DeThinkingService
ReThinkingDesign

First Nordic Conference on
Service Design and Service Innovation
Oslo, 24th–26th November 2009

Sketching web services backends with SERPE

Walter Alberto Aprile

IO-Studiolab, Faculty of Industrial Design, Room 10-2A-11 to 23

Delft University of Technology

Landbergstraat 15, 2628 CE Delft, The Netherlands

w.a.aprile@tudelft.nl

http://studiolab.io.tudelft.nl/aprile/

Summary

In the face of current strong commercial interest in services that, from an implementation point of view, consist of databases provided with a web API and/or a front end, there is a scarcity of tools that allow quickly sketching the service backend in order to deliver an interactive prototype. SERPE consists of an online Python programming environment that hides the complexities of the MVC model, databases, objects and object relational mapping and builds on Google's services, letting the service designer think in terms of a simple tag-indexed persistent store. A Twitter clone and a service that allows control of a large lamp are presented as typical SERPE projects.

Introduction

In our experience as teacher in the field of interaction design, we have noticed that students are strongly influenced in their design activities by the prototyping and sketching(Buxton, 2007) tools at their disposal. The tools do not simply influence the range of ideas that they can test out, but also their thinking: this is probably a consequence of the fact that we insist on prototyping early and often. It follows that, to extend the range of solutions, we have to extend the range of interactive sketching tools.

What qualities should we look for in these tools? Paul Dourish (Dourish, 2001) brings back to the field of design the Heideggerian distinction between two states of an object, ready-to-hand (*zuhanden*) and present-at-hand (*vorhanden*) – see also (Winograd & Flores, 1986). An object that is ready-to-hand is ready for use: the subject uses it to achieve something. In contrast, and object that is present-at-hand is being observed rather than being used. The distinction depends on the state of the observing and interacting subject: the same tool may

DeThinkingServiceReThinkingDesign

be ready-to-hand or present-at-hand to the same person, depending on circumstances. For a craftsman, his tools are ready-to-hand; for a historian of technology, they are present-at-hand. When involving interactive technology in the design process it is important that it is ready-to-hand for the designers to use. At a basic level, we don't want designers sketching a car in foam to become very interested in the properties of the foam: we want them to be concentrating on the car form. This is not in contrast with design's traditional interest for the properties of materials, but rather draws a distinction between the materials of sketching and prototyping, that are temporary and easily manipulated and the materials of delivered goods. Such distinctions exist in software: for example, GUI designers will frequently build sketches in Photoshop or in Flash, while the delivered system will be written in Java or other lower level languages. Designers who work in hardware will use highly flexible prototyping boards such as Arduino (Mellis, Banzi, Cuartielles, & Igoe, 2007) to work in a sketch-like manner (Buxton, 2007), using high-level programming tools and modular sensor systems (Hummels, Overbeeke, & Klooster, 2007). In this way design iteration can be performed quickly steering clear of the complex and time consuming engineering work that becomes necessary for the delivered system.

Existing tools and limits

In the field of service design there exist numerous tools that enable a measure of experiential sketching with varying degrees of fidelity. Designers use HTML + JavaScript or Flash to build web pages and try them out as touchpoints. More specialized tools such as Axure ("Axure,") and Balsamiq Mockups("Balsamiq Mockups,") place themselves at opposite ends of the fidelity spectrum (an excellent resource specifically on wireframing is (Linowski, 2009)). Fidelity, though, is only one of the axis on which tools can be classified. (McCurdy, Connors, Pyrzak, Kanefsky, & Vera, 2006) propose five dimensions of fidelity, namely

