulating computer networks and/or distributed systems issues
15. Others	In this category, we gathered domains with at most 5 publications, covering several divergent topics, such as Chemistry, Geometry and Engineering, among others

In addition, it is important to highlight that TABLE II is missing some important domains due the total amount of manuscripts included in this study. Hereby, we cite one sample publication of these domains that were left over: healthcare [12], pervasive computing [13], graphics [14], cloud/grid computing [15], robotics [16], ontology [17], games [18], multi-agent systems [19], requirements engineering [20], bioinformatics [21], mobile apps [22], multimedia [23], user interface [24], hardware description [25], automation [26].

IV. MAIN FINDINGS

In this section, each topic presents the findings of a research sub-question, highlighting evidences gathered from the data extraction process. These results populate the classification scheme, which evolves while doing the data extraction. It is important to mention that this study is not going to enumerate all the references we found, as it makes no sense at all to list 1440 references. Instead, we are going to choose sample references to demonstrate our results.

Our first results are shown in Figure 3, which presents the distribution of papers according to Facet 1 – Classes of research. As can be seen, there is a majority number of Solution Proposals, which indicates that there are many proposals yet to be validated. The number of Validation and Evaluation Research together represents about one third of those proposals, which means that a representative number of proposals are somehow tested in industry and/or academy.

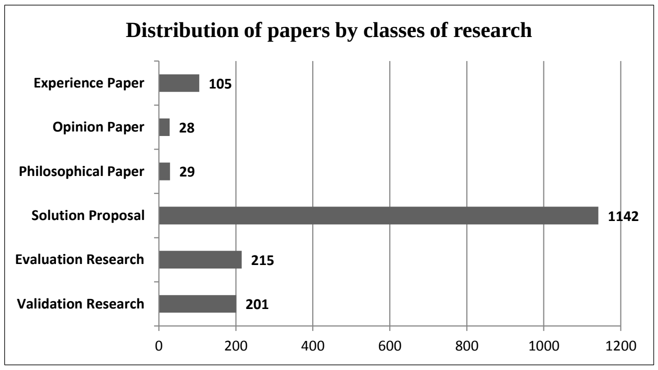

Figure 3. Distribution of papers by classes of research.

A. *Techniques, Methods and/or Processes for Handling DSLs*

Several techniques, methods and/or processes could be found during the execution of this mapping study. Methods for software construction using generative techniques are not new as we can see in [27], although they did not directly mention the construction of DSLs.

At an abstract level, a language is a means of communication; in the case of computing that communication is generally between a human and a machine. In order to be usable, a language needs to have a way that participants can share communications (syntax) and an agreed shared meaning (semantics). Languages may form parts of larger languages (e.g. the sub-part of English used only in computing could be detached and reattached to the main language); they may be parameterisable (e.g. American and British English can be seen as variations on the single, abstract, language English); they may have variable syntaxes (e.g. Serbian is written in both the Cyrillic and Latin alphabets); and so on [28].

One of the processes for handling DSL catalogued by this mapping study is called *Language Factories* [28]. Language Factories break languages down into components, including the following parts:

- **Abstract syntax**: The single definition of its Abstract Syntax Tree (AST).
- **Concrete syntax(es) and syntactic mapping**: A definition of its concrete syntax(es) specified as e.g. a context free grammar, and a mapping from that concrete syntax to the abstract syntax.
- **Semantic aspect(s)**: Each semantic aspect defines (a possibly incomplete part of the) semantics. Semantic aspects may overlap with each other (e.g. an operational and denotational semantics) or describe completely different elements of the semantics (e.g. semantics of language types and semantics for text editors supporting tool-tips).
- **Constraints**: Describes constraints on how the language can be composed with others (both in terms of what the component provides, and what it requires of other components).

These parts of language development could help us in citing the findings of this study. The development of formal DSLs contains concepts of metamodels or grammars (syntax) [29], [30], context conditions (static analysis and quality assurance) as well as possibilities to define the semantics of a language [31]. Many references highlight techniques directly related to compiler construction [11], [32], [33]. Along with the concept of DSL, we catalogued some publications describing DSMLs and its peculiarities [34]. Over the last few decades, DSLs have proven efficient for mastering the complexities of software development projects. The natural adaptation of DSLs to the model-driven technologies has in turn established domain-specific modeling languages (DSMLs) as vital tools for enhancing design productivity.

