Publishing Your Prototype Tool on the Web

PUPTOL, a Framework

Axel Belinfante and Arend Rensink

University of Twente, P.O.Box 217, NL-7500 AE Enschede, the Netherlands
{Axel.Belinfante,rensink}@cs.utwente.nl

Abstract. We describe an approach to reduce the effort involved in disseminating prototype academic (command-line) tools for wider use and inspection. This helps in preserving the effort involved in the development of such tools, and in raising the standards in computer science for conducting repeatable experiments. For this purpose we propose a light-weight, flexible framework to make such tools available via web forms.

1 Introduction

Prototype tools are a prime method to show the feasibility of formal methods-related ideas or algorithms. However, results obtained using such prototypes are typically only disseminated in the form of a (tool) paper. This does not do justice to the effort that went into producing the tool, and makes it practically impossible, at at least very hard, to repeat the experiments.

There are at least two ways of improving on this: 1) making the prototype tool and examples available for downloading, or 2) making the tool available via a web form. In this paper we concentrate on the second scenario: we offer a framework, called PUPTOL, that enables web-based tool access to prototype tools. Constraints and requirements are:

- We concentrate on *non-interactive* tools, which in their normal mode of operation transform input (given as command-line options, standard input or files) into output (on the standard output or in output files).
- We cater for tools that have not been developed with online access in mind. Typically, prototype tools are designed only to show certain functionality, without any a priori thought on interoperability or online access.
- Usage has to be lightweight: the overhead involved in publishing a tool should be small enough to motivate tool authors to actually do so.

Figure 1 gives a high-level view of the overall system. A user accesses the system via his Browser. The Server processes requests from the users' Browser, invokes a Tool with the tool input data and parameters that are given in the request, captures the output data of the Tool result, and returns it to the Browser.

We now first, in Section 2, describe the functionality of PUPTOL, and then, in Section 3, discuss how we use PUPTOL to provide access to a number of tools.

2 Functionality and Usage

In terms of Fig. 1, PUPTOL supports two types of backend: for tools that run locally, and for tools that are invoked via a remote web server. For tools that run locally, results appear in the users Browser as soon as they become available; for tools that are invoked via a remote web server, results are only captured when the remote tool has finished running. The sequence diagram of Fig. 2 shows the dynamic behaviour, where the polling part only applies to tools that run locally. A single PUPTOL instance can provide multiple users access to multiple tools. PUPTOL is implemented in Go (see `http://golang.org/`) and runs on Windows and Unix.

In order to publish his tool, the *tool author* provides the tool itself (if it is to run locally) and specifies 1) the contents of the web page through which the user interacts with the tool, 2) how input items in this page control the invocation of the tool, 3) how results of the tool are shown (in the same page, or in a new page so the user can easier save them), and 4) optional *scenarios*: predefined settings of input items to simplify replay of examples. The web page is specified in terms of input items (text fields, check-buttons etc.), output items (placeholders for text, or images, to be filled with output from the tool, or for additional input items that are to be added dynamically based on output from the tool), buttons (used to activate the tool), and container items (used to group items together). The specification is given in the lighweight JSON notation (see `http://www.json.org/`) To control the layout of the web page the tool author can provide a cascading style sheet. To integrate the tool access provided by PUPTOL in a web page, the tool author can provide a custom *tool page template* and use an inline html frame to hide the link to PUPTOL. To reduce the dependency on the PUPTOL administrator, a tool author can edit specification, scenarios and tool page template via a web interface, and make PUPTOL fetch and install new versions of the tool.

Invocation of the tool is specified, for tools that are run locally, via a mapping of the input items onto command line options, environment variable settings, and data that is to be written to the standard input of the tool. For tools that are invoked via a remote web server, the invocation is specified in terms of a URL, a request type (typically GET or POST) and a mapping of the input items onto name-value pairs that are passed in the request on the remote web server.

