HAL
open science

An approach for supporting distributed user interface orchestration over the Web

Sergio Firmenich, Gustavo Rossi, Marco Winckler, Philippe Palanque

To cite this version:

Sergio Firmenich, Gustavo Rossi, Marco Winckler, Philippe Palanque. An approach for supporting distributed user interface orchestration over the Web. International Journal of Human-Computer Studies, 2014, vol. 72 (n° 1), pp. 53-76. 10.1016/j.ijhcs.2013.08.014 . hal-01153939

HAL Id: hal-01153939

https://hal.science/hal-01153939v1

Submitted on 20 May 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive TOULOUSE Archive Ouverte (OATAO)
OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : http://oatao.univ-toulouse.fr/
Eprints ID : 13297

To link to this article : DOI: 10.1016/j.ijhcs.2013.08.014
URL : http://dx.doi.org/10.1016/j.ijhcs.2013.08.014

To cite this version : Firmenich, Sergio and Rossi, Gustavo and Winckler, Marco Antonio and Palanque, Philippe *An approach for supporting distributed user interface orchestration over the Web.* (2014) International Journal of Human-Computer Studies, vol. 72 (n° 1). pp. 53-76. ISSN 1071-5819

Any correspondance concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

An approach for supporting distributed user interface orchestration over the Web

Sergio Firmenich [a], Gustavo Rossi [a], Marco Winckler [b,*], Philippe Palanque [b]

[a] *Universidad Nacional de la Plata and CONICET, Argentina*
[b] *ICS-IRIT, Université Paul Sabatier, France*

Keywords:
Distributed user interfaces
Task and process modeling
Web application
Web augmentation
Collaborative Web tasks

A B S T R A C T

Currently, a lot of the tasks engaged by users over the Web involve dealing with multiple Web sites. Moreover, whilst Web navigation was considered as a lonely activity in the past, a large proportion of users are nowadays engaged in collaborative activities over the Web. In this paper we argue that these two aspects of collaboration and tasks spanning over multiple Web sites call for a level of coordination that require Distributed User Interfaces (DUI). In this context, DUIs would play a major role by helping multiple users to coordinate their activities whilst working collaboratively to complete tasks at different Web sites. For that, we propose in this paper an approach to create distributed user interfaces featuring procedures that are aimed to orchestrate user tasks over multiple Web sites. Our approach supports flexible process modeling by allowing users to combine manual tasks and automated tasks from a repertoire of patterns of tasks performed over the Web. In our approach, whilst manual tasks can be regarded as simple instructions that tell users how to perform a task over a Web site, automated tasks correspond to tools built under the concept of *Web augmentation* (as it *augments* the repertoire of tasks users can perform over the Web) called *Web augmenters*. Both manual and automated tasks are usually supported by specific DOM elements available in different Web sites. Thus, by combining tasks and DOM elements distributed in diverse Web sites our approach supports the creation of procedures that allows seamless users interaction with diverse Web site. Moreover, such an approach is aimed at supporting the collaboration between users sharing procedures. The approach is duly illustrated by a case study describing a collaborative trip planning over the Web.

1. Introduction

Currently, many tasks users engage over the Web involve dealing with different Web sites; for example, planning a simple trip would require the visit of a first Web site for booking a hotel, a second one for booking flights and many more for finding interesting sightseeing places at the destination… Despite the fact that users would consider such Web navigation as being part of the same task (i.e. planning a trip) most Web sites will run independently with little support to the actual users' concern (Firmenich et al., 2010). Moreover, although Web navigation was regarded in the past as a solitary activity, a large proportion of users are nowadays engaged in collaborative activities (Morris, 2008); for example sharing with colleagues the results of a search for cheap hotels, explaining to friends how to book a seat next to yours in a flight, outsourcing tasks such as asking the community to suggest nice sightseeing places at the destination… For a motivating example, in Fig. 1 we illustrate a scenario for collaborative trip planning to attend a conference. For accomplishing this common goal, two users need to gather general information about a conference such as the conference dates and location, buy flights, book hotel, etc. Each user can perform the required tasks individually. However, if users want to travel together, some coordination and communication will be required for booking the same flights, hotels etc. Moreover, users might decide to share the work, for example one user can book for both participants. In the scenario presented by Fig. 1 it is possible to notice the many Web sites that will be visited by users. As we shall see, despite the fact that there is dependency between the information provided, Web sites are not integrated. While building service-based software such as mashups can be a solution for combining data and information from different providers, many times this approach might have limitations as they can hardly integrate all possible opportunistic tasks that users might have in mind; for example, checking the traffic situation on the way to the airport, booking the preference users' restaurant at the destination, etc.

We argue that there is a huge set of processes and tasks (such as planning a trip collaboratively) performed nowadays over the

* Corresponding author. Tel.: +33 5 61 55 62 58.
E-mail address: winckler@irit.fr (M. Winckler).

http://dx.doi.org/10.1016/j.ijhcs.2013.08.014

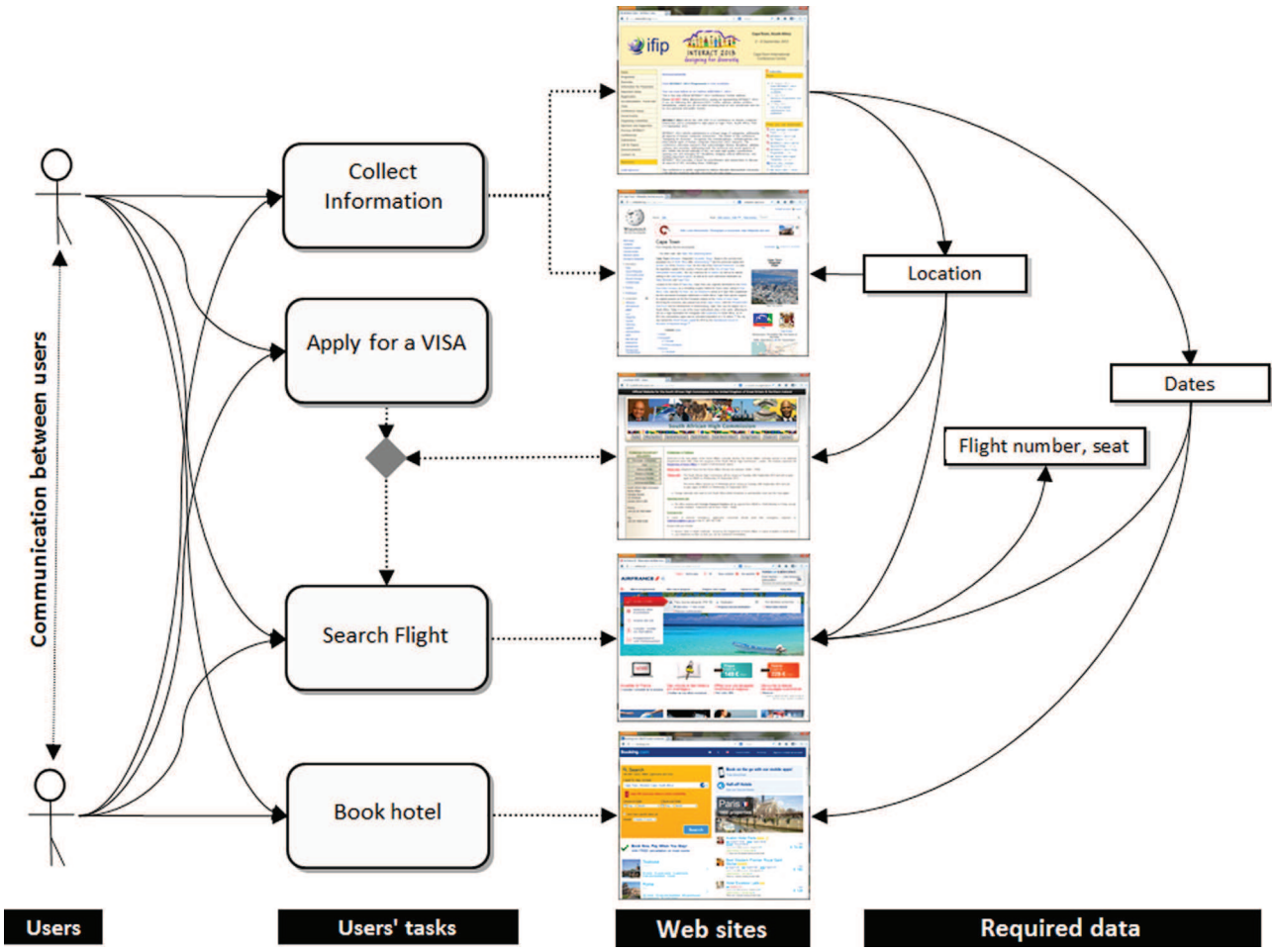

Fig. 1. Distributed user interaction for planning a trip including information sharing between users.

Web which requires a level of coordination that can only be achieved by Distributed User Interfaces (DUI). Distributed User Interfaces (DUI) has been recently defined as a "*user interface whose components are distributed across one or more of the dimensions input, output, platform, space and time*" (Gallud et al., 2011). The example shown by Fig. 1 highlights two important aspects of user activity over the Web that appeal for distributed user interfaces: (i) users tasks are distributed at multiple applications, i.e. Web sites, that are not directly connected to each other; (ii) users working collaboratively share information to accomplish a common goal (i.e. travel together).

In this paper we propose an approach to building distributed user interfaces that aim at ensuring a smoother user interaction whilst users are performing their tasks across multiple Web sites. Our approach combines individual user tasks to create procedures that seek to orchestrate user tasks over multiple Web sites. Such an approach is aimed at: (i) allowing flexible tasks modeling so that users can create ad-hoc processes for describing how to accomplish tasks in different Web sites; (ii) helping users to share such processes with friends and colleagues who might benefit of the guidance provided by prior task planning; (iii) support a seamlessly integration between data available over Web pages and the actual tasks performed by users across different Web sites; (iv) support the automation of user tasks by the means of Web augmentation tools that are aimed at helping users in their tasks; (v) mediate user interactions with the Web site via the execution of the so-called Web augmentation tools.

Our approach is built upon the concept of Web augmentation (Bouvin, 1999; Brusilovsky, 2007; Brusilovsky et al., 2007) that defines strategies for implementing tools that can extend the set of elementary tasks users can do whilst navigating the Web. For that, we have developed a dedicated framework called Context Sensitive Navigation (CSN) (Firmenich et al., 2010) which implements a set of Web augmentation tools called *augmenters* (Firmenich et al., 2011). These augmenters are the basic building blocks for extracting information and DOM elements from diverse Web sites for creating distributed user interfaces. In Section 2 we revise the main concepts that are necessary to understand how our proposal is related to the state of the art on DUIs. Section 3 describes the CSN framework and individual augmenters. Later on in Section 4 we properly present an overview of our approach for building DUIs by composing individual augments and a Domain Specific Language (DSL) that formalizes the composition of procedures made of the assembly of individual tasks (Section 4.2). The section about the approach ends with the description of the corresponding tool support (Section 4.3). Section 4.4 presents a case study of a DUI for collaborative trip planning using the tool support that demonstrates the feasibility of our approach. Section 6 reports the preliminary results of a usability testing with 11 participants. In Section 2.4 we discuss the contributions of our approach to the research in distribute user interfaces (DUI). Finally, Section 7 presents conclusion, lessons learned and future work.

2. Related work

The field of Distributed User Interfaces is broad and spans over the development of Web applications. For the sake of conciseness

we will concentrate on those research works involving Web applications which are close to our intent.

2.1. Collaborative work on the Web

There are many tasks that users can perform collaboratively in small groups (i.e., colleagues, family, or friends) such as researching group projects or reports, arranging joint travel, or planning shared entertainment opportunities. With the advent of Web 2.0 paradigm, several Web applications implement tools that allow users to interact and collaborate with each other in a social media (Morris, 2008). The collaboration using Web 2.0 technology is often focused a single Web site where the context of the task they allowed to do is defined by the Web site (e.g. edit a document, annotate page, upload images, etc.); a popular Web 2.0 tool is Google Docs (docs.google.com) where multiple users can edit collaboratively a single document.

However, more recent tools extended the concept of collaboration for supporting distributed user interfaces over multiple devices. As discussed by Paul and Morris (2009), most of these tools support awareness features (e.g., sharing of group members' query histories, browsing histories, and/or comments on results) and division of work features (e.g., chat systems, the ability to manually divide search results or URLs among group members, and/or algorithmic techniques for modifying group members' search results based on others' actions).

