

Domain-Specific Languages

Edited by

Sebastian Erdweg¹, Martin Erwig², Richard F. Paige³, and
Eelco Visser⁴

- 1 TU Darmstadt, DE, erdweg@informatik.tu-darmstadt.de
- 2 Oregon State University, US, erwig@eecs.oregonstate.edu
- 3 University of York, GB, richard.paige@york.ac.uk
- 4 TU Delft, NL, e.visser@tudelft.nl

Abstract

This report documents the program and outcomes of Dagstuhl Seminar 15062 “Domain-Specific Languages”, which took place February 1–6, 2015. The seminar was motivated on the one hand by the high interest in domain-specific languages in academia and industry and on the other hand by the observation that the community is divided into largely disconnected subdisciplines (e.g., internal, external, visual, model-driven). The seminar included participants across these subdisciplines and included overview talks, technical talks, demos, discussion groups, and an industrial panel. This report collects the abstracts of talks and other activities at the seminar and summarizes the outcomes of the seminar.

Seminar February 1–6, 2015 – <http://www.dagstuhl.de/15062>

1998 ACM Subject Classification D.2 Software Engineering, D.3 [Programming Languages] Compilers, D.3.1 [Formal Definitions and Theory] Semantics, F.3.2 Semantics of Programming Languages


Keywords and phrases Internal DSLs, External DSLs, Domain-specific modeling, Extensible languages, Language workbenches, Textual/graph-based/visual languages, Language design, Language implementation techniques

Digital Object Identifier 10.4230/DagRep.5.2.26

Edited in cooperation with Daco Harkes

1 Executive Summary

Sebastian Erdweg
Martin Erwig
Richard F. Paige
Eelco Visser

License  Creative Commons BY 3.0 Unported license
© Sebastian Erdweg, Martin Erwig, Richard F. Paige, and Eelco Visser

Software systems are the engines of modern information society. Our ability to cope with the increasing complexity of software systems is limited by the programming languages we use to build them. Domain-specific languages (DSLs) successfully address this challenge through linguistic abstraction by providing notation, analysis, verification, optimization, and tooling that are specialized to an application domain. DSLs are already ubiquitous in industrial software development with prominent examples such as HTML, SQL, Make, AppleScript, Matlab, or Simulink.



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Domain-Specific Languages, *Dagstuhl Reports*, Vol. 5, Issue 2, pp. 26–43
Editors: Sebastian Erdweg, Martin Erwig, Richard F. Paige, and Eelco Visser



Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There is a wide range of methods and techniques for the development of DSLs. Each of these makes different trade-offs that enable different usage scenarios. After the initial design of a DSL, switching to another approach can be very expensive or even impossible. Therefore, the trade-offs and implications of different approaches must be well understood by practitioners from the beginning. However, there is no clear account of what exactly these trade-offs are; neither in industry nor in academia.

The goal of the proposed seminar was to bring together key representatives from the communities that address DSLs from different perspectives: (1) internal DSLs, (2) external DSLs, (3) domain-specific modeling, (4) extensible languages, (5) graph-based languages, and (6) formal semantics. To enable constructive exchange between seminar participants from different communities, the seminar started with one introductory talk per community by a representative. These introductory talks were essential for raising awareness for each other's discipline, the challenges involved, and the problems already solved.

The first day of the seminar was concluded with a poster session. Before the seminar, the organizers invited each participant to prepare and bring a poster that describes their position with respect to the seminar topic. Many participants followed this invitation or used a flip chart for an impromptu presentation. During the poster session, the participants alternated between presenting their own poster and receiving introductions by others. While the seminar did not feature a separate round of introductions at the beginning of the first day, this did not at all hinder discussion and interaction during the talks prior to the poster session. The organizers of this seminar would like to encourage other organizers to consider a poster session as replacement for an introduction round.

After the community and personal introductions on the first day, the second day featured four talks about the “design history” of four existing DSLs. The presenters reported on how the design of their DSLs began, what features turned out to be good, what features turned out to require revision, and how modifications of the design were formed, decided, and implemented. Beyond reporting on their experience, the four talks provided concrete examples of DSLs that could be referred to by all participants during the remainder of the seminar. Subsequently to the design histories, the seminar featured a session on DSL evaluation followed by an industrial panel on industrial DSL requirements.

In the morning of the third day, the participants had the chance to present their latest research results in lightning talks. These were the only talks during the seminar without precise instructions by the seminar organizers. In total, there were eight lightning talks. We observed a high degree of interaction across communities. In the afternoon most participants joined for the excursion: A hike around Schloss Dagstuhl.

