PolyJuS: A Squeak/Smalltalk-based Polyglot Notebook System for the GraalVM

Fabio Niephaus
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
fabio.niephaus@hpi.uni-potsdam.de

Eva Krebs
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
eva.krebs@student.hpi.uni-potsdam.de

Christian Flach
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
christian.flach@student.hpi.uni-potsdam.de

Jens Lincke
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
jens.lincke@hpi.uni-potsdam.de

Robert Hirschfeld
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
hirschfeld@hpi.uni-potsdam.de

ABSTRACT

Jupyter notebooks are used by data scientists to publish their research in an executable format. These notebooks are usually limited to a single programming language. Current polyglot notebooks extend this concept by allowing multiple languages per notebook, but this comes at the cost of having to externalize and to import data across languages. Our approach for polyglot notebooks is able to provide a more direct programming experience by executing notebooks on top of a polyglot execution environment, allowing each code cell to directly access foreign data structures and to call foreign functions and methods. We implemented this approach using GraalSqueak, a Squeak/Smalltalk implementation for the GraalVM. To prototype the programming experience and experiment with further polyglot tool support, we build a Squeak/Smalltalk-based notebook UI that is compatible with the Jupyter notebook file format. We evaluate PolyJuS by demonstrating an example polyglot notebook and discuss advantages and limitations of our approach.

CCS CONCEPTS

• **Software and its engineering** → **Integrated and visual development environments**; • **Mathematics of computing** → *Exploratory data analysis.*

KEYWORDS

Polyglot Programming, Notebooks, GraalVM, GraalSqueak

ACM Reference Format:
Fabio Niephaus, Eva Krebs, Christian Flach, Jens Lincke, and Robert Hirschfeld. 2019. PolyJuS: A Squeak/Smalltalk-based Polyglot Notebook System for the GraalVM. In *Companion of the 3rd International Conference on Art, Science, and Engineering of Programming (Programming '19), April 1–4, 2019, Genova, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3328433.3328434

Programming '19, April 1–4, 2019, Genova, Italy

ACM ISBN 978-1-4503-6257-3/19/04…$15.00
https://doi.org/10.1145/3328433.3328434

1 INTRODUCTION

Reproducible computational workflows in the form of Jupyter notebooks [3] have become an important tool of the scientific community. These notebooks combine text, program code, and computation results into one document. Since the scientific community uses many different programming languages, especially in fields such as data analysis and machine learning, one can choose from a broad list of languages to use in a notebook. The Jupyter project, for example, started with support for Julia, Python, and R and today supports over 60 additional languages.

But this freedom of choosing a language is limited. Since only one language can be used per notebook, it is hard for scientists to use code written in another programming language. To overcome this limitation, polyglot notebooks [9] allow code cells to be written in different languages. This way, scientists can choose the most appropriate libraries and frameworks per use case. Current polyglot notebook systems, however, come at a cost: each code cell has to import and export data and intermediate results because each cell is executed in its own language-specific execution environment (*notebook kernel*). Serialization can be trivial for simple data structures, but may also require significant programming effort for more advanced use cases. Scenarios that require the modification of the same object graphs from different languages are usually not supported.

To solve this problem, a system is needed that is capable of representing and executing code written in different languages. More importantly, such a system should make the interaction between languages as directly and effortlessly as possible. A notebook backend powered by a system like this allows code cells of a notebook not only to exchange data with each other, but also to re-use behavior and actually work on the same objects in memory.

In this paper, we present an approach for combining a notebook system with a polyglot runtime environment with focus on the programming experience. Based on this, we have implemented PolyJuS, a Squeak/Smalltalk-based notebook system that also demonstrates how tools for live object exploration can further advance the notebook concept. We discuss advantages and limitations of our approach, compare it with related work, and give an outlook for future work.