There is strong evidence that people frequently share the results of their searching of the Web (Melchior et al.; Morris, 2008; Schmid et al., 2012). In the last years much attention has been focused on collaborative Web search (Rädle et al., 2012). Several tools, such as CoSearch (Amershi and Morris, 2008), SearchTogether (Morris and Horvitz, 2007), CoSense (Paul and Morris, 2009) and Twister-Search (Rädle et al., 2012), support collaborative activities of searching over the Web. CoSearch (Amershi and Morris, 2008) features a specialized browser that implements a queue of queries performed by the group and users can associate notes to individual Web pages. Users can share the work by downloading distinct subsets of search results to their mobile phones. SearchTogether (Morris and Horvitz, 2007) is a prototype that enables remote users to synchronously or asynchronously collaborate when searching the Web by implementation mechanisms for awareness (people can see the activities of other participants), division of labor (search can be split among individuals), and persistence (results are available for later use). SearchTogether is so far a tool for supporting the investigation of sensemaking (which means how people value or give a meaning to information) in collaborative search. In order to investigate more precisely how sensemaking activities occur in Web navigation, Amershi and Morris (2008) have specifically designed the tool CoSense. CoSense supports collaborative searching as active discussion among participants around the results found by the group.

2.2. Web augmentation and data integration tools

After having found useful information over the Web, it is very likely that information will be used to accomplish other tasks. Indeed, quite often the contents users need to accomplish their goals are scattered around different Web sites. In more recent years, several tools have been developed to support the integration of Web contents into a seamlessly user interface. By doing so, the interface produced is “augmented” with respect to what users can do at the original Web sites. We can identify two coarse-grained approaches for developing such Web augmentation tools: (i) mashing up contents or services in a new application and (ii) augmenting the original application, generally by running adaptation scripts on the client side. Mashups (Yu et al., 2008; Wong and Hong, 2007) are an interesting alternative for final users to combine existing resources and services in a new specialized application. Most of the approaches aim at allowing users without programming skills to produce these new applications. Visual and intuitive tools such as (Daniel et al., 2009; Yahoo Pipes!, 2012) simplify the development of these applications. Some of the approaches are focused on UI integration, but others – such as Yahoo Pipes! (Yahoo Pipes!, 2012) – are focused on data integration. One of the main problems about mashups is that they, usually, are based on Web services. The fact is that most Web applications do not provide Web services to access their functionality or information. In order to solve this problem, Han and Tokuda (2008) propose an approach to integrate contents of third party applications by describing and extracting these contents at the client-side and then, using these contents later by generating virtual Web services that allow accessing them.

As we said before, the goal of mashups tools is to integrate UI or data in order to generate a new application. Clearly, this means that Web applications are taken off their original versions, which could not be desirable for many users; moreover Web applications are really dynamic about functionalities or contents, for example, news, special offers, etc. In contraposition, several tools for Web augmentation exist which aim to modify the preferred Web sites of users, while maintaining intrinsic features of these. Usually, these tools are implemented as browser's plug-ins that allow users to either install or generate scripts for modifying the Web pages they visit; for example GreaseMonkey (Diaz et al., 2010; GreaseMonkey, 2012; Pilgrim, 2005) and Scriptish (Scriptish, 2012). Nowadays, the most popular tool is GreaseMonkey, a browser plug-in for client-side scripting. Basically GreaseMonkey executes scripts (JavaScript files installed and created by users) when a target Web page is loaded. Then the Web page DOM is modified before it is shown to the user. Most of the GreaseMonkey scripts adapt Web pages in a static way, i.e. that the adaptation goals do not contemplate which is the user task or concern. Although passing information among Web applications with GreaseMonkey scripts is not so easy to achieve, this is not the common goal of GreaseMonkey scripts.

However, there are other approaches for supporting users' task in several ways. For example, MozillaUbiquity (MozillaUbiquity, 2012) – another Firefox plug-in – tries to improve user experience by empowering his browser with commands for dispatching operations related with his task. With MozillaUbiquity users execute commands (developed by the community) for specific operations; while some of them are more task-related (for instance, to publish some text from the Web site he is navigating in a specific social network) others can be addressed for adapting the current Web page in some way (for example, highlighting some text). MozillaUbiquity could be used even for building mashups. These commands are executed under user demand, and adaptations are not made automatically. Although MozillaUbiquity allows users to develop and execute commands for making shorter the distance between two distinct Web Applications, moving information (and performing adaptations with this information) from one of them to another is not fully exploited. A similar tool is Operator (Operator, 2012). Operator is another Firefox plug-in that performs this kind of operation by basing the available ones in correspondence to the Microformats found in the Web page the user is visiting. By finding a specific Microformat, Operator can use it for consuming Web services from Flickr, Google Calendar or del.icio.us, in order to support users with a specific task, for example, to add some event in Google Calendar based on some Microformat about dates or events found in any Web page.

Some tools have been specifically designed to automate repetitive users' tasks over the Web; for instance, CoScripter

(Bogart et al., 2008) (another Firefox Plug-in developed by IBM) allows users to record their interactions – Web pages visited, DOM events like clicks over DOM elements, etc. – and then, they can repeat the process automatically later. Although most of the scenario recorded is rigid, the approach is a bit flexible because it allows users to repeat the same scenario (Web sites, interactions, etc.) while changing the information entered in form inputs when the scenario was recorded. In this way, CoScripter is not useful when users need to change slightly the process, for example by changing one of the involved Web applications. This is not a minor issue since it is normal that users utilize diverse Web applications for the same goal (a clear example is booking a hotel room). While CoScripter is more of a tool for supporting repetitive business processes, we think that users' tasks cannot be only supported by filling forms but that there may exist several kinds of DOMs interventions for improving users' experience. CoScripter is not the only "high level API" for supporting processes on the Web. For example, ChickenFoot (Bolin et al., 2005) is a tool for programming users scripts and it extends JavaScript by defining new commands like "click()", "enter()", "find()", etc., which make easier the development of Web automation scripts. Although ChickenFoot has a powerful and expressive language, it hardly contemplates fine changes in the adaptation process and it prevents reuse of scripts as these are really DOM-dependent. The tool Koala (Little et al., 2007) proposes a more natural scripting language; for example, the expression "click('search button')" in ChickenFoot would be equivalent to "Click 'search button'" in Koala. Selenium (Selenium, 2012) is another tool based on recording users interactions. Although it was originally for performing UI tests, it is not limited to that and, in the same way as ChickenFoot and CoScripter, it could be helpful for performing repetitive tasks.

Although these tools are really useful and we share the philosophy behind them (in terms of putting the power on users hands, trying to make as easy as possible the end-user development, and trying to improve user experience), we think that the use of the Web is not as rigid as these approaches need. All these tools are based on registering all events (at least most of them) that occur in the Browser in order to be able to repeat the same sequence of events later. Anyway, behind each event that occurs in the browser there is a user intend, concern or a task in mind. This semantic weight behind each event is not taken into account, and for us, it is important to do it in order to understand in a better way which is the real user concern. In this way, we think that these tools are more for automatic-use Web applications rather than adapting them in order to add new contents (or adapt these), functionalities (or adapt these), which could be useful in the context of users tasks.

Moreover, we believe that it is necessary to go a step further in those aspects which imply an integration of information among several Web pages. In Firmenich et al. (2011) we showed how to use the actual user concern (expressed in his navigational history) as an additional parameter to adapt the target application. The main constraint in our previous work was that the development of the artifacts could be made only by experienced JavaScript programmers. This is not only counterproductive for creating new scenarios, but this mechanism is not good to analyze real behavior of users on the Web.

2.3. Task and process modeling

Task analysis is widely recognized as one fundamental way to focus on the specific user needs and to improve the general understanding of how users may interact with a user interface to accomplish a given interactive goal (Diaper and Stanton, 2004). In the field of HCI many task model notations have been proposed for capturing in models the various elements of user activity in interactive systems Card et al. (1983).

In the last decades, several tasks notations have been developed as means to describe work carried out by users whilst interacting with a system (Paternò et al., 1997). Despite the fact that various specific task notations exist, they are mainly structured around two concepts: task decomposition (often represented as a hierarchy) and task flow (for showing the order in which tasks are executed) (Limbourg et al., 2001). When adequately combined, these concepts can provide an exhaustive and complete representation of large quantity of information in a single model. However, as discussed by Paternò and Zini (2004), when applied to real-life systems, tasks notations end up in very large, hard-to-manage models thus making task modeling a time-consuming and sometimes painful activity.

More recent task model notations such as HAMSTERS (Martinie et al., 2011) integrate structuration mechanisms such as modularity in the tasks representation, reuse of sub-tasks elements, data flow and communication features between task elements. These features allows more flexibility in the organization of task-model diagrams and allow us to envisage the use of tasks models for describing complex interactive systems build upon the composition of individual tasks and their distribution into the user interface (Luyten and Coninx, 2005).

However, the experience with task-model for building DUIs is mainly driven but the construction of an application whose interface is going to be distributed among different devices and platforms (Luyten and Coninx, 2005; Manca and Paterno, 2011; Melchior et al.). In order to integrate tasks that can be performed in multiple Web sites, Daniel, Soi and Casati (Daniel et al., 2009) propose the concept of user interface orchestration that is aimed to describe how distributed user interfaces can be coordinated around a single business process. These works are promising and yet they demonstrate the value of tasks models for helping to build complex distributed user interfaces.

2.4. Positioning the contribution with respect to the DUI paradigm

Systems which support the management of complex tasks and of a huge amount of information usually require Distributed User Interfaces (DUIs) for allowing multiple users to perform their tasks in a (possibly) concurrent way. Several definitions of this kind of interfaces co-exist and present interestingly complementary viewpoints Vanderdonckt (2010).

In Lin et al. (2009), a UI distribution is defined as "*the repartition of one or many elements from one or many user interfaces in order to support one or many users to carry out one or many tasks on one or many domains in one or many contexts of use, each context of use consisting of users, platforms, and environments*". This definition focusses on the notion of repartition of UI elements in several locations. Another definition proposed by Elmqvist (2011) identifies five dimensions for the distribution of UI components: input, output, platform, space and time. This definition emphasizes the fact that there is no need to distribute on all these dimensions to be eligible for DUI paradigm. Demeure et al. (2008) also propose a reference framework (called 4C) to analyze DUIs, which is composed of four concepts: computation, coordination, communication and configuration. It builds on top of the design space identified by Salber based on Production, Coordination and Communication for groupware (Laurillau and Nigay, 2002).

In the contribution presented in this paper, we propose a tool-supported architecture allowing information exchange and dynamic tasks allocation between multiple distributed users involved in a common task. This contribution heavily relies on the exploitation of innovative Web technologies (as demonstrated in the case study presented in Section 4.4). However, the proposed

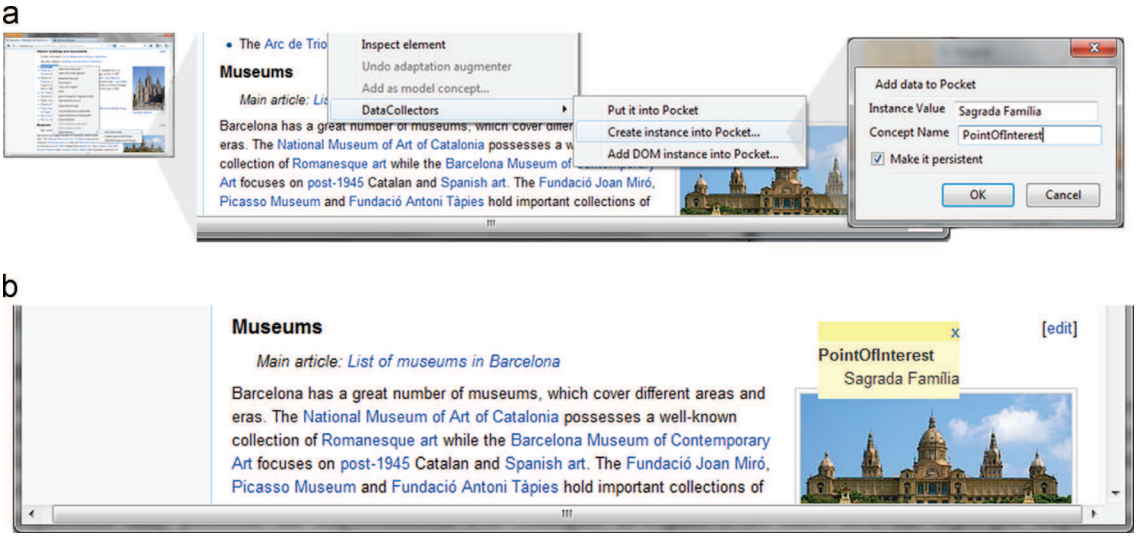

Fig. 2. Illustration of the use of an augmenter pocket. (a) Menu in the augmenter tool for collecting data whilst navigating a Web page. (b) Items collected with the augmenter pocket as they appear on Web pages.

framework could be transposed to any platform supporting multi-user interactions. Indeed, the construction of a procedure (that will be then allocated to a remote collaborator) is based on the blending in one single entity of several user interfaces components extracted from multiple Web sites. The same could be performed in a similar way between multiple interactive applications.