Thursday morning was reserved for four talks on DSL type systems. The four talks illustrated different ways of addressing DSL type systems. From a distinguished metalanguage and to automated mechanization to type-system embedding and attribute grammars. The presented work was not mature enough to allow for a meaningful discussion of benefits and disadvantages of the individual approaches. On Thursday afternoon the participants split into two breakout groups on Language Design Patterns and Name Binding. Some participants of the breakout groups decided to continue exchange and discussion after the seminar. The breakout groups were followed by tool demonstrations, where participants could freely move between demos.

Finally, on Friday morning the seminar ended with a session on establishing a research agenda, that is, relevant research questions that should be addressed by the DSL community. Moreover, the participants found that no new dedicated venue for DSLs needs to be established, because there are sufficiently many venues for DSL research available already.

This report collects the abstracts of the talks, and summarises other activities (including a panel and a discussion on a research agenda). The summaries and abstracts suggest outcomes and potential directions for future scientific research.

2 Table of Contents

Executive Summary

Sebastian Erdweg, Martin Erwig, Richard F. Paige, and Eelco Visser 26

Overview of Talks

A status update on Ensō	
<i>William Cook</i>	31
Type Systems for the Masses: Deriving Soundness Proofs and Efficient Checkers	
<i>Sebastian Erdweg</i>	31
Semantics-Driven Language Design	
<i>Martin Erwig</i>	31
Macros and Extensible Languages	
<i>Matthew Flatt</i>	32
DSL type systems: experiences from using reference attribute grammars	
<i>Görel Hedín</i>	32
Graphical DSLs	
<i>Steven Kelly</i>	32
Model-Driven Grant Proposal Engineering	
<i>Dimitris Kolovos</i>	33
Resugaring: Lifting Evaluation Sequences through Syntactic Sugar	
<i>Shriram Krishnamurthi</i>	33
Policy languages	
<i>Shriram Krishnamurthi</i>	33
An operational semantics for QL	
<i>Peter D. Mosses</i>	34
Semantic Modularity for DSLs	
<i>Bruno C. d. S. Oliveira</i>	34
Evaluating DSLs	
<i>Richard F. Paige</i>	34
Domain Specific Languages in Practice – An Example	
<i>Julia Rubin</i>	34
Through the Looking Glass: A Design History of DSLs for Self-Reconfigurable Robots	
<i>Ulrik Pagh Schultz</i>	35
Streams a la Carte: Extensible Pipelines with Object Algebras	
<i>Yannis Smaragdakis</i>	35
Constraint based language tools	
<i>Friedrich Steimann</i>	36
Language workbenches: textual and projectional	
<i>Tijs van der Storm</i>	36
Interpreter composition	
<i>Laurence Tratt</i>	36

EMF/Ecore-based DSL engineering	
<i>Dániel Varró</i>	37
Incremental queries for DSMLs	
<i>Dániel Varró</i>	37
A Short History of Name Biding in Stratego/XT and Spoofox	
<i>Eelco Visser</i>	37
A Theory of Name Resolution	
<i>Eelco Visser</i>	38
Domain-specific type systems	
<i>Guido Wachsmuth</i>	38
Cognitive Dimensions for DSL Designers	
<i>Eric Walkingshaw</i>	39
Demo Session	
MetaEdit+: Industrial Strength Graphical DSLs	
<i>Steven Kelly</i>	39
Epsilon	
<i>Dimitris Kolovos</i>	39
Diagram Editors – Layout and Change Tracking	
<i>Sonja Maier</i>	39
Incremental model queried in EMF-IncQuery	
<i>Dániel Varró</i>	40
Composition of Languages and Notations with MPS	
<i>Markus Völter</i>	40
The Spoofox Language Workbench	
<i>Guido Wachsmuth and Daco Harkes</i>	40
Working Groups	
Language Design Patterns	40
Open Problems	
Research Agenda Discussion	41
Panel Discussions	
Industrial Panel	
<i>Ralf Lämmel</i>	41
Participants	43

3 Overview of Talks

3.1 A status update on Ensō

William Cook (University of Texas at Austin, US)

Joint work of Cook, William; van der Storm, Tijs

I gave an update on Ensō, which is a experimental workbench for building and integrating interpreted domain-specific specification languages. Ensō is also implemented in itself.