1. level of visual refinement
2. breadth of functionality
3. depth of functionality
4. richness of interactivity
5. richness of the data model

and observe that prototypes can be high fidelity along one dimension, and low fidelity along a different one. In our teaching activities with students from Industrial Design and Engineering schools, we have observed that the tools we have mentioned tend to be single-user, single-session tools: the prototype (or the sketch, as the case may be) is tried out by a single user during a session that is frequently observed by a researcher. Back end functions, such as data storage are trivially faked, or the test is conducted in such a way that no actual back end is necessary. We argue that the absence of tools for back end sketching does not allow designers to sketch out the full breadth of a service. We formed this hypothesis on observing that, out of various groups of industrial design students tasked with designing a game that encourages physical activity in young urban girls (van der Helm, Aprile, & Keyson, 2008), not one had chosen a design concept that went beyond very local solutions, despite suggestions on our part to add a high score table or other components that would allow sharing of the game experience over a geographic area. Since the course we were teaching insists on sketching as a form of design exploration (as opposed to prototyping as a form of design confirmation), we realized that the lack of designer friendly tools for back end

sketching was constraining the breadth of the design solutions the students were able to explore.

Design rationale

If we concentrate ourselves on services that are accessible through the Internet, it is safe to say that most of these services contain, at the highest possible level of description, a database, a web application framework more or less integrated with an object relational mapper (ORM), business logic and presentation logic. To master the design and implementation of such a system requires that the designer, from a computer science point of view, understands objects, databases and probably also the model-view-controller paradigm, first formalized by (Reenskaug, 1979) and applied to web development by (Leff & Rayfield, 2001).

It is not realistic to expect industrial designers to reach this level of knowledge in the course of their curriculum, not without converting them into software engineers. In design curricula, though, it is easy to find some form of programming, typically of a procedural nature. Is it possible to use simple programming techniques to sketch service back ends without bringing into play the full arsenal we mentioned above? This is what we set forth to do when building SERPE.

What we expect as a result of using SERPE is that

- services will be tested from an early stage on more varied devices
- multi-user experiences will be sketched and tested early. Many services make sense and reveal their true complexity only when multiple users engage with them on different time scales. This is extremely difficult to test on paper or using screen oriented tools like the ones we discussed before.

At every step in the design of SERPE we have striven for a smooth learning curve which took its inspiration from the Python programming language itself. Python can be used, at a simple level, even if the programmer does not understand what a function or an object are; for some contexts of use, this understanding is not necessary. In the same guise, a SERPE developer should not need to understand all, or even the greater part, of the system to get a simple application off the ground.

In the same vein, we have striven for reasonable defaults for every function, and we have kept the entire application in one single file, in order to avoid the typical web progamming experience of having to edit various configuration files.

Architecture

SERPE (SERvice Prototyping Environment) is composed of a simplistic HTML and HTTP manipulation library, aimed at the non-technical developer, a store/retrieve module for persistent information, URL-to-function mapping rules and a restricted execution environment.

On-line code editing has been chosen in preference to the more frequent approach of developing on an IDE and uploading code to a server. In our teaching experience, we have noticed that on-line environments, while being generally inferior in responsiveness and

integration to applications running on a local computer, offer certain advantages that could be critical for our target users, namely easy cross platform compatibility, ease of distribution, immediate availability and a very low entrance threshold. Additionally, an on-line development environment simplifies (or removes outright) deployment and it can be easily used even in tightly controlled computing environments.

Notice that nothing limits SERPE to returning web pages. We have written examples that produce RSS feeds or messages in low level binary protocols.

Code execution

The user interacts with the on-line code editor and submits his Python code to SERPE. Subsequent HTTP access triggers the execution of the user code according to the following mapping rules:

- http://host/ and http://host/index.html call the index() function, if the programmer has defined it: otherwise, all of the user code is simply executed.
- http://host/foo calls the function foo, if the user has defined it: otherwise, an error message is displayed.

The user code executes in a special environment created with the *exec()* function. The environment's global variables include the URL that has been requested, the contents of the form fields and controls if appropriate, and information about the current user.

