A Domain Specific Language for Orchestrating User Tasks Whilst Navigation Web Sites

Sérgio Firmenich[1,2], Gustavo Rossi[1,2], and Marco Winckler[3]

[1] LIFIA, Facultad de Informática
[2] Universidad Nacional de La Plata and Conicet Argentina
[3] IRIT, Université Paul Sabatier, France
{gustavo,sergio.firmenich}@lifia.info.unlp.edu.ar,
winckler@irit.fr

Abstract. In this paper we claim that there are a lot of processes over Web applications that require a high level of coordination between individuals and tasks featuring procedures. We propose hereafter a Domain Specific Language (DSL) for describing the asynchronous orchestration users' tasks including manual users' tasks (i.e. simple instructions that tell users what to do during the navigation) and automated tasks (i.e. tasks that can be partially or completely automated by client-side scripts). The approach is illustrated by examples and a case study showing the tools, for which an empiric evaluation is presented.

Keywords: task and process modeling, Web application, Web augmentation.

1 Introduction

Although Web navigation was regarded in the past as a solitary activity, nowadays, many users are engaged in repetitive and collaborative activities that are supported by uncountable Web applications [6]; for example booking a seat in a flight or explaining friends how to book a seat next yours in a flight… Moreover, many of these tasks involve dealing with different Web sites, which run independently with no support to the actual users' concern [4].

This lack of integration of different Web resources has motivated the development of mash-ups tools that are able merge into a specialized applications a set resources that are scattered among different Web sites [8]. The problem is that mash-up are used straightforward, when most of tasks users perform are volatile and do not really require the creation of a new an entirely new applications.

The integration of data across applications can also be done by Web augmentation artifacts, which perform interventions over Web applications DOMs. Some Web augmentation approaches [1][4] aim to support users task by adapting the Web pages visited accordingly.

In this paper we propose a Domain Specific Language (DSL) for describing *procedures* that are aimed to orchestrate user tasks over multiple Web sites. It supports flexible process modeling by allowing users to combine manual task and automated tasks from a repertoire of patterns of tasks performed over the Web. Whilst manual tasks can be regarded as simple instructions, automated tasks correspond to *Web* *augmentation* [4] tools (i.e. *augmenters*). The approach is duly illustrated by a case study describing a trip planning over the Web.

The rest of the paper is organized as follows: section 2 motivates and presents related works; section 3 introduces the approach. Section 4 present the DSL followed by the corresponding tool support (section 5). In section 5 we also present a comparative study using our tools; and lately section 6 presents the conclusions and future work.

2 Motivation and Related Work

Web Augmentation is not a new concept, and it is becoming really important from the point of view of users, who are expecting new kinds of mechanisms for personalizing their experience while navigating the Web. Large communities of scripting such as GreaseMokey prove the value of this technique. There are other similar approaches. Mashup tools, for instance, have the same final goal: improve the users' experience. Neither mash-ups nor existing Web augmentation techniques provide a definite and flexible solution for supporting users tasks. Here, we compare our approach with others DSL/tools for supporting users tasks.

Some approaches allow users to specify the steps involved in certain tasks in order to repeat these steps later. For example CoScripter [4] records the user interactions (based on DOM events) and then the user may reproduce the same steps automatically. Other approaches define DSLs that aim to help to automate tasks. For instance, ChickenFoot [1] extends JavaScript with new sentences (e.g. "click()", "enter()", etc.). In this way, to develop a script for automating Web use is easier. Both ChickenFoot and CoScripter are powerful approaches but these do not contemplate changes in the process, since it is completely DOM-dependent. With the same philosophy we can mention Selenium [7], which can be used for this task automation, although it was originally defined for testing. While all these approaches may help users by allowing them to automate only primitive tasks, our approach mixes these with augmentation ones, which adapt Web pages accordingly to the current user tasks. It implies that not only repetitive processes may be defined but complex scenarios of adaptation. Besides that, the manual execution of certain tasks gives the control to users. In this way, sensitive tasks (for example payments, or sensible information use) are not performed by automatic tasks in which users may not trust.