According to Elmqvist's definition of DUIs (Elmqvist, 2011) we can position our work as follows:

- Input: user input is distributed amongst the various users exploiting multiple Web sites.
- Output: in our approach, there is no specific contribution on output beyond the fact that output is tightened to input (which is distributed as explained above).
- Platform: even though the contribution does not address explicitly the distribution over various platforms, there is a high potential of distribution due to the close connection of the contribution with Web technologies.
- Space: the choice of Web technologies was explicitly made to support distribution of user and of their tasks in space. Indeed, the contribution directly targets at supporting multiple remote users involved in collaborative activities as demonstrated in the case study.
- Time: the contribution here mainly focusses on asynchronous collaborations as augmenters are meant to be "sent" to remote users who will exploit them at their will. Synchronous mechanisms could be added to the framework to provide communication support for performing the tasks but this has not been presented here.

3. Overview of an Web augmentation

Our approach for building distributed user interfaces is based on the CSN framework (Firmenich et al., 2010) and it relies on the existence of Web augmentation tools that can support users' tasks that use contents from multiple Web sites. Hereafter we show how to use Web augmentation tools to support users' tasks whilst the navigation over the Web. We introduce Web augmentation tools and the CSN framework. Later, we describe how such Web augmentation tools built with our framework can be combined to feature distributed user interfaces that can support processes performed by multiple users.

3.1. *Web augmentation tools and the CSN framework*

The concept of Web augmentation, as defined by Bouvin (1999), is used to refer to a set of tools that extend tasks users can usually perform on the Web. Several strategies for implementing Web augmentation tools exist (Diaz et al., 2010; Firmenich et al., 2010; GreaseMonkey, 2012; Pilgrim, 2005; Scriptish, 2012). In our previous work (Firmenich et al., 2010) we have presented our own framework called CSN that support the development of such Web augmentation tools called *augmenters.* Augmenters feature generic functions that can be executed on client-side (i.e. on the Web browser) and perform adaptation of DOM elements. Tasks supported by augmenters might include *automatic filling in forms*, *highlighting text*…

Fig. 2 illustrates the execution of an *augmenter* called *pocket*. This *augmenter* was implemented under the metaphor of a real pocket; thus users can collect data into the pocket and then carry on such data whilst navigating the Web. Data can be simple text values (such as "*Sagrada Familia*", "*Barcelona*") or text values with semantic meanings (such as like "*PointOfInterest*" or "*City*"). Fig. 2a shows how the augmenter is activated from a contextual menu over a Web page to create an instance of the concept *PointOfInterest* for the previous selected text "*Sagrada Família*". Once collected, the augmenter modifies the Web page in the client by creating a new DOM element featuring an electronic post-it to display the information collected by the user, as shown in Fig. 2b.

Fig. 3 illustrates another augmenter called *highlight* that can be activated directly from the information previously collected by the users using the augmenter *pocket.* In Fig. 3a, we show how the *Highlight* augmenter is triggered by right-click the target Pocket element, which opens the contextual menu with all the augmenters available. Fig. 3b shows the Web page already augmented, after executing the augmenter. Note that the *PointOfInterest* was collected from Wikipedia and is available on other Web sites. Since the augmenter was executed with the concept *PointOfInterest*, all its instances were contemplated for being highlighted. The augmenter can be executed only with one instance, for example "*La Rambla*".

Currently, a large set of augmenters are deployed with the CSN framework. These augmenters are grouped in Table 1 according to the following categories: data collectors (which at aimed to help users to collect information from Web pages), filling forms (which are focusing on users tasks involving Web forms), select

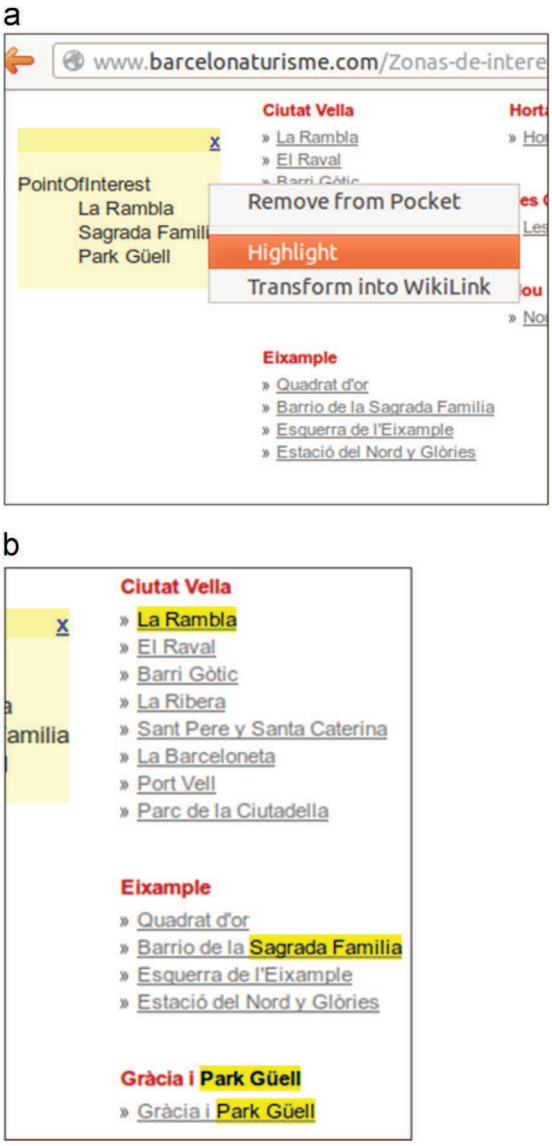

Fig. 3. Illustration of the use of an augmenter highlight. (a) Triggering the augmenter highlight. (b) Highlight effect on Web page.

data (in the Web page by editing CSS properties), navigation (which adds elements to Web pages for helping users to improve navigation) and accessibility (which concerns edits on the Web page that are aimed to improve the accessibility of the Web page).

It is interesting to notice that augmenters can be used in combination for supporting sequences of tasks. Fig. 4 shows how the augmenters *ConceptInstanceCollector* and *IconifiedLink* can be executed in the sequence in a row to create navigation from Web pages to the Google Maps site using data previously collected.

Augmenters can be used to support users or to automate tasks. For example, the augmenter *UntypedDataCollector,* which is a particular implementation of the *augmenter pocket* without semantic meaning for the values collected, acts as a memo or electronic post-it that follows users through his navigation on different Web sites. This augmenter helps users to remember important information whilst navigating the Web. Another example is the *CopyDataIntoInput* that can be used to automatically fill in form fields with data previously collected by any augmenter in the category *data collector*. These two examples clearly show that these augmenters are not only extending what users can do on Web pages (e.g. create electronic post-its) but that user performance can be improved by automating some of the user actions (i.e. automatically fill-in forms). It is also interesting to notice that *augmenters* provide users with an alternative for performing the tasks. For example, users can type the information previously collected into a form field or trigger the *augmenter CopyDataIntoInput* to perform the same action. At some extension, such augmenters allow distributed control over tasks that users perform on Web sites.

So far, we have shown examples where only plain text is collected. However, this is not the only kind of information that may be *moved* from one application to another. Sometimes is convenient to show several parts of the user interface from different Web sites at one time, such as what Mashups or Web Integration approaches do. In this work, users can manage DOM elements with the same flexibility as plain text, i.e.: collect DOM elements into the *Pocket* under some semantic label, and then performing adaptations based on this semantic label using augmenters. As a first example, in Fig. 5a we show how a user can collect a specific DOM element from Booking.com and then insert it in other Web site. This could be made by executing the corresponding augmenter *UseForReplace* that replaces the element which is selected by the user with the collected DOM element into the Pocket.

Fig. 5b shows the execution of this augmenter in the Airfrance.com Web site. The result of the adaptation is shown in Fig. 5c. Note that although the augmenter can be executed from the concept *HotelInfo*, and in this way the DOM element inserted may vary if a different DOM element was collected under this semantic tag.

The example from Fig. 5 shows how to apply concern-sensitive integration of UI components. Here, the user has finished the task of booking a hotel room, and when he was going to buy flight tickets, informations about the hotel (name, address, both check-in and check-out dates, etc.) were available in other Web page. This kind of integration allows users to interact with parts of the UI from an external Web site in the current one. For example, if the collected DOM element contains links or forms, then the user can interact with this element when inserted in other existing Web page.

3.2. *Users of the CSN framework*

The CSN framework is supported by the means of a Firefox plugin. A small set of basic Web augmenters are deployed with the installation of the plugin including the before mentioned *highlight* and *pocket*. However, it is possible to create and install additional augmenters into the plugin. Therefore, there are two kinds of users of the CSN framework:

(i) End-users who take advantage of existing augmenters deployed with the CSN plugin; these users benefit of these augmenters during navigating by "collecting" information to be used when adapting the user interface.
(ii) Developers who create and distribute new augmenters that can be played in the Web browser. Despite the fact that a large set of augmenters already exist, creative developers can use the framework to implement new augmenters at they will. With this structure we can put the power on the crowd, allowing any Web developer to create his own augmenter. Moreover, new augmenters can be shared by other users of the tools implemented with the CSN framework. This crowdsourcing development has been proved successful in other communities, for example with those using GreaseMonkey scripts (GreaseMonkey, 2012).

3.3. *Creating and describing a simple augmenter in the CSN framework*

This section shows how new *augmenters* can be created and instantiated into the CSN Framework. Overall, there is no need to

Table 1
Set of augmenters currently deployed with the CSN framework.

Augmenter	Category	Description	Execution mode
UntypedDataCollector	Data collector	Quickly add some data into the pocket without semantic weight	User/system
ConceptInstanceCollector	Data collector	Add conceptualized data into the pocket, for example, for the concept "*City*"" collect the text "Barcelona"	User/system
DOM Element Collector	Data collector	Collect a specific DOM element into the Pocket	User/system
Remote Collector	Data collector	Collect data from a remote Web application (not the current Web site)	System
RegularExpressionBasedCollector	Data collector	Add data into the pocket based in a regular expression, for example, for collecting e-mail addresses.	System
MicroformatBasedDataCollector	Data collector	Add data into the pocket based in Microformats annotations.	User/system
CopyDataIntoInput	Filling forms	Copy a value form the Pocket to the select input	User/system
AnnotateInputWithMicroformat	Filling forms	Annotate the Web forms inputs with Microformats	User/system
MicroformatFormFiller	Filling forms	Based in the Microformats found in the current Web page	User/system
Highlight	Select data	Highlight selected data from pocket in the current Web pages	User/system
Remove	Select data	Delete selected element from the current Web page	User/system
WikiLinkConvertion	Navigation	Convert selected data into links to the corresponding Wikipedia article	User/system
IconifiedLink	Navigation	Add links to Google Maps and Flickr	User/system
PopUpMessage	Navigation	Shows a Popup Message, under certain circumstances, where is suggested to the user some links for being followed.	System
NewNavigationMenu	Navigation	Add new navigational menu	System
ConvertPocketIntoMenu	Navigation	Transform each Pocket element into a Link relevant for the current application or user concern	System
DOM Inject	Integration	Inject the selected element of the Pocket (only DOM Instance elements) as a child of the selected element	User/system
Replace DOM element	Integration	Replace an element of the current Web page for the selected element of the Pocket (only DOM Instance elements)	User/system
Add floating DOM element	Integration	Insert as a floating UI component the selected element of the Pocket (only DOM Instance elements)	User/system
ReplaceImageWithAltText	Accessibility	Replace all Images an put in place the alt text	System
IncreaseTextSize	Accessibility	Increase the size of the target text	System

create new augmenters to follow our approach for building distribute user interface. However, the example below illustrates the potential of the framework to create new behavior and user interfaces that can be useful for building DUIs for the Web.

Generally speaking the CSN framework provides a template that can integrate a file containing a JavaScript functions in which a new object has to be defined inheriting from *AbstractAugmenter*, as illustrated by Fig. 6. In order to create an *augmenter* the following elements should be described: (i) Metadata information used to identify the augmenters; (ii) a method *getAugmenterInstance* which is used to get augmenter instances; (iii) a class definition; in the example the class *HighlightingAugmenter* is the new augmenter object. It is also necessary to provide a value to the label instance variable; this value is used to create the menu showing to users that the augmenter is available for use; and (iv) an extension point of heritance: the last line from the example shows how the new augmenter is a subclass of *AbstractAdapter* (the extension point for augmenters).