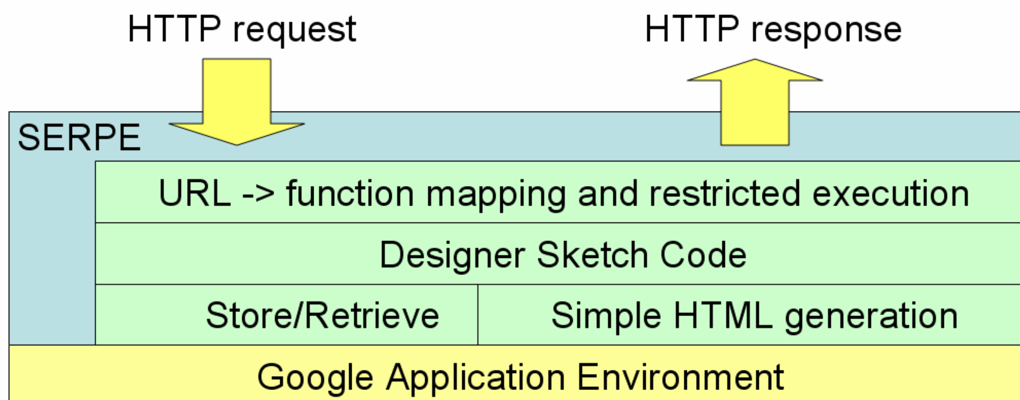

Fig. 1 the SERPE architecture

Before execution, the code is prefixed with debugging instructions, if necessary. From the point of view of the programmer, if he created a field called "telephone" in a form, his script will have access to a variable called exactly "telephone". There is no need to understand the CGI mechanism.

Editing and saving

A syntax-coloring JavaScript code editor called Edit Area (Dolivet, 2009), accessible through a special URL, lets the user write his code and save it to the system. SERPE programs are automatically versioned, and all previous versions are kept, for unlimited undo capability. This encourages an explorative and iterative development style.

Program structure

At the most simple, a SERPE program is just a sequence of Python statements, not necessarily wrapped in functions or classes. For example, this SERPE program

```
htmlStart(title="Formatting fun")
out(p(bold("Important information:")))
out(p("Writing HTML by hand is bad for your health"))
htmlEnd()
```

will produce a simple message when http://host/ or http://host/index are accessed. Of course the user is allowed to write any valid Python program (within the limits of the Google Application Environment platform): the functions are executed according to the URL mapping rules we have mentioned before. If the service is to be accessed via a Web browser, the programmer can access a large set of HTML generation functions that should largely remove the need to write HTML by hand, while at the same time guaranteeing reasonable defaults. For example, the *form()* function will produce an HTML form, and it will automatically include a Submit button if the user does not include one. It is our hope that this and other features in the library will reduce the frustration felt by beginning service developers.

Database hiding

An important part of services is stored information. All permanent information in the Google Application Environment must be stored in database tables, since no creation of files is allowed. A simple system of multi-key associative array has been implemented on top of GAE database objects. Arbitrary Python objects are associated with a set of keys, whose interpretation is entirely up to the user. The retrieval operation can specify also up to four keys, and returns the set of all objects whose keys match the request. For example, after storing some values:

```
store(Object1,”Delft”,”historic”,”Netherland”,”small”)
store(Object2,”Rotterdam”,”contemporary”,”Netherland”,”large”)
store(Object3,”Hamburg”,”contemporary”,”Germany”,”large”)
store(Object4, “Hamburg”, “seaport”)
```

the programmer can access what he has stored by retrieving:

```
retrieve(“Delft”)
```

returns a list containing Object1

```
retrieve(None,”contemporary”)
```

returns a list containing Object2 and Object3, and lastly

```
retrieve(“Berlin”)
```

returns an empty list.

This simple mechanism is sufficient, in our experience, for representing most real-word information structures. Of course, we are not suggesting that this is what application programmers should use for real-word programming! Our store/retrieve module is very far from providing the structural richness of a database, relational or not It is appropriate only

for an interactive sketch, that is to say throwaway code that does not need to be easily maintainable or durable in any way.