3 Overview of Our Approach for Orchestration of Web Tasks

This section provides a view at glance of our approach and the type of users' tasks supported which include: *primitive* and *augmentation tasks*.

We refer as *primitive* tasks to a basic set of tasks that are already supported by the Web browser. These tasks include actions such as "*go to a Web page*", "*fill in a form*", etc. Primitive tasks used in our approach are heavily inspired by previous works that have already proposed a taxonomy for these user tasks [7].

For us, *augmentation* tasks are those ones that require advanced scripts programming (based on Web augmentation techniques) to be executed over the Web browser. Some of these tools are able to perform changes in DOM's changing Web pages on the client side. In previous work [4] we have developed a set of Web augmentation tools, called *augmenters,* using the CSN framework. The CSN framework is a tool that supports the development of scripts aimed to adapt Web sites accordingly to the actual users' concern. *Augmenter* are integrated into the Web browser via the framework. Once installed, augmenters are accessible to the user via a contextual menu. The framework has two main user roles: i) developers: are users with programming skills who can extend the framework by creating augmenters; ii) final users: who use augmenters to improve their performance whilst navigating the Web. For example, Figure 1 shows the activation of the augmenter *DataCollection* used to collect data from Web pages. The data collected is presented as a kind of floating post-it called *Pocket*. In the example the user is collecting point of interest under the name of "*PoI"*. As we shall, the collection of Web page data requires an advanced script (i.e. an augmenter), it modifies the DOM page (by creating a floating DIV element) and extend what users can do over a Web page (i.e. create electronic post-its); so that when a user runs the *DataCollection* augmenter he in fact performing an *augmentation* task.

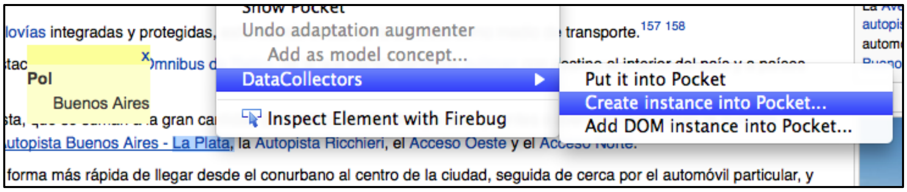

Undo adaptation augmenter
Add as model concept...
DataCollectors
Inspect Element with Firebug
Put it into Pocket
Create instance into Pocket...
Add DOM instance into Pocket...
PoI
Buenos Aires

Fig. 1. Example of the use of the augmenter *DataCollection*

Augmenters can also be used in combination to create complex sequences of tasks. Figure 2 shows the combined execution of augmenters. In this example a user executes the augmenter *CreateGoogleMapsLink* from the *Pocket* element (2.a). This action adds an anchor to *GoogleMaps* next to each occurrence of the concept "*PoI"* (2.b) that can then navigated to the corresponding *GoogleMaps* web site (2.c).

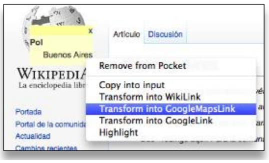


Fig. 2.a. Triggering augmenter using *GoogleMapLink*

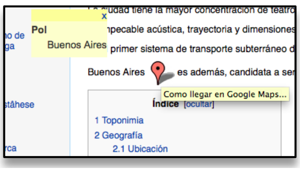


Fig. 2.b. Adaptation performed by the augmenter *GoogleMapLink*

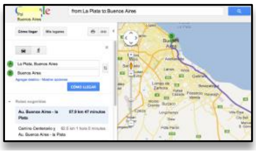


Fig. 2.c. Navigation to GoogleMaps

3.1 Overview of the Approach

The goal is to allow users to create complex processes, called *procedures*, by composing *primitive* and *augmentation* tasks. The composition is a sequence of tasks formalized by a DSL and stored as a XML file. A dedicated tool parses that XML file and executes the procedures on the Web browser. Figure 3 provides a view at glance of the approach. As we shall see, the approach include three phases, as follows:

- Definition of tasks: it concerns the inclusion of task to be composed. This phase requires skilled Web developers who program augmenters. This is technically demanding, but the work should be done once and it will benefit all users. Nonetheless, the framework provides a large set of both primitive and augmentation tasks.
- During *Composition* phase, users create a sequence of tasks available in the *repository*, which is exported by the *factory* and defined by the means of a DSL describing all tasks in the procedure. This artifact, defined by the DSL, may be shared with other users in order to support them in the accomplishment of the same task.
- *Execution*: this phase features a player that is concerned by the execution of the procedure previously encoded by the DSL.