A widespread approach to the design of a DSML is to make use of the so-called profile mechanisms and to reuse the UML metamodel as the base language. By extending UML elements with stereotypes and their attributes, it is possible to define new concepts to better represent elements of a domain. Despite the ever increasing number of profiles defined and successfully applied in many applications.

The technique of UML profile is mentioned in 21 publications of our catalogue, as for example, [34–36]. We noticed that many of those techniques are well supported by tools, as we exemplify in the corresponding section.

We found quite a large number of techniques, methods and/or processes as can be seen in Figure 4. These are some examples of techniques for creating new DSLs: [37–39]. A total number of 160 publications mention some topic related to DSL creation, 69 other publications mention DSML creation and 53 mention embedded DSL creation.

Among different methods/processes for creating [40], implementing [41] and evolving [42], [43] a DSL, one of the methods that caught attention was the one that mentions directly the concept of *Language-Oriented Programming* [44] or even DSL oriented software engineering. The authors' fundamental principle is promoting the use of the right domain specific tool for each problem, instead of some universal tool coupled with a way of working that tries to wrap it so that it becomes usable in various contexts. The primary meta-tool promoted in [44] is usage of high level, strictly domain specific languages, based on formal concepts used and widely understood by domain experts who may have limited or no software engineering knowledge. This concept of language-oriented programming is fully aligned with other similar concept called *Language Factories* [28], already mentioned.

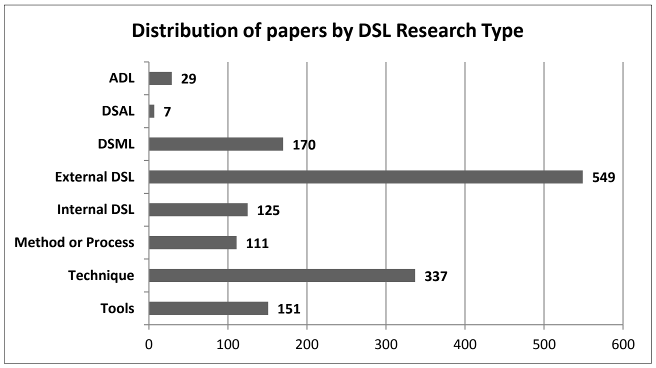

Figure 4. Distribution of papers by DSL research type.

Moreover, other relevant aspect identified in this study involves DSL integration/composition, as shown in [45–47]. Development of and tooling for a single DSL is well-studied, but surprisingly little is known about the interplay of different DSLs in a single system. Multiple DSLs are required when moving from toy examples to real enterprise applications. Methods and tool support are needed if multiple DSL development is to succeed. One of these methods is described in [48]. The method specifically tackles the problem of overlapping concerns between different DSLs. It has three steps: 1) *Identification*, 2) *Specification*, and 3) *Application*. The purpose of the *Identification* step is to uncover the overlaps between different languages and identify connections among them. The *Specification* step encodes these connections in a way that will make them amenable to various analyses. The last step of the method is *Application* where the encoded connections from the previous two steps are used. The authors also provide tools and case studies for using their method.

B. *Domain-Specific Languages and their Respective Domains*

As can be seen in Figure 5, several DSLs were catalogued according to their domain. We separated the studies that simply report the usage of a DSL in two categories: external DSL and internal (embedded DSL). For each embedded DSL, we also identified in which technology it was implemented. The most common technology in which DSLs are embedded is Haskell with 46 concurrencies, as for example in [49]. However many other host languages are used, such as Java, C/C++, Ruby, Scala, SmallTalk, Python, Prolog, XML and even some unpopular languages like Clean, Galois, Dylan and Curry.

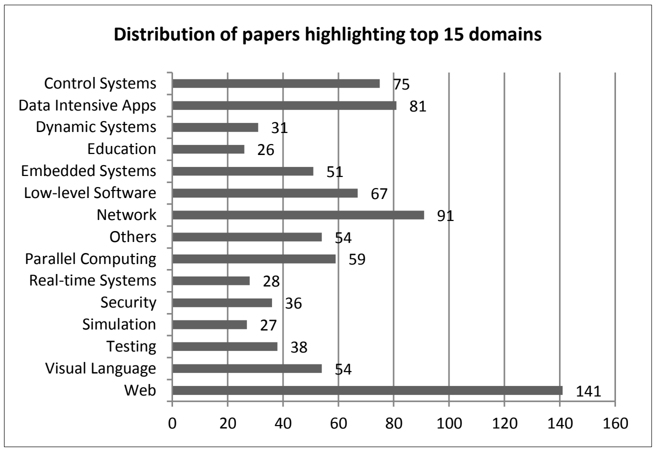

Figure 5. Distribution of papers highlighting top 15 domains.