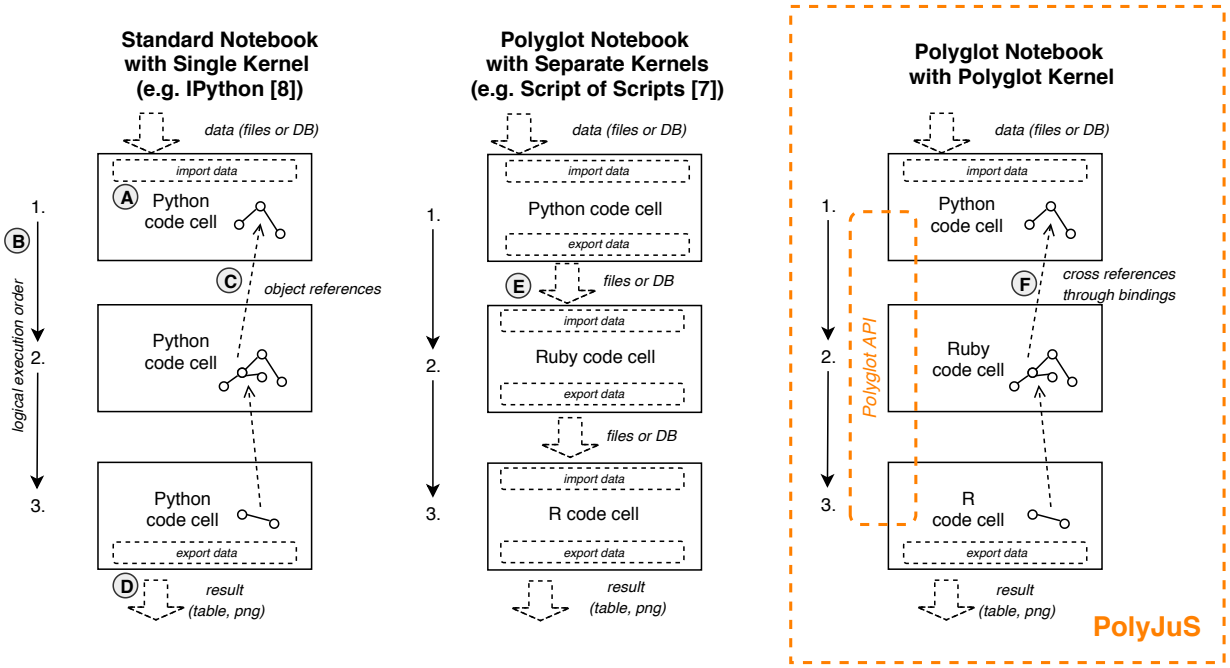

Figure 1: From Standard Notebooks to Polyglot Notebooks.

2 BACKGROUND

In this section, we introduce different notebook systems as well as polyglot runtime environments.

The Jupyter Project. Jupyter notebooks evolved from IPython [7], an interactive Python command shell. These notebooks can consist of code and text cells. A user can choose to evaluate code cells on a language runtime (*kernel*), which often runs on a remote, high spec server to allow computationally intensive workloads. As shown in Figure 1, code cells (A) are executed (B) in one session sharing a common mutable state (C). Inspired by Literate Programming [4], code cells can be annotated with text cells that can contain further instructions, explanations, or discussions of intermediate results, which in turn are often visualized in form of a table or a plot (D). Notebooks can be saved in a JSON-based file format[1] and shared with others. Since source code, text cells, and output cells can be persisted in this format, it is possible to view a shared notebook without having access to a compatible kernel.

Polyglot Notebooks. Sharing data between code cells of a polyglot notebook through data serialization is relatively straightforward when each cell is a functional projection and the overall program structure is a data flow as shown in Figure 1 (E). But when the overall program structure does not fit a functional paradigm, it gets more complicated. Once data is represented in an object-oriented programming system such as Python or Smalltalk, exporting an arbitrary object structure into a form that can be read from another system becomes hard. This might be manageable when the programmer has full control over the code and can design such requirements in the first place, but it becomes nearly impossible when the runtime data (objects) that the user is interested in is produced by external code written in a different language (e.g. a library, code from stackoverflow.com, or from a fellow researcher).