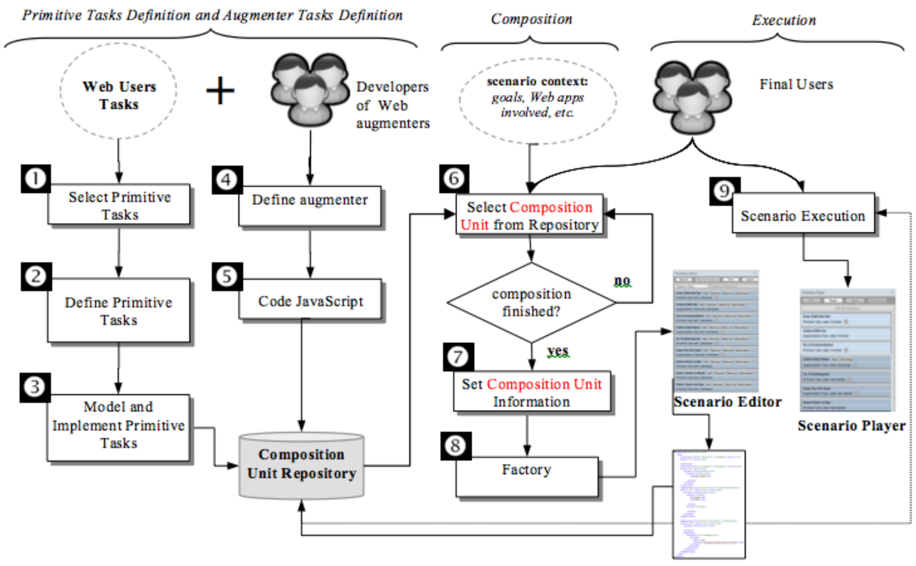

Primitive Tasks Definition and Augmenter Tasks Definition
Composition
Execution
Web Users Tasks
+
Developers of Web augmenters
scenario context: goals, Web apps involved, etc.
Final Users
1 Select Primitive Tasks
2 Define Primitive Tasks
3 Model and Implement Primitive Tasks
4 Define augmenter
5 Code JavaScript
Composition Unit Repository
6 Select Composition Unit from Repository
composition finished?
no
yes
7 Set Composition Unit Information
8 Factory
9 Scenario Execution
Scenario Editor
Scenario Player

Fig. 3. Overview of the approach

4 A DSL for Web Task Composition

Procedures will be defined according with the DSL metamodel shown in Figure by a UML class model. This metamodel defines those elements contemplated by the DSL and their relations. Basically, the DSL defines a procedure as a XML file containing a list of tasks. Primitive tasks supported are based on [3]. The set of augmenters depends on what was developed by users. Composed tasks are used to group several tasks in a single block. Tasks have three main properties: *repetition* property for specifying if the task may be executed more than once. The *optional* property allows skipping the execution of the task. If *automatic* property is *true*, then the player automatically triggers the task.

