Towards A Squeak/Smalltalk-based Python IDE

An Interpreter-level Integration of Python with Smalltalk

Fabio Niephaus
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
fniephaus@acm.org

ABSTRACT

In this paper, we present how we integrated Python with a Smalltalk environment on interpreter level in order to be able to reuse concepts and tools from Smalltalk for Python development.

CCS CONCEPTS

• **Software and its engineering → Integrated and visual development environments**; *Interpreters*;

KEYWORDS

Smalltalk, Python, IDE, interpreters, debugging

ACM Reference format:
Fabio Niephaus. 2017. Towards A Squeak/Smalltalk-based Python IDE. In *Proceedings of Programming '17, Brussels, Belgium, April 03-06, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3079368.3079370

1 INTRODUCTION

The Smalltalk programming language is one of the few languages that comes with its own development environment [6]. It is largely implemented in itself, which enables many features that are missing in other programming languages and integrated development environments (IDEs).

One example of such a feature is Smalltalk's interactive debugger, which allows stop-edit-continue programming at runtime. In contrast, Python's default debugger PDB [15] uses traditional breakpoints to stop-and-inspect the runtime. It does not allow developers to modify program code and continue afterwards, or to interrupt the program at arbitrary points. Python IDEs provide UI-based debugging tools which can be more convenient, but their functionality is ultimately a graphical layer over PDB.

We believe that the advantages of Smalltalk's tools, such as the debugger, as well as the ability to rapidly build new tools would also be beneficial for developers working with other programming languages. We intend to lay the groundwork for this effort by demonstrating a mechanism to integrate other dynamic, object-oriented languages with Smalltalk, using Python as our prototype. In our design, we integrate Python and Smalltalk at an equal level rather than one running on top of the other. Not only will this result in better performance, but more importantly it will allow us to adopt Smalltalk tools, so that they can be used for Python development. On the other hand, this also makes thousands of Python packages available to the Smalltalk community, including popular and mature packages such as NumPy [10], Django [7], or scikit-learn [11].

With our work, we aim to make the following contributions:

(1) Demonstrate an approach to deeply integrate a foreign programming language with Smalltalk;
(2) Lay the groundwork for adapting Smalltalk concepts and tools for Python development;
(3) Allow the Smalltalk community to easily reuse Python libraries and frameworks.

2 IMPLEMENTATION

Our approach is based on interpreter composition of RPython-based virtual machines (VMs) [1, 2]. We use RSqueak/VM [4] for Squeak/Smalltalk [8] and PyPy [13] for Python. Since we want to be able to interact with the Smalltalk environment while a Python program is running, we need to find a way to run the two interpreter loops concurrently.

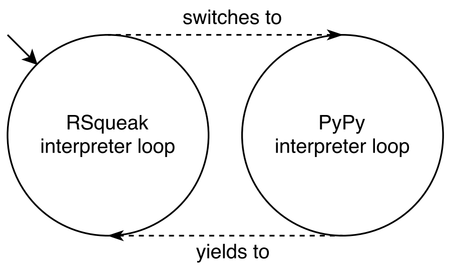

Figure 1: The VM switches to the Python interpreter loop whenever a Smalltalk-level Python process is scheduled.

Smalltalk has a concept of processes which can be scheduled dynamically [6]. Figure 1 shows how we leverage this and integrate the execution of Python bytecodes with a Smalltalk-level process, leaving the decision when to run more Python bytecodes up to the Smalltalk scheduler. This scheduler uses priority-based round-robin scheduling [6] to run Smalltalk processes, and therefore ensures that Smalltalk's standard processes, such as the UI process, are being scheduled alongside with Python-executing processes. This keeps the Smalltalk environment responsive and even allows us to interrupt a Python process to inspect it from Smalltalk.

In a second step, we introduce a special `PythonObject` class in Smalltalk and add appropriate primitives to the VM in order to be able to interact with and send messages to Python objects. From now on, we can execute Python code from Smalltalk by calling a `primEval:filename:mode:` method, which, just like Python's `compile()` built-in, expects Python source code, a filename, and a mode ("eval", "exec", or "single"). The result of such a call is either a `PythonObject`, or a Smalltalk object in case the VM is able to automatically convert a primitive data type. When sending a message to a `PythonObject`, the VM first attempts to call a corresponding Python method or to get a Python attribute. If this was unsuccessful, it performs a normal Smalltalk lookup. This means that `PythonObjects` are hybrid objects, because they can have both, methods written in Python and in Smalltalk. As an example for the interaction with `PythonObjects`, calling

```
(PythonObject
  primEval: 'dict(x=123, y=456)'
  filename: '<string>'
  mode: 'eval') __getitem__: 'x'
```

results in a SmallInteger 123 in Smalltalk, and is equivalent to calling `dict(x=123, y=456).__getitem__('x')` in Python.