Application

In the current academic year we will start deploying SERPE in classes, and we already have used SERPE ourselves for writing (outside of toy programs such as a forum and a simple Twitter clone) a simple system to control the groups of lamp inside a large interactive luminaire. In this case, no HTML is generated but rather a specific binary protocol to control a set of shift registers inside the luminaire.

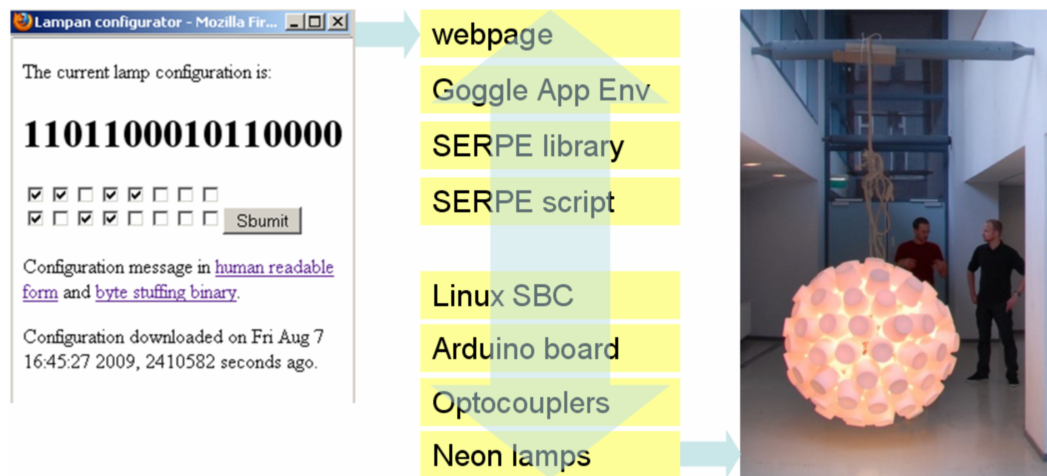

Fig 2. SERPE-generated interface to a large computer controlled luminaire.

For the coming months, we are planning an application of SERPE to design, share, store and execute sequences of operations on an automated household appliance.

Conclusion

We have described here an environment that enables service designer to sketch service backends with a small effort. The SERPE environment has been designed to present a simple procedural programming model that hides the complexities of object orientation and of the underlying databases. In the future, we will extend and improve SERPE on the basis of what we observe in class.

References

Axure. Retrieved 1st September 2009 from http://www.axure.com/

Balsamiq Mockups. Retrieved 1st September 2009, from http://www.balsamiq.com/products/mockups

Buxton, W. (2007). *Sketching user experience – Getting the design right and the right design*. San Francisco, CA: Morgan Kaufmann.

Dolivet, C. (2009). Edit Area. Retrieved 1st September, 2009, from http://www.cdolivet.com/index.php?page=editArea

Dourish, P. (2001). *Where the Action Is: The Foundations of Embodied Interaction*. Cambridge, MA: MIT Press.

Leff, A., & Rayfield, J. T. (2001). *Web-application development using the model/view/controller design pattern*.

Linowski, J. (2009). *Fluidia: agile UI prototyping.* Delft University of Technology, Delft.

McCurdy, M., Connors, C., Pyrzak, G., Kanefsky, B., & Vera, A. (2006). *Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success*.

Mellis, D. A., Banzi, M., Cuartielles, D., & Igoe, T. (2007, April 28 - May 3, 2007). *Arduino: An Open Electronics Prototyping Platform.* Paper presented at the CHI 2007, San Josè, USA.

Reenskaug, T. (1979). *Models-views-controllers*: Xerox PARC

van der Helm, A., Aprile, W., & Keyson, D. (2008). Experience Design for Interactive Products: Designing Technology Augmented Urban Playgrounds for Girls. *PSYCHNOLOGY JOURNAL, 6*(2), 173-188.

Winograd, T., & Flores, F. (1986). *Understanding computers and cognition: A new foundation for design*: Ablex Publishing Corporation.