An Open Workflow Environment to Support Learning Data Science

BOISVERT, Charles, DOMDOUZIS, Konstantinos and LOVE, Matthew

Published version

BOISVERT, Charles, DOMDOUZIS, Konstantinos and LOVE, Matthew (2016). An Open Workflow Environment to Support Learning Data Science. In: Apprentissage Instrumente de l'Informatique, Font-Romeu (France), 30 January - 2 February 2017.

Repository use policy

An Open Workflow Environment to Support Learning Data Science

Charles Boisvert[1], Konstantinos Domdouzis[2], and Matthew Love[3]

[1] Sheffield Hallam University, City Campus, Howard Street, Sheffield S1 1WB, UK,
c.boisvert@shu.ac.uk
[2] k.domdouzis@shu.ac.uk
[3] m.love@shu.ac.uk

Abstract. The vast majority of visual tools to learn computing focus on imperative and object-oriented programming. This paper outlines a graphical tool and language which is makes functional programming accessible to inexperienced learners, while also supporting open access to the data and executable results for study and deployment. We believe that both the broadening of the range of programming paradigms and the open approach embedded in the tools make the materials valuable for learning.

Keywords: computer science education, open data, data science, workflow, functional programming

1 Introduction

The increasingly complex and broad agenda of 'big data' raises the bar of what our students need to learn and practice. Yet the teaching community is not responding fast to the challenge. The strongest focus of the numerous courses and software tools is learning procedural and object-oriented programming, with multiple visual, block-based environments such as ALICE [3], and studies like those of Kölling [6].

By contrast, for example, the nifty assignments repository of computing assessment ideas ([11], [10]) contains 101 assignments, collected for their quality, but only 7 of these incorporate work with a real data set. So the practice has not yet disseminated to analysing and processing data.

Of particular concern to us, at Sheffield Hallam University's Data Intelligence group, is adapting our tools, teaching methods and resources in order to facilitate access to and process of data by students at any level. We also believe that such facilitation could support the broader public, such as school pupils or citizen groups. Two authors of this paper worked particularly with Open Data with a local advocacy group [8].

To pursue this idea, we are now developing the Open Piping project, an open-source functional programming environment [2] and visual workflow interface for data processing.

2 Open piping: lowering the barriers to data science

Open Piping is a system of "pipes" - of graphical functional programming for SOA and data processing applications. Visual worklow environments are common ([9], [7]), including some in commercial and scientific use ([5], [1]). But in many cases, the value of the tools is limited due to the poor transparency of the processes and technology they implement - sometimes, though not always, deliberately so.

Take the case of the popular - until its end in 2015 - Yahoo pipes [9]. Any member of the public could define, share and execute workflows, but to execute the process so defined on systems of their choice, had to go through a complex, uncertain process of exporting the pipe in JSON, then compiling it with a utility such as pipes2py [4]. When Yahoo support for pipes ended, users had no choice, if they still wished to run their processes, but to recreate them or work through this export, compilation and redeployment in another environment.

2.1 Open by design

Our ambition is to propose a graphical tool that can prolong the availability of open data with a system for user-defined processes, which would include, by design, the transparency and flexibility needed to apply user-defined processes in a range of languages and environments. Open piping aims to be at once:

- *Open*. By which we mean, Open Source; the system's source code is available under the GNU licence. But the notation used to define the process - a simple S-expression defining a function, encoded in JSON for convenience - is itself open. The executable process, defined from the function, can then be made available by transforming it in any of a range of target programming languages.
- *Interoperable*. The openly available, human-readable JSON format for specifying and interchanging S-expressions, ensures the interoperability of the system with any manner of services, such as alternative end-user interfaces, new languages or new process hosting and remote execution tools.
- *Easy to use*. The user interface makes it easy to define workflows and shows clearly the relation between workflow, resulting S-expression, and executable functionality.
- *With resulting processes easy to deploy*. The ability to choose from multiple languages and standards for services and content integration, would facilitate the re-use of user-defined processes in different environments, such as within content-management systems, as web or application widgets, or within a service-oriented architecture.

