Domain-Specific Languages for Workflows. A Systematic Literature Review.

Akif Quddus Khan
Dept. of Information Technology
Norwegian University of Science and Technology
Gjøvik, Norway
akif.q.khan@ntnu.no

***Abstract*—This paper aims to provide an overview of the complete process in the development of a Domain-Specific Language (DSL). It explains the construction steps such as preliminary research, language implementation, and evaluation. Moreover, it provides details for different key-components which are commonly found in the DSLs such as the abstraction layer, DSL metamodel and the applications. It also explains the general limitations related to the Domain-Specific Languages for Workflows.**

***Index Terms*—WorkflowDSL, DSL, Domain-Specific Language, Workflow.**

I. Introduction

In recent years, Scientific Workflows [1] have been intensively applied in astronomy [2], seismology [3], genomics [4], etc. A workflow has a sequence of jobs that perform required functionality and it has control or data dependencies between jobs. Or, as defined by the Workflow Management Coalition, workflow is the automation of a business process, in whole or parts, during which documents, information or tasks are passed from one participant to another for actions, according to a set of procedural rules [5]. Systems like Pegasus [6], Pregel [7], GBase [8], Trinity [9], Hama [10], Giraph [11], HipG [12], PBGL [13], ScaleGraph [14], KDT [15], etc. are a few examples of Big Data workflows.

A DSL, on the other hand, is a "mini" language built on top of a hosting language (such as C, Java, and JavaScript) that provides a common syntax and semantics to represent concepts and behaviors in a particular domain. Maximilien et al., [16], summarized that using or designing a DSL generally helps achieving the high-level linguistic constructs, terse code, simple-and-natural syntax, ease of programming, and code generation. In the research communities of software engineering and programming language, Domain-Specific Language is an important and efficient way to reduce the programming complexity as well as improve the productivity [17]. Essentially, DSL aims to realize the concept of meta programming [18], which makes a program designed to read, generate, analyse or transform other programs, and even modify itself while running. A DSL that requires itself to be embedded within the host language can be called an Internal DSL. A DSL with its own syntax and that is not required to be embedded within another language can be called an External DSL [19]. One of the biggest advantage of using DSLs is, since they are tailored to a specific application domain, they exploit domain knowledge for productivity and efficiency [20].

It is commonly agreed that the development of custom DSLs and the related model-driven workflows is a complex task that should be addressed with an iterative process [21]. Hence, all kinds of purposes may be fulfilled by domain-specific languages. In diverse ways and among varying kinds of people, they may be used. Some DSLs are designed for programmers to use and are thus more advanced, while others are meant for anyone who is not a programmer to use less geeky concepts and syntax.

This papers covers the following topics related to DSLs for Workflow: Key-components in DSLs for Workflow, major/common construction steps in DSLs for Workflows and the limitations of the DSLs in the workflows. This paper is organized as follows. In Section II, We have described the complete research methodology for this paper in detail. In Section III, we have summarized the answers for each research question in a separate sub-section. In Section IV, we have concluded the research results and have presented the results in a concise form.

II. Research Methodology

Jesson et al defines the systematic review as "a review with a clear stated purpose, a question, a defined search approach, stating inclusion and exclusion criteria, producing a qualitative appraisal of articles" [22].

The complete research process is shown in figure 1.

A. Define the research question

For defining the research question, We initially set a bigger topic of study, Big Data Workflows. It contains multiple sub topics such as Visualization, Workflow Initialization, Workflow specification, Domain-Specific Languages, Cloud infrastructure etc. So for this paper, We focused on the Domain-Specific Languages (DSL) for Workflows.

After reading the relevant literature, following are the research questions we formulated.

1) What are the key-components of a Domain-Specific Language for Workflows?
2) What are common steps in the construction process of a DSL Language for Workflows?
3) What are the limitations of DSLs for Workflows?