Different types of DSLs have been identified other than the ones we previously knew. We identified FSML, ADL and DSAL.

A Framework-Specific Modeling Language (FSML) [50] is a kind of Domain-Specific Modeling Language that is used for modeling framework-based software. FSMLs enable automated round-trip engineering over non-trivial model-to-code mappings and thereby simplify the task of creating and evolving framework-based applications.

An Architecture Description Language (ADL or ADSL) [51] is a language that directly expresses a system's architecture. In this sentence, "*directly*" means that the language's abstract syntax contains constructs for all the ingredients of the conceptual architecture. Developers can thus use the language to describe a system on the architectural level.

A Domain-Specific Aspect Language (DSAL) [52] is a custom language that allows special forms of crosscutting concerns to be decomposed into modularized constructs. Examples of domain-specific aspect languages include languages for dealing with coordination concerns, object marshaling concerns, and class graph traversal concerns.

Many different domains that make use of DSL could be identified in our study. The most popular domain was the horizontal domain of web applications, in which several publications states the use of web services, and terms like *services composition*, *services orchestration* and *services mash up* are common. Figure 6 shows a full cross reference view of the DSL research type and their respective domains.

In this context, web services composition refers to the creation of new (web) services by combining functionalities provided by existing ones. A number of domain-specific languages for service composition have been proposed, with consensus being formed around a process-oriented language known as WS-BPEL (or BPEL). The kernel of BPEL consists of simple communication primitives that may be combined using control-flow constructs expressing sequence, branching, parallelism, synchronization, etc. Some examples of BPEL identified in this study: [53–55].

C. *Tools*

Tools play an essential role in software engineering and it is not different when we are talking doing language engineering. Our study identified 151 manuscripts that are related to DSL tools.

Some studies do not actually describe a new tool, but discuss about other tools as in [56] or just make use of a set of tools and report the experience as in [57]. Although, there are few studies comparing DSL tools, we were able to identify two of them as can be seen in [8], [58].

Observing the available publications, we could identify 3 subcategories of tools:

- **Tools for using DSLs**: this type of tool is actually the more comfortable for the user once he/she is supposed to be familiarized with the domain being manipulated. No knowledge about language engineering or domain engineering is necessary for using this type of tool, as well as it is projected be used by domain experts. A good example listed in our study is the tool Scratch [59], appropriated for introductory programming courses.
- **Tools for DSL creation (specification)**: these are a more intuitive way of creating compilers. At this level, the tool is nothing more than a compiler of compilers and, in the end of the process of DSL creation, there will be no integration with other software engineering tools (IDEs), pretty printer,