By following the simple template above, developers can create a huge set of augmenters to feature adaptations on the DOM of the Web pages where the augmenter is triggered. The use of the template ensures that the JavaScript functions are compatible with other augmenters in the framework. Thus, a community of developers can create augmenters and share them with all other developers and end-users of the CSN plugin.

4. Building DUIs with Web augmenters

In this section we present how we have extended the CSN framework to create distribute user interfaces.

4.1. The approach in a nutshell

The overall idea behind our approach is that it supports collaboration between users that accomplish complex processes through distributed interactions over many Web sites. Such processes, called procedures here, are built by assembling basic building blocks provided by the CSN framework, each augmenter supporting a single user tasks. Different tasks in a procedure can be performed in different Web sites but one augmenter is instantiated to support the user interaction with a specific Web site. The user interface of each augmenter features elements that are extracted from the user interface of the Web site instantiated by the augmenter. The corresponding user interfaces for the whole procedure is thus a composition of elements that are originally distributed across different Web sites.

Not all user tasks are (yet) supported by a specific augmenter; so that the user interface might include instructions for the user to perform certain tasks manually using the functions already supported by the Web browser. For that purpose we use the definitions made in previous works (Byrne et al., 1999; Heath, 2010) that describe what users can do on a Web browser. These tasks, that we call here "*primitive tasks*" include actions such as "*go to a Web page*", "*locate information*", "*configure the Web browser*", "*fill in a form*", etc. The primitive tasks used in our approach are heavily inspired by the work of Byrne et Ali (Byrne et al., 1999). However we assume that, according to new features implemented by most recent Web browser, the set of primitive tasks could be extended in the future. Fig. 7 provides a view at glance of the approach.

The composition of the user interface is done by creating a sequence of individual tasks, which is formalized by a Domain Specific Language (DSL) and stored as a XML file. A tool called procedure player is able to parse that XML file and reconstruct the procedures on the Web browser. It includes functions for supporting the collaboration between users. As we shall see, the approach includes three phases, as follows:

- Definition of *Primitive* and *augmentation* tasks: it concerns the inclusion of augmenters in the CSN framework. This phase required a highly skilled Web developer who is able to program augmenters. Whilst this task is demanding, the work should be done once and it will benefit many more users. The development of new augmenters is out of the scope of this paper. Nonetheless, the

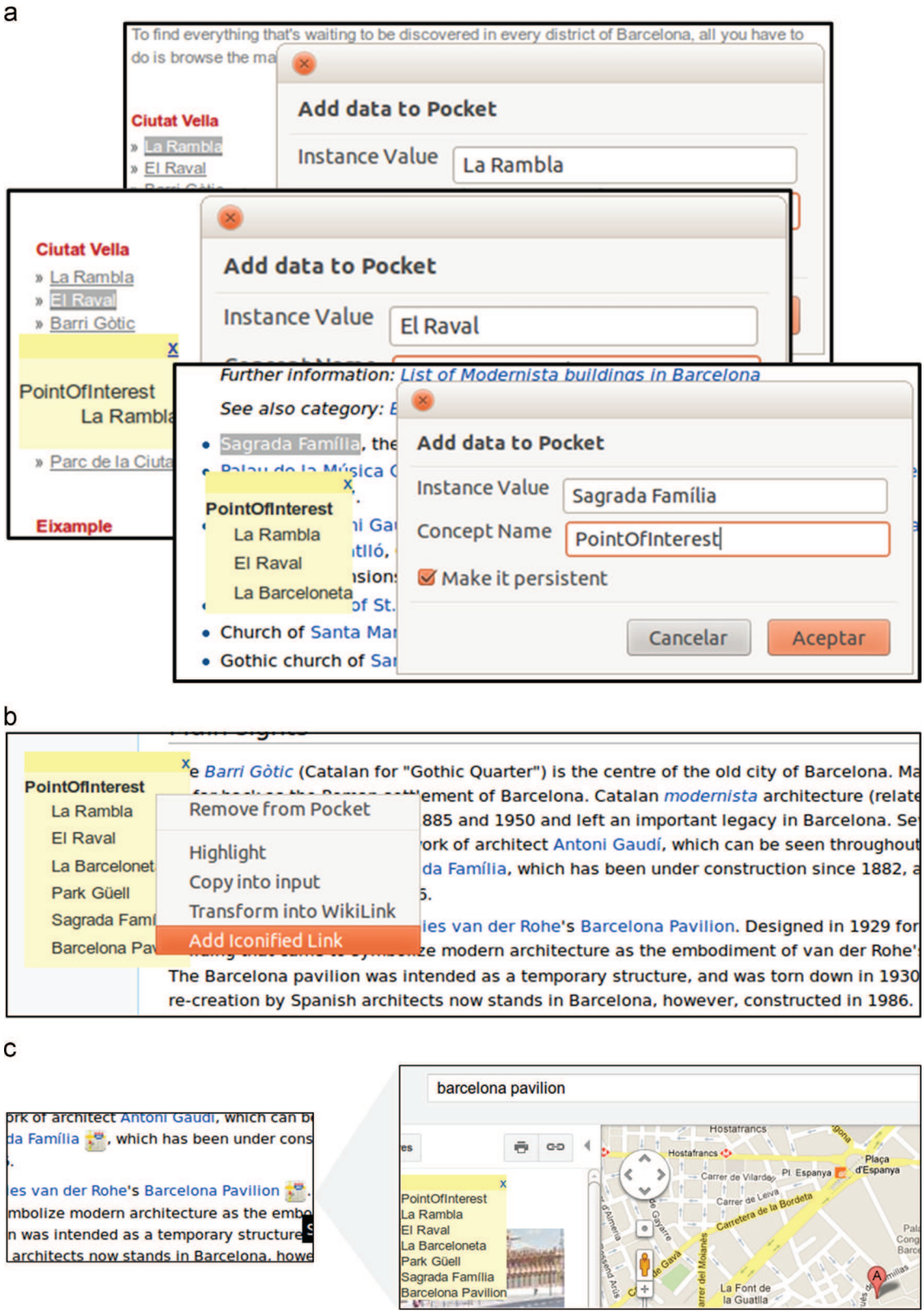

Fig. 4. Combined use of augmenters ConceptInstanceCollector and IconifiedLink. (a) Collecting 'Points of interest'. (b) Applying the augmenter IconifiedLink over collected 'Points of Interest'. (c) Result of the augmenters IconifiedLink over the previously collected point of the interest. At the left, a Web page augmented with the icons that feature a link to the Google Map Web site. At the right, the corresponding navigation which integrates the previously collected 'Points of interest' as an entry point for Google Maps.

approach is delivered with a large set of tasks, so that users do not need to create new *augmenters* to create their procedures.

- *Composition* refers to the definition of procedures by compositing tasks available in the *repository*. Users have to select a task from the repository to create a sequence of tasks. This sequence is stored in the *factory* by the means of a DSL describing all tasks in the procedure; this step is repeated until the composition is finished.

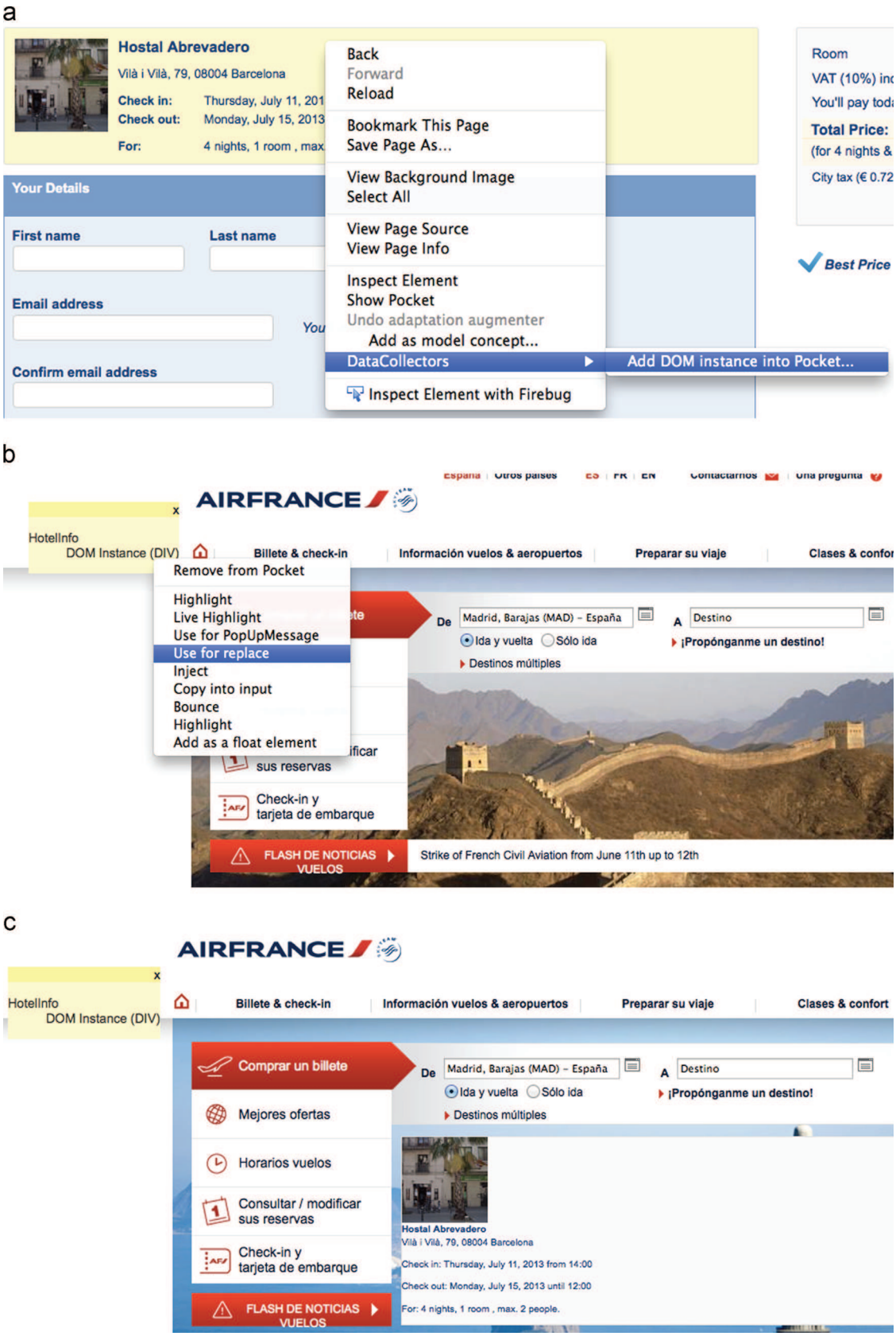

Fig. 5. Collecting DOM elements with DOMElementCollector augmenter and using the element in other Web page. (a) Collecting DOM element from Booking.com. (b) Execution of UseForReplace augmenter. (c) Result of applying Use For Replace augmenter.

```
// augmenter identification
var metadata = {"name":"My Augmenter Name", "author":"Author Name", "description": "This augmenter ........"};
// getter of augmenter instances
function getAugmenterInstance(){
   return new HighlightingAugmenter ();
};
// class definition
function HighlightingAugmenter(){
   this.label = "Highlight";
};
// point of inheritance in the CSN framework
HighlightingAugmenter.prototype = new AbstractAdapter();

```

Fig. 6. JavaScript template for building augmenters accordingly to the CSN framework.