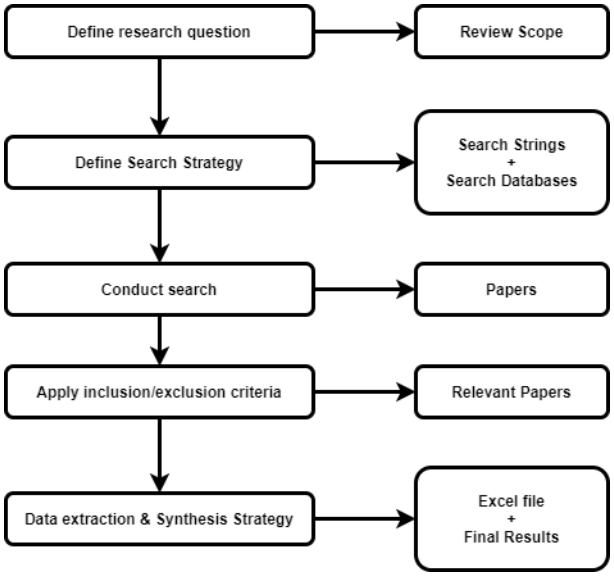

Fig. 1: Complete Research Methodology for this paper [23]

B. Inclusion/Exclusion Criteria

We defined the following Inclusion/Exclusion criteria for this paper.

- Only Primary Studies, No secondary studies
- Only Peer reviewed studies
- Studies published in English Language
- Studies that answers at least one research question

C. Research Keywords

Our main research keywords are ”DSL”, ”Workflow”, ”WorkflowDSL” and ”Domain-Specific language”. These keywords produced results to help answer the first three research questions. Since we got a lot irrelevant results by searching in ”All Metadata” and too few results by searching in title, so we decided to search our terms in Abstract.

For IEEE, we used the following query:

```
(("Abstract":DSL AND
"Abstract":Workflow)
OR
("Abstract": "Domain-Specific
Language"
AND "Abstract":Workflow)
OR
("Abstract":WorkflowDSL))
```

For ACM, we used the following query:

```
[[Abstract: dsl] OR
[Abstract: "Domain-Specific
language"] OR
[Abstract: workflowdsl]]
AND
[Abstract: workflow]
```

D. Research Results Overview

In the first round, we analysed the Title and Abstract of the papers. We applied the Inclusion/Exclusion criteria and following are the results.

	Total	Relevant
IEEE	41	13
ACM	28	7

TABLE I: Databases results overview

In the second round, We analysed the complete papers to see if they answer at least one of the research question. After the second round, we got the following results:

IEEE: 10 and **ACM:** 5

E. Snowball method

Since we could only find a limited number of relevant papers, to compliment the answers of our research questions, we used snowball approach [24] in some articles. It is a way of finding literature by using a key document on the subject as a starting point. Then to find other relevant articles and papers, consult the bibliography in the key document. One disadvantage of this approach is that the literature we find in the bibliography, might be old. To overcome this challenge, We made sure that the source is relatively newer, if not, highly relevant.

III. Analysis

In this section, We present the results of our literature review on DSLs for Workflows. We started by formulating a summary of the answers found in the relevant papers, which is followed by the presentation of the commonalities in the literature.

A. Key-components of a Domain-Specific Language in Workflow

Since the use of computer systems has increased substantially overtime, there has been an exponential growth in the size of data being collected, which in turn creating a shortage of capable analysts [25]. Hence, visualization has proved to be an effective tool for exploring and gathering insight from large quantities of data [26]. Karl Smeltzer at Oregon State University, designed a prototype for a DSL for data visualization. He described the following key-components for the DSL [27]:

- Abstraction layers: There could be multiple abstraction layers each according to user needs. This would allow the implementation details to be hidden when desired.
- Underlying Systematic Model: An underlying model of visualizations rather than flat images. Since the process of data analysis is iterative [28], this would allow users to steer clear of creation of new visualizations at the start of each iteration. Instead, existing visualizations can be transformed as per the new requirements.

A team of researchers at the University of Firenze developed a DSL-supported Workflow for the Automated assembly of large stochastic models [29]. In their approach, they used

TMDL (Template Models Definition Language). For TMDL, they defined the key-components as follows:

- TMDL "Library": A library is composed of a set of template elements. Each template has a distinguished name, and a set of parameters.
- TMDL "Scenario": A scenario is composed of a set of classes. Each class has a distinguished name, and references a specific template in the model library. Moreover, a class may contain a set of assignments, which specify concrete values.