The results of the tool are specified, for tools that are run locally, in terms of file descriptors that PUPTOL will monitor for output when the tool is running. In addition to standard output and standard error, which PUPTOL monitors by default, additional file descriptors can be specified, such that the tool (or a shell script that wraps the actual invocation of the tool) can write results to PUPTOL on as many concurrent “channels”

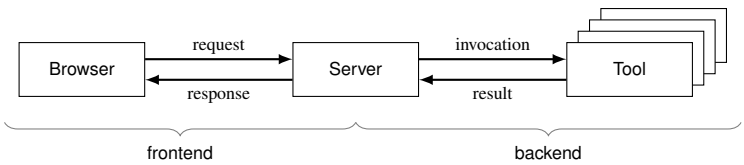

Fig. 1: High-level view on the architecture of PUPTOL.

as it needs. For each file descriptor an associated output item (and thus: an associated placeholder in the web page) is specified. For each output item a type is specified, that indicates how to display the result: add it as text to the placeholder in the page, or use it to replace the current placeholder contents; treat it as the filename of an image that is to be shown in the placeholder, or treat it the specification of additional input elements that are to be added to the page, etc. In Section 3 we give examples of how we used this functionality.
For tools that are invoked via a remote web server, the results are specified in terms of the textual output returned by the remote web server, where, when only part of that output is to be shown, the start and end of the part that is to be extracted from the output of the remote web server can be indicated.

The PUPTOL *system* uses the specification provided by the tool author to allow the user to interact with the tool via a web page, as follows. When the user accesses the page for a tool (request getPage() in Fig. 2), the Server initially serves the *tool template* page to the Browser. This template always contains an html container to be populated later and a reference to a javascript file at the Server (the tool author can include any additional information in the template). When the Browser displays the template page, it automatically requests the javascript file (request getScript() in Fig. 2), which then is

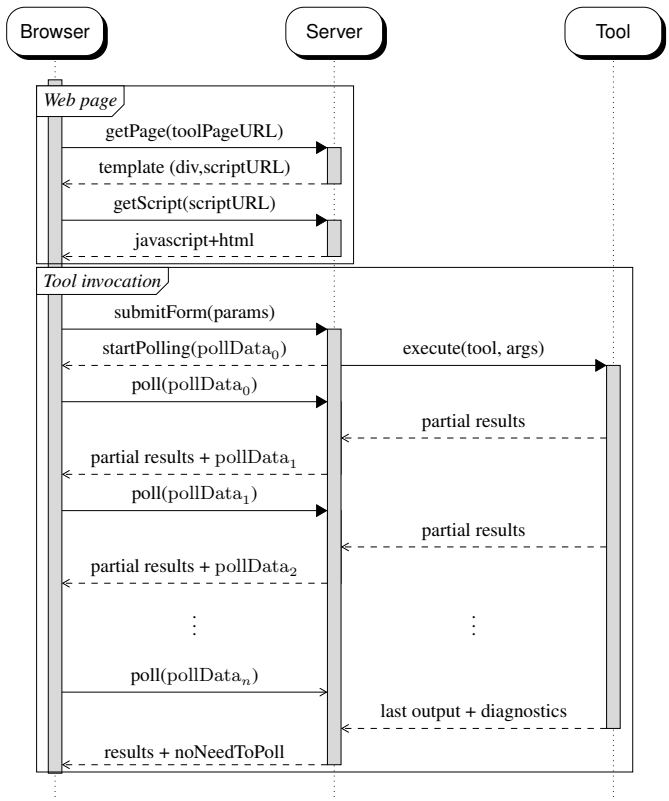

Fig. 2: Sequence diagram showing interaction between Browser, Server and Tool.

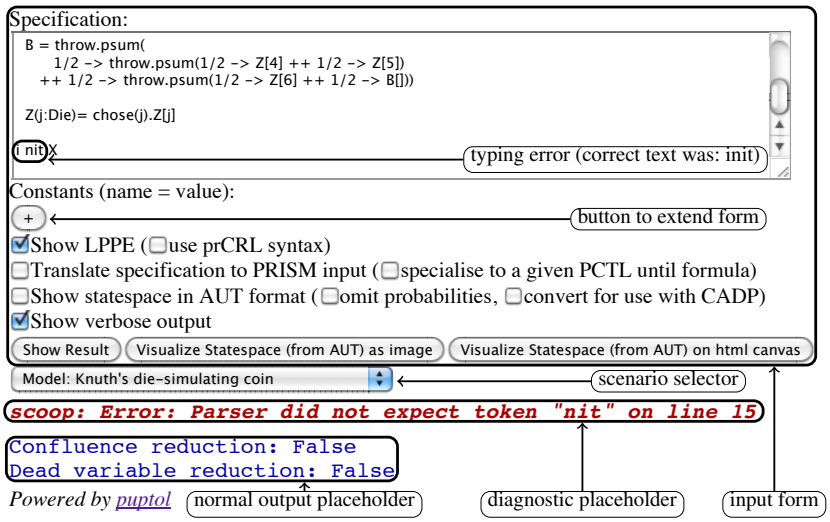

Fig. 3: SCOOP web page synthesized by PUPTOL (simplified to reduce space).