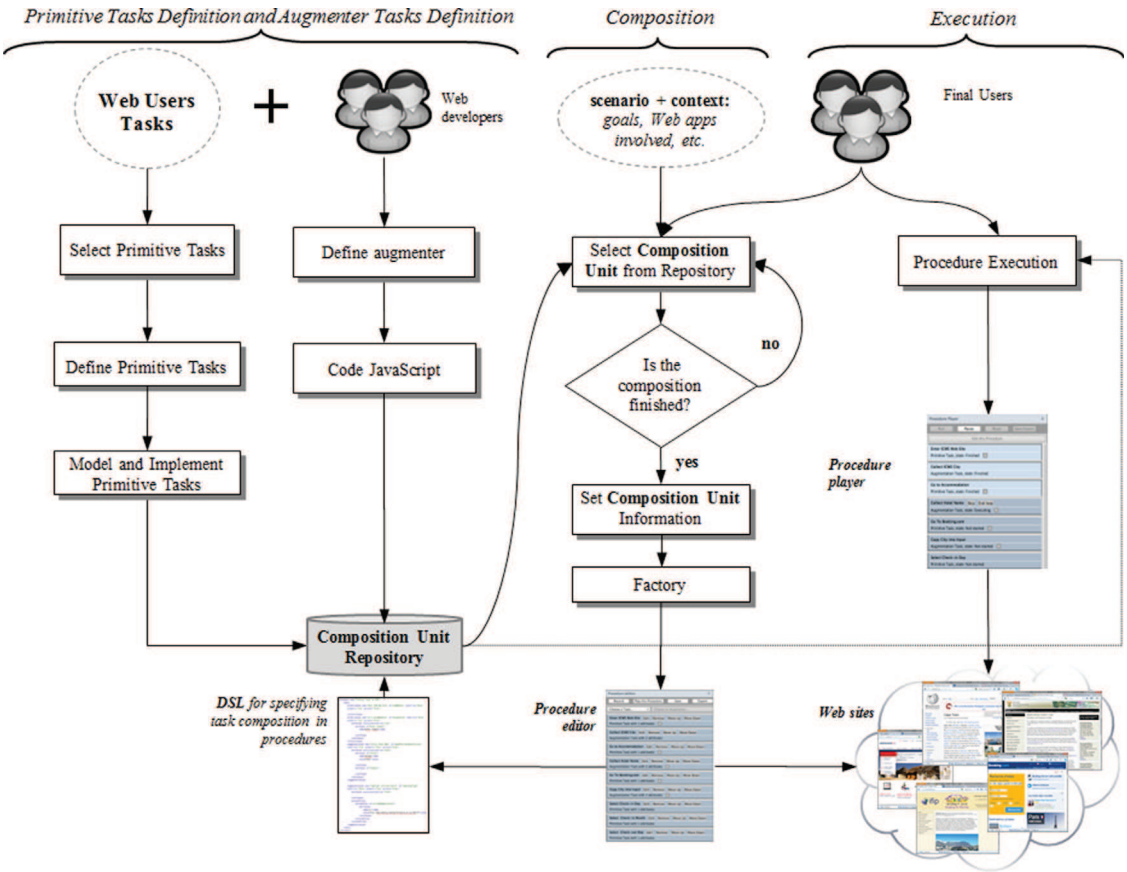

Fig. 7. Overview of the process leading to the creation of DUIs using task composition and Web augmenters.

- *Execution*: this phase features a player that is concerned by the execution of the procedure previously encoded by the DSL.

As shown in the figure above, procedures provide an alternative for the interaction with Web sites by automating users' tasks but they do prevent users from interacting directly with Web. Thus the user input is shared (or distributed) between the procedure player and the Web sites.

It is important to say that procedure player integrate functions that also support the collaboration between users, according to the policies for task synchronization described by the means of a Domain Specific Language (DSL).

4.2. A DSL for describing composition of users tasks on Web applications

In this section we provide a view at glance about the DSL that we have developed to specify the procedures and tasks in our approach. Task composition is defined according with the DSL metamodel shown in Fig. 8. This metamodel defines those elements contemplated by the DSL and their relations.

Basically, a procedure defined with our DSL is realized as an XML file containing a list of tasks. For each task additional properties can be added including *preconditions*, *postconditions* and *attributes*. The DSL elements and the corresponding syntax include the following:

- **Primitive tasks**: the tag `<primitivetask/>` describes these tasks. The current list of primitive tasks supported is based on Byrne et al. (1999).
- **Augmenters**: the tag `<augmentationtask/>` describes these elements. The list of augmenters depends on what is currently installed on the users' browser.
- **Composed tasks**: using the tag `<composedtask/>`, composed tasks are used to group a set of subtasks (either primitives or augmenters) in a single block.

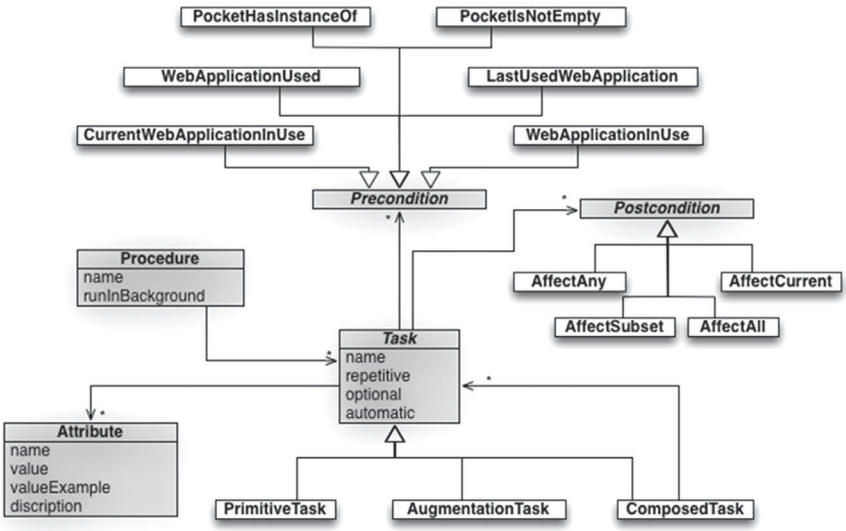

Fig. 8. The DSL metamodel.

- **Preconditions**: preconditions are used to decide whether (or not) tasks will be executed or not according to the information is available. These are described in the DSL with the tag `<precondition/>` which can be a:
  - **Precondition about collected data**: for conditioning the execution of a task according to the collected data. For example the precondition *PocketHasInstanceOf*() allows specifying that a task will be executed if the Pocket has an instance of a particular concept, for example, a *PointOfInterest* instance.
  - **Precondition about navigational history**: for conditioning the execution of a task according to the Web applications used. For example, *WebApplicationUsed* checks whether a particular Web site (defined by a parameter: a URL) ever appears in the browser navigational history.
- **Post-conditions**: post-conditions are specified to determine the effect of executing a particular task, and the tag `<post-condition/>` is used for this purpose. As shown by Fig. 8, there are four possible post-conditions. For example, *AffectCurrent* is used to specify that the execution will modify the current Web site.
- **Repetition, optional and automatic properties**: both Primitive and Augmentation tasks have three intrinsic properties. A *repetition* property specifies if the task may be executed more than once. The *optional* property allows skipping the execution of the current task. If the property *automatic* is *true*, then the player automatically triggers the task. In this case, all the attributes needed by the tasks need to have an associated value. Defaults values, *false* in the three cases, will be used if a task definition does not provide a value. In addition to these properties, each task can be set as ***synchronized***.
- **Attributes**: refer to data required to accomplish tasks. Ex.: the task *Provide URL* needs an attribute URL. Attributes, with their name and values, must be defined for each task in their specifications.
- **DUI component**: this property, only valid for Augmentation tasks, refers to DOM elements that may be extracted from Web sites and then associated with the task. The DUI component may be a specific DOM Element for static ones, and an xPath value in order to get the DOM element dynamically when the tasks is executed.

4.3. Tool support

We have developed tools as a proof of concept for our approach. Our tools help to edit, execute and synchronize the execution of procedures performed by multiple users on multiple Web sites. The tool presented hereafter is an extension of the previously described CSN framework, which allows developers to create and install new augmenters. The tools are delivered with a set of basic augmenters plus a set of primitive tasks that can be used in the composition of procedures. Fig. 9 shows the general architecture of our tools which relies on two main components:

- A task repository is a cloud service hosted in a Web server where users store tasks, augmenters, procedures and history information about the execution of procedures; and,
- A browser plugin available to the client that supports both the edition mode (allowing composing new sequences of tasks) and the execution mode (play task sequences).

In Fig. 9 we show how the user 1 can use the tool (i.e. task edition mode) in order to create a new procedure by composing basic tasks. Once the procedure is created, he can play and share his creation with other users. Once a procedure is created, the user can upload it into the repository. Other users can then download the procedure from the task repository and execute it in their Web browser using the plugin (i.e. task execution mode). It is noteworthy that the edition and execution of procedures are not supported simultaneously by our tools; indeed, a user cannot modify a procedure that is being executed by another user.

The synchronization between users, which is an important aspect for supporting collaboration, is implemented and delivered as part of the architecture of the tool. In the current implementation the user who creates and publishes a procedure will be notified when another user executes it. Usually, the synchronization between users will always occur at the end of the execution of a procedure. Nonetheless, tasks defined as ***synchronized*** will be updated in the task repository immediately after their execution in the client. Users can at any moment explore the task repository and check the current state of execution of the procedures they have published there.

The task repository has several responsibilities, including:

- Managing user profiles: users profiles are needed since several preferences and partners (other users with which usually collaborate in procedures executions) may be specified.
- Storing augmenters: this is a repository from where users can download and install new augmenters.
- Storing procedure: this is a procedure repository from where users can get procedures defined by other users.

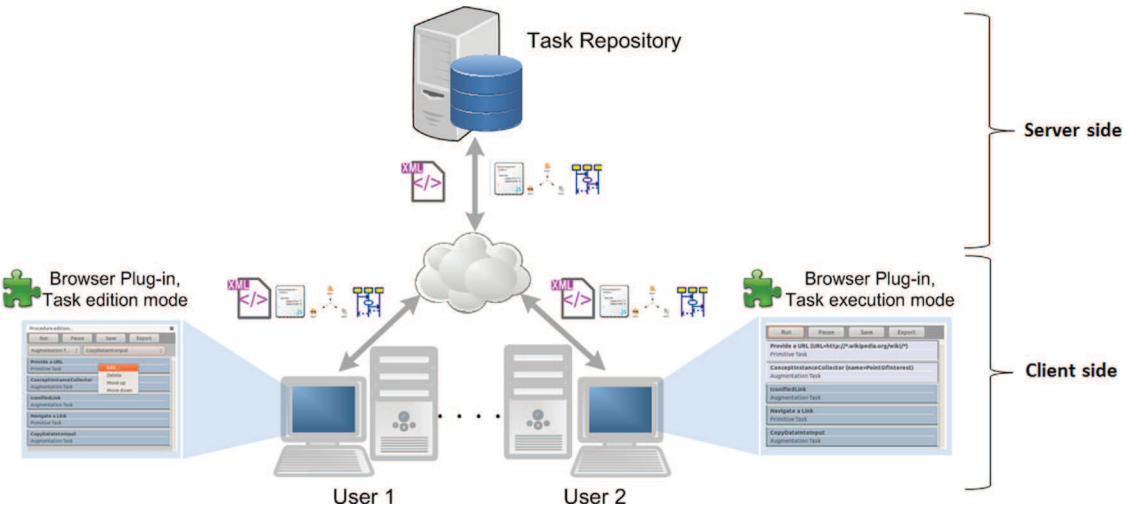

Fig. 9. Overall architecture of the approach describing the composition of tasks in the DSL.

- Storing procedure execution states: when users are collaborating in order to facilitate the procedure execution of each other, they can upload the states of their own execution; then, other users may take advantage of tasks which have been already executed with concrete data.
- Allowing synchronization between users: this is achieved by allowing users to share their procedures execution states with specific partners.

The client-side plugin allows users to:

- Edit and create new procedures by combining tasks (edition mode): this implies to modify the composition of tasks (both augmentation and primitive tasks) and the information these tasks are expecting to be executed.
- Execute a specific procedure (execution mode): by selecting one of the procedures locally installed, users may run it and then facilitate their tasks.
- Share both new procedures and information about execution of these: in the middle of an execution, we allow users to share the state of the procedure execution. Then, users may share with their partners the information used for the procedure.

The same client-side tool edits and executes procedures. Fig. 10 shows some screenshots of the tool in the edition mode. Each of the three screenshots (a, b and c) show at the top the main menu the buttons "run", "save" and "export", that correspond to commands related to the whole procedure being edited. Just above the main menu, Fig. 10a shows a vertical menu that is used to select a task that will be used in the composition of a procedure. The options: *Primitive Task*, which can be deployed in a second menu as shown by Fig. 10b; and *Augmentation Task*, as shown by Fig. 10c. These menus are built dynamically with the tasks locally available. If Augmentation tasks installed locally are not enough to perform some task, the user can install new augmenters. Once a particular task is selected; it is included in a list that appears just below these menus as shown by Fig. 11.

Using the tool in the edition mode, users can initialize tasks for working with a specific Web site. For example, Fig. 11a shows how the user edits the attribute "*Provide a URL*" task by accessing the task's options from the contextual menu. In this case, the task "*Provide a URL*" only has one attribute to be specified: the URL. Note that in Fig. 11b the URL entered is a regular expression (notice the "*"). This option is just for giving flexibility to the whole procedure. With this value for the attribute, the task would be considered completed once the user visits a Web site whose URL matches with this regular expression. The execution mode is illustrated by Fig. 11c. The tool shows five tasks, in which the first two (i.e. "*Provide a URL*" and "*ConceptInstanceCollector*") were already performed; this is made visible by the different colors used. In the case of "*Provide a URL*", the value has been changed to a specific Wikipedia article about Barcelona.