For executable system models, formal languages like UML [30] or SysML [31] are increasingly used to model them. Furthermore, an effective way to reduce development and usage effort for complex software systems is the model-driven development (MDD) approach and the model=driven architecture (MDA) [32]. Sven et al., [33] developed a Domain-Specific Language for Executable system models. The defined the key-components as four different model layers which are as follows:

- The meta metamodel layer (M3)
- The metamodel layer (M2)
- The model layer (M1)
- The instance layer (M0)

Each layer uses the elements of the upper layer to describe the elements for modeling the lower layer. The important thing to notice in this approach is that the four-layer structure is not fixed for every application. It depends on the specific modeling problem. For example, they used three-layer approach for their system.

A team of researchers at the McGill University, Canada developed a DSL for crisis management systems. The described the following key-component in their approach [34].

- A metamodel: Based on the information from the use cases
- A Usecase map: A diagram to show the handling of a given type of emergency from one actor's perspective.

Embedded DSL developed for Progressive Spatiotemporal Data Analysis and Visualization has the following key-components [35]:

- Abstract Data Type: A new built-in data type that abstracts the common modalities of the scientific data.
- Explicit Data Publishing Hints: A hinting mechanism to facilitate incremental production of the results of ongoing computations by clearly indicating the current state as well as the available appropriate opportunities.
- Generalized multidimensional Iterators: A generic iterator for loops that can be performed in any order for example for computing an average. They would also permit nonlinear evaluation of the loop body.

Arjan et al., 2013 described following as the Ingredients of key-components for any DSL [36]:

- Abstract Syntax or Metamodel: Describes the domain concepts and their relations.
- Concrete Syntax or language: Describes the representation of an instance of the metamodel. Usually they are separated to manage the complexity [37], hence separating concept definition from their representation. Regardless of that, the abstract and concrete syntax significantly overlap [38], thus they need to be consistently defined [39].
- Instances of the DSL: Generating a language infrastructure that recognizes the Domain-Specific keywords as defined in the language.

The implementation is based on the Eclipse Platform and its "Modeling" components [40]. The TMDL meta-model has been defined as an Ecore model; Xtext [41] has then been used to define a textual syntax, and to generate the editor, parser, and syntax highlighter. A prototype version of the transformation algorithm has been implemented using the ATLlanguage [3].

Hiroaki et all., 2017 [42] developed a DSL called ContextWorkflow, a DSL for compensable and Interruptible executions. ContextWorkflow is developed as a DSL embedded in Scala using the Monad interface in scalaz, a functional programming library [43]. The key modules for this specific DSL are as follows:

- Atomic Action and Workflow: An atomic action consists of a normal action (i.e., a Scalaexpression) and a compensation action, and a workflow is a sequence of atomic actions. Both of them are represented as objects of type Workflow, which is basically a compensation monad [44].
- Interruption and Context: Interruptions can only occur between the atomic actions. To tackle this issue, they proposed a two steps scheme:
  1) To represent time-varying context
  2) To check the context associated to a workflow at the start of the each atomic action

Skitter [45], is a DSL for distributed reactive workflows. It is implemented on top of the Elixir programming language. Elixir was chosen for its focus on distributed systems. It has a proven track record of scaling to large systems and is being used by Amazon, Facebook etc. Moreover, it uses an actor model [46] which provides a natural way to treat a component instance as as independant execution unit.

Following are the key-components of Skitter [45]:

- Component Definition: The functions that define how the component reacts to incoming data, as well as the information that how the component can be embedded inside a workflow and the effects it may generate when reacting.
- Workflow Definition: It consists of further three entities:
  1) A set of reactive component instances
  2) A source
  3) A set of links
- Workflow execution: Write programs using Skitter to solve the problems on hand. In the given case, Skitter uses the dataflow model [47] to achieve parallelism.

B. Common steps in the construction process of a DSL Language for Workflow:

There are two common approaches to develop a DSL: Start from the general-purpose UML and constrain and refine it's usage to better embrace Domain-Specifications. Or use a generic language that relates to domain modeling concepts [48].

Following approach was used by the researchers to develop a DSL for a crisis management system [34]:

1) Preliminary Research: To realize the goal of the research and gather relevant information of the system for which the DSL is being developed.
2) Use Cases: To formulate the the concrete requirements aimed to be fulfilled by the DSL.