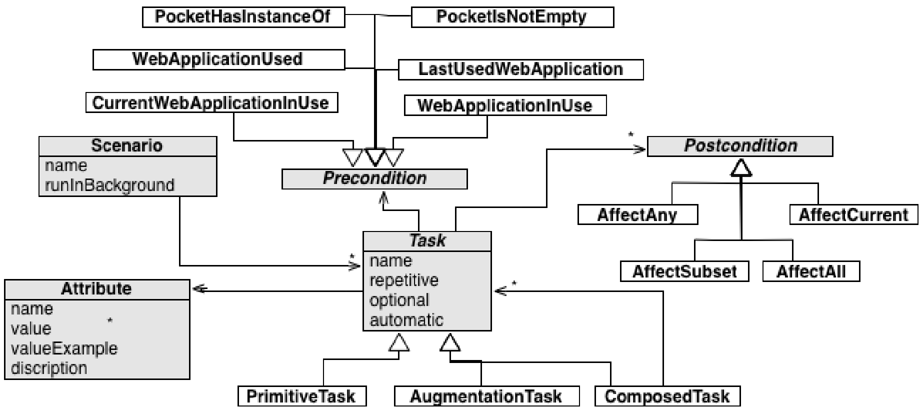


Fig. 4. The DSL metamodel

Besides these properties, for each task additional properties can be added including *preconditions*, *postconditions* and *attributes*:

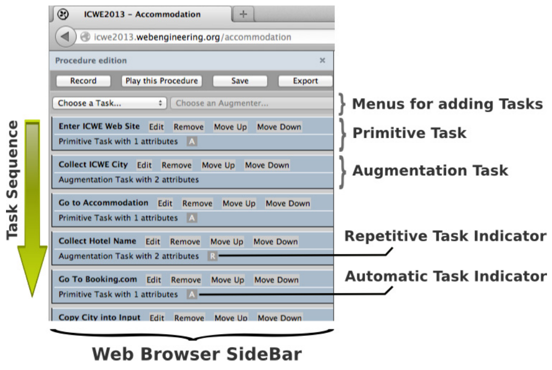
- **Preconditions:** preconditions are used to decide if the task will be executed or not according to which information is available. There are two main kinds of preconditions. On the one side, *preconditions about collected data*: for conditioning the execution of a task according to the collected data. On the other side, *preconditions about navigational history*: for conditioning the execution of a task according to the Web applications used.
- **Post-conditions:** post-conditions are specified to determine the effect of executing a particular task. For example, *AffectCurrent* is used to specify that the execution will modify the current Web site.
- **Attributes:** refer to data required to accomplish tasks. Attributes (with name, values, etc.) are specified as metadata for each task.

ComposedTask allows creating dependencies in the DSL. With this kind of task a finite sequence of tasks can be manage altogether in order to mark as *repetitive* or *optional* this entire block.

In the example from Figure 3, we have used both pre and post conditions. For example in the augmentation task *IconifiedLink* we have specified the *AffectSubset* precondition with a regular expression that matches with all Wikipedia articles. In this way, when a new *"PoI"* is collected, all Wikipedia articles will be adapted by adding the corresponding link to Google Maps (the focused Wikipedia article and any other opened in non-focused Browser tabs). In order to show an example of precondition, we have used the *PocketHasInstanceOf* one in order to execute the augmenter only if an instance of "*PoI*" was collected.

5 Tool Support

We have developed two tools: an editor for creating procedures and a procedure player for parsing and executing procedures.



Enter ICWE Web Site Edit Remove Move Up Move Down
Primitive Task with 1 attributes A

Collect ICWE City Edit Remove Move Up Move Down
Augmentation Task with 2 attributes

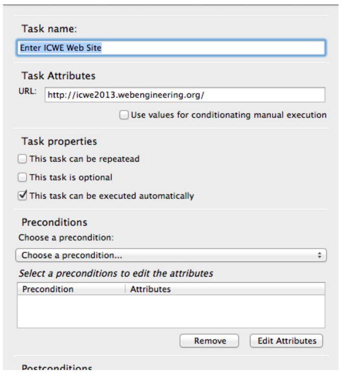
Go to Accommodation Edit Remove Move Up Move Down
Primitive Task with 1 attributes A

Collect Hotel Name Edit Remove Move Up Move Down
Augmentation Task with 2 attributes R

Go To Booking.com Edit Remove Move Up Move Down
Primitive Task with 1 attributes A

Copy City into Input Edit Remove Move Up Move Down

Fig 5.a. General view of the tool



Task name:
Enter ICWE Web Site

Task Attributes
URL: http://icwe2013.webengineering.org/
☐ Use values for conditionating manual execution