It is also possible to group sequences of tasks as shown in Fig. 12. For that, users must select two or more tasks, and then an option "Group tasks" on the contextual menu will appear. Similarly, if tasks are grouped, an option will allow ungrouping them. It is also possible to change the order of tasks in the list by selecting the options "*Move up*" or "*Move down*".

Once a procedure has been completed, users might share it with friends and colleagues. This task can be done by selecting the button "*Export*" as shown in Fig. 13. In edition mode (see Fig. 13a) the user can decide to: upload the procedure into the repository, send it via email to other users, or both. When the procedure is uploaded into the repository, it becomes available in a Web server for all users using the plugin. The option 'share with' is used when a procedure is intended to be shared with a few users. Fig. 13b shows the export window for the execution mode. In this mode, share the procedure means to allow other users to see with which information the user has executed the procedure. Since the user could have used confidential data for performing the tasks, he can choose from the list those tasks for which the attributes values must not be shared. This is made by selecting the tasks with private information from the list.

Finally, in Fig. 14 we show the main options to manage procedures. On the one hand, the main menu allows users to create a new procedure. On the other hand, by choosing "Manage procedures" option, the user can manage the procedures already installed in his browser. In Fig. 14, we show the only procedure installed, named "Conference trip planning". By selecting one of the procedures listed, the user can execute it, edit it, or see the previous executions made by himself and those executions shared by his partners.

4.4. Distributing UI components inside of procedure tool

In this section we show how both the approach and the tools support DUI by combining the use of some augmenters and some properties contemplated in the DSL. For that we propose two scenarios concerning (i) the inclusion of DOM elements into

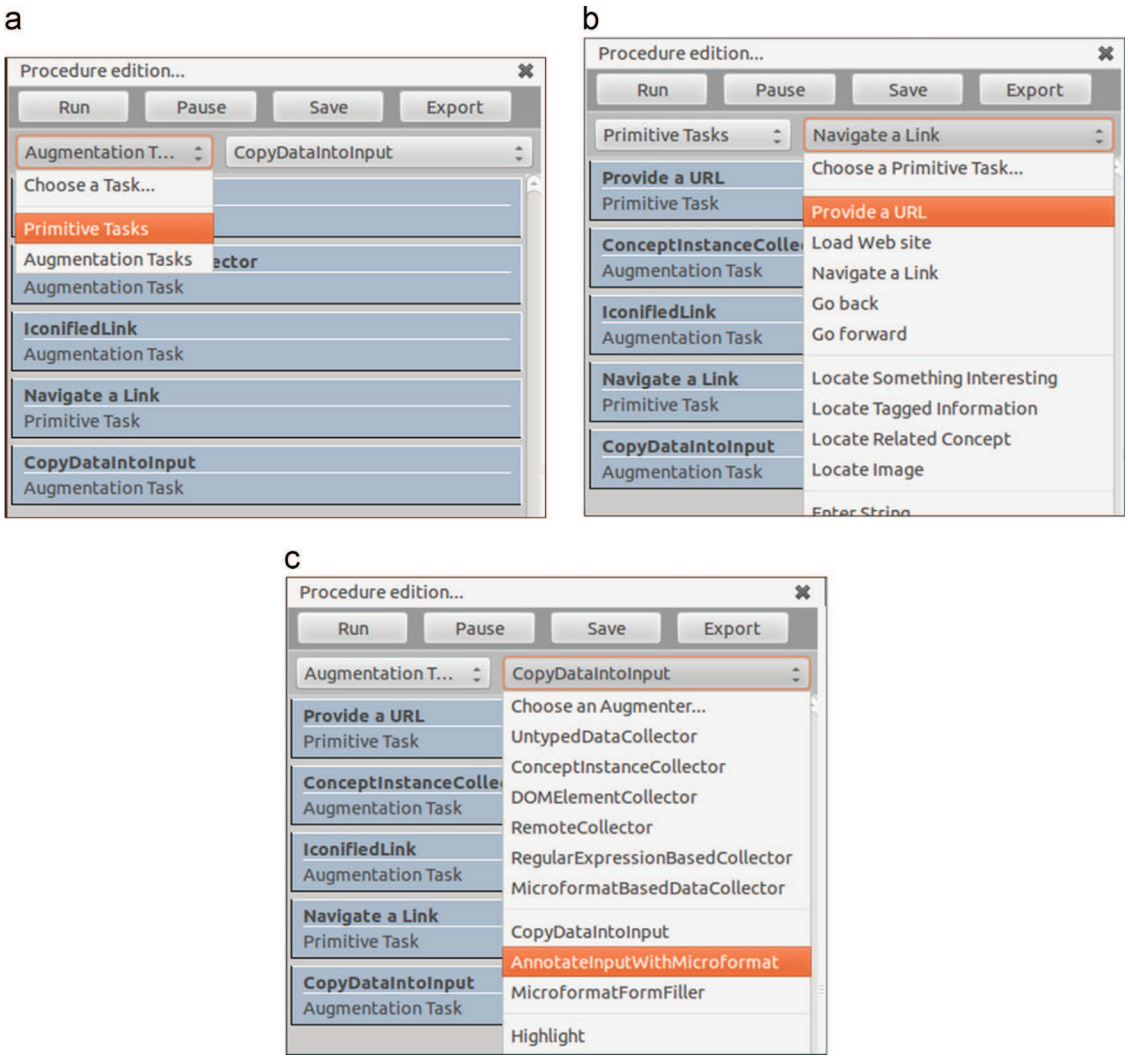

Fig. 10. Edition mode allowing task selection (either primitive tasks or augmenters). (a) Menu of tasks (primitive/augmenter). (b) Selection of primitive tasks. (c) Selection of an existing augmenter.

procedures and their corresponding interactors issued from the original Web site; and (ii) user collaboration via interaction with DUIs components embedded into procedures.

4.4.1. *Embedding DOM elements into procedures*

Users can use the procedure tool to collect graphical components of Web sites that are encoded as DOM elements and embed them into procedures. This operation is done from the window task edition as shown in Fig. 15. From there, users can select the option "*Select DOM Element*" and then explore the Web page to pick the corresponding DOM element. DOM elements are automatically detected by the tool and shown in highlight (in yellow) when the user moves the mouse over then. The selection of the DOM element is made by clicking on a highlighted area; this action makes the corresponding DOM elements embedded into the procedure tool as show by Fig. 15.

By collecting DOM elements, users are not only collecting information but they are also capturing the inner behavior necessary for interacting to the original Web site. Indeed, collected DOM elements might contain anchors, links and other interactors which users can use in the distribution of the user interface. Such as interactors becomes available when users share their procedures with other users. These elements are not only static copies DOM elements. To illustrate this, we provide in Fig. 16 a small scenario: let us assume a user wants to share for a while his Gmail mailbox with a friend without providing his login and password information. For that, the user creates a procedure embedding the form fields for logging into Gmail as illustrated by Fig. 16a. At first the user creates a task and associates the corresponding DOM element that contains all the form fields for performing a login into Gmail. The embedded DOM elements are shown in Fig. 16b. Then, the user types his login information on the procedure as shown in Fig. 16c. In the sequence, the user records the procedure and sends it his friend.

Fig. 17 illustrates the user's friend view of the procedure in the player mode. As we shall see, the procedure contains the DOM elements (i.e. the login) required to access a Gmail account in Spanish as this was the version of the Web site used for creating the procedure. Indeed, the first user created the procedure under an English version of Firefox running on Mac OS (Fig. 16) whilst his friend plays the procedure under an English version of Firefox running on Window 7 (Fig. 17). Assuming that the other user can read Spanish, the only thing he needs to do to access his friend's account is to click on the button "Acceder" (i.e. submit in Spanish) which is embedded into the procedure as part of the login form

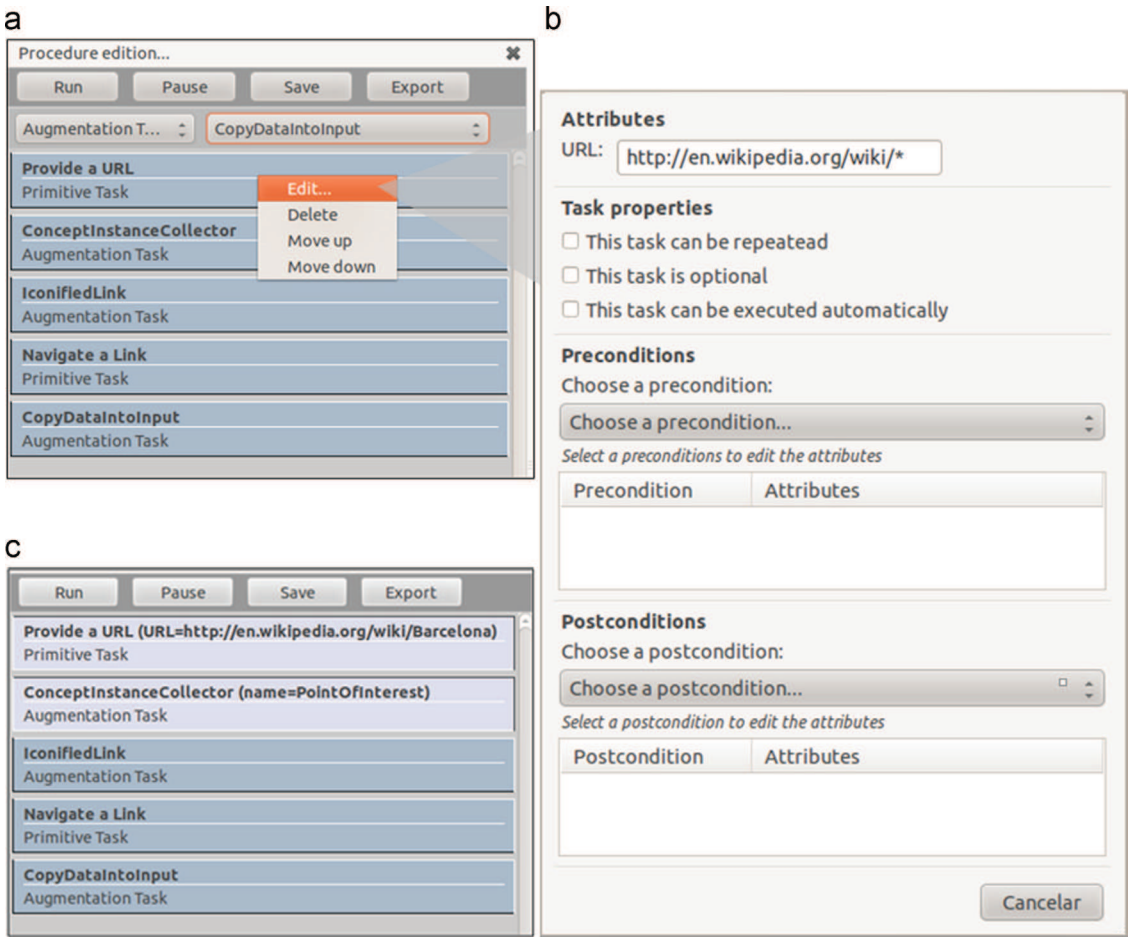

Fig. 11. Edition mode showing how to define properties of tasks (a and b) and execution mode (c). (a) Edition mode. (b) Edit task attributes. (c) List of tasks in the execution mode. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

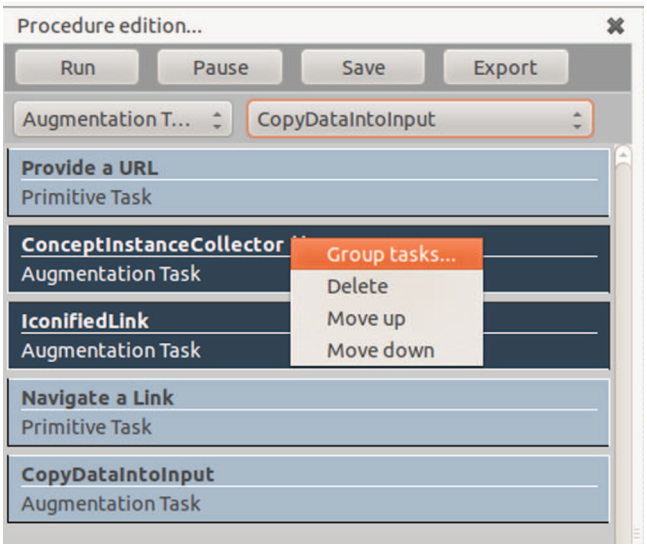

Fig. 12. Grouping tasks.

shown in Fig. 17; this action will grant the access to Gmail (as shown in the right side of Fig. 17) without knowing his friend's password. This scenario might look artificial at first sight but it illustrates how powerful our approach is with respect to the distribution of the user interface.