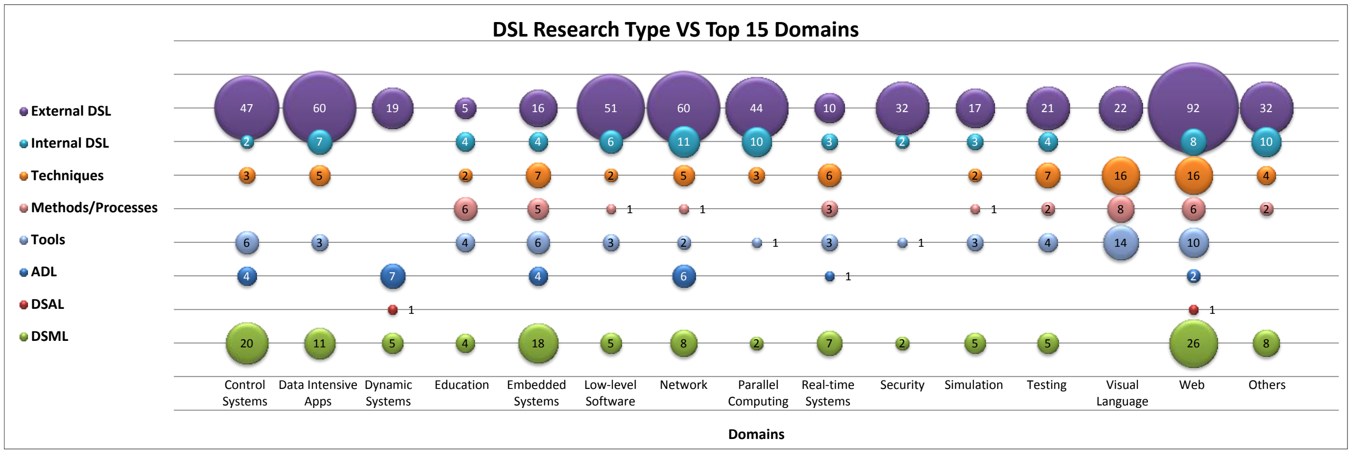

Figure 6. DSL Research Type VS Top 15 Domains.

code assistant and so on. An example of this type of tool in our study is JTS (Jakarta Tool Suite) [60].

- **Language workbench**: these tools support DSL creation not just in terms of parsing and code generation but also in providing a better editing experience for DSL users. In particular, language workbenches let a DSL author create custom DSL editors of similar power to modern IDEs. Language workbenches are still in their early days, but if their potential is realized, they could change the face of programming [10]. Our study identified some examples of language workbenches: XText [61], MetaEdit+ [62], Spoofax [63], and MPS JetBrains [64].

Another way of classifying tools is considering textual and visual languages as described in [65]. Instead, we decided to try other classification to highlight the real power of language workbenches. Unfortunately, the number of publications regarding language workbenches is still low. However, some relevant studies have been published such as [64–66].

V. Threats to Validity

There are some threats to the validity of our study. First one is regarding our set of research questions. The set we defined might not have covered the whole DSL research field, mainly because language implementation in general overlaps other several research fields, for example, model-driven approaches. As we considered this as a feasible threat, we had several discussion meetings and decided to use questions as broad as possible. This way, we knew that the number of primary studies would be bigger but there would be a smaller chance of leaving any important study out of this MS.

In addition, it is possible that we have not chosen the most appropriate keywords. In general, several research fields that use computer science as a mean to solve problems also use DSLs to provide practical solutions where the domain experts can be more effectively involved. However, these types of research and their associated publications may not directly mention DSL keywords. To mitigate this threat we added the terms “generative programming” and “domain-specific modeling language”, although we noticed that rarely the term DSL is left off completely.

Other two possible threats to the validity of our study are: Search engines providing incoherent information in BibTeX and, to mitigate this threat, we developed a tool to extract BibTeX information which considers the peculiarities of each search engine, reducing the number of possible mistakes; and we may have not selected the most representative studies but, to mitigate this threat, we revised the paper selection spreadsheet in pairs until we reached a common sense.

VI. Concluding Remarks

The main motivation for this work was to investigate the state-of-the-art in engineering DSLs, through systematically mapping the literature in order to determine what issues have been studied, as well as by what means, and provide a guide to aid researchers in planning future research.

After performing this mapping study, we catalogued 1440 relevant studies from an initial set of 4450, which helped us to investigate several approaches regarding different aspects of DSL engineering. Our findings could show which are the domains where DSLs are most suitable. For instance, four domains of applications draw our attention, as following (with the respective number of publications): *Web* (141), *Network* (91), *Data Intensive Apps* (81), and *Control Systems* (85). In addition, we were able to catalogue which types of DSL are being created (*internal/external DSL*, *DSML*, *ADL*, *DSAL*), we listed several techniques, methods/processes to handle DSL, and we identified different tools to create and maintain DSLs, including language workbenches.

Moreover, in Figure 6, this study presents a bubble chart with a full cross reference view of DSL research types and their respective domains. This way, it is easy to identify which areas in this research field have been deeply explored and which are lacking attention from academy/industry with only a few publications listed.

In our future agenda, we will investigate more deeply the area of language workbenches through a SR, gathering even more evidence of the area. Moreover, we intend submit an extended version of this study to a journal because the page limit here is restraining us to present more details.

ACKNOWLEDGMENTS

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES [67]), funded by CNPq and FACEPE, grants 573964/2008-4, APQ-1037-1.03/08 and APQ-1044-1.03/10 and Brazilian Agency (CNPq processes number 475743/2007-5 and 140060/2008-1).