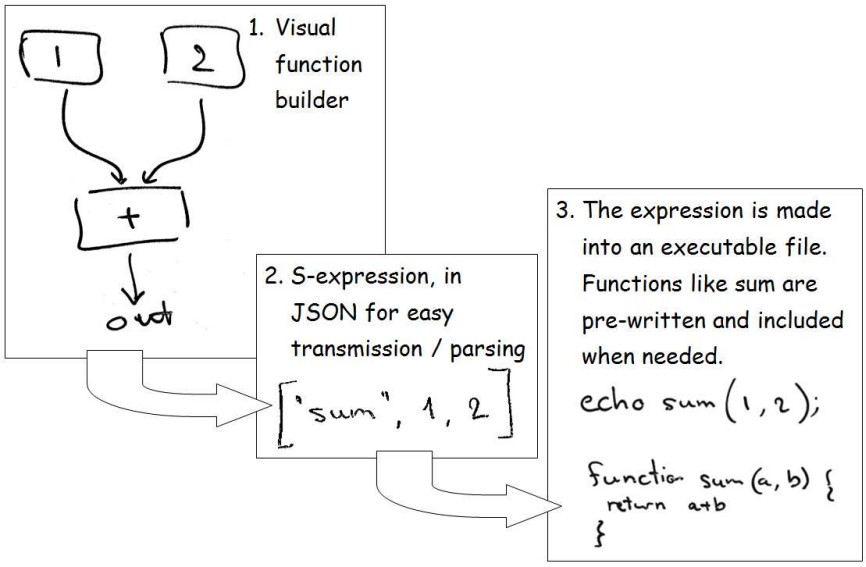
Altogether, these characteristics ensure that users can easily define the processes they want to operate on data, while also retaining control of these processes to use them in new environments.

2.2 Open piping architecture

An end-user would program a service by defining a function in a visual functional language. The language offers access to a set of pre-defined primitive functions, which at once define the primary graphical blocks available to the end-user, provide basic access to processing capabilities, and limit that access to a chosen set.

The user's visually-defined function is translated into an S-expression in JSON. The S-expression can be compiled into an executable function in any number of languages, provided that calls to the primitive functions can be defined and securely executed. Currently this project demonstrates the compilation from the S-expression to JavaScript, but multiple environments common on web server and clients are considered - e.g. JQuery, PHP, node.js, etc. Once compiled, the resulting process can be executed, but could also be deployed in new systems.

Fig. 1. Open piping principle



This architecture has several advantages:

- Limits to processing capabilities are not inherent to the language used, but instead to the environment in which the functions are deployed, for example by setting a processing time limit.
- The execution environment is limited by the set of pre-defined functions, supporting secure remote execution.
- The visual language is loosely coupled to the execution environment, by producing a function definition in an open syntax; this ensures that changes to

the visual interface, to the target language, and to the execution environment are independent.

The tools so far let the user define a function, then traverse the user's tree to produce the matching S-expression. The S-expression is translated in JavaScript and users can run the resulting code, displaying the result.

Functions defined are a small set of mathematical and logic primitives: arithmetic and boolean operations, plus conditional execution and function definition. A set of test S-expressions is provided to test the language and the compilation process.

The language remains limited, in particular as it does not yet support higher-order functions. But it demonstrates how end users can be supported with visual, yet flexible, access to develop in a functional language. We believe there is an opportunity to develop tools that demonstrate a broader range of programming paradigms, and allow learners to experience and compare such different paradigms. Furthermore, we intend to support open access to source code, standards used, executable results, and deployment environments, enabling autonomous learning through accessibility.

References

1. E. H. Beni. A survey on workflow middleware.
2. C. Boisvert. Open piping project software. https://github.com/boisvert/open-piping. Accessed: 2016-10-10.
3. S. Cooper, W. Dann, and R. Pausch. Alice: a 3-d tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, volume 15, pages 107–116. Consortium for Computing Sciences in Colleges, 2000.
4. G. Gaughan. A project to compile yahoo! pipes into python. https://github.com/ggaughan/pipe2py. Accessed: 2016-10-10.
5. D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic acids research*, 34(suppl 2):W729–W732, 2006.
6. M. Kölling. The problem of teaching object-oriented programming. *Journal of Object Oriented Programming*, 11(8):8–15, 1999.
7. D. Le-Phuoc, A. Polleres, G. Tummarello, and C. Morbidoni. Deri pipes: visual tool for wiring web data sources. *)ˆ(Eds.):Book DERI pipes: visual tool for wiring web data sources(2008, edn.)*, 2008.
8. M. Love, C. Boisvert, E. Uruchurtu, and I. Ibbotson. Nifty with data: Can a business intelligence analysis sourced from open data form a nifty assignment? In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, pages 344–349, New York, NY, USA, 2016. ACM.
9. T. O'Reilly. Pipes and filters for the internet. http://radar.oreilly.com/2007/02/pipes-and-filters-for-the-inte.html. Accessed: 2016-10-10.
10. N. Parlante. Nifty assignments. http://nifty.stanford.edu. Accessed: 2016-01-12.
11. N. Parlante, J. Popyack, S. Reges, S. Weiss, S. Dexter, C. Gurwitz, J. Zachary, and G. Braught. Nifty assignments. In *ACM SIGCSE Bulletin*, volume 35, pages 353–354. ACM, 2003.