Bacatá: a generic notebook generator for DSLs

Mauricio Verano Merino
TUe
The Netherlands
m.verano.merino@tue.nl

Jurgen J. Vinju
CWI / TUe
The Netherlands
Jurgen.Vinju@cwi.nl

Tijs van der Storm
CWI / University of Groningen
The Netherlands
storm@cwi.nl

Abstract

Interactive notebooks, such as provided by the Jupyter platform [2], are gaining traction in scientific computing, data science, and machine learning. Developing a Jupyter kernel machinery for a new language, however, requires considerable effort. In this extended abstract, we present Bacatá, a language-parametric bridge between Jupyter and the Rascal language workbench [3]. Reusing existing language components, such as a parsers, interpreters, Read-Eval-Print Loop (REPLs) and autocomplete, Bacatá generates a Jupyter kernel machinery so that the DSL can be used in notebook form. We sketch the architecture of Bacatá and demonstrate it in action using a DSL for image processing, called Amalga.

1 Introduction

Notebooks are gaining traction in scientific computing, data science, and machine learning, due to the capabilities they provide in terms of development, documentation, execution and results visualization. Notebooks support live code, results computation, and narrative text, all in the same rich media document. Consequently, they are often used for computational storytelling: explaining languages, libraries, algorithms, etc. in an interactive way.

Jupyter [2] is the most popular platform for constructing notebook applications. Jupyter notebooks platform can be divided into two main parts: the web server and the notebook document. The web server contains different *kernels* (interpreters), which runs and introspect user's code. Those kernels enable the communication between the platform and the language.

Constructing a Jupyter kernel for a new language is a time-consuming and error-prone task. It requires understanding and handling messages through the implementation of the Jupyter's wire protocol and connecting it to a language-specific read-eval-print-loop (REPL) which computes results, displays errors, and produces rich media representations. In this extended abstract, we present Bacatá, a language-parametric framework for creating Jupyter notebooks with minimal effort. Bacatá acts as a bridge between Jupyter and languages developed within the Rascal language workbench [3]. As a result, language components that are already developed within Rascal can be reused to drive the generated Jupyter kernel.

DSLDI'2017, October 2017, Vancouver, Canada
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

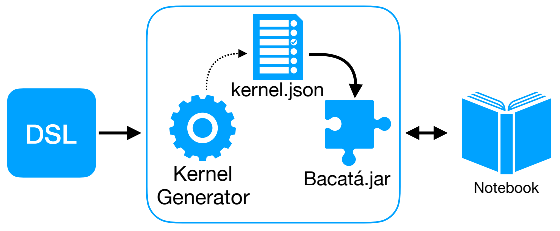


Figure 1. High-level overview of Bacatá

We illustrate Bacatá in the context of a DSL for image processing algorithms (i.e. Amalga). We expect Bacatá to be a first step towards extending the editor services provided by language workbenches [1] to include interactive notebooks.

2 Bacatá

Bacatá provides a bridge between the Jupyter notebook platform and Rascal [3]. Rascal is a functional meta-programming language for source code analysis and transformation, and language workbench for DSL development. It has been successfully used for developing DSLs in the domains of finance, digital forensics, and game economies. Bacatá extends Rascal's current set of IDE services with support for interactive notebooks.

Figure 1 summarizes Bacatá's architecture. A DSL implementation is the input to a Jupyter kernel generator, which implements how the generic Bacatá executable should process requests and responses from and to the Jupyter notebook server.

To create a notebook interface using Bacatá, the language developer needs to perform the following high-level steps:

1. Define a Rascal module *M* containing a function which returns a `REPL` value (see below).
2. Call the function `genKernel`, with the name and location of module *M*, and other configuration parameters. This generates a JSON file (i.e. kernel.json) containing the proper invocation to Bacatá when it is loaded into the Jupyter platform.
3. Call the function `startNotebookServer` to start the Jupyter Web server to load the kernel given as parameter (e.g. *M*); Bacatá will start Rascal with *M*, and from then on relay communication from Jupyter to the proper handlers defined by the obtained `REPL` value.

The Bacatá executable is language-parametric; all language-specific configuration is defined in Rascal through the `REPL` datatype returned by `main`, which is defined as follows:

```
data REPL = repl(
  Result(str), // the REPL evaluator;
  Completion(str,int), // the completor
  Mode // the syntax highlighting mode
);
```

The first argument is a function representing the interpreter; it receives the entered source string and returns a `Result`. A result consists of an HTML element returned as a string and/or a set of error messages. The second argument is a function for tab-completion: given the current input and the position of the cursor, it returns a list of suggestions which will be shown in the editor. Finally, the last argument is a value describing a CodeMirror syntax-highlighting mode for use in the Jupyter code editor[1]; such modes can optionally be automatically derived from a Rascal context-free grammar.