Task properties
☐ This task can be repeatead
☐ This task is optional
☑ This task can be executed automatically

Preconditions
Choose a precondition:
Choose a precondition...
Select a preconditions to edit the attributes

Precondition	Attributes

Remove Edit Attributes

Postconditions

Fig. 5.b. Edition of a single task

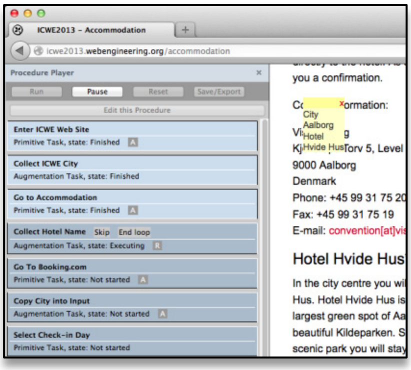
Figure 5.a shows the editor: a sidebar that allows users to specify tasks into the procedure while analyzing Web sites. The tool provides an assisted mode: users may *record* their interaction with the Web and the corresponding tasks will be added to the procedure automatically. This mode contemplates both primitive and augmentation tasks. Figure 5.b shows how to edit a task. It allows users to specify the name, pre- and post-conditions as well as values for both properties and attributes.

The Procedure Player is shown in Figure 6. When the user selects a procedure to be executed, this appears in the Procedure Player. Once it is running, the Procedure Player may execute automatically a task (if the tasks was marked as automatic). Those tasks that have been executed appear with different styles, in order to give visual feedback to users when a task was finished. For manual tasks the Procedure Player waits to the corresponding user interaction. When this happens the task state changes and the following task in the sequence is executed. When the procedure has finished, the user may share the procedure execution (which includes both tasks definition and data used in each task) for future executions or even for share with partners.

5.1 A Simple Case Study Using the Tools

Figure 6 shows the execution of a procedure for planning a trip to ICWE2013. The first task "*Enter ICWE Web Site*" is automatic and it loads the ICWE2013 Web site. Then the procedure waits a manual task, which require from users to collect a *City* into the Pocket. Once it is made, the procedure loads the accommodation page.

The task "*Collect Hotel name*" allows user to collect hotel names. After that, an automatic task opens the site booking.com for searching rooms. Figure 6.b shows the booking.com loaded with the "Destination" input filled with the city previously collected. The procedure follows with augmenters for highlighting the selected hotels.



Enter ICWE Web Site
Primitive Task, state: Finished A

Collect ICWE City
Augmentation Task, state: Finished

Go to Accommodation
Primitive Task, state: Finished A

Collect Hotel Name Skip End loop
Augmentation Task, state: Executing R

Go To Booking.com
Primitive Task, state: Not started A

Copy City into Input
Augmentation Task, state: Not started A

Select Check-in Day
Primitive Task, state: Not started

City
Aalborg
Hotel
Hvide Hus

you a confirmation.

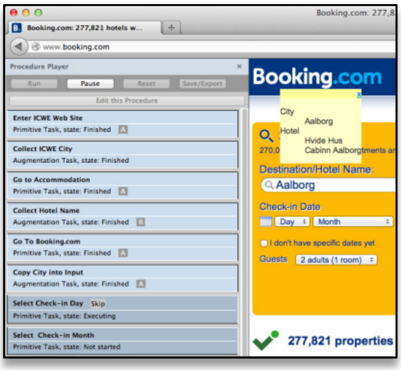
Co ormation:

Vi g
Kj Torv 5, Level
9000 Aalborg
Denmark
Phone: +45 99 31 75 20
Fax: +45 99 31 75 19
E-mail: convention[at]vi

Hotel Hvide Hus

In the city centre you wi
Hus. Hotel Hvide Hus is
largest green spot of Aa
beautiful Kildeparken. S
scenic park you will stay

