Domain-Specific Languages

Walid Taha
Department of Computer Science
Rice University
Houston, TX 77005 USA
Email: taha@rice.edu

***Abstract*—Recently, there has been a growing interest in what have come to be known as domain-specific languages (DSLs). This paper introduces a definition for DSLs, explains how DSLs can have a far-reaching impact on our lives, and discusses why DSLs are here to stay.**

INTRODUCTION

Computer science is undergoing a revolution today, in which language designers are shifting attention from general-purpose programming languages to so-called domain-specific languages (DSLs) [1], [2], [3]. General-purpose languages like Java, C#, C++, and C have long been the primary focus of language research. The idea was to create one language that would be better suited for programming than any other language. Ironically, we now have so many different general-purpose languages that it is hard to imagine how this goal could be attained. In fact, the stream of such languages seems endless, and we are continually seeing new ones being introduced.

Instead of aiming to be the best for solving any kind of computing problem, DSLs aim to be particularly good for solving a specific class of problems, and in doing so they are often much more accessible to the general public than traditional programming languages. A widely known example of a DSL is Excel, Microsoft's spreadsheet application. Spreadsheets may not be the best language for writing scientific computing codes, for creating three-dimensional animations, or for programming embedded and real-time systems, but they are nevertheless excellent for expressing many important types of calculations. In addition, they require only limited training beyond what is needed to understand the calculation that one actually wants to perform. As a result, Excel is probably the most widely used programming language in the world [4].

There are several seminal papers on the topic, such as [5], [6], [7], [8], [9], [10], that provide a very good introduction for the specialist. This paper, on the other hand, aims to prepare the general reader for the oncoming DSL revolution. In particular, three key questions are addressed:

I) What is a DSL,
II) How DSLs will transform our lives, and
III) Why DSLs are here to stay.

My hope is that this introduction will provide the reader with both a basic appreciation for the potential that DSLs hold and an understanding of how he or she can collaborate with programming-language experts to create a DSL.

I. WHAT IS A DSL

Today, there is no universally accepted definition of what constitutes a DSL. It is easy to recognize specific examples — in addition to Excel, other widely used examples include AutoCAD for architectural design, ProEngineer for mechanical modeling, Verilog for hardware description, and Mathematica for symbolic computing. However, the absence of a more abstract definition for DSLs is problematic. Having such a definition would provide a basis for evaluating DSLs and would guide the development of the principles and methods for designing and implementing new DSLs.

I propose that DSLs can be defined by four key characteristics, as follows:

A) The domain is well-defined and central.
B) The notation is clear.
C) The informal meaning is clear.
D) The formal meaning is clear and is implemented.

The first three characteristics define what constitutes a *jargon*, and the last characteristic tells us when a jargon can be considered a *DSL*. In the rest of this section, we explain each of these characteristics in the context of familiar, everyday domains.

A. The Domain is Well-Defined and Central

There are two qualitatively different ways in which a domain can be defined. The first is when the domain is well-defined mathematically. Examples of such domains include linear algebra and differential equations. The second way is societal. Here, a domain is defined purely by human activity. Examples of such domains range from washing dishes to building nanotubes. Often, we find that there is a group of people who have special expertise in a particular domain, and frequently such *domain experts* will have a special language, or a *jargon*, to communicate ideas relating to their domain. Legal language is a good example of such jargon. Focusing on the jargon of a problem domain rather than on the jargon of a computer implementation is a pervasive characteristic of successful DSLs.

DSL development is a natural candidate for collaboration between domain experts and programming-language experts. While being well-defined mathematically makes a domain more amenable to the development of a DSL, we believe that being defined by human activity is more important for the success of a DSL. Therefore, we advocate a DSL design

strategy that begins with a focus on an area of activity and then moves to finding the mathematical tools that can help serve the purpose of the activity.

B. The Notation is Clear

Notation is the form a language takes as it is transmitted from the source to the target. Notation can take on many forms. In spoken language, it is sound, and in typed language it is sequences of key strokes. Modern musical notation [11] represents another distinct and widely used form. The domain for musical notation is obvious, and musicians are most comfortable using this notation. It is also worth noting that this notation existed independently of computers, and it was then natural to adopt as a way to instruct a computer to generate a particular musical sequence.

Part of the design of a DSL is finding a good notation, and for practical reasons of storage and processing it is often convenient to use symbols that are easy to enter using a keyboard, mouse, or similar input devices. It should be noted, too, that using a keyboard does not limit us to English alphabet. The QWERTY keyboard is easily used for entering written forms of other languages and specialized notations ranging from musical notation to mathematical formulae.