5. Case study

This section illustrates our approach by the means of a case study concerning a trip planning to attend a conference. Our main concern here is to demonstrate the creation and execution of complex procedures that combine both manual and automated tasks. For that, let us assume that two friends named Peter and John are planning to go together to a scientific conference. To accomplish this goal they should visit the conference Web site for getting general information (such as location, dates, etc.), collect points of interest at the destination, book flights and hotels. The first step is to define a sequence of minimal actions that they should do to accomplish this goal. Then, they can share informations during the process. Both Peter and John have previously installed the plugin as described in Section 4.3.

Peter decided to create a procedure as it is shown at the left-side of the browser window in Fig. 18. The procedure is built incrementally while Peter is navigating the different Web sites required to accomplish the task. As he collects data during his navigation, it becomes recorded as part of the procedure itself. For example, the first tasks in the list is "*Provide a URL*" where we can read the parameter "URL=http://www.interact2013.org" indicating the Web site visited by the user. We can also read that "*Provide a URL*" is a primitive task, which means that users should inform themselves which Web site they want to make part of the procedure. Because users might have many browser windows opened at the same time to perform different tasks, it is important to let users inform whether or not a specific Web site been visualized should be part of the procedure he has in mind. That

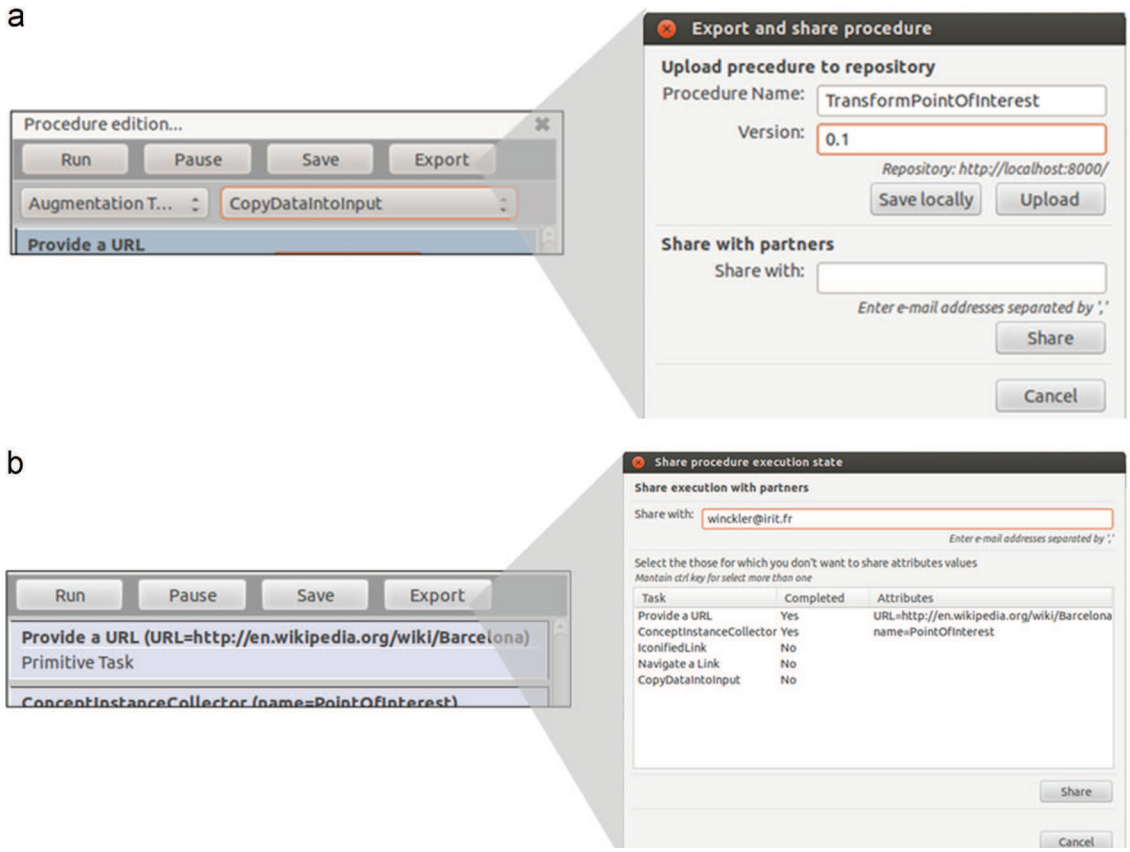

Fig. 13. Using functions *upload* and *sharing* in the tool. (a) Edition mode. (b) Execution mode.

Fig. 14. Main menu and procedure manage window.

is why this task is set as *primitive* and therefore performed manually by the user.

As we shall see in Fig. 18, the second, third and fourth elements in the list (lines 2–4) concern the augmenter *ConceptInstanceCollector* that was used three times to collect data from the Web site; respectively the *destination*, *dateFrom* and *dateTo*. For that, the only thing Peter had to do is to point at information on the Web page and trigger the augmenter, so that a copy operation is done. The data became part of the procedure (see the parameters *name* and *value* next to the augmenter definition). Moreover, these data become available when Peter is visiting the Web site for booking his flight at expedia.com (line nine). As that data is recorded as part of the procedure Peter can reuse it when filling in the form for booking the flight; moreover, Peter can configure the augmenter *CopyDataIntoInput* to automatically fill in the form with the appropriate data as described in line 10 of Fig. 18. The data used for a procedure remain available during the entire execution. Fig. 19 shows the manual tasks (i.e. primitive tasks) that are defined to indicate what users have to do for the payment. We assume that Peter has represented these tasks to provide contextual help for accomplish the whole tasks. Indeed, in some case manual tasks do not need to be explicitly represented.

The whole procedure created by Peter is detailed in Table 2. Notice that tasks have been grouped and that for the sake of clarity the Web sites where tasks are performed have been included. The numbers in the first column indicate the order of execution of tasks. Preconditions and post-conditions have not been specified for the case study (Figs. 20 and 21).

So far we have shown how Peter has created the planning trip procedure. But then, he decides to share it with his friend John. For

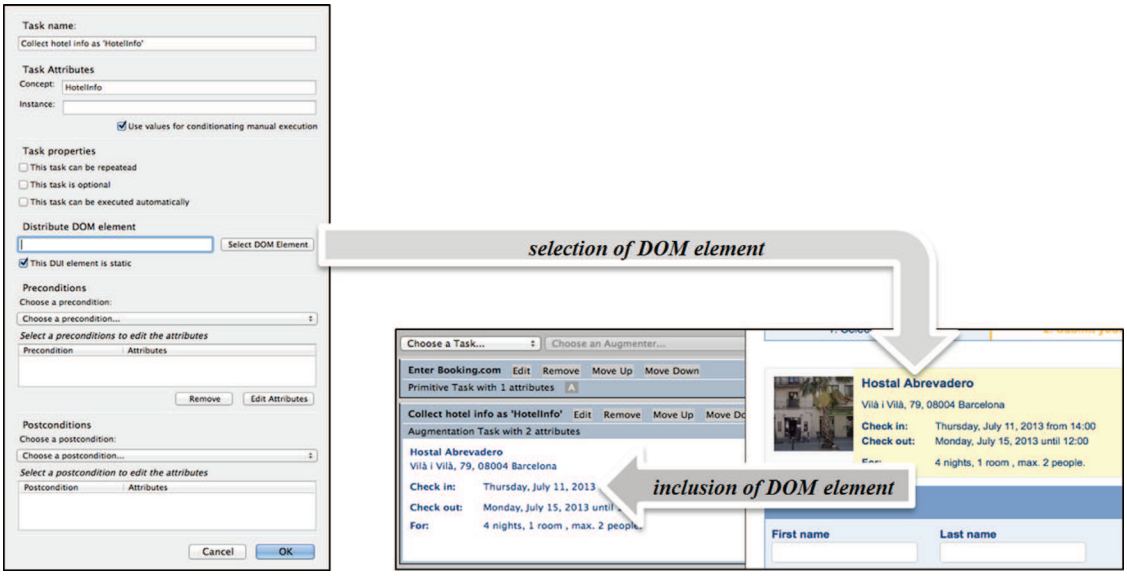

Fig. 15. DOM selection from Web site and later integration inside of procedure tool. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

that Peter triggers the option "export" in the main menu, execution mode (see Fig. 13b) and then a dialog menu appears with the list of tasks in that procedure. It is noteworthy that the procedure contains all data used by Peter, including his credit card information that he does not want to share with John (Fig. 20). So that Peter cleans the values associated to these tasks whilst sending the procedure to John.

John will be notified by email that a new procedure is made available by his friend John. By clicking in the URL provided in the email he can download that procedure that will appear in his browser as shown in Fig. 21. Notice that John sees the procedure in the execution mode of the tool. Most of the tasks contain attributes with data previously provided by Peter, which makes John's tasks easier to perform, except by data associated to tasks that Peter decided do not share with John.

When John finishes his procedure he can share this information with Peter. The procedure for sharing information is similar to that of sharing a procedure. The data is sent to the task repository that will notify Peter that John has being executing the procedure. By selecting the combo box as shown in Fig. 22, Peter can see the values he has used (Fig. 22a) and those provided by his friend John (Fig. 22b).

Note that, for example, the execution perform by John is not finished (note the two visible tasks in the list, which appear as not completed). Besides that, comparing both executions, we can see that John has used different information in the tasks; for instance, he has collected different *Point of Interest*.

6. Evaluation of the tools

The tools presented above are fully operational and they can be used to coordinate distributed user interactions over the Web. In order to highlight the contributions of these tools, we present in this section a preliminary evaluation.

The evaluation concerns the procedure player. We assume that user performance could be an important factor for adopting the tools proposed in this paper. To investigate this, we have performed a user testing experiment that was aimed at investigating whether users perceive our tool as usable and whether they were able to perform tasks faster using our tools than navigating individual Web sites. The user testing basically consisted of observing users' performance and comments whilst users accomplish tasks (Han and Tokuda, 2008). The study was run in a controlled room equipped with a PC running Unix and a Web browser Firefox 19 which integrates the procedure tool and a full set of augmenters. A chronometer was used to record user performance but sessions were not video-taped.

At first participants were asked to fill in a pre-questionnaire in order to collect demographic data and their experience with Web applications. Then, we have explained to the participants the goals of the study and they were asked to think aloud during the testing session. Participants were also introduced to our tools and get a short explanation about how it works. In order to make sure that all participants would receive the same information the instructions were provided by a 6 min video demonstrating the use of the procedure player.

Then we have asked participants to envisage a professional travel to the conference INTERACT with a colleague, named John, who is keen to share his trip plan with him. The travelers had to book a hotel in the city where the conference takes place. We assume that John had found the cheapest hotel offers at the Web site booking.com and that he was keen to share this hint. However, the participants had to make the arrangements for the trip alone. This general scenario encompasses the main tasks:

- Task 1 (T1): To get general information about the conference and the cheaper hotel.
- Task 2 (T2): To search at booking.com for a room in the hotel during conferences dates (check-in: first day of the conference; check-out: last day of the conference).
- Task 3 (T3): To book and pay for the room.

In order to compare the value added by our approach, we have asked participants to perform this trip planning with and without our tools. Whilst not using our tools, the trip planning featured a printed version of an email that contained all John's trip arrangements; then the participant had to visit the Web sites indicated in the email to perform the tasks. Whilst using our tools, the scenario included the trip planning made by John using the procedure editor; in this case John's email contained a link from where it was possible to download the procedure from the task repository. Thus, to execute the procedure the participant should have to perform the following additional task:

- Task 0: To import the procedure.