3.2 Type Systems for the Masses: Deriving Soundness Proofs and Efficient Checkers

Sebastian Erdweg (TU Darmstadt, DE)

License © Creative Commons BY 3.0 Unported license
© Sebastian Erdweg

Joint work of Erdweg, Sebastian; Grewe, Sylvia; Mezini, Mira

The correct definition and implementation of non-trivial type systems is difficult and requires expert knowledge, which is not available to developers of domain-specific languages and specialized APIs in practice. We propose an approach that automatically derives soundness proofs and efficient and correct algorithms from a single, high-level specification of a type system. Our approach supports the modularization of the specification and the composition of derived proofs and algorithms in order to scale the underlying verification procedure up to real-world languages and to enable specification reuse for common language features.

3.3 Semantics-Driven Language Design

Martin Erwig (Oregon State University, US)

License © Creative Commons BY 3.0 Unported license
© Martin Erwig

Joint work of Erwig, Martin; Walkingshaw, Eric

Main reference M. Erwig, E. Walkingshaw, “Semantics First! – Rethinking the Language Design Process,” in Proc. of the 4th Int’l Conf. on Software Language Engineering (SLE’11), LNCS, Vol. 6940, pp. 243–262, Springer, 2011; pre-print available from author’s webpage.

URL http://dx.doi.org/10.1007/978-3-642-28830-2_14


URL <http://web.engr.oregonstate.edu/~erwig/papers/abstracts.html#SLE11>

The design of languages is still more of an art than an engineering discipline. Although recently tools have been put forward to support the language design process, such as language workbenches, these have mostly focused on a syntactic view of languages. While these tools are quite helpful for the development of parsers and editors, they provide little support for the underlying design of the languages.

Convention dictates that the design of a language begins with its syntax. We argue that early emphasis should be placed instead on the identification of general, compositional semantic domains, and that grounding the design process in semantics leads to languages with more consistent and more extensible syntax. We demonstrate this semantics-driven design process through the design and implementation of a DSL for defining and manipulating calendars, using Haskell as a metalanguage to support this discussion. We emphasize the importance of compositionality in semantics-driven language design.

3.4 Macros and Extensible Languages


Matthew Flatt (University of Utah – Salt Lake City, US)

License  Creative Commons BY 3.0 Unported license
© Matthew Flatt

Macros and extensible languages naturally implement a DSL as an “internal” or “embedded” DSL. Using the QL survey language as an example, we illustrate its implementation in Racket. We start with an S-expression notation, demonstrate techniques for improving syntax errors and adding type checking, show how to make the language look more “external” by putting its in its own module, and show how non-S-expression syntax can be supported with editing support in DrRacket.

3.5 DSL type systems: experiences from using reference attribute grammars

Görel Hedin (Lund University, SE)

License  Creative Commons BY 3.0 Unported license
© Görel Hedin

Reference attribute grammars support the building of very modular compilers. The technique has been used for implementing complex domain-specific languages with user-defined types and inheritance, like Modelica, as well as general-purpose languages like Java. In this talk, I discuss a number of design strategies that have emerged for using RAGs to implement type systems.

3.6 Graphical DSLs

Steven Kelly (MetaCase – Jyväskylä, FI)

License  Creative Commons BY 3.0 Unported license
© Steven Kelly

Main reference S. Kelly, J.-P. Tolvanen, “Domain-Specific Modeling: Enabling Full Code Generation,” 448 pages, ISBN 978-0-470-03666-2, Wiley, 2008.

URL <http://dsmbook.com/>

The idea of graphical languages is anathema to many working with textual languages – either from lack of familiarity, familiarity with the all-too-common bad examples, or being burned earlier by a failure in their own attempt to use them. This talk will briefly examine the fundamental and empirical evidence for the utility of graphical languages, and in particular how they differ from textual languages for the language user, the language engineer, and the maker of a language workbench.

3.7 Model-Driven Grant Proposal Engineering

Dimitris Kolovos (University of York, GB)

License © Creative Commons BY 3.0 Unported license
© Dimitris Kolovos

Joint work of Kolovos, Dimitris; Matragkas Nicholas; Williams, James; Paige, Richard

Main reference D. S. Kolovos, N. Matragkas, J. R. Williams, R. F. Paige, “Model-Driven Grant Proposal Engineering,” in Proc. of the 17th Int’l Conf. on Model-Driven Engineering Languages and Systems (MODELS’14), LNCS, Vol. 8767, pp. 420–432, Springer, 2014.

URL http://dx.doi.org/10.1007/978-3-319-11653-2_26

During this talk, we will demonstrate the application of Model-Driven Engineering techniques to support the development of research grant proposals. In particular, we will report on using model-to-text transformation and model validation to enhance productivity and consistency in research proposal writing, and present unanticipated opportunities that were revealed after establishing an MDE infrastructure. We will discuss the types of models and the technologies used, reflect on our experiences, and assess the productivity benefits of our MDE solution through automated analysis of data extracted from the version control repository of a successful grant proposal; our evaluation indicates that the use of MDE techniques improved productivity by at least 58%.