When programmers want to express parts of their program behavior in two or more different programming languages (*polyglot programming*), the different parts have to communicate and share data. They have to be executed in separate systems that can only exchange data by sending messages via Remote Procedure Calls (RPCs) or by passing data through external means (e.g. pipes and files). Both cases require the programs to be very explicit and to make use of external data formats (e.g. JSON, XML, a database, etc.) when handing data from one language to another.

Polyglot Runtime Environments. Polyglot runtime environments, such as GraalVM [10] or Squimera [6], support the execution of multiple programming languages and provide means for seamless language interoperability. For this, these runtimes usually expose some kind of polyglot Application Programming Interface (API) that can be used to invoke code from other languages. Moreover, they often provide tools, such a debugger, as well as a Just-in-time compiler (JIT) (to increase runtime performance) that all work across language boundaries. Additionally, such an environment manages data and objects in the same operating system level process which makes it possible to directly share them across different languages. Common language integration techniques such as RPCs or Foreign Function Interfaces (FFIs), on the other hand, are usually boundaries for tools, often impose overhead in terms of runtime performance, and more importantly require additional work on the user side for sharing data structures between languages.

[1] https://nbformat.readthedocs.io

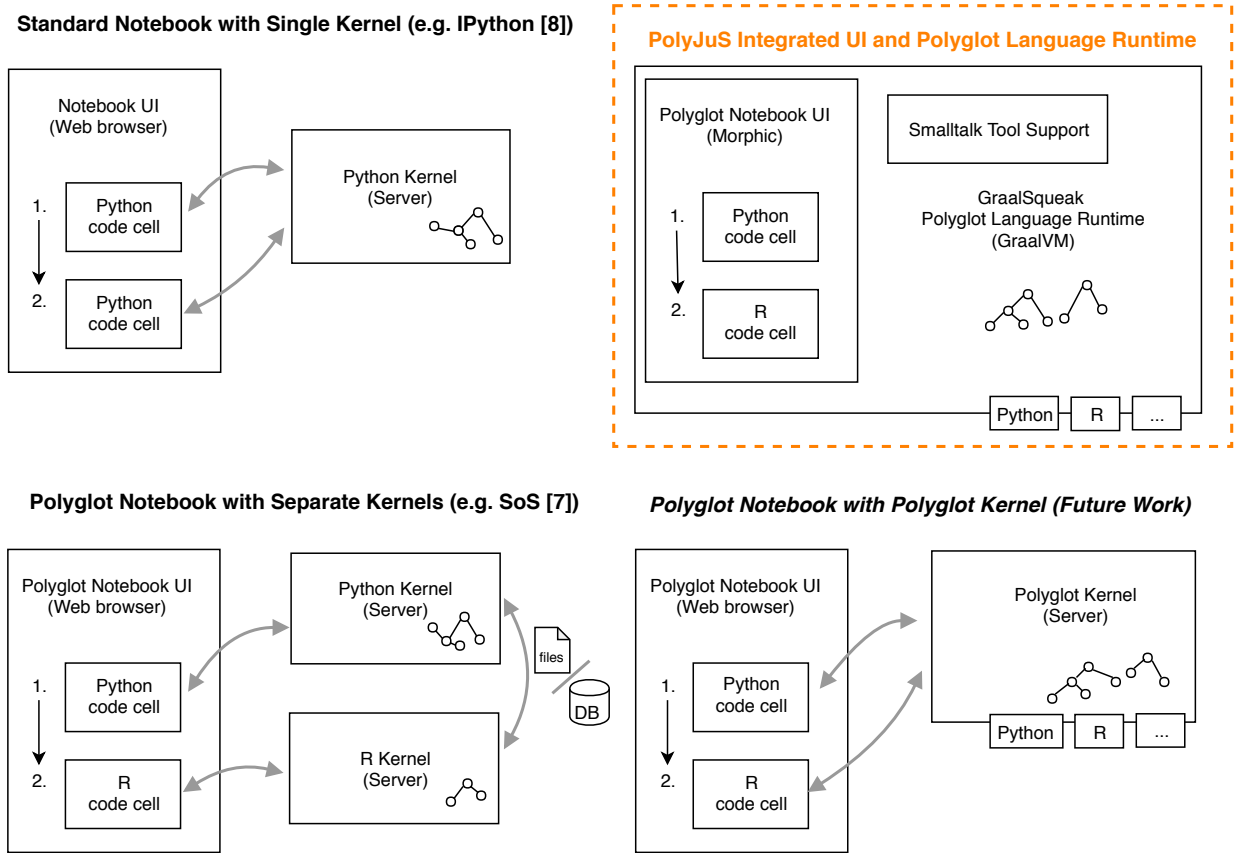

Figure 2: Different notebook system architectures.