Van et al., describes the following typical steps in the development of a DSL [49]:

- Analysis:
  1) Identify the domain of the problem
  2) Collect knowledge of the domain
  3) Arrange it into small number of concepts
  4) Design a DSL that describes the target applications
- Implementation:
  1) Build a library that implements the Use cases/concepts collected in step 1
  2) Design and develop a compiler that executes DSL programs
- Use: Write DSL programs for the applications in the target domain

Bonachea et al,. [50] presented a practical case study of developing a DSL for costumer user profiling. They advocated a completely iterative model of the process, with iteration taking place between most activities. They reported that the following activities were being implemented:

1) Interview domain experts
2) Models Development
3) Write programs that observes the models
4) Shape the language
5) Use the new DSL to write programs
6) Develop and Implement Runtime system and language compiler

Cleaveland [51] proposesd an approach of process modelling for building application generators. They are a particular case of DSL in which a compiler translates high-level specifications into a regular low-level programming language. Their proposed process contains seven steps:

1) Identify/Recognize Domains
2) Define boundaries for domains
3) Define a model underlying it
4) Define the components of variants and in-variants
5) Identify products
6) Develop the generator

Khalaoui et al. have examined the success factors for domain specific modeling activities and compiled a list of qualitative criteria with positive and negative impacts [52] which is presented in figure 2.

C. Limitations of DSLs for Workflows

Tharido et al., 2018 [53] developed a DSL called "WorkflowDSL" for Data Analysis applications. WorkflowDSL language does not have a support for cyclic operations. Moreover, it is ennforced via Xtext validation rules. In addition to that, since this very language was designed to focus on data-intensive workflows, it cannot convey flows of a workflow. Other limitations include, grammar expression capabilities and high dependency on language constructs.

Exedra [20], a domain-specific language for large graph analytics workflows has the following limitations.

- The current version of Exedra grammar is limited to graph reading, and graph analysis algorithms such as degree distribution calculation, clustering etc.
- This version of Exedra grammar does not have the capability to do graph traversal operations.
- There is only implementation of compartments for ScaleGraph and KDT libraries, hence restricting the other mediums.
- The communication between different compartments introduces an additional overhead of data format conversion.
- Lastly, the Dipper framework is not able to accommodate the changes made to the libraries or the middle ware using which the compartments run.

Apart from the development, evaluation of a DSL is also an integral and challenging task. Hence, the question of evaluating DSL solutions is gaining more and more attention in the scientific community [54]. Khalaoui et al. have investigated the success factors for Domain-Specific modeling activities and prepared a list of qualitative criteria with positive and negative impacts [55]. To evaluate a DSM solution, Mohagheghi et al. [56] suggested using both quantitative and qualitative criteria, while taking into account the stakeholder's interests.

To do so, there are following limitations [54]:

- As the complexity of the DSL model increases, simple evaluation metrics do not fulfill the requirements.
- To validate the generator classes etc, separate evaluation metrics are required.
- The implementation cost and the learning curve for using the DSL is also a challenging factor.

Nikolov et al., [57] proposed a DSL for scalable execution of big data workflows with the use of software containers. They put a major focus on workflow definition using a domain-specific language.A workflow step can be defined with a communication medium and triggers. But the metamodel does not have any element to describe the storage system for the workflow. Since the DSL generates a YAML file for cloud orchestration tools, network, ports and build parameters can also be added. Moreover, it is also missing the ability for resource provisioning/management.

IV. Results

After analysing the relevant literature, the common key-components in all the Domain-Specific Languages in Workflow are presented in Table II.