3.8 Resugaring: Lifting Evaluation Sequences through Syntactic Sugar

Shriram Krishnamurthi (Brown University – Providence, US)

License © Creative Commons BY 3.0 Unported license
© Shriram Krishnamurthi

Joint work of Pombrio, Justin; Krishnamurthi, Shriram

Main reference J. Pombrio, S. Krishnamurthi, “Resugaring: Lifting Evaluation Sequences through Syntactic Sugar,” in Proc. of the 35th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI’14), pp. 361–371, ACM, 2014.

URL <http://dx.doi.org/10.1145/2594291.2594319>

Syntactic sugar is pervasive in language technology. It is used to shrink the size of a core language; to define domain-specific languages; and even to let programmers extend their language. Unfortunately, syntactic sugar is eliminated by transformation, so the resulting programs become unfamiliar to authors. Thus, it comes at a price: it obscures the relationship between the user’s source program and the program being evaluated.

This presentation motivates the problem, explains it through working examples, and outlines the solution.

3.9 Policy languages

Shriram Krishnamurthi (Brown University – Providence, US)

License © Creative Commons BY 3.0 Unported license
© Shriram Krishnamurthi

A brief survey of policy languages, especially ones used in security and networking.

3.10 An operational semantics for QL

Peter D. Mosses (Swansea University, GB)

License © Creative Commons BY 3.0 Unported license
© Peter D. Mosses

Joint work of Churchill, Martin; Mosses, Peter D.; Sculthorpe, Neil; Torrini, Paolo
Main reference M. Churchill, P.D. Mosses, N. Sculthorpe, P. Torrini, “Reusable components of semantic specifications,” Transactions on Aspect-Oriented Software Development Vol. XII, LNCS, Vol. 8989, pp. 132–179, Springer, 2015; pre-print available from author’s webpage.

URL http://dx.doi.org/10.1007/978-3-662-46734-3_4
URL <http://www.plancomps.org/taosd2015>

We present and discuss an abstract syntax and operational semantics for QL using I-MSOS, which is a highly modular variant of structural operational semantics. We also raise some questions about the intentions of the language designer as expressed in the informal language specification available at <http://www.languageworkbenches.net/wp-content/uploads/2013/11/QL.pdf>.

3.11 Semantic Modularity for DSLs

Bruno C. d. S. Oliveira (University of Hong Kong, HK)

License © Creative Commons BY 3.0 Unported license
© Bruno C. d. S. Oliveira

This talk discusses how to implement modular components for DSLs. The idea is to have collections of language components (abstract syntax and semantics) that can be easily combined and reused in language implementations. Semantic modularity means that such components can, not only, be separately defined, but also be given precise interfaces, be type-checkable and separately compiled. Using Object Algebras, a recently introduced designed pattern, this talk shows how semantic modularity can be achieved in common OO languages such as Java, Scala or C#.

3.12 Evaluating DSLs

Richard F. Paige (University of York, GB)

License © Creative Commons BY 3.0 Unported license
© Richard F. Paige

Introduction to the session on evaluating DSLs. Includes an introduction to a tagging/coding exercise on DSL qualities, and an overview of evaluating DSLs in the context of a course.

3.13 Domain Specific Languages in Practice – An Example

Julia Rubin (MIT – Cambridge, US)

License © Creative Commons BY 3.0 Unported license
© Julia Rubin

In this talk, we present an example of a real-life domain-specific language (DSL). We use this example to highlight problems faced by organizations that use DSLs in practice:

- difficulties to identify a desirable syntax for the language;
- difficulties to modify, extend and evolve the language;
- difficulties to educate new developers in the use of the language;
- the lack of an ecosystem around DSL management operations, e.g., refactoring and dead-code elimination.

3.14 Through the Looking Glass: A Design History of DSLs for Self-Reconfigurable Robots

Ulrik Pagh Schultz (University of Southern Denmark – Odense, DK)

License © Creative Commons BY 3.0 Unported license
© Ulrik Pagh Schultz

Joint work of Bordignon, Mirko; Stoy, Kasper; Christensen, Johan

Main reference U. Schultz, M. Bordignon, K. Stoy, “Robust and reversible execution of self-reconfiguration sequences,” *Robotica*, 29:35–57, 2011.