3 APPROACH

We propose to execute code of Jupyter-like notebooks on a polyglot runtime environment. A suitable runtime environment must support a sharing mechanism for language interoperability purposes. Object memory can often not be shared directly among different programming languages due to potential name clashes for example. Therefore, we suggest to use a dedicated namespace for sharing objects in the form of a key-value store that is accessible from all languages. To further improve the programming experience, the notebook kernel can hide some of the interaction with the runtime's interoperability API by extending code execution requests appropriately. For example, it could translate a code cell's language selection into the appropriate API call. Additionally, it can prepend common import statements to code, so that the shared namespace or required modules for polyglot programming are available automatically.

A notebook system like this allows users to select a programming language per code cell. In contrast to existing polyglot notebook systems such as SoS notebooks (see Figure 1 (E)), data and objects can be shared across different languages with direct access from all code cells (see Figure 1 (F)). This eliminates not only potential performance overheads imposed by serialization mechanisms. It also reduces the implementation overhead on the side of the user, which in turn improves the programming experience.

Moreover, this experience can further be improved by live object inspection tools, such as the tools of a Smalltalk programming environment. These inspection tools help to shorten feedback loops as they can give valuable insights on the current state of the kernel including all objects. When writing polyglot code, such feedback can be very useful to better understand and coordinate the interaction between languages.

4 IMPLEMENTATION

We have implemented PolyJuS, a polyglot notebook system based on Squeak/Smalltalk and its Morphic user interface (UI) framework. It uses GraalSqueak [5], a Squeak/Smalltalk virtual machine implementation for the GraalVM, as its polyglot runtime environment. Consequently, the prototype does not use any kind of client-server architecture unlike standard Jupyter notebooks (see Figure 2). PolyJuS has full access to GraalVM's polyglot API through GraalSqueak's `PolyglotPlugin`. This also includes GraalVM's polyglot bindings object which PolyJuS integrates as a namespace for sharing objects between languages.

Figure 3 shows a screenshot of PolyJuS. We will discuss the actual purpose of this notebook in more detail in the next section. The main column on the left contains text and code cells. The language of each cell is displayed in the cell's title bar. It is highlighted with a color and can be changed via the cell's context menu. This context