Success Factors	Positive Impact	Negative Impact	EAI	Assessment Criteri
Domain expertise	**Good knowledge of domain concepts, vocabulary and terminology => Expressive DSML**	**Incomplete domain knowledge => Inexpressive DSML, lacking functionalities**	***Abstract syntax***	***Expressiveness Completeness***
Domain scoping	**Fitting business needs**	**Domain too broad or too narrow**	***Abstract syntax Concrete syntax***	***Simplicity Suitability Completeness***
Effective supporting tools	**Faster, better and cheaper DSML development**	**Costly DSML**	***All elements***	***Cost-effectiveness Productivity***
Effective meta-model	**Easy definition and upgrade of expressive, high-level abstract DSML.**	**Ambiguity Poor semantics**	***Abstract syntax Semantic***	***Formality Expressiveness Comprehensibility Scalability***
Effective underlying generator	**Better exploitation of DSML models**	**Models exclusively intended for documentation purposes**	***Abstract syntax Semantic***	***Utility Transformability Interpretability***
Domain engineering environment	**Specialized and dedicated teams for DSML definition**	**Weak domain expertise**	***All elements***	***Supportability Suitability***
High level of abstraction	**Close to the real-world separation of concerns**	**Platform-dependent DSML**	***Abstract syntax***	***Simplicity Expressiveness***
Language expertise	**Coherent models, useful and non-redundant functionality.**	**Incoherent models' useless, redundant functionalities**	***All elements***	***Uniqueness Coherence***
View point orientation	**Specialized DSMLs**	**Some stakeholders may be neglected**	***Views***	***Utility Suitability***
Purpose-orientation	**Focused DSMLs**	**Incoherent models**	***Views***	***Consistency Utility***
Domain expert support	**More support and fast adoption**	**Resistance and sabotage**	***All elements***	***Usability Utility***
Effective DSML definition process	**Well-established practices for DSML definition**	**Individual approaches' unintended results**	***All elements***	***Scalability Maintainability***

Fig. 2: Success Factors and Assessment Criteria Mapping [52]

Component Name	Short Description	Other Terms Used
Abstraction Layer	Abstracts the common modalities of the of the DSL. Describes the domain concepts and their relations.	Abstract Syntax, Metamodel, Atomic Action, or Language Library
Metamodel Layer	Describes the representation of an instance of the metamodel. Composed of a set of classes and each class contains the definition of a specific component in the abstraction layer.	Scenario, Concrete Syntax, Component Definition,
Instance Layer	A language infrastructure that recognizes the domain specific keywords as defined in the language.	Context, Workflow Execution, Instantces of DSL

TABLE II: Key-components of DSLs for Workflows

The common construction steps to develop a new DSL for Workflows are presented in table III:

Step Name	Common Tasks Involved
Prelimenary Research	Identify problem domain, Collect domain knowledge, Interview domain experts, Prepare Use cases
Implementation	Write programs that observes the use cases. Compiler Construction
Evaluation	Writing programs using the new DSL and its benchmark evaluation

TABLE III: Common construction steps for DSL for Workflows

V. Conclusion

In this paper we covered the major elements involves in any DSL for Workflows, that is, the different layers of a Domain-Specific Language. In addition to that, based on the relevant case studies, what could be the common steps to build those components. There are several stakeholders involved in the development of a DSL. From Software Engineers, Quality Experts, Tool Developers to Domain Experts, End Users. Each stakeholder plays its integral role, either it is implementing the language, requirement gathering, language evolution, scalability or usability/reusability. Concurrency in a Domain-Specific language for Workflows is a challenge, yet to be solved. Furthermore, the complexity is directly proportional to language complexity, that is, the more domain artifacts a DSL covers, the larger its complexity would be.

References

[1] I. J. Taylor, E. Deelman, D. B. Gannon, M. Shields *et al.*, *Workflows for e-Science: scientific workflows for grids*. Springer, 2007, vol. 1.

[2] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su, "Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand," in *Optimizing Scientific Return for Astronomy through Information Technologies*, vol. 5493. International Society for Optics and Photonics, 2004, pp. 221–232.

[3] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya *et al.*, "Scec cybershake workflows—automating probabilistic seismic hazard analysis calculations," in *Workflows for e-Science*. Springer, 2007, pp. 143–163.

[4] "Usc epigenome center."

[5] W. M. Coalition, "Workflow management coalition terminology and glossary," *Workflow Management Coalition*, 1999.

[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[7] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–146.

[8] U. Kang, H. Tong, J. Sun, C.-Y. Lin, and C. Faloutsos, "Gbase: a scalable and general graph management system," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 1091–1099.

[9] B. Shao, H. Wang, and Y. Xiao, "Managing and mining large graphs: systems and implementations," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 589–592.

[10] S. Seo, E. J. Yoon, J. Kim, S. Jin, J.-S. Kim, and S. Maeng, "Hama: An efficient matrix computation with the mapreduce framework," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE, 2010, pp. 721–726.