URL <http://dx.doi.org/10.1017/S0263574710000664>

An overview of my experience in developing domain-specific languages for self-reconfigurable robots, focusing on the main path of development. An analysis of how the development took place is presented, based in part (1) on relating different steps to different sources of inspiration, (2) classifying the kinds of steps that took place and the forces that affected the development, and (3) proposing a list of so-called language design patterns that influenced the development.

3.15 Streams a la Carte: Extensible Pipelines with Object Algebras

Yannis Smaragdakis (University of Athens, GR)

License © Creative Commons BY 3.0 Unported license
© Yannis Smaragdakis

Joint work of Biboudis, Aggelos; Palladinos, Nick; Fourtounis, George; Smaragdakis, Yannis

Main reference A. Biboudis, N. Palladinos, G. Fourtounis, Y. Smaragdakis, “Streams à la carte: Extensible Pipelines with Object Algebras,” to appear in Proc. of the 29th Europ. Conf. on Object-Oriented Programming (ECOOP’15); pre-print available from author’s webpage.

URL <http://cgi.di.uoa.gr/~biboudis/streamalg.pdf>

Streaming libraries have become ubiquitous in object-oriented languages, with recent offerings in Java, C#, and Scala. All such libraries, however, suffer in terms of extensibility: there is no way to change the semantics of a streaming pipeline (e.g., to fuse filter operators, to perform computations lazily, to log operations) without changes to the library code. Furthermore, in some languages it is not even possible to add new operators (e.g., a zip operator, in addition to the standard map, filter, etc.) without changing the library.

We address such extensibility shortcomings with a new design for streaming libraries. The architecture underlying this design borrows heavily from Oliveira and Cook’s object algebra solution to the expression problem, extended with a design that exposes the push/pull character of the iteration, and an encoding of higher-kinded polymorphism. We apply our design to Java and show that the addition of full extensibility is accompanied by high performance, matching or exceeding that of the original, highly-optimized Java streams library.

3.16 Constraint based language tools

Friedrich Steimann (Fernuniversität in Hagen, DE)

License © Creative Commons BY 3.0 Unported license
© Friedrich Steimann

Main reference F. Steimann, “From well-formedness to meaning preservation: model refactoring for almost free,” *Software & Systems Modeling*, 14(1):307–320, 2015.

URL <http://dx.doi.org/10.1007/s10270-013-0314-z>

It is shown how based a single constraint-based specification of static semantics: well-formedness can be checked; bindings can be computed; errors can be fixed; and refactorings can be performed using a constraint solver.

References

- 1 Friedrich Steimann: From well-formedness to meaning preservation: model refactoring for almost free. *Software & Systems Modeling* 14:1 (2015) 307–320.
- 2 Friedrich Steimann, Bastian Ulke: Generic Model Assist. In *Proc. of MoDELS (2013)* 18–34.

3.17 Language workbenches: textual and projectional

Tijs van der Storm (CWI – Amsterdam, NL)

License © Creative Commons BY 3.0 Unported license
© Tijs van der Storm

Main reference S. Erdweg, T. van der Storm, M. Völter, M. Boersma, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, G. D. P. Konat, P. J. Molina, M. Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. A. Vergu, E. Visser, K. van der Vlist, G. H. Wachsmuth, J. van der Woning, “The State of the Art in Language Workbenches,” in *Proc. of the 6th Int’l Conf. on Software Language Engineering (SLE’13)*, LNCS, Vol. 8225, pp. 197–217, Springer, 2013.

URL http://dx.doi.org/10.1007/978-3-319-02654-1_11

Language workbenches are IDEs consisting of meta languages to build languages and IDEs. In this talk I will sketch a brief history of Language Workbenches, summarize current tools and illustrate how they work internally. In particular I’ll distinguish textual language workbenches (based on parsing) from projectional ones (based on structure editing). Concretely, I’ll show the questionnaire DSL in the Rascal workbench. Although language workbenches can greatly improve productivity in developing domain-specific languages (DSLs), there are trade-offs in comparison to internal approaches (embedding, macros, etc.) to DSL implementation.

3.18 Interpreter composition

Laurence Tratt (King’s College London, GB)

License © Creative Commons BY 3.0 Unported license
© Laurence Tratt

Joint work of Tratt, Laurence; Barrett, Edd; Bolz, Carl Friedrich; Vasudevan, Navaneetha

Language composition is typically crude, and synonymous with the concept of “foreign function interface”. In this talk, I show that fine-grained language composition is possible, even down to the level of cross-language variable scoping. By using meta-tracing, we are able to make such compositions perform close to their mono-language peers. This opens up entirely new possibilities for language composition.