generated by the Server from the web page specification provided by the tool author. The javascript contains html code for the page items. When the Browser loads and executes the javascript, the html code is inserted in the above-mentioned container. When the user presses a button to activate a tool (request submitForm() in Fig. 2), the Server invokes the tool using form data from the request, captures the results, and returns those as formatted output to the Browser.

When the tool is run locally, after invoking the tool[1], the Server starts to capture the results, and sends a "start polling" response (startPolling() in Fig. 2) to the Browser. The Browser then shows an "Abort" button, disables all other form buttons, and starts sending polling requests (poll() in Fig. 2) to the server, to which the Server responds with (partial) results received from the tool, and an indication whether to continue polling. When the user presses the "Abort" button, the Browser sends an abort request to the Server, upon which the Server aborts the tool invocation; results produced by the tool until that moment are sent to the Browser using the polling mechanism, as described above.

The *user* interacts with the tool via his Browser. For instance, Fig. 3 shows (a somewhat older version of) the web page for the SCOOP tool [14], in which we have indicated input form, two output items, and the scenario selector that is automatically generated

[1] All command line tools are invoked via memtime [9] to control resource consumption. Memtime is a program that starts a given program, with given command line arguments, and then monitors and reports—as specified via a command line option, either periodically, or only when the given program terminates—the resource usage of the started program. Memtime can also be invoked with upperbounds for memory or cpu time consumption, in which case it aborts the running program when an upperbound is exceeded. For PUPTOL, we extended memtime with a command line option to specify the file descriptor onto which memtime writes its usage reports, to allow PUPTOL to treat (i.e. show) these separate from tool output.

from the scenarios specified by the tool author. The page results from the following interaction. First, scenario "Knuth's die-simulating coin" was selected, which caused the text of the model to automatically appear in the input form. Then, this text was edited to introduce a syntax error (a space was inserted in "init"). The "Show verbose output" checkbox was checked, and then the "Show Result" button was pressed which caused the results to appear.

For easy sharing of experiment set-ups, the user can also press a button "Permalink" (see Fig. 4; this button is not shown in Fig. 3) to store the current settings of the input form on the PUPTOL server. When this button is pressed, a URL is returned that, when entered in a web browser, will show the same form, populated with the stored settings. (For each tool configured in PUPTOL, a "Permalink" button is automatically added; the tool author can adjust its place in the input form via the PUPTOL tool specification and the cascading style sheet.)

The view in Fig. 3 is static. However, the structure is extensible 1) via direct user interaction, or 2) as side effect of a tool invocation. In case 1), buttons like the "+" next to `Constants` are used to extend the input form with extra items or shrink it back, for a tool that accepts a variable number of arguments. In case 2), pressing a button to invoke a tool causes an additional input item to appear; for example, a radio button allowing the user to use an invocation result (or an uploaded file) as input for a next invocation. This is shown in Fig. 4, where the two lines with "dataset 0" and "dataset 1" were addded as result of presses on the "Show Plot and store dataset" button. The plot shown in this figure was obtained by checking the check-boxes of these datasets, and pressing the "Plot selected data sets in combined plot" button. Stored results will remain available to the user until his session expires.

3 Experience and evaluation

We have used PUPTOL for tools CVPP [8], PROMOVER [13], GROOVE [3], SCOOP [14], IMCA [6], the tool chain MAMA [7] that combines SCOOP and IMCA, the VERCORS tool [1], and DFTCALC [2]. For CVPP and PROMOVER we use PUPTOL as "wrapper" around basic web interfaces that already existed, and specified examples from [8] and [13] as scenarios. For command line tools GROOVE, SCOOP, IMCA and DFTCALC we created web interfaces to access their functionality, where the user can choose between visual and textual representation of the results. The author of SCOOP refered in [14] to the SCOOP page at PUPTOL, and demonstrated SCOOP using PUPTOL in the presentation of [14]. The authors of PROMOVER and SCOOP seamlessly integrated PUPTOL-generated content in their tool websites. This proves that PUPTOL meets its requirements concerning the light-weight application to tools not originally designed for web-based access.