[11] M. Dayarathna and T. Suzumura, "A first view of exedra: a domain-specific language for large graph analytics workflows," in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 509–516.

[12] E. Krepska, T. Kielmann, W. Fokkink, and H. Bal, "Hipg: parallel processing of large-scale graphs," *ACM SIGOPS Operating Systems Review*, vol. 45, no. 2, pp. 3–13, 2011.

[13] D. Gregor and A. Lumsdaine, "Lifting sequential graph algorithms for distributed-memory parallel computation," *ACM SIGPLAN Notices*, vol. 40, no. 10, pp. 423–437, 2005.

[14] M. Dayarathna, C. Houngkaew, and T. Suzumura, "Introducing scalegraph: an x10 library for billion scale graph analytics," in *Proceedings of the 2012 ACM SIGPLAN X10 Workshop*, 2012, pp. 1–9.

[15] A. Lugowski, D. Alber, A. Buluç, J. R. Gilbert, S. Reinhardt, Y. Teng, and A. Waranis, "A flexible open-source toolbox for scalable complex graph analysis," in *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 2012, pp. 930–941.

[16] E. M. Maximilien, H. Wilkinson, N. Desai, and S. Tai, "A domain-specific language for web apis and services mashups," in *International Conference on Service-Oriented Computing*. Springer, 2007, pp. 13–26.

[17] M. Fowler, "Domain-specific languagesdsl: Domain-specific languages," 2011.

[18] T. Sheard and S. P. Jones, "Template meta-programming for haskell," in *Proceedings of the 2002 ACM SIGPLAN workshop on Haskell*, 2002, pp. 1–16.

[19] R. D. Kelker, *Clojure for domain-specific languages*. Packt Publishing Ltd, 2013.

[20] M. Dayarathna and T. Suzumura, "A first view of exedra: A domain-specific language for large graph analytics workflows," in *Proceedings of the 22nd International Conference on World Wide Web*, ser. WWW '13 Companion. New York, NY, USA: Association for Computing Machinery, 2013, p. 509–516. [Online]. Available: https://doi.org/10.1145/2487788.2487985

[21] ¡a href="contents/contents.php?query=Volter"¿Marcus Volter¡/a¿, "Md* best practices," *Journal of Object Technology*, vol. 8, no. 6, pp. 79–102, Sep. 2009, (column). [Online]. Available: http://www.jot.fm/contents/issue_2009_09/column6.html

[22] J. Jesson, L. Matheson, and F. M. Lacey, *Doing your literature review: Traditional and systematic techniques*. Sage, 2011.

[23] A. Q. Khan, S. Khan, and U. Safaev, "Serious games and gamification: A systematic literature review," 2020.

[24] LibGuides, "Information literacy history: Search methods." [Online]. Available: https://libguides.rug.nl/c.php?g=470628&p=3218096

[25] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, A. Hung Byers *et al.*, *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute, 2011.

[26] D. Keim, "al.(2006). challenges in visual data analysis," *Tenth International Confer*, 2006.

[27] K. Smeltzer, "A language for visualization variation and transformation," in *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2014, pp. 195–196.

[28] J. J. Van Wijk, "The value of visualization," in *VIS 05. IEEE Visualization, 2005.* IEEE, 2005, pp. 79–86.

[29] L. Montecchi, P. Lollini, and A. Bondavalli, "A dsl-supported workflow for the automated assembly of large stochastic models," in *2014 Tenth European Dependable Computing Conference*. IEEE, 2014, pp. 82–93.

[30] OMG, "Unified modeling language tm (omg uml), version 2.5," 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5/PDF

[31] ——, "Systems modeling language (omg sysml), version 1.3," 2012. [Online]. Available: http://www.omg.org/spec/SysML/1.3/

[32] ——, "Model driven architecture (mda) mda guide rev. 2.0," 2014. [Online]. Available: http://www.omg.org/cgi-bin/doc?ormsc/14-06-01

[33] S. Jäger, R. Maschotta, T. Jungebloud, A. Wichmann, and A. Zimmermann, "Creation of domain-specific languages for executable system models with the eclipse modeling project," in *2016 Annual IEEE Systems Conference (SysCon)*. IEEE, 2016, pp. 1–8.