3.19 EMF/Ecore-based DSL engineering

Dániel Varró (Budapest University of Technology & Economics, HU)

License © Creative Commons BY 3.0 Unported license
© Dániel Varró

The Eclipse Modeling Framework (EMF) is an industrial technology to define the abstract syntax of domain-specific modeling languages (DSMLs). After defining the metamodel of the language (in a so-called Ecore model) EMF generates a domain-specific libraries to support model manipulation (interfaces and implementations), model persistency, notifications and commands, undo and redo support as well as a tree based model editor and JUnit test cases demonstrating the use of the API. The generated EMF tool forms the basis of supporting more advanced language and editor features with graphical or textual concrete syntax, model validators, incremental graphical views, model transformations and code generators etc.

3.20 Incremental queries for DSMLs

Dániel Varró (Budapest University of Technology & Economics, HU)

License © Creative Commons BY 3.0 Unported license
© Dániel Varró

Main reference Z. Ujhelyi, G. Bergmann, Á. Hegedüs, Á. Horváth, B. Izsó, I. Ráth, Z. Szatmári, D. Varró, “EMF-IncQuery: An integrated development environment for live model queries,” *Science of Computer Programming*, 98(1):80–99, 2105.

URL <http://dx.doi.org/10.1016/j.scico.2014.01.004>

EMF-IncQuery is a framework for defining declarative graph queries over EMF models, and executing them efficiently without manual coding in an imperative programming language such as Java. EMF-IncQuery enables to (1) define model queries using a high level query language (2) execute the queries efficiently and incrementally, with proven scalability for complex queries over large instance models and (3) integrate queries into domain-specific modeling environments to support incremental graphical views, constraint validator, derived features or data binding.

3.21 A Short History of Name Biding in Stratego/XT and Spoofox

Eelco Visser (TU Delft, NL)

License © Creative Commons BY 3.0 Unported license
© Eelco Visser

Highlights of the development of methods for the implementation and specification of name binding in the Stratego/XT transformation framework and the Spoofox Language Workbench.

3.22 A Theory of Name Resolution

Eelco Visser (TU Delft, NL)

License  Creative Commons BY 3.0 Unported license
© Eelco Visser

Joint work of Pierre Neron, Andrew Tolmach, Guido Wachsmuth


Main reference P. Neron, A. Tolmach, E. Visser, G. Wachsmuth, “A Theory of Name Resolution,” in Proc. of the 24th Europ. Symp. on Programming (ESOP’15), LNCS, Vol. 9032, pp. 205–231, Springer, 2015.

URL http://dx.doi.org/10.1007/978-3-662-46669-8_9

We describe a language-independent theory for name binding and resolution, suitable for programming languages with complex scoping rules including both lexical scoping and modules. We formulate name resolution as a two-stage problem. First a language-independent scope graph is constructed using language-specific rules from an abstract syntax tree. Then references in the scope graph are resolved to corresponding declarations using a language-independent resolution process. We introduce a resolution calculus as a concise, declarative, and language-independent specification of name resolution. We develop a resolution algorithm that is sound and complete with respect to the calculus. Based on the resolution calculus we develop language-independent definitions of α -equivalence and rename refactoring. We illustrate the approach using a small example language with modules. In addition, we show how our approach provides a model for a range of name binding patterns in existing languages.

3.23 Domain-specific type systems

Guido Wachsmuth (TU Delft, NL)

License  Creative Commons BY 3.0 Unported license
© Guido Wachsmuth

Joint work of Gabriël Konat, Vlad Vergu, Eelco Visser

This talk starts with a short history of specification techniques for typing rules in the Spoofox language workbench. First, we show how typing rules used to be specified with Stratego rewrite rules. In this approach, declarations are first stored in scoped dynamic rules, before declarations and references are renamed using globally unique identifiers, and types are calculated and checked. Next, we introduce new meta-languages for the declarative specification of name binding rules (NaBL) and type systems (TS). We discuss their core concepts, the underlying incremental analysis framework, and show example specifications for language constructs in domain-specific languages such as QL (questionnaires), SDF3 (syntax definition), and Green-Marl (graph processing).

3.24 Cognitive Dimensions for DSL Designers

Eric Walkingshaw (Oregon State University, US)

License © Creative Commons BY 3.0 Unported license
© Eric Walkingshaw

Joint work of Green, Thomas; Petre, Marian

Main reference T. R. G. Green, M. Petre, “Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework,” *Journal of Visual Languages & Computing*, 7(2):131–174, 1996.