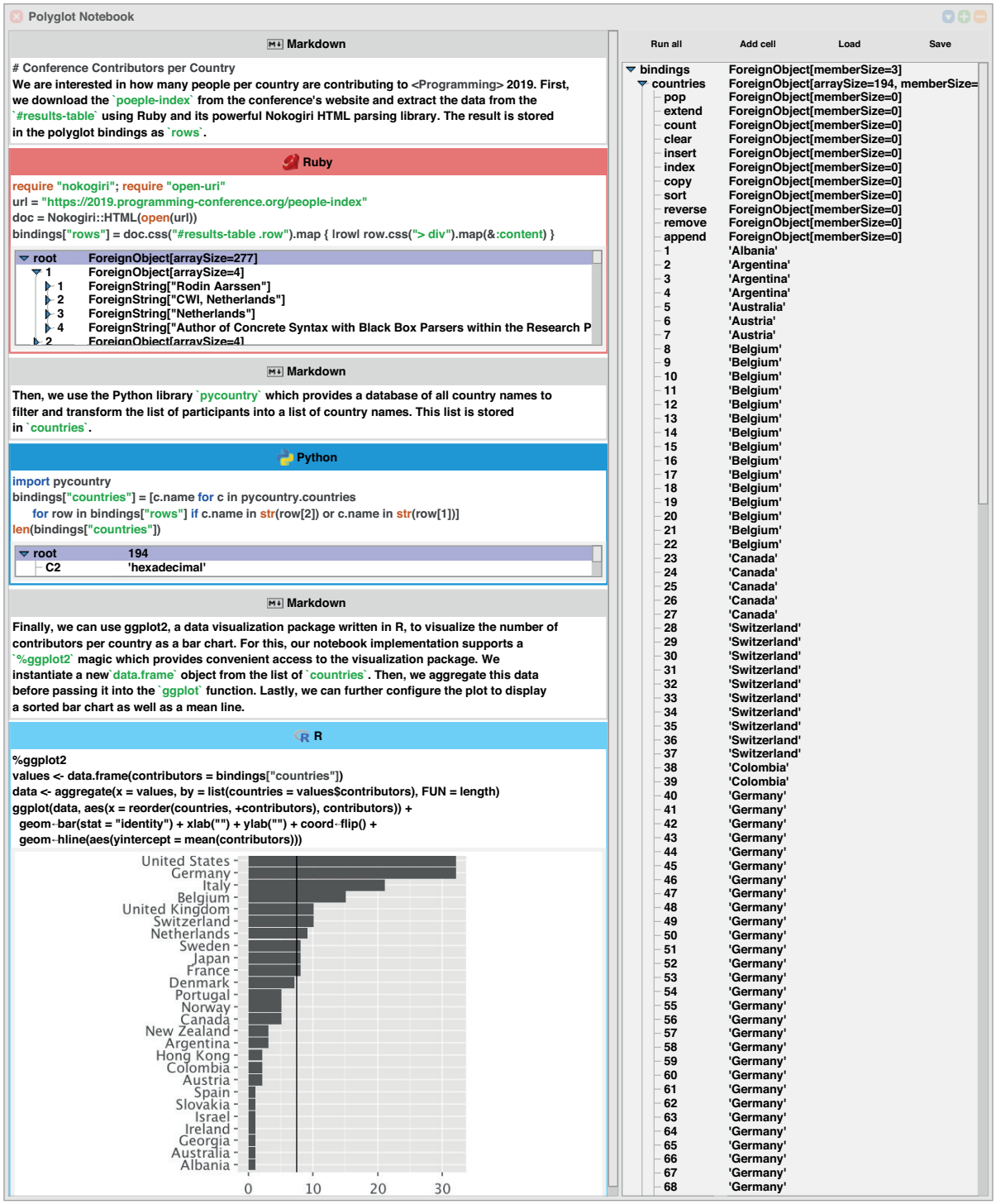

Figure 3: A polyglot notebook for visualizing the number of conference contributors per country using Ruby, Python, and R, created using PolyJuS, and shared on GitHub at https://gist.github.com/fniephaus/d0e95ff0ac2c8c17870a07f5d1d9f898.

menu lists all languages supported by the underlying GraalVM as well as cell actions for executing or removing the cell and for moving it up and down. The Ruby library Rouge is used to provide syntax highlighting in all cells, which was straightforward to integrate in this polyglot environment. Intermediate results of code cells can be inspected in an explorer-like object viewer. The result of the last line in the red Ruby code cell, for example, is an array of arrays of strings. If the result is an image or a morph, it is displayed instead. This is the case in the last code cell whose output is a diagram.

The sidebar on the right contains buttons to run all code cells sequentially, to add a new cell, and to load and save Jupyter notebook files. Additionally, it contains an object explorer for the polyglot bindings object. This way, the user is able to see and inspect all objects shared between the languages at all times.

Since our prototype is implemented in an interactive programming environment, it is always possible to select code in cells, run it, and print the result inline through Smalltalk's *printIt* mechanism, or to open additional inspection tools on objects (*inspectIt* and *exploreIt*). Similarly, objects such as the diagram can be cloned and moved out of a notebook and into other Squeak/Smalltalk applications.

