Conception and Standarization of Online Execution Tools

M. Colom

miguel.colom@cmla.ens-cachan.fr
http://mcolom.perso.math.cnrs.fr/

July 21, 2013

1 Conception and Standarization of Online Execution Tools

In general, two kinds of online execution tools are installed on a web server that runs demos in a reproducible research environment.

- The base system, that is the back-office part that interacts with the user and manages the execution of the demos. For example, in the case of IPOL, this would be the Python web application.
- The rest of external tools that don't belong to the base system, but are useful to automate some tasks or simplify them. For example, in IPOL they'd be the tools that allows to automatically create a demo from a textual description or the program that sends an email to the authors when their program fails durings its execution in the production server (work in progress ...).

The base system has to be flexible enough to support the needed features that most of the demos require. For example, handling several types of configurable controls in the parameter selection, as sliders, combo-boxes, selection of sets of images, etc. To decide if a new feature should be supported or not, the only way is to see if there's a real demand for such a feature. If so, the base system has to be adapted.

The outlier or non-standard demos that operate in a different way from the standard demos imply a risk, since they require an special attention and are difficult to maintain. For example, if they use their own mechanism to represent 3D data, they won't be aware of changes in that part of the base system. In general, the demos non-standard demos should not be allowed or at least, strongly discouraged.

However, the level of flexibility of the base system has a limit that depends on the complexity that is added when implementing a new feature. For example, the demo system should be able to generate new demos from a simple textual description. If the demos need complex descriptions, the solution is using external tools. For example, in IPOL an automatic code generator is available right now. It allows to create the source code of an IPOL demo from a complex textual description, without messing up the server with the complexity of this external tool.

The external tools not only prevent that the complexity of the base system increases, but also allow for the automation of some tasks, freeing the IPOL Tech team from doing them manually. For example, in IPOL the logcheck tool automatically sends an email to the IPOL Tech team each time a program fails. Right now, what is done is to manually use the objdump tool to disassembly the program and warn the author. This task could be to be automated with an script to receives the notification of logchecks, calls objdump with the proper parameters and sends an email to the author. To do it, it's necessary that some of the external tools have access to some centralized data store, with the list of the demos, the name of the authors, their emails, etc.

In the case of the base system, it's important the some standard mechanisms and interfaces are defined, in order to manage the many demos easily. In general, the demos require the following:

- Asynchronous communication (AJAX) for interactive data presentation (i.e. 3D points clouds, large zoomable audio plots, etc).
- Flexible but fixed data formats for the data. For example, the data structure for representing a cloud of 3D points and how to store it and transmit it.

The necessity of having well defined data structures for large data is clear, since it decouples the complexity of the presentation of the data from the generation of the data itself. And this allows to change the tool that renders the graphics on the browser for some other in the future. Not only the format of the data structures should be well defined, but also the interaction between the client and the server when using the asynchronous communication. For example, when a demo in the client wants to draw a part of a audio signal, it should call an specific function of the server API with a parameter that indicates the level of zoom. If it wants to get samples between two times, it should call a different function.

The APIs for the 2D and 3D data should define exactly what are the formats and how to interact asynchronously with the base system. This way, the API offers a level of indirection that decouples the data from its processing in the demo application and from its representation in the client

browsers. At the same time, it makes easier the management and reusability of the system.

About the development of the base system, it's important to take into account that right now there's a lot of demos running and strongly coupled to the current API of the demo system. Therefore, is obvious that any change in the demo system shouldn't break the functionality. This can be achieved by:

- Keeping full backward compatibility with previous version of the base system.
- Only perform small changes in the base system. This allows for the creation of automatic tools that change the code of all the existing demos at the same time. This can be seen as *evolutive development*.