URL <http://dx.doi.org/10.1006/jvlc.1996.0009>

The cognitive dimensions provide a terminology for discussing human factors in programming languages and the tools we use to work with them. This work from the psychology of programming has many applications for DSL designers. They can improve communication by helping us state claims about usability more precisely and to effectively motivate design rationale. They support the design process by providing a way to enumerate common usability issues and by making the tradeoffs between these issues clear. Finally, they support the qualitative evaluation of languages and tools.

4 Demo Session

4.1 MetaEdit+: Industrial Strength Graphical DSLs

Steven Kelly (MetaCase – Jyväskylä, FI)

<http://www.metacase.com/>

4.2 Epsilon

Dimitris Kolovos (University of York, GB)

<http://eclipse.org/epsilon/>

4.3 Diagram Editors – Layout and Change Tracking

Sonja Maier (Universität der Bundeswehr – München, DE)

<http://www.unibw.de/sonja.maier>

In the first part of the demo, we presented a layout framework, which is specifically designed for diagram editors. The key idea is that so-called layout patterns encapsulate certain layout behavior. Its main strengths are that several layout patterns may be applied to a diagram simultaneously, even to diagram parts that overlap and that the layout is continuously maintained during diagram modification.

In the second part of the demo, we presented a framework whose purpose it is to keep track of diagram changes. The key ideas are that all diagram changes are recorded, and that these low-level changes are filtered, aggregated and visualized. The goal is to enable new functionality and to improve usability.

4.4 Incremental model queried in EMF-IncQuery

Dániel Varró (Budapest University of Technology & Economics, HU)

<http://eclipse.org/incquery>

4.5 Composition of Languages and Notations with MPS

Markus Völter (Völter Ingenieurbüro – Stuttgart, DE)

<http://www.jetbrains.com/mps/>

<http://mbeddr.com/>

4.6 The Spoofox Language Workbench

Guido Wachsmuth and Daco Harkes (TU Delft, NL)

<http://metaborg.org/spoofox/>

5 Working Groups

5.1 Language Design Patterns

We met in a breakout group of ca. 15 participants to discuss whether systematic approaches to (domain-specific or general-purpose) programming language design could be communicated in the form of language design patterns. The breakout group was organized by Tillmann Rendel, Ulrik Pagh Schultz, and Eric Walkingshaw. Main topics of discussion included:

- Is a pattern language appropriate to talk about language design?
- Which granularity of language design should the patterns address?
- How would domain-specific patterns differ from general-purpose patterns?
- What are good examples of language design patterns?
- How should language design patterns be evaluated?
- How can the community work together to find and describe language design patterns?

The breakout session resulted in the creation of a mailing list to further discuss these and other questions related to language design patterns or the communication of systematic approaches to language design in general:

- <https://listserv.uni-tuebingen.de/mailman/listinfo/language-design>

Some participants consider to work more on language design patterns, maybe co-author a book on the topic and/or organize a workshop to collaboratively work on language design patterns.

6 Open Problems

6.1 Research Agenda Discussion

There was consensus not to have (or revive) a conference dedicated to DSLs specifically, because existing conferences, such as SLE or GPCE, provide a suitable platform for presenting DSL research. It was suggested to create a repository of DSL success stories that researchers could point to (in papers) to emphasize the importance and impact of DSLs. (The DSM community has this already. The DSM Forum site (dsmforum.org) has 50 published case studies and about 100 publications)

As for research questions that deserved to be studied in the future, the following topics were mentioned:

- Languages with visual syntax, heterogeneous syntax
- Domain-specific type systems
- Composition of DSLs (specific to DSLs)
- Debugging of DSLs and DSL programs
- End-user/domain experts as DSL developers, scientists who can program
- Customizability: evolving one DSL into another
- Integration: opening of tools to provide access for others and hooking tools into other tools without standardization
- User studies: formative and summative evaluation
- Study (repositories of) DSL artefacts: like what language constructs are used

7 Panel Discussions

7.1 Industrial Panel

Ralf Lämmel (University of Koblenz-Landau – DE)

License  Creative Commons BY 3.0 Unported license
© Ralf Lämmel

Summary compiled by the panel moderator.

7.1.1 Acknowledgment

I am grateful to the panelists (listed below) for contributing to the discussion. The design of the panel is mainly due to the Dagstuhl organizers and specifically Martin Erwig, Sebastian Erdweg, Eelco Visser, and Richard Paige.

7.1.2 Objective of the panel

The overall objective of the panel was to discuss the match between academic research on DSL and the priorities in industry. To this end, the panelists were asked about their thoughts on these questions:

- Do the academic approaches to DSL development work for industry?
- What does industry need to get useful DSLs?
- What is the actual practice for DSL development in your industrial setting?

