Computer Languages, Systems & Structures 44 (2015) 24–47

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Evaluating and comparing language workbenches
Existing results and benchmarks for the future

Sebastian Erdweg [d,*], Tijs van der Storm [a], Markus Völter [e], Laurence Tratt [b], Remi Bosman [f], William R. Cook [c], Albert Gerritsen [f], Angelo Hulshout [g], Steven Kelly [h], Alex Loh [c], Gabriël Konat [i], Pedro J. Molina [j], Martin Palatnik [f], Risto Pohjonen [h], Eugen Schindler [f], Klemens Schindler [f], Riccardo Solmi [i], Vlad Vergu [i], Eelco Visser [i], Kevin van der Vlist [k], Guido Wachsmuth [i], Jimi van der Woning [l]

[a] *CWI, The Netherlands*
[b] *King's College London, UK*
[c] *University of Texas at Austin, USA*
[d] *TU Darmstadt, Germany*
[e] *voelter.de, Stuttgart, Germany*
[f] *Sioux, Eindhoven, The Netherlands*
[g] *Delphino Consultancy, The Netherlands*
[h] *MetaCase, Jyväskylä, Finland*
[i] *TU Delft, The Netherlands*
[j] *Icinetic, Sevilla, Spain*
[k] *Sogyo, De Bilt, The Netherlands*
[l] *Young Colfield, Amsterdam, The Netherlands*

A R T I C L E I N F O

Article history:
Received 16 February 2015
Received in revised form
1 July 2015
Accepted 18 August 2015
Available online 28 August 2015

Keywords:
Language workbenches
Domain-specific languages
Questionnaire language
Survey
Benchmarks

A B S T R A C T

Language workbenches are environments for simplifying the creation and use of computer languages. The annual Language Workbench Challenge (LWC) was launched in 2011 to allow the many academic and industrial researchers in this area an opportunity to quantitatively and qualitatively compare their approaches. We first describe all four LWCs to date, before focussing on the approaches used, and results generated, during the third LWC. We give various empirical data for ten approaches from the third LWC. We present a generic feature model within which the approaches can be understood and contrasted. Finally, based on our experiences of the existing LWCs, we propose a number of benchmark problems for future LWCs.

1. Introduction

Language workbenches, a term popularized by Martin Fowler in 2005 [1], are tools that lower the development costs of implementing new languages and their associated tools (IDEs, debuggers, etc.). As well as easing the development of

[*] Corresponding author.
E-mail address: erdweg@cs.tu-darmstadt.de (S. Erdweg).

http://dx.doi.org/10.1016/j.cl.2015.08.007
1477-8424/© 2015 Elsevier Ltd. All rights reserved.

traditional stand-alone languages, language workbenches also make multi-paradigm and language-oriented programming environments (see e.g. [2,3]) practical.

For almost as long as programmers have built languages, they have built tools to ease the process, such as parser generators. Perhaps the earliest tool which we would now think of as a language workbench was SEM [4], which was later followed by tools such as MetaPlex [5], Metaview [6], QuickSpec [7], and MetaEdit [8], Centaur [9], the Synthesizer generator [10], the ASF+SDF Meta-Environment [11], Gem-Mex/Montages [12], LRC [13], and Lisa [14]. Most of these systems operated on textual languages and were intended to work with formal specifications of General Purpose Languages (GPLs) [15]. Nevertheless, many of them were used to build practical Domain-Specific Languages (DSLs) [16].

Informally, modern language workbenches are often referred to as being textual, graphical, or projectional. Extant textual workbenches like JastAdd [17], Rascal [18,19], Spoofax [20], and Xtext [21] can be seen as successors of the original language workbenches, often making use of advances in IDE or editor technology. Many extant graphical workbenches such as MetaEdit+ [22], DOME [23], and GME [24] were originally developed for box and line style diagrams. Projectional workbenches are a recent addition, with JetBrains MPS [25] and the Intentional Domain Workbench [26] reviving and refining the old idea of syntax directed editors [27], opening up the possibility of mixing textual and non-textual notations.

Since language workbenches have come from industry, it is perhaps unsurprising that many real-world projects have used them. As an indicative sample (in approximate chronological order): the Eurofighter Typhoon used IPSYS's HOOD toolset [28]; Nokia's feature phones [29] and Polar's heart rate monitors [30] used MetaEdit+; WebDSL [31] and Mobl [32] were developed using Spoofax.

In short, not only are the uses for language workbenches growing, but so are the number and variety of the workbenches themselves. One disadvantage of this growing number of systems is that the terminology used and features supported by different workbenches are so disparate that both users and developers have struggled to understand common principles and design decisions. Our belief is that a systematic overview of the area is vital to heal this rift.

The Language Workbench Challenge (LWC) was thus started to promote understanding of, and knowledge exchange between, language workbenches. Each year a language engineering challenge is posed and submissions (mostly, but not exclusively, by the developers of the tools in question) implement the challenge; documentation is required, so others can understand the implementation. All contributors then meet to discuss the submitted solutions. Tackling a common challenge allows a better understanding of the similarities and differences between different workbenches, the design decisions underlying them, their capabilities, and their strengths and weaknesses.

Contributions and structure: In this paper, we describe the challenges posed by the 4 LWC editions run so far (Section 2), before explaining why we focus on the results generated by its third incarnation, LWC'13. We then make the following contributions:

- We establish a feature model that captures the design space of language workbenches as observed in the previous LWCs (Section 3).
- We present and discuss the 10 language workbenches participating in LWC'13 by classifying them according to our feature model (Section 4).
- We present empirical data on 10 implementations of the LWC'13 assignment: a questionnaire DSL (Section 5).
- Based on the experiences from the previous LWCs, we propose benchmark problems to be used in future LWCs (Sections 6.1 and 6.5) and also two examples for evaluating the benchmarks in Section 7.

This paper is an extended version of [33]. The discussion of the various editions of the LWC (at the beginning of Section 2) and the benchmarks (in Sections 6.1 and 6.5) are new in this version.

2. Background

The idea for the LWC came from discussions at the 2010 edition of the Code Generation conference. Since then, four LWCs have been held, each posing a different challenge. We first describe each year's challenges, before explaining why we focus in this paper on data collected from the third LWC. We start out with a note on terminology.

2.1. Terminology

In this paper we use terminology from different areas, including DSL engineering (“program”, “abstract syntax”), model-driven engineering (“metamodel”, “model-to-text”, “model-to-model”), and language-oriented programming (“language extension”). The reason is that the various tools as well as the authors come from this variety of backgrounds. We decided to not try to unify the different terminologies into a single one because doing this well would amount to its own paper. We believe that each of the terms is clear in whatever context it is used in the paper.