C. The Informal Meaning is Clear

Expressions (utterances) in a DSL are often concise and to the point. For example, musical notation may not be well-suited to express arbitrary ideas. However, it does have a clear and well-defined meaning to trained musicians. Simple examples include traffic lights and traffic signs. Note that while they might have different meanings in different countries, it is important that traffic lights have a well-defined meaning in each. Similarly, notations exist for specifying football game strategies and for dance routines. A key part of what makes these notations work is that they have a clear meaning shared by all those who use them to communicate.

The programming-language literature provides the principles and the aesthetics [12] for achieving clarity of informal meaning, and I believe that we can teach issues relating to principles and aesthetics to a broad audience. Although there is a shortage of such teaching and tutorial materials today, we expect to see many of these developed over the coming years. And it is important to realize that, with or without such materials, successful development of a DSL requires involving both users and programming-language experts throughout the process, including the design of the notation and evaluation of the clarity of informal meanings and the aesthetics of the language.

D. The Formal Meaning is Clear and is Implemented

This feature distinguishes a DSL from a jargon: it means being amenable to rigorous, formal treatment, and as such being well-suited for sensible implementation by a machine. For example, musical notation can be given a formal mathematical meaning by mapping each note to a frequency, a volume, and a duration. While this definition is too simplistic to be acceptable for a musician (e.g., it does not address the issue of producing a reasonable imitation of a musical instrument), it captures the essential idea necessary for achieving such a goal. Assigning such a meaning to notes does not preclude humans from being allowed to assign them a different meaning each time they play them, but for processing by a computer it is essential to assign at least one rigorous meaning.

The programming-languages literature also provides the formal tools [13] needed to deal with the formal meaning and implementation. I believe that the design, implementation, and formal aspects require extensive training. The only way to prepare the workforce of programming-language experts is to teach these formal aspects to computer science students at the undergraduate level.

II. How DSLs Will Transform Our Lives

Because DSLs focus on a particular domain of life rather than on the computer, they are often *much more accessible to the interested layman* than traditional programming languages. This accessibility means that DSLs have the potential to impact our daily lives much more dramatically than any other advance in programming technology.

To give the reader an idea of the impact that DSLs can have on our lives, I will give some examples that shed light on different ways in which DSLs can be useful.

A. The Household

The realm of the household has an abundance of domains that are readily amenable for formalization, and we can save significant time and energy if we are able to design and implement DSLs that help us in this realm. We can envision a world in which we are able to specify our grocery shopping needs and our expectations in terms of maintaining an orderly house or maintaining a garden. However, these may seem like "luxury domains," so we will focus on a more obvious and almost universal domain: cooking. Recipes are essentially a sequence of instructions, and as such they are a classic example of a program. Yet, today it is common for at least one person in each household to spend an hour or more preparing at least one meal every day. Of course, often it can be significantly more than that. Eating prepared meals is an alternative, but it is not without cost — in addition to paying someone else to do this work for us, there is a vast loss of control over the preparation. Preparing one's own food is also healthier because you can control the quality of the ingredients, the amount of salt used, and the fat, protein, nutrients, and portion sizes much more precisely. A DSL that could enable us to implement recipes would save us time, provide us with more control over our diet, and even allow us to perfectly reproduce healthy, gourmet dishes with a snap of a finger.

B. Personal Security

Many factors affect personal security. To illustrate the role that DSLs can have in this realm, we consider the two closely related areas of insurance and privacy. As we consider these

two domains, we point out several parallels between the role that DSLs can play in each of them.

Our insurance and privacy needs are greater today than they have ever been. Financial instruments provide insurance for our cars, homes, belongings, computers, and even our very lives. Yet, insurance policies are formidable documents that are often hard to understand when we sign them, and we often forget the details soon afterward. Thus, when we are confronted with the question of which of our insurance policies (if any) covers a certain type of incident that may have taken place, finding the answer can be difficult. Similarly, electronic commerce and various modes of sharing data have given rise to new concerns about privacy. In the United States, a significant body of law exists to regulate how a company keeping personal information may or may not use or share this information.

Similarly, every utility company, bank, or merchant with which we interact sends us a mailing at least once a year containing their privacy policy, which represents how they take care not to misuse our personal data. Not only is this a lot of material for us to read, close inspection of privacy policies reveals that they often contain ambiguities, gaps, and in some cases inconsistencies [14].