7.1.3 Format of the panel

Each of the five panelists presented a two-minutes opening remark with the help of one slide. The idea was that each panelist would focus on the single most important issue or problem that should be addressed. Possibly, that issue or problem could be related to the questions listed above. The majority of the 90 minutes of panel time were dedicated to discussions while giving equal time to contributions from the audience and responses from the panelists.

As a follow-up to the panel, if possible, the most important issue or problem should be synthesized. However, such a synthesis turned out to be infeasible, as the panelists had somewhat complementary, albeit related priorities.

7.1.4 The panelists' positions

Five panelists were invited on the grounds of their current or recent involvement in DSL-related projects in industry. The panel was moderated by Ralf Lämmel, Software Languages Team, University of Koblenz-Landau. Here is the list of panelists and a short indication of their 'single most important issue or problem', as communicated to and edited by the moderator:

Hassan Chafi, Oracle The important issue is *arriving at successful lifecycles for DSLs and their development*. More specifically, how can DSLs be gradually introduced into existing projects (lots of valuable code already exists and would benefit from DSLs)? What are interoperability models between DSLs and general purpose languages that either host them or glue them to other parts of the system?

Steven Kelly, MetaCase The important issue is *moving from characters to objects*. This entails, for example: i) Don't rebuild understanding from scratch, store the understood structure. ii) Manipulate, co-operate, link on that level, not as characters. iii) Allow free representation and persistent layout, not forced automatic layout iv) Together, these allow scalability in model and team size and complexity, and seem to be significant factors in cases where good graphical DSLs have been shown to be 5-10x faster than programming.

Oleg Kiselyov, Tohoku University The important issue is this: *What exactly is a DSL?* To quote from an email by the panelist: The issue came up at the Shonan seminar which we organized past May. We were talking about some DSL for HPC, and then someone asked: why do we call it DSL? What exactly is 'domain-specific' here? What is the domain? HPC is so broad that the classification is almost useless. The issue is especially blurred for embedded DSLs. How is that different from a library?

Julia Rubin, MIT The important issue is *maintainability*. This entails specific questions like these: How to keep the language current when new requirements emerge or when new technologies are introduced? Also, how to combine domain specific languages and how to keep the language attractive, e.g., to new employees that are unfamiliar with the language? How to build an ecosystem of solutions around DSLs, e.g., for code analysis, refactoring, debugging, etc.?

Markus Völter, Völter Ingenieurbüro The important issue is *scalability*. There are several dimensions. i) Scaling languages and language ecosystems in terms of complexity: more declarative and analyzable language specifications, fewer specs for more language aspects. ii) Scaling Languages, Domains and Audiences: wider ranges of notations and 'language experiences', more stripped down, user-friendly tools. iii) Scaling some aspects of tools: incremental generation for big systems, incremental type checking for non-trivial type system rules, and others.

Participants

- Lennart Augustsson
Standard Chartered Bank –
London, GB
- Hassan Chafi
Oracle Labs – Belmont, US
- William R. Cook
University of Texas – Austin, US
- Sebastian Erdweg
TU Darmstadt, DE
- Martin Erwig
Oregon State University, US
- Matthew Flatt
University of Utah – Salt
LakeCity, US
- Andrew Gill
University of Kansas, US
- Daco Harkes
TU Delft, NL
- Görel Hedin
Lund University, SE
- Steven Kelly
MetaCase – Jyväskylä, FI
- Oleg Kiselyov
Tohoku University – Sendai, JP
- Dimitris Kolovos
University of York, GB
- Shriram Krishnamurthi
Brown University, US
- Ralf Lämmel
Universität Koblenz-Landau, DE
- Sonja Maier
Universität der Bundeswehr –
München, DE
- Peter D. Mosses
Swansea University, GB
- Bruno C. d. S. Oliveira
University of Hong Kong, HK
- Klaus Ostermann
Universität Tübingen, DE
- Richard F. Paige
University of York, GB
- Tillmann Rendel
Universität Tübingen, DE
- Julia Rubin
MIT – Cambridge, US
- Ulrik Pagh Schultz
University of Southern Denmark –
Odense, DK
- Yannis Smaragdakis
National Kapodistrian University
of Athens, GR
- Friedrich Steimann
Fernuniversität in Hagen, DE
- Laurence Tratt
King's College London, GB
- Tijs van der Storm
CWI – Amsterdam, NL
- Dániel Varró
Budapest University of
Technology & Economics, HU
- Eelco Visser
TU Delft, NL
- Markus Völter
Völter Ingenieurbüro –
Stuttgart, DE
- Guido Wachsmuth
TU Delft, NL
- Eric Walkingshaw
Oregon State University, US