REFERENCES

[1] M. Fowler, *Domain-Specific Languages*, 1st ed. Addison-Wesley Professional, 2010, p. 640.

[2] M. Mernik, J. Heering, and A. Sloane, “When and how to develop domain-specific languages,” *ACM Computing Surveys (CSUR)*, vol. 37, no. 4, pp. 316–344, 2005.

[3] D. Ross, “Origins of the APT language for automatically programmed tools,” *ACM SIGPLAN Notices*, vol. 13, no. 8, pp. 61–99, 1978.

[4] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic mapping studies in software engineering,” in *12th International Conference on Evaluation and Assessment in Software Engineering*, 2008, pp. 71–80.

[5] P. J. Landin, “The Next 700 Programming Languages,” *Communications of the ACM*, vol. 9, no. 3, pp. 157–166, 1965.

[6] J. Bentley, “Programming pearls: little languages,” *Communications of the ACM*, vol. 29, no. 8, pp. 711–721, 1986.

[7] A. van Deursen, P. Klint, and J. Visser, “Domain-specific languages: An Annotated Bibliography,” *ACM SIGPLAN Notices*, vol. 35, no. 6, pp. 26–36, Jun. 2000.

[8] N. Vasudevan and L. Tratt, “Comparative Study of DSL Tools,” *Electronic Notes in Theoretical Computer Science*, vol. 264, no. 5, pp. 103–121, Jul. 2011.

[9] B. Kitchenham, “Guidelines for performing systematic literature reviews in software engineering, version 2.3,” Keele University, EBSE Technical Report. EBSE-2007-01, 2007.

[10] M. Fowler, “A pedagogical framework for domain-specific languages,” *Software, IEEE*, vol. 26, no. 4, pp. 13–14, 2009.

[11] T. R. Henry, “Teaching compiler construction using a domain specific language,” *ACM SIGCSE Bulletin*, vol. 37, no. 1, p. 7, Feb. 2005.

[12] J. Munnelly and S. Clarke, “ALPH: a domain-specific language for crosscutting pervasive healthcare concerns,” in *Proceedings of the 2nd workshop on Domain specific aspect languages*, 2007, p. 4–es.

[13] P. Barron and V. Cahill, “YABS: a domain-specific language for pervasive computing based on stigmergy,” in *Proceedings of the 5th international conference on Generative programming and component engineering - GPCE ’06*, 2006, pp. 285–294.

[14] F. Jacob, “CUDACL+: a framework for GPU programs,” in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, 2011, pp. 55–58.

[15] A. Manjunatha, A. Ranabahu, A. Sheth, and K. Thirunarayan, “Power of Clouds in Your Pocket: An Efficient Approach for Cloud Mobile Hybrid Application Development,” in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, 2010, pp. 496–503.

[16] M. Bordignon, U. P. Schultz, and K. Stoy, “Model-based kinematics generation for modular mechatronic toolkits,” in *Proceedings of the ninth international conference on Generative programming and component engineering*, 2010, pp. 157–166.

[17] I. Ceh, M. Crepinsek, T. Kosar, and M. Mernik, “Ontology driven development of domain-specific languages,” *Computer Science and Information Systems*, vol. 8, no. 2, pp. 317–342, 2011.

[18] P. Moreno-Ger, R. Fuentes-Fernández, J.-L. Sierra-Rodríguez, and B. Fernández-Manjón, “Model-checking for adventure videogames,” *Information and Software Technology*, vol. 51, no. 3, pp. 564–580, 2009.

[19] M. Amor, A. Garcia, and L. Fuentes, “Agol: An aspect-oriented domain-specific language for mas,” in *Proceedings of the Early Aspects at ICSE Workshops in AspectOriented Requirements Engineering and Architecture Design*, 2007, pp. 4–11.

[20] P. Sawyer, N. Bencomo, D. Hughes, P. Grace, H. J. Goldsby, and B. H. C. Cheng, “Visualizing the Analysis of Dynamically Adaptive Systems Using i* and DSLs,” in *Second International Workshop on Requirements Engineering Visualization REV*, 2007, pp. 1–10.

[21] T. Antao, I. Hastings, and P. McBurney, “Ronald: A Domain-Specific Language to study the interactions between malaria infections and drug treatments,” in *International Conference on Bioinformatics Computational Biology*, 2008, pp. 747–752.