Personalized health records are an area with significant privacy concerns. As Google Health, Microsoft HealthVault, Health Revolution and others come online and attempt to interface and interact with hospital and clinical electronic medical record systems, there will be pressing need for effective management of privacy concerns.

A DSL that could provide clear formal meaning to an insurance policy or a privacy policy would enable mechanical checking, which means that we would be able to give such policies to a computer, ask it to check them for internal consistency, ask it whether a certain event is covered by the policies, or ask it if they satisfy our own preferences for what information we are or are not willing to allow the company to share with others. For privacy policies, such a DSL would also allow us to address another key weakness in current industrial practice: enforcement. In particular, corporations would be able to mechanically check that their work flow and information systems abide by the rules of the policies that they have promised to follow.

C. The Arts

Art is fundamentally about self expression, and a work of art communicates a thought that was not expressible in any other form. Art is full of opportunities for DSLs that can radically change the way artists work. For example, scriptwriters and playwrights constantly compete for the attention of producers around the world. Playwrights often have to work hard to convince producers to take the time to read through their scripts and visualize the imaginary world that they have so carefully and meticulously constructed. This experience would be dramatically different if scripts were in a DSL that we could give to a computer to animate. Granted, it would require that the script carefully explain movements on the stage and how certain parts should be delivered, and the process could be computationally intensive, but it would allow one artist to produce, single handedly, his or her work to the point where he or she could put up a video on YouTube, for example. Once we have reached that point, the world is our audience, and it becomes much easier for artists to independently make the case for the value of their creative works.

Interestingly, while writing this paper I heard Mimi Holloway, Houston's Theatre Southwest producer, saying that "If you see something on a page, it's just not the same as seeing it on stage." This was said in the opening remarks of the Eleventh Annual Festival of Originals. Although each of the five plays presented that evening was exceptional, one wonders how the fate of the rest of the three hundred submissions could have been different if such a DSL had been available to them.

III. Why DSLs are Here to Stay

There are many reasons to believe that we are just opening the floodgates to the idea of DSLs. Here are the ones that I find most compelling:

Codification is a natural, age-old process. A key step in the birth of a DSL is codification, which is the process of taking informal ideas and giving them form. This codification is something that humans have been doing for millennia and will continue to do. It is one of the most important tools that we have for organizing the world around us. Designing a DSL is not fundamentally different from codifying an oral tradition, which is an activity that began as soon as script was devised.

Codification is an iterative process. Codification, even for non-computing domains, tends to be an iterative process. A case in point is the Napoleonic Code [15], [16], which was introduced in the early nineteenth century and is widely considered to be the first successful codification of European law. While this codification was done quickly, it built on numerous prior codifications of European law that had been created over at least half a century. Interestingly, the key features of such a code, often described as "clarity" and "accessibility," are also what we often find in a successful DSL such as Excel or AutoCAD.

As we express bigger ideas, new patterns emerge. Languages are patterns for expressing ideas. Since it is inevitable that we will continue to express more complex and sophisticated ideas, it is therefore also inevitable that new patterns will continue emerge. Such emergent patterns can be found in domains as diverse as dance [17] and cell biology [18].

Conclusion

I hope that this paper has provided the reader with a new perspective on computing and how it can have a broader role in all aspects of our lives through DSLs. The reader is encouraged to try out this new perspective and to contact me with any questions or insights about DSLs.

Acknowledgments

I wish to thank Prof. Hossam M. Fahmy and the organizers of The 2008 IEEE International Conference on Computer

Engineering and Systems (ICCES 2008) for inviting me to give the plenary talk and write this paper.

Kirsten Jones, Prof. Robert (Corky) Cartwright, Prof. Jeremy Gibbons, Dr. Edwin Westbrook, Kapil Dev, and Dr. Abd-Elhamid Taha provided me with detailed comments on an earlier draft of this paper. Cherif Salama kindly assisted me with formatting and proof reading. Mathias Ricken, Dr. Stephen Wong, Dr. John Greiner, and Tony Elam provided me with helpful feedback on an earlier draft.

This work is supported by NSF CAREER award #0747431, NSF EHS award #0720857, and NSF SoD award #0439017.

References

[1] D. Spinellis, "Notable design patterns for domain specific languages," *Journal of Systems and Software*, vol. 56, no. 1, pp. 91–99, Feb. 2001.

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

[3] "IFIP Working Conference on Domain Specific Languages (DSL WC)," 2009, call for papers. Available online at http://www.smart-generators.org/DSLWC.

[4] S. P. Jones, A. Blackwell, and M. Burnett, "A user-centred approach to functions in Excel." *ICFP*, pp. 165–176, 2003.