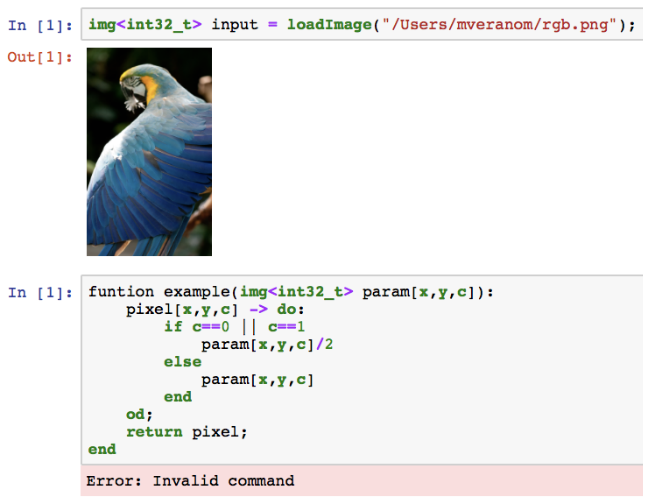
Apart from communicating between the evaluator and completor on the one hand and the Jupyter server, on the other hand, Bacatá also intercepts the standard output and error streams of the interpreter and connects them to their corresponding Jupyter streams. This means that an existing, console-based REPL for a DSL, can be reused as is in the context of Jupyter.

3 Case Study: Amalga

Amalga is a DSL for image processing algorithms which have been implemented in Rascal. The aim of Amalga is to abstract some of the programming complexity for people who need to write image processing algorithms, but lack a background in computer science. These algorithms have to satisfy some restrictions in terms of performance and portability. Amalga captures these constraints explicitly, and translates high-level algorithms to Halide [4], a language for image processing and computational photography. Halide is an embedded DSL in C++ which produces highly optimized code for multiple platforms.

We have implemented a notebook interface for Amalga using Bacatá. Commands entered in the notebook are compiled to C++, then to native code, and executed in the background. If the output of execution is an image, it is shown directly in the browser by having the "interpreter" (in the `REPL` data type) returning an `img` tag as a result, loading the image from disk.

Figure 2 shows an example of the Jupyter notebook generated using Bacatá for Amalga. As it can be seen in the figure, some of the commands produce a rich output, but some others do not provide an explicit output (i.e. they produce intermediate steps). Moreover, Bacatá also supports the display of error messages. The type of output displayed by



```
In [1]: img<int32_t> input = loadImage("/Users/mveranom/rgb.png");
Out[1]:
```

```
In [1]: funtion example(img<int32_t> param[x,y,c]):
            pixel[x,y,c] -> do:
                if c==0 || c==1
                    param[x,y,c]/2
                else
                    param[x,y,c]
                end
            od;
            return pixel;
        end
        Error: Invalid command
```

Figure 2. Example of an Amalga's notebook

each command is entirely implemented in the Amalga REPL module.

4 Conclusion

Interactive notebooks are becoming a popular way to interact with programming languages. In this extended abstract, we have presented Bacatá, a language-parametric framework for defining Jupyter kernels for DSLs defined in the Rascal language workbench. Bacatá abstracts the low-level wire protocol of Jupyter by offering a high-level registration API that can be used directly from within Rascal. As result, existing language artifacts (grammars, completors, REPLs etc.) can be reused with only little glue code. We expect notebooks to be an integral part of a language workbench's interactive language services.

As future direction for Bacatá, we plan to explore better code completion by allowing the REPL interpreter to share its state with the completor function and provide support for rich, interactive output and visualization through the embedding of Rascal's Salix GUI library[2].

References

[1] Sebastian Erdweg, Tijs van der Storm, Markus Völter, and Laurence Tratt et al. 2015. Evaluating and comparing language workbenches. *Computer Languages, Systems & Structures* 44 (2015), 24–47.

[2] Jupyter. 2015. Jupyter notebook. (2015). Retrieved July 24, 2017 from http://jupyter.org

[3] Paul Klint, Tijs van der Storm, and Jurgen Vinju. 2011. EASY Meta-programming with Rascal. In *GTTSE III*. 222–289.

[4] Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. 2012. Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines. *ACM Trans. Graph.* 31, 4, Article 32 (July 2012), 12 pages. https://doi.org/10.1145/2185520.2185528

[1] https://codemirror.net/demo/simplemode.html

[2] https://github.com/cwi-swat/salix