Fig. 6.a. Task Execution: collecting accommodation



Enter ICWE Web Site
Primitive Task, state: Finished A

Collect ICWE City
Augmentation Task, state: Finished

Go to Accommodation
Primitive Task, state: Finished A

Collect Hotel Name
Augmentation Task, state: Finished R

Go To Booking.com
Primitive Task, state: Finished A

Copy City into Input
Augmentation Task, state: Finished A

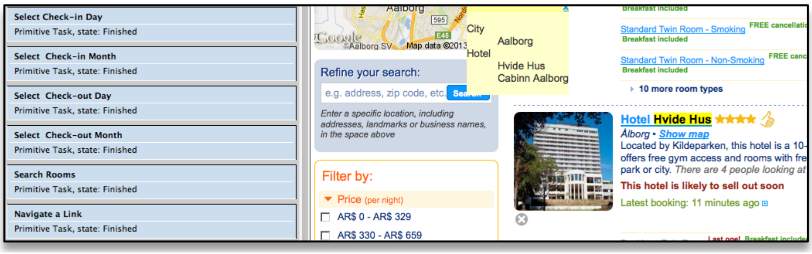
Select Check-in Day Skip
Primitive Task, state: Executing

Select Check-in Month
Primitive Task, state: Not started

City
Aalborg
Hotel
Hvide Hus
Cabinn Aalborg

Destination/Hotel Name: Aalborg
Check-in Date Day Month
I don't have specific dates yet
Guests 2 adults (1 room)
277,821 properties

Fig. 6.b. Task Execution: looking for hotels rooms



Select Check-in Day
Primitive Task, state: Finished

Select Check-in Month
Primitive Task, state: Finished

Select Check-out Day
Primitive Task, state: Finished

Select Check-out Month
Primitive Task, state: Finished

Search Rooms
Primitive Task, state: Finished

Navigate a Link
Primitive Task, state: Finished

City
Aalborg
Hotel
Hvide Hus
Cabinn Aalborg

Refine your search:
e.g. address, zip code, etc. Search
Enter a specific location, including addresses, landmarks or business names, in the space above

Filter by:
Price (per night)
ARS 0 - ARS 329
ARS 330 - ARS 659

Standard Twin Room - Smoking FREE cancellatio
Breakfast included
Standard Twin Room - Non-Smoking FREE canc
Breakfast included
10 more room types

Hotel Hvide Hus ★★★★
Ålborg • *Show map*
Located by Kildeparken, this hotel is a 10-
offers free gym access and rooms with fre
park or city. *There are 4 people looking at*
This hotel is likely to sell out soon
Latest booking: 11 minutes ago

Fig. 7. Trip to ICWE procedure execution: searching and highlighting collected hotels

Figure 7 shows the procedure state once the user has finished several primitive tasks for searching for rooms. Once the results are shown, the hotel names collected are used by the task "*Highlight collected Hotels*" which adapt the current Web page for highlighting the relevant hotels. Once the hotel room payment is finished, the procedure gives the same support for buying flight tickets: it opens expedia.com, prefills the forms for search (it uses the geolocation component provided by the framework), etc. since some tasks are marked as automatic. Finally, it supports to the user in the task of filling forms with his personal data.

5.2 Evaluation

We have evaluated empirically the approach by performing the same task in different ways: manually, automatically with other tool (Selenium) and with procedures. We defined three procedures with different levels of automation: i) repeating the task structure but reentering all information, ii) repeating task structure and reusing information from previous execution, iii) fully automated. We assessed quantitatively the interactions made by the user using GOMS-Keystroke (KLM) model [6]. The GOMS-Keystroke (KLM) allows to simulate the performance of a trained user proposing the average time to perform basic action (for instance, *reach for mouse* takes 0,40 sec). Thus, provided a detailed scenario of user actions including low-level user actions, it is possible to estimate user performance (i.e. speed).

The task was *Planning a Trip to ICWE, which* implied to use three different Web sites: i) ICWE2013 home page to get information about the conference; ii) Expedia.com to buy flights tickets; and iii) boking.com to book a room in one of the conference hotel.

Table 1 summarizes the results obtained with each approach. The task was decomposed into smaller ones in order to show when the use of a tool makes the difference. A first task, *Create Artifact*, is only valid when a tool for automating tasks is used.