[5] P. J. Landin, "The next 700 programming languages," *Communications of the ACM*, vol. 9, no. 3, 1966.

[6] D. L. Parnas, "On the design and development of program families," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 1, pp. 1–9, Mar. 1976.

[7] K. J. Gough, "Little language processing, an alternative to courses on compiler construction," *SIGCSE Bulletin*, vol. 13, no. 3, pp. 31–34, 1981.

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

[9] P. Hudak, "Domain specific languages," 1997, available from author on request.

[10] D. M. Weiss and C. T. R. Lai, *Software product-line engineering: a family-based software development process*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[11] Wikipedia, "Musical noation — Wikipedia, the free encyclopedia," 2008, accessed September 18th, 2008. Available online from http://en.wikipedia.org/wiki/Musical_Notation.

[12] B. J. MacLennan, *Principles of Programming Languages: Design, Evaluation, and Implementation*, 3rd ed. Oxford University Press, 1999.

[13] B. Pierce, *Types and Programming Languages*. MIT Press, 2002.

[14] C. A. Brodie, C.-M. Karat, and J. Karat, "An empirical study of natural language parsing of privacy policy rules using the sparcle policy workbench," in *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*. New York, NY, USA: ACM, 2006, pp. 8–19.

[15] Wikipedia, "Napoleonic code — Wikipedia, the free encyclopedia," 2008, accessed September 18th, 2008. Available online from http://en.wikipedia.org/wiki/Napoleonic_Code.

[16] Britannica, "Napoleonic code — Britannica online encyclopedia," 2008, accessed September 18th, 2008. Available online from http://www.britannica.com/EBchecked/topic/403196/Napoleonic-Code.

[17] I. Hagendoorn, "Emergent patterns in dance improvisation and choreography," in *Proceedings of the International Conference on Complex Systems*, 2002.

[18] S. J. Morrison, N. M. Shah, and D. J. Anderson, "Regulatory mechanisms in stem cell biology," *Cell*, vol. 88, pp. 287–298, 1997.

[19] W. Taha and T. Sheard, "MetaML: Multi-stage programming with explicit annotations," *Theoretical Computer Science*, vol. 248, no. 1-2, 2000.

[20] K. Czarnecki1, J. O'Donnell, J. Striegnitz, and W. Taha, "DSL implementation in metaocaml, template haskell, and C++," in *Dagstuhl Workshop on Domain-specific Program Generation*, ser. LNCS, Batory, Consel, Lengauer, and Odersky, Eds., 2004.

[21] S. Fogarty, E. Pasalic, J. Siek, and W. Taha, "Concoqtion: indexed types now!" in *PEPM '07: Proceedings of the 2007 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*. New York, NY, USA: ACM, 2007, pp. 112–121.

[22] S. Ellner and W. Taha, "The semantics of graphical languages," in *PEPM '07: Proceedings of the 2007 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*. New York, NY, USA: ACM, 2007, pp. 122–133.

[23] J. Gillenwater, G. Malecha, C. Salama, A. Y. Zhu, W. Taha, J. Grundy, and J. O'Leary, "Synthesizable high level hardware descriptions: using statically typed two-level languages to guarantee verilog synthesizability," in *PEPM '08: Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*. New York, NY, USA: ACM, 2008, pp. 41–50.

[24] R. Kaiabachev, W. Taha, and A. Zhu, "E-FRP with priorities," in *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*. New York, NY, USA: ACM, 2007, pp. 221–230.

About the Author

Walid Taha is a professor at Rice University, Houston, TX, USA. His interests span programming language semantics, type systems, compilers, program generation, real-time systems, and physically safe computing. His research on DSLs focuses on building tools for rapidly constructing efficient implementations of DSLs [19], [20], [21] and on graphical languages [22]. In collaboration with researchers and practitioners at Intel, Schlumberger, and National Instruments, he has developed DSLs for hardware description [23] and for reactive and real-time systems [24].

Prof. Taha is the principal investigator on a number of National Science Foundation (NSF), Texas Advanced Technology Program (ATP), and Semiconductor Research Consortium (SRC) research projects. He is the principal designer of MetaOCaml, Acumen, and the Verilog Preprocessor (VPP) system. He founded the ACM Conference on Generative Programming and Component Engineering (GPCE), the IFIP Working Group on Program Generation (WG 2.11), and the Middle Earth Programming Languages Seminar (MEPLS). He is the program chair for the IFIP Working Conference on Domain-Specific Languages [3].