Moreover, for both SCOOP and DFTCALC the PUPTOL web interface offers functionality not available in the command-line tools.
For SCOOP, the web interface offers, in addition to visualization of a state space generated by SCOOP, also integrated access to model-checker IMCA (not shown in Fig. 3) (i.e. it offers access to the MAMA tool chain). For both these functions, a single request from the user results in the running of multiple programs (e.g. SCOOP itself, the translator from SCOOP output to input for the visualizer, the visualizer itself, or, when the model-

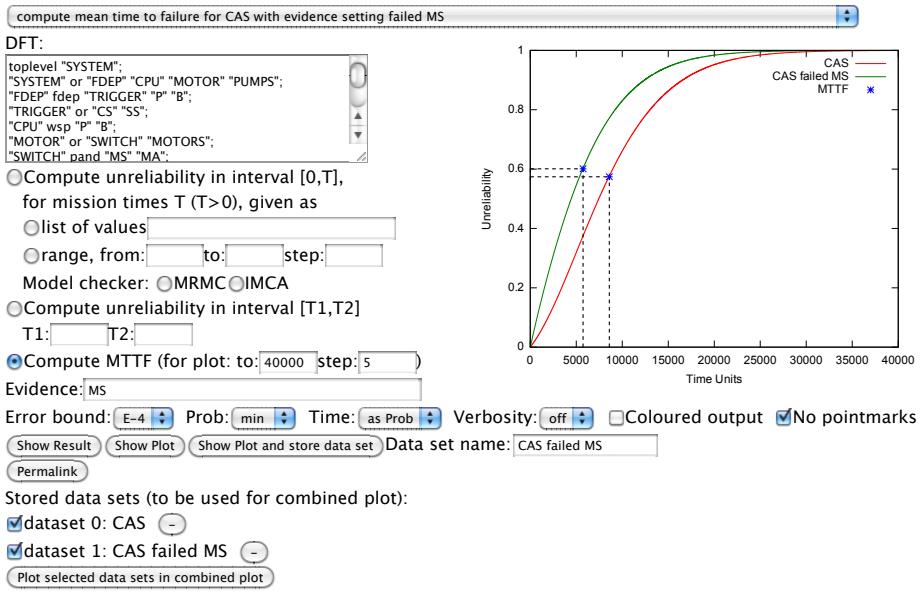

Fig. 4: DFTCALC web page synthesized by PUPTOL, showing two dataset items that were dynamically added to the input form.

checker functionality is invoked, IMCA) that all contribute to the user request; for each of these programs, the web page for SCOOP shows individual cpu and memory usage statistics that are dynamically updated while the program is running. This is realized as follows, using the PUPTOL feature to monitor as many file descriptors as needed. The user request causes a shell script to be executed: a shell script that invokes each of the programs, necessary to honour the request, via a separate instance of memtime. Each such memtime instance writes, while its program is running, periodically (every second) usage statistics to a separate file descriptor that is monitored by PUPTOL, which allows PUPTOL to show the output of each memtime instance in a separate placeholder in the web page.

For DFTCALC, the web interface offers access to plotting functionality, where plot data-sets can be stored on the PUPTOL server, to subsequently be used to create combined plots that show multiple data-sets; this is shown in Fig. 4. When the "Show Plot and store data set" button is pressed, multiple placeholders in the DFTCALC web interface will be updated with tool results, as follows. At the bottom of the page memtime statistics (dynamically updated while the tool is running) will be shown. Above that, a textual result will be shown. (These two items are not visible in Fig. 4). Above that, a dataset selection input item will be added to the input form, and at the right side of the page a plot will be displayed. All these items are passed to PUPTOL via separate file descriptors, using the framework functionality described in Section 2.

PUPTOL can handle multiple concurrent users. Currently, tools are run locally on the PUPTOL server. If this results in too high a load, the architecture of PUPTOL makes it easy to change the configuration to run tools on one or more dedicated machines.

Future Work. Planned extensions to enhance the functionality of PUPTOL include: adding the ability to create online tutorials as in `go-tour.appspot.com` and [11]; further reducing the dependency on the PUPTOL administrator by improving the PUPTOL administrative web interface.