With this, we can start to adopt and build new tools in Smalltalk to control and modify a Python program at runtime as well as use Python libraries and objects in Smalltalk applications.

3 RELATED WORK

There are several implementations of programming languages which run on top of another language, such as JRuby [9], which is a Ruby implementation on top of Java. Moreover, the idea to compose interpreters in RPython is also not a new one [2, 3]. However, in these approaches debuggers and other tools for the foreign language cannot share runtime concepts and need to be built from scratch. The Helvetia [12] project aims to make Smalltalk tools reusable, but targets embedded languages. Eco [5] is a language composition editor which allows to write application in multiple languages at the same time, but does not execute them.

4 FUTURE WORK

In the following months, we are going to adopt Smalltalk tools, such as the debugger, the system browser, or the test runner, so that they can be used for Python development. Moreover, adapted versions of the workspace and inspection tools would allow live, interactive exploration of Python objects. In addition, we want to evaluate whether building application-specific tools in Smalltalk, for example using Vivide [14], translates to other languages. We have already implemented simple applications which reuse the Python standard library. But it would also be interesting to see a Smalltalk project using more powerful Python packages such as scikit-learn in order to further improve the interaction between the two languages. We also want to generalize our approach and apply it to other programming languages, such as integrating Ruby, to demonstrate that the design is not limited to Python. Further on, we want to investigate if and how we can integrate languages with other programming paradigms, such as Prolog.

5 CONCLUSION

In this paper, we demonstrated how we designed and implemented a VM with support for Smalltalk and Python. Since our approach is based on interpreter composition, we only compromise little performance for the ability to control the execution of Python code from Smalltalk. With this prototype VM, we can start adopting Smalltalk tools, such as its interactive debugger, for Python development.

REFERENCES

[1] D. Ancona, M. Ancona, A. Cuni, and N. D. Matsakis. Rpython: A step towards reconciling dynamically and statically typed oo languages. In *Proceedings of the 2007 Symposium on Dynamic Languages*, DLS '07, pages 53–64, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-868-8. doi: 10.1145/1297081.1297091. URL http://doi.acm.org/10.1145/1297081.1297091.

[2] E. Barrett, C. F. Bolz, and L. Tratt. Unipycation: A case study in cross-language tracing. In *Proceedings of the 7th ACM workshop on Virtual machines and intermediate languages*, pages 31–40. ACM, 2013.

[3] E. Barrett, C. F. Bolz, and L. Tratt. Approaches to interpreter composition. *Computer Languages, Systems & Structures*, 44:199–217, 2015.

[4] C. F. Bolz, A. Kuhn, A. Lienhard, N. D. Matsakis, O. Nierstrasz, L. Renggli, A. Rigo, and T. Verwaest. *Back to the Future in One Week — Implementing a Smalltalk VM in PyPy*, pages 123–139. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-89275-5.

[5] L. Diekmann and L. Tratt. Eco: A language composition editor. In *International Conference on Software Language Engineering*, pages 82–101. Springer, 2014.

[6] A. Goldberg and D. Robson. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983. ISBN 0-201-11371-6.

[7] A. Holovaty and J. Kaplan-Moss. *The definitive guide to Django: Web development done right.* Apress, 2009.

[8] D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay. Back to the future: the story of squeak, a practical smalltalk written in itself. In *ACM SIGPLAN Notices*, volume 32, pages 318–326. ACM, 1997.

[9] C. O. Nutter, T. Enebo, N. Sieger, O. Bini, and I. Dees. *Using JRuby: Bringing Ruby to Java.* Pragmatic Bookshelf, 1st edition, 2011. ISBN 9781934356654.

[10] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[12] L. Renggli, T. Gîrba, and O. Nierstrasz. Embedding languages without breaking tools. In *European Conference on Object-Oriented Programming*, pages 380–404. Springer, 2010.

[13] A. Rigo and S. Pedroni. Pypy's approach to virtual machine construction. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 944–953, New York, NY, USA, 2006. ACM. ISBN 1-59593-491-X. doi: 10.1145/1176617.1176753. URL http://doi.acm.org/10.1145/1176617.1176753.

[14] M. Taeumel, B. Steinert, and R. Hirschfeld. The vivide programming environment: connecting run-time information with programmers' system knowledge. In *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*, pages 117–126. ACM, 2012.

[15] G. van Rossum. *Python Library Reference.* Centrum voor Wiskunde en Informatica (CWI), 1995.