[22] H. Behrens, “MDSD for the iPhone Developing a Domain-Specific Language and IDE Tooling to produce Real World Applications for Mobile Devices,” in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, 2010, pp. 123–128.

[23] X. Amatriain and P. Arumi, “Frameworks Generate Domain-Specific Languages: A Case Study in the Multimedia Domain,” *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 544–558, 2011.

[24] S. Michels and R. Plasmeijer, “iTask as a new paradigm for building GUI applications,” in *Proceedings of the 22nd international conference on Implementation and application of functional languages (IFL’10)*, 2010, pp. 153–168.

[25] C. Kulkarni, G. Brebner, and G. Schelle, “Mapping a domain specific language to a platform FPGA,” in *Proceedings of the 41st annual conference on Design Automation DAC 04*, 2004, pp. 924–927.

[26] M. Jiménez, F. Rosique, P. Sánchez, B. Álvarez, and A. Iborra, “Habitation: A Domain-Specific language for home automation,” *Software, IEEE*, vol. 26, no. 4, pp. 30–38, 2009.

[27] J. M. Neighbors, “The Draco approach to Constructing Software from Reusable Components,” *IEEE Transactions on Software Engineering*, vol. 10, no. 5, pp. 567–574, 1984.

[28] T. Clark and L. Tratt, “Language factories,” in *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA ’09*, 2009, pp. 949–955.

[29] H. Meng, “Semiautomatic acquisition of semantic structures for understanding domain-specific natural language queries,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 1, pp. 172–181, 2002.

[30] C. Brabrand and M. I. Schwartzbach, “The metafront system: Safe and extensible parsing and transformation,” *Science of Computer Programming*, vol. 68, no. 1, pp. 2–20, Aug. 2007.

[31] J. Evermann and Y. Wand, “Toward formalizing domain modeling semantics in language syntax,” *IEEE Transactions on Software Engineering*, vol. 31, no. 1, pp. 21–37, Jan. 2005.

[32] K. Kennedy et al., “Telescoping Languages: A System for Automatic Generation of Domain Languages,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 387–408, Feb. 2005.

[33] C. Consel and F. Latry, “A generative programming approach to developing DSL compilers,” in *Fourth International Conference on Generative Programming and Component Engineering (GPCE)*, 2005, pp. 29–46.

[34] F. Lagarde, H. Espinoza, F. Terrier, C. André, and S. Gérard, “Leveraging Patterns on Domain Models to Improve UML Profile Definition,” in *FASE’08/ETAPS'08 Proceedings of the Theory and*

practice of software, 11th international conference on Fundamental approaches to software engineering, 2008, vol. 4961, pp. 116–130.

[35] T. Ritala and S. Kuikka, "UML Automation Profile: Enhancing the Efficiency of Software Development in the Automation Industry," in *5th IEEE International Conference on Industrial Informatics*, 2007, pp. 885–890.

[36] I. Weisemöller and A. Schürr, "A Comparison of Standard Compliant Ways to Define Domain Specific Languages," in *ACM/IEEE 10th International Conference On Model Driven Engineering Languages And Systems (MoDELS 2007)*, 2008, pp. 47–58.

[37] F. Javed, M. Mernik, and A. Sprague, "Incrementally inferring context-free grammars for domain-specific languages," in *Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering - SEKE'06*, 2006, pp. 363–368.

[38] H. Krahn, B. Rumpe, and S. Völkel, "Integrated definition of abstract and concrete syntax for textual languages," in *10th International Conference Model Driven Engineering Languages and Systems (MoDELS 2007)*, 2007, pp. 286–300.

[39] R. T. Lindeman, L. C. L. Kats, and E. Visser, "Declaratively defining domain-specific language debuggers," in *Proceedings of the 10th ACM international conference on Generative programming and component engineering - GPCE '11*, 2011, pp. 127–136.

[40] R. Martinho, J. Varajão, and D. Domingos, "Using the semantic web to define a language for modelling controlled flexibility in software processes," *IET Software*, vol. 4, no. 6, p. 396, 2010.

[41] B. Selic, "A systematic approach to domain-specific language design using UML," in *Proc. 10th IEEE Int'l Symp. Object and Component-Oriented Real-Time Distributed Computing*, 2007, pp. 2–9.

[42] L. Tratt, "Evolving a DSL implementation," in *ICSE '08 Proceedings of the 30th international conference on Software engineering*, 2008, pp. 425–441.