Moreover, our notebook implementation also improves the interaction with the polyglot API. In some languages, for example, the user would normally need to import a polyglot module before using the API. PolyJuS takes care of this if necessary. It also prepends a language-specific import statement to each code execution request to make the polyglot bindings object always accessible from within all code cells. Furthermore, it also integrates ggplot2, a comprehensive data visualization package available in R, with a %ggplot2 magic command. When this magic command is used, PolyJuS adds appropriate glue code to display plots directly in the notebook.

5 EXPERIENCE REPORT

In this section, we report our experience using our PolyJuS prototype for creating polyglot notebooks. As an example, let's assume we want to analyze how many people per country are contributing to a conference, such as ‹Programming›. Given is a table of all contributors which can be found on the conference website.

Although it would be possible to solve this task using a single programming language, it might not be straightforward to implement considering that the data must (a) be downloaded and extracted from an HTML file, (b) cleansed, and (c) visualized as a chart.

Since our polyglot notebooks seamlessly integrate various programming languages, we can always pick the language that we believe fits best for each subtask. For this example, we may choose to solve the task using Ruby, Python, and R. First, we download the HTML file using Ruby and then extract the table of contributors with `nokogiri`, a powerful parsing library for Ruby. The Python library `pycountry` helps us to detect country names within the extracted data. Lastly, we use R's ggplot2 package through the %ggplot2 magic command to visualize the result.

Figure 3 shows the notebook we created with PolyJuS including all sources as well as explanations of its logic. The object explorer below the Ruby code cell helped us to verify that we successfully extracted all 277 rows from the HTML table. The output box of the Python cell, on the other hand, helped us to understand how many countries were detected. Initially, we only searched for country names in the country column. Extending the search to include the affiliation column (`"c.name in str(row[1])"`) yielded about 50 additional data points for example. Furthermore, we were able to inspect all shared objects using the explorer in the sidebar. The screenshot shows the `countries` object, which was created when executing the Python code cell. The explorer confirmed that this is an actual Python list containing 194 country names ready to be aggregated. Since other conferences use the same content management system, it was easy to compare conference contributions. For this, we moved copies of the diagram out of the notebook and created new ones by re-running all cells after changing the URL to a different conference website in the Ruby cell. For sharing the notebook with others, we saved it in the Jupyter notebook file format and uploaded it to GitHub (see caption of Figure 3), which has support for rendering the format in the browser.

6 DISCUSSION

As shown with our experience report, polyglot notebooks give users a much broader choice in terms of libraries and languages they can use, which is especially useful in the data analysis and machine learning domains.

Our prototype uses a Squeak/Smalltalk-based UI which runs on the same execution environment providing language interoperability capabilities. This uncommon setup, however, may be unsuitable for computationally intensive operations, such as distributed map-reduce, as it can only run on a single machine. But this is not a requirement. It would also be possible to use an existing web-based notebook UI in combination with a remote kernel if this kernel supports multiple programming languages.

On the other hand, the Squeak/Smalltalk environment provides tools for live object inspection which allow users to explore intermediate results as well as the state of the notebook's kernel. In the case of PolyJuS, this state includes the namespace of each language as well as the shared polyglot namespace. We believe this is a useful addition to the notebook ecosystem and helps to further reduce feedback loops.

7 RELATED WORK

Script of Scripts (SoS) Notebook [9] is a multi-language data analysis environment that supports multiple connections to different Jupyter kernels at the same time. Our approach, on the other hand, uses a single polyglot kernel. This avoids data synchronization between different kernels as data and objects can directly be passed to other languages without serialization overhead.

BeakerX [8] is a collection of Jupyter language kernels for JVM languages, Python, and JavaScript. BeakerX supports polyglot notebooks allowing different languages for cells. Similar to our approach, the cells can communicate through accessing a shared "beakerx" object. This sharing mechanism, however, is limited to primitive values, arrays, and dictionaries that can be serialized to JSON. This "Autotranslation" feature is a performance bottleneck at the moment and only supports to transfer a few megabytes of data from

each cell to another[2]. Our PolyJuS prototype, on the other hand, allows exchange of all kinds of objects between languages.

Galaaz [1] is a system that integrates functions and packages from R into the Ruby programming language, so that Ruby can be used for scientific computing and machine learning. Similar to our prototype, it has special support for the ggplot2 package and is designed to run on top of GraalVM.