Related work. Though we are aware of several approaches to improve accessability of tools that share our motivation to improve the repeatability of experiments, such as JETI [10] and SHARE [5], there are huge differences in the chosen approach. In particular, we have gone for a lightweight framework, which allows easy installation but (in contrast to the aforementioned approaches) does not allow performance measurements, tool comparison or interoperability studies. Web sites with similar functionality as the ones generated by PUPTOL are Microsoft's research web tool demo site [11] and the GO language playground [4]; with respect to these, PUPTOL lacks ability to create tutorials, but has the advantage of self-maintainance and easy configurability, and the ability for tool authors to specify scenarios and seamlessly integrate PUPTOL-generated content in existing web-sites.

Acknowledgements. We thank Mark Timmer for his enthousiastic cooperation in the production of the SCOOP web demo.

References

1. Amighi, A., Blom, S.C.C., Huisman, M., Zaharieva-Stojanovski, M.: The vercors project: Setting up basecamp. In: Sixth Workshop Programming Languages meets Program Verification (PLPV 2012), Philadelphia, USA. pp. 71–82. ACM, New York (January 2012)
2. Arnold, F., Belinfante, A., van der Berg, F., Guck, D., Stoelinga, M.I.A.: DFTCalc: A tool for efficient fault tree analysis (extended version). Technical Report TR-CTIT-13-13, CTIT, University of Twente, Enschede (June 2013)
3. Ghamarian, A.H., de Mol, M.J., Rensink, A., Zambon, E., Zimakova, M.V.: Modelling and analysis using GROOVE. STTT 14(11), 15–40 (2011), see `http://groove.cs.utwente.nl`
4. The Go playground. `http://golang.org/doc/play/`
5. Gorp, P.V., Mazanek, S.: SHARE: a web portal for creating and sharing executable research papers. In: ICSS. Procedia Computer Science, vol. 4, pp. 589–597. Elsevier (2011)
6. Guck, D., Han, T., Katoen, J.P., Neuhausser, M.: Quantitative timed analysis of interactive markov chains. In: 4th International Symposium on NASA Formal Methods, NFM 2012, Norfolk, VA, USA. Lecture Notes in Computer Science, vol. 7226, pp. 8–23. Springer Verlag, Berlin (April 2012)
7. Guck, D., Hatefi, H., Hermanns, H., Katoen, J.P., Timmer, M.: Modelling, reduction and analysis of markov automata (extended version). Technical Report arXiv:1305.7050, Cornell University, Ithaca, NY, USA (May 2013)
8. Huisman, M., Gurov, D.: CVPP: A tool set for compositonal verification of control-flow safety properties. In: Beckert, Marche (eds.) FoVeOOS 2010. LNCS, vol. 6528, pp. 107–121. Springer (2010), see `http://www.csc.kth.se/~siavashs/cvpp/cvpp.html`
9. Johan Bengtsson: Memtime project website at freshmeat. `http://freshmeat.net/projects/memtime/` (2002), our version with modifications for PUPTOL is available at `http://fmt.ewi.utwente.nl/gitweb/memtime.git`.
10. Margaria, T., Nagel, R., Steffen, B.: Remote integration and coordination of verification tools in jeti. In: ECBS. pp. 431–436. IEEE Computer Society (2005)

11. Microsoft research web tool demo site. http://rise4fun.com/
12. PUPTOL – FMT tools. http://fmt.cs.utwente.nl/tools/puptol/, example web pages: http://fmt.ewi.utwente.nl/puptol/
13. Soleimanifard, S., Gurov, D., Huisman, M.: Procedure-modular verification of control flow safety properties. In: FTfJP 2010. p. 5. ACM (2010), see http://www.nada.kth.se/~siavashs/ProMoVer
14. Timmer, M.: SCOOP: A tool for SymboliC Optimisations Of Probabilistic Processes. In: Palamidessi, C., Riska, A. (eds.) QEST. IEEE Computer Society, Los Alamitos, USA (2011), see http://www.cs.utwente.nl/~timmer/prcrl/

Appendix: Tool Availability

The main PUPTOL website [12] describes how to download and install PUPTOL. We have seen PUPTOL running on Windows and Unix.

Note that to install PUPTOL one also needs to install a development environment for the GO language (available via http://golang.org). At this time of writing, the PUPTOL source unfortunately still targets GO version R.60; an update to the current GO version (currently GO 1.1) is high on our agenda.