[34] N. B. Khzam and G. Mussbacher, "Domain-specific software language for crisis management systems," in *2018 IEEE 8th International Model-Driven Requirements Engineering Workshop (MoDRE)*. IEEE, 2018, pp. 36–45.

[35] C. Christensen, J.-W. Lee, S. Liu, P.-T. Bremer, G. Scorzelli, and V. Pascucci, "Embedded domain-specific language and runtime system for progressive spatiotemporal data analysis and visualization," in *2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE, 2016, pp. 1–10.

[36] A. J. Mooij, J. Hooman, and R. Albers, "Gaining industrial confidence for the introduction of domain-specific languages," in *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, 2013, pp. 662–667.

[37] F. Fondement and T. Baar, "Making metamodels aware of concrete syntax," in *European Conference on Model Driven Architecture-Foundations and Applications*. Springer, 2005, pp. 190–204.

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

[39] M. Rossi, J. Sprinkle, J. Gray, J.-P. Tolvanen *et al.*, "Proceedings of the 9th oopsla workshop on domain-specific modeling (dsm'09)," 2009.

[40] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[41] "Xtext-language development made easy."

[42] H. Inoue, T. Aotani, and A. Igarashi, "A dsl for compensable and interruptible executions," in *Proceedings of the 4th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems*, ser. REBLS 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 8–14. [Online]. Available: https://doi.org/10.1145/3141858.3141860

[43] E. Bainomugisha, J. Vallejos, C. De Roover, A. L. Carreton, and W. De Meuter, "Interruptible context-dependent executions: a fresh look at programming context-aware applications," in *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*, 2012, pp. 67–84.

[44] G. Ramalingam and K. Vaswani, "Fault tolerance via idempotence," in *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2013, pp. 249–262.

[45] M. Saey, J. De Koster, and W. De Meuter, "Skitter: A dsl for distributed reactive workflows," in *Proceedings of the 5th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems*, 2018, pp. 41–50.

[46] G. Agha, "An overview of actor languages," *ACM Sigplan Notices*, vol. 21, no. 10, pp. 58–67, 1986.

[47] T. Von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active messages: a mechanism for integrated communication and computation," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, pp. 256–266, 1992.

[48] 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.

[49] A. Van Deursen, P. Klint, and J. Visser, "Domain-specific languages," in *DRAFT DRAFT ANNOTATED BIBLIOGRAPHY. DRAFT ACM SIGPLAN NOTICES. DRAFT*. Citeseer, 2000.

[50] D. Bonachea, K. Fisher, A. Rogers, and F. Smith, "Hancock: A language for processing very large-scale data," in *Proceedings of the 2nd conference on Domain-specific languages*, 1999, pp. 163–176.

[51] W. B. Frakes and S. Isoda, "Success factors of systematic reuse," *IEEE software*, vol. 11, no. 5, pp. 14–19, 1994.

[52] A. Kahlaoui, A. Abran, and E. Lefebvre, "Dsml success factors and their assessment criteria," 2008.

[53] T. Fernando, N. Gureev, M. Matskin, M. Zwick, and T. Natschläger, "Workflowdsl: Scalable workflow execution with provenance for data analysis applications," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 01, 2018, pp. 774–779.

[54] T. Wegeler, F. Gutzeit, A. Destailleur, and B. Dock, "Evaluating the benefits of using domain-specific modeling languages: an experience report," in *Proceedings of the 2013 ACM workshop on Domain-specific modeling*, 2013, pp. 7–12.

[55] A. Kahlaoui, A. Abran, and E. Lefebvre, "Dsml success factors and their assessment criteria," *Metrics News*, vol. 13, no. 1, pp. 43–51, 2008.

[56] P. Mohagheghi and Ø. Haugen, "Evaluating domain-specific modelling solutions," in *International Conference on Conceptual Modeling*. Springer, 2010, pp. 212–221.

[57] N. Nikolov, Y. D. Dessalk, A. Q. Khan, A. Soylu, M. Matskin, A. H. Payberah, and D. Roman, "Conceptualization and scalable execution of big data workflows using domain-specific languages and software containers," *Internet of Things*, p. 100440, 2021.