The prototype editor Eco [2] lets developers write composed programs using Python, HTML, and SQL. For this purpose, it uses *language boxes* for code written in the different languages. Compared to polyglot notebooks, these boxes can be nested while the order of execution is predefined by the structure of a program.

8 CONCLUSION AND FUTURE WORK

We presented an approach for combining a Jupyter notebook-like system with a polyglot runtime environment allowing users to use multiple programming languages in the same notebook. Compared with existing notebook systems, our PolyJuS prototype supports language interoperability through GraalVM's polyglot API which allows direct exchange of objects between languages. Additionally, it improves the polyglot programming experience through an enhanced interaction with the polyglot API and through live object inspection tools known from Smalltalk.

In the future, we plan to create more notebook examples analyzing larger datasets than we used in our previous example. To further reduce the cognitive complexity when working with objects from different languages, we want to look into ways to map commonly used APIs automatically across languages. Additionally, it would be interesting to build a GraalVM-based Jupyter kernel that, as shown in Figure 2, provides the same grade of language interoperability as PolyJuS for existing web-based notebook UIs.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of Oracle Labs[3], HPI's Research School[4], and the Hasso Plattner Design Thinking Research Program[5].

REFERENCES

[1] Rodrigo Botafogo. 2018. Ruby Plotting with Galaaz: An example of tightly coupling Ruby and R in GraalVM. https://medium.com/p/520b69e21021

[2] Lukas Diekmann and Laurence Tratt. 2014. Eco: A Language Composition Editor. In *Proceedings of the 7th International Conference on Software Language Engineering*, Benoît Combemale, David J. Pearce, Olivier Barais, and Jurgen J. Vinju (Eds.). Springer International Publishing, 82–101. https://doi.org/10.1007/978-3-319-11245-9_5

[3] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In *ELPUB*. 87–90. https://doi.org/10.3233/978-1-61499-649-1-87

[4] D. E. Knuth. 1984. Literate Programming. *Comput. J.* 27, 2 (01 1984), 97–111. https://doi.org/10.1093/comjnl/27.2.97

[5] Fabio Niephaus, Tim Felgentreff, and Robert Hirschfeld. 2018. GraalSqueak: A Fast Smalltalk Bytecode Interpreter Written in an AST Interpreter Framework. In *Proceedings of the 13th Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems (ICOOOLPS '18)*. ACM, New York, NY, USA, 30–35. https://doi.org/10.1145/3242947.3242948

[6] Fabio Niephaus, Tim Felgentreff, Tobias Pape, Robert Hirschfeld, and Marcel Taeumel. 2018. Live Multi-language Development and Runtime Environments. *The Programming Journal* 2, 8 (2018). https://doi.org/10.22152/programming-journal.org/2018/2/8

[7] Fernando Perez and Brian E. Granger. 2007. IPython: A System for Interactive Scientific Computing. *Computing in Science Engineering* 9, 3 (May 2007), 21–29. https://doi.org/10.1109/MCSE.2007.53

[8] Two Sigma Open Source, LLC. 2019. BakerX. http://beakerx.com

[9] Chris Wakefield, Di Du, James Melott, John N Weinstein, Jun Ma, Yulun Chiu, Bo Peng, Gao Wang, and Man Chong Leong. 2018. SoS Notebook: an interactive multi-language data analysis environment. *Bioinformatics* 34, 21 (05 2018), 3768–3770. https://doi.org/10.1093/bioinformatics/bty405

[10] Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko. 2013. One VM to Rule Them All. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward! 2013)*. ACM, New York, NY, USA, 187–204. https://doi.org/10.1145/2509578.2509581

[2] https://github.com/twosigma/beakerx/blob/1deffe1996a458ca1df5c6e72428c8171c06ce6d/doc/groovy/GeneralAutotranslation.ipynb (accessed 2019-01-30)

[3] https://labs.oracle.com/

[4] https://hpi.de/en/research/research-school.html

[5] https://hpi.de/en/dtrp/