[43] S. Wenzel and U. Kelter, "Analyzing model evolution," in *Proceedings of the 13th international conference on Software engineering - ICSE '08*, 2008, pp. 831–834.

[44] A. Vajda and J. Eker, "Return to the language forrest," in *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10*, 2010, pp. 389–392.

[45] H. Krahn, B. Rumpe, and S. Völkel, "MontiCore: a framework for compositional development of domain specific languages," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 5, pp. 353–372, 2010.

[46] M. Brambilla, P. Fraternali, and M. Tisi, "A Transformation Framework to Bridge Domain Specific Languages to MDA," in *ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2008)*, 2009, pp. 167–180.

[47] E. Wyk and E. Johnson, "Composable Language Extensions for Computational Geometry: A Case Study," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007, pp. 258–267.

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

[49] P. Thiemann, "An embedded domain-specific language for type-safe server-side web scripting," *ACM Transactions on Internet Technology*, vol. 5, no. 1, pp. 1–46, Feb. 2005.

[50] M. Antkiewicz, K. Czarnecki, and M. Stephan, "Engineering of Framework-Specific Modeling Languages," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 795–824, Nov. 2009.

[51] M. Voelter, "Architecture as Language," *IEEE Software*, vol. 27, no. 2, pp. 56 – 64, 2010.

[52] M. Shonle, K. Lieberherr, and A. Shah, "XAspects: an extensible system for domain-specific aspect languages," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications - OOPSLA'03*, 2003, pp. 28–37.

[53] S. Brahe and B. Bordbar, "A pattern-based approach to business process modeling and implementation in web services," in *ICSOC'06 Proceedings of the 4th international conference on Service-oriented computing*, 2007, pp. 166–177.

[54] W. Hummer, P. Gaubatz, M. Strembeck, U. Zdun, and S. Dustdar, "An integrated approach for identity and access management in a SOA context," in *Proceedings of the 16th ACM symposium on Access control models and technologies - SACMAT'11*, 2011, pp. 21–30.

[55] J. Boubeta-Puig, I. Medina-Bulo, and A. García-Domínguez, "Analogies and Differences between Mutation Operators for WS-BPEL 2.0 and Other Languages," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011, pp. 398–407.

[56] D. Spinellis, "The Tools We Use," *IEEE Software*, vol. 24, no. 4, pp. 20–21, Jul. 2007.

[57] T. Kosar, M. Mernik, and P. E. M. Lopez, "Experiences on DSL Tools for Visual Studio," in *2007 29th International Conference on Information Technology Interfaces*, 2007, pp. 753–758.

[58] M. Freudenthal, "Using DSLs for developing enterprise systems," in *Proceedings of the Tenth Workshop on Language Descriptions Tools and Applications*, 2010, pp. 11:1–11:7.

[59] M. Resnick et al., "Scratch: Programming for All," *Communications of the ACM*, vol. 52, no. 11, p. 60, Nov. 2009.

[60] D. Batory, B. Lofaso, and Y. Smaragdakis, "JTS: tools for implementing domain-specific languages," in *Proceedings. Fifth International Conference on Software Reuse*, 1998, pp. 143–153.

[61] M. Eysholdt and H. Behrens, "Xtext - Implement your Language Faster than the Quick and Dirty way," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion - SPLASH'10*, 2010, pp. 307–309.

[62] R. Pohjonen, "Metamodeling made easy–metaedit+ (tool demonstration)," in *Generative Programming and Component Engineering (GPCE 2005)*, 2005, pp. 442–446.

[63] L. C. L. Kats and E. Visser, "The spoofax language workbench," *ACM SIGPLAN Notices*, vol. 45, no. 10, p. 444, Oct. 2010.

[64] M. Voelter, "Embedded software development with projectional language workbenches," in *Proceedings of the 13th international conference on Model driven engineering languages and systems Part II (MoDELS 2010)*, 2010, pp. 32–46.

[65] B. Merkle, "Textual modeling tools: overview and comparison of language workbenches," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion - SPLASH '10*, 2010, pp. 139–148.

[66] M. Völter and E. Visser, "Language extension and composition with language workbenches," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion - SPLASH '10*, 2010, pp. 301–304.

[67] *National Institute of Science and Technology for Software Engineering (INES)*. Available in: www.ines.org.br. Last accessed in September, 2012.