Table 1. Results of the evaluation

			Procedures		
Task	**Normal Use**	**Selenium**	**Semi automatic**	**Semi automatic with data reutilization**	**Automatic**
Create Artifact	-	*9,5*	-	*472,2*	-
Execute Artifact	-	-	9,5	9,5	9,5
Get information about the conference	12	9,5	14,2	14,2	0
Search Flights	35,9		1,7	1,7	0
Select Flights	5		6.3	6.3	0
Enter Passenger Information	25,5		25,5	0	0
Pay Flights	59,7	9,5	59,7	1,7	0
Search Room	19,9		3,6	3,6	0
Select Room	6,5		5,1	5,1	0
Enter Passenger Information	29,4		27,5	0	0
Pay Room	51,4	9,5	51	4,8	0
Total	**245,3**	**28.5**	**202,8**	**46,9**	**9,5**

Table also shows how much time was necessary with each approach. The most time consuming was *the normal use* (245,3s). Selenium consumed 28.5s. The automatic *procedure* was the fastest. However it can be counterproductive since users lose the control over task. Semi-automatic execution only reproduced automatically those aspects like prefilling forms, and opening URLs when the previous task is finished, etc. Semi-automatic execution with data reutilization implies more automation by reusing data used in previous executions of the procedure such as prefilling forms with passenger information, credit card information, etc. In this case each confirmation steps (i.e. clicking search buttons) were performed manually. Finally, the full-automated procedure performs even these last actions, but leaving the user unable to control the task. Defining the procedure took 472,2 sec. This time would be lower/higher accordingly to the automation level used. We only measured the case we thought was the best choice in our approach.

6 Conclusions and Future Work

We presented an approach and DSL for orchestrating user tasks over the Web. The approach allows easy integration of client-side scripts to build procedures that can be share with other users. The DSL provides a certain level of abstract that could be used to analyze the sequences of users' tasks used in procedures compositions. Each task may be pre-conditioned, and the data is not fixed a priori (the approach contemplates data collection as tasks); which gives flexibility. Manual tasks are contemplated too, in order to give control to users who may feel uncomfortable if the whole task is delegated in an automatic tool.

The case study presented shows that the tools are completely functional. An empiric evaluation shows how the approach improves the performance in the execution of complex tasks. However we need additional studies to explore the usability and potential of user adoption of such tools. In addition with user testing of the tools, future work will address the possibility of having synchronous communication between users performing procedures. Our ultimate goal is to allow users who create and share procedures with friends, be able to follow the execution of the procedures.

The approach opens up the way for potential collaboration between users. By sharing *procedures* or even synchronize users' *procedures* execution would allow users to collaborate in order to accomplish a task altogether or even to share a procedure execution with a partner.

References

1. Bolin, M., Webber, M., et al.: Automation and customization of rendered web pages. In: UIST 2005, pp. 163–172. ACM Press (2005)
2. Card, S., Moran, T., Newell, A.: The psychology of human-computer interaction, p. 448. Lawrence Erlbaum Associates, Hillsdale (1983)
3. Byrne, M.D., John, B., Wehrle, N., Crow, D.: The tangled Web we wove: a taskonomy of WWW use. In: Proc. of Conf. on Human factors in computing systems (CHI 1999), pp. 544–551. ACM, New York (1999)
4. Firmenich, S., Winckler, M., Rossi, G., Gordillo, S.: A Framework for Concern-Sensitive, Client-Side Adaptation. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 198–213. Springer, Heidelberg (2011)
5. Leshed, G., Haber, E., Matthews, T., Lau, T.: CoScripter: automating & sharing how-to knowledge in the enterprise. In: Proc. of ACM SIGCHI 2008, pp. 1719–1728. ACM Press (2008)
6. Morris, M.R.: A survey of collaborative web search practices. In: Proc. of ACM SIGCHI 2008, pp. 1657–1660. ACM Press (2008)
7. Selenium, `http://jroller.com/selenium/` (last visit: February 26, 2013)
8. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding Mashup Development. IEEE Internet Computing 12, 44–52 (2008)