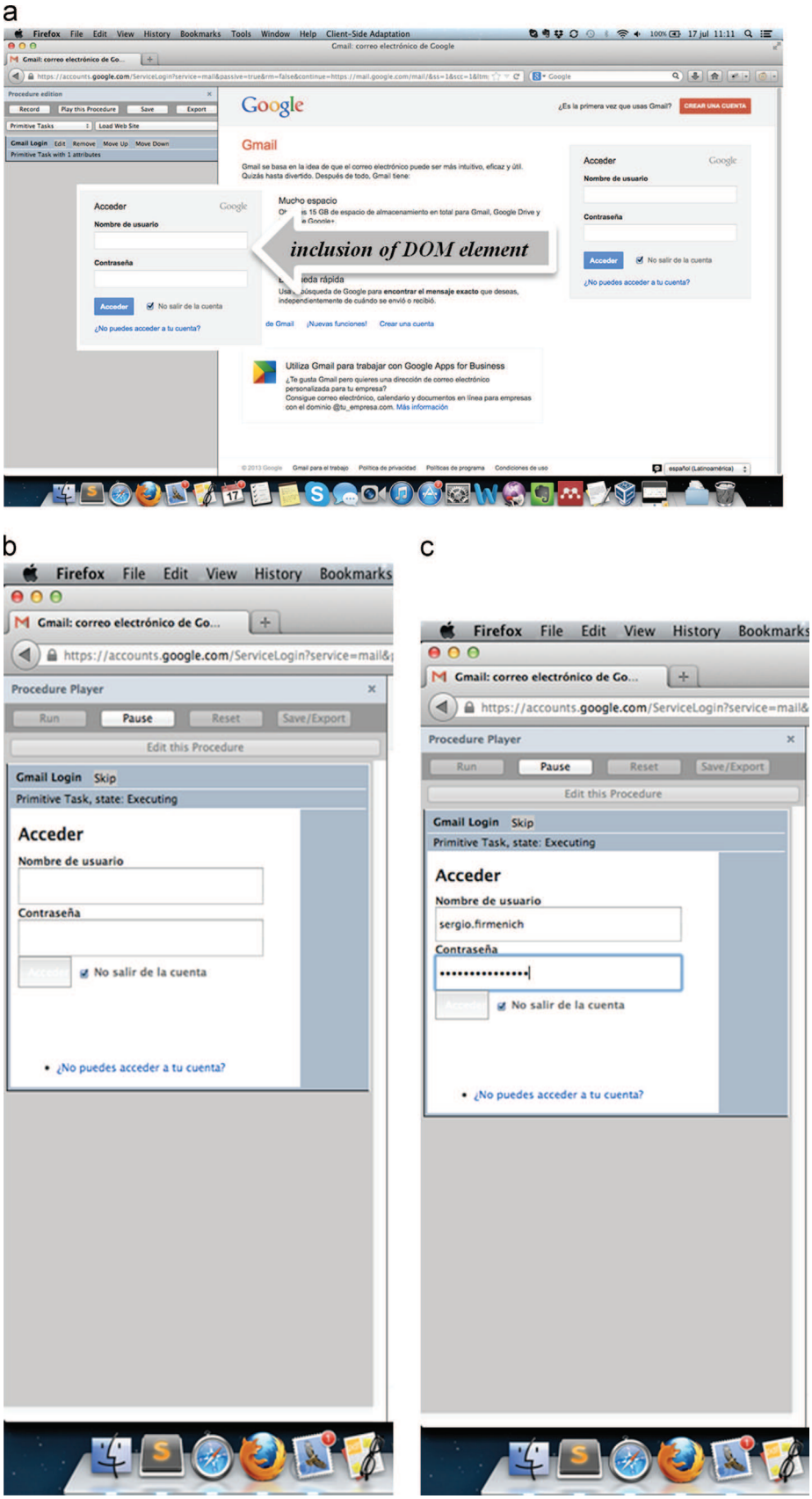

Fig. 16. Creation of a procedure embedding DOM element login using the procedure plugin in English version of Firefox for Mac OS, with Gmail in Spanish. (a) Inclusion of DOM element: login form. (b) Embedded DOM. (c) Customizing DOM.

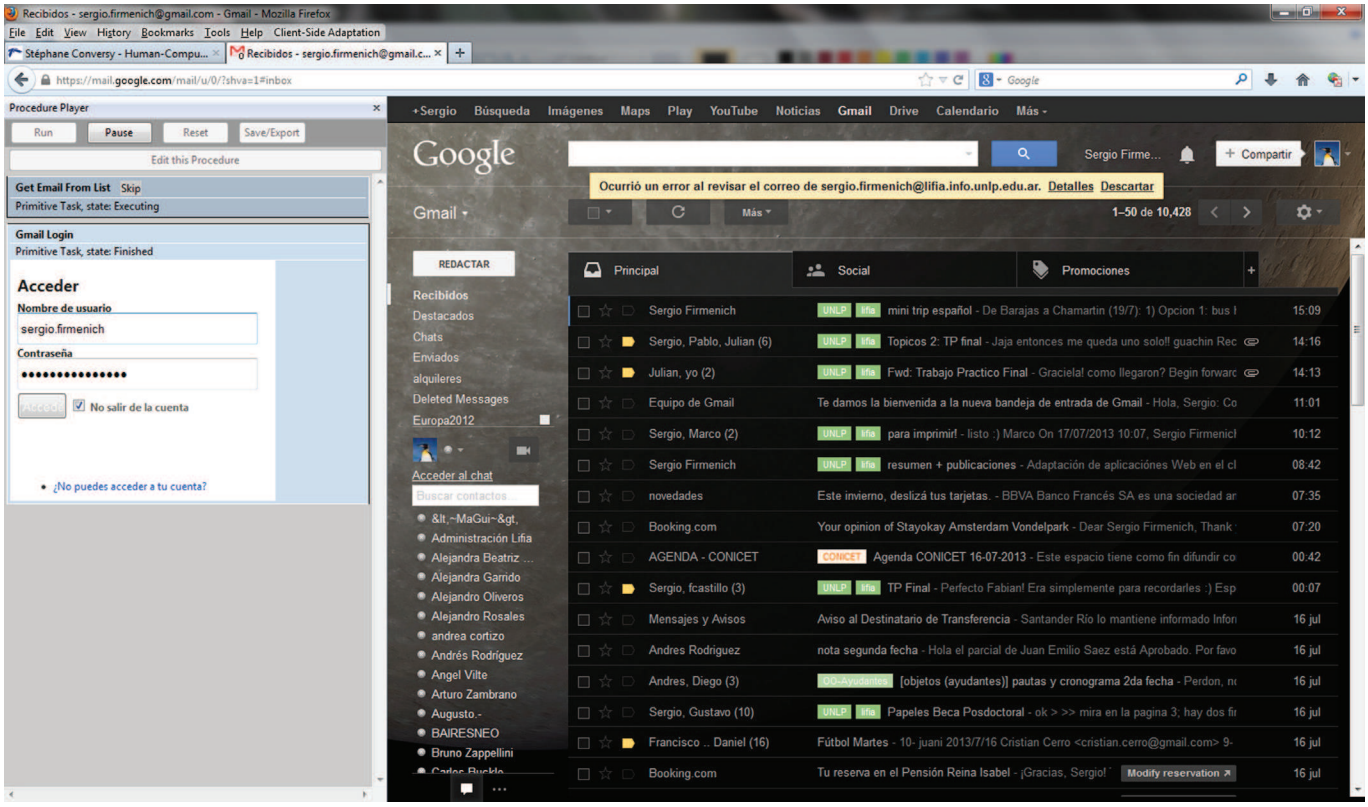

Fig. 17. Running the procedure plugin under an English version of Firefox for Windows 7.

Fig. 18. A procedure for planning a trip as it is visualized when navigating the Web. At the left-side, the plugin with the list of tasks (including both primitive and augmenters). At the right-site the current Web site.

After performing every task, participants were asked to assess from 1 to 5 the difficulty to perform the task (1 from very easy to 5 very difficult). In order to measure efficiency we took the time consumed by each user task. Effectiveness was measured in terms of the number of tasks users could finish.

At first they were asked to perform the tasks without the tools. Then they were allowed to perform the tasks using the procedure player. After accomplishing the tasks with and without the procedure player, participants were asked to fill in a form used to determine the satisfaction degree, including: what they liked in the tools, what they did not like, whether they experienced difficulties in activating the tools, which task were difficult to perform, whether (or not) they were aware of the actions using the tools. Finally, a System Usability Scale questionnaire (i.e. SUS (Brooke, 1996)) was used to determine tools usability.

For this study about the usability of our tools, eleven participants (9 males and 3 females, aged from 20 to 33 years old,

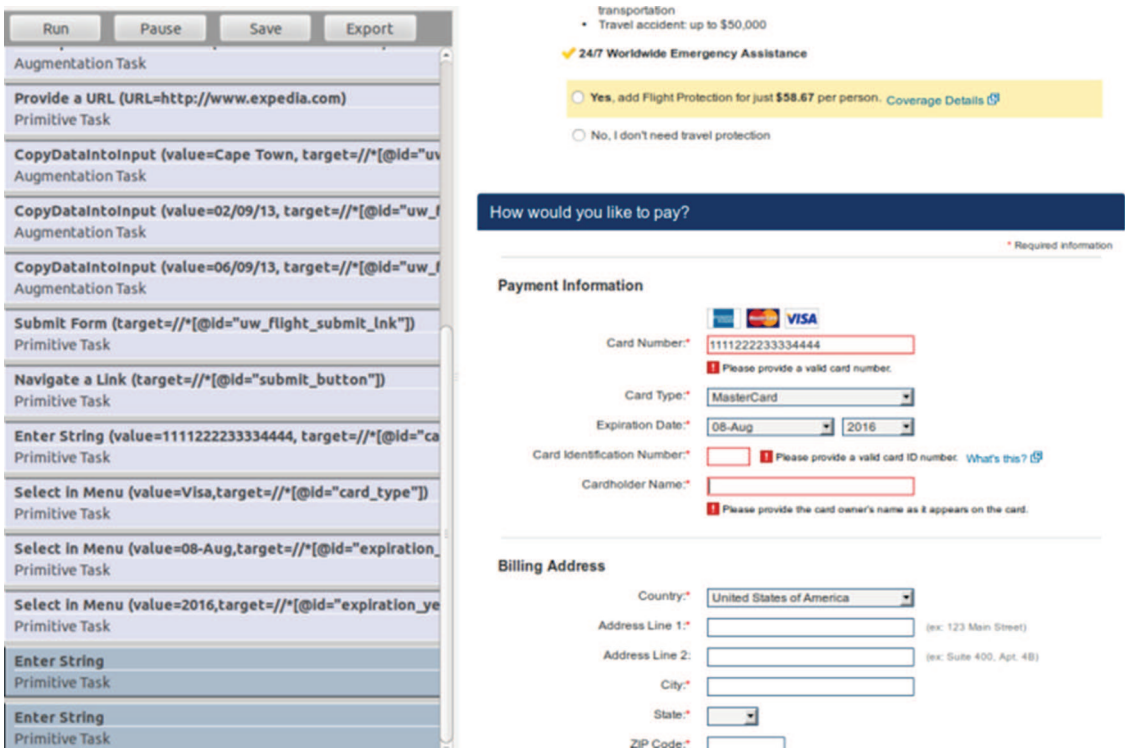

Fig. 19. Tasks related to payment in a procedure.

Table 2
Peter's procedure for planning a trip to attend a conference.

User tasks	Web site	Task (*primitive/augmenter*)	Attributes
1. Visit conference Web site	–	Primitive task: *provideURL*	url
2. Collect data about the conference	www.interact2013.org	*ConceptInstanceCollector*	conference place
		ConceptInstanceCollector	destination
		ConceptInstanceCollector	dateFrom
		ConceptInstanceCollector	dateTo
3. Visit site for getting information about destination	–	Primitive task: *provideURL*	url
4. Collect points of interest at destination	www.interact2013.org	∗*ConceptInstanceCollector*	point of interest
5. Visit site for booking a flight	–	Primitive task: *provideURL*	url
6. Search for flights	www.expedia.com	*CopyDataIntoInput*	destination
		CopyDataIntoInput	dateFrom
		CopyDataIntoInput	dateTo
		Primitive task: *submitForm*	
7. Go to payment page	www.expedia.com	Primitive task: *clickButton*	
8. Pay for flights booking	www.expedia.com	Primitive task: *enterString*	card number
		Primitive task: *selectMenu*	card type
		Primitive task: *selectMenu* Primitive task: *selectMenu*	expiration month
		Primitive task: *enterString*	expiration year
		Primitive task: *submitForm*	card holder
9. Collect data about flight booking	www.expedia.com	*ConceptInstanceCollector*	flights number
		ConceptInstanceCollector	flight set
10. Visit site for booking hotel	–	Primitive task: *provideURL*	url
11. Search hotels	www.booking.com	*CopyDataIntoInput*	destination
		CopyDataIntoInput	dateFrom
		CopyDataIntoInput	dateTo
		Primitive task: *submitForm*	
12. Pay for hotel booking	www.booking.com	Primitive task: *enterString*	card number
		Primitive task: *selectMenu*	card type
		Primitive task: *selectMenu* Primitive task: *selectMenu*	expiration month
		Primitive task: *enterString*	expiration year
		Primitive task: *submitForm*	card holder
13. Collect data about hotel		*ConceptInstanceCollector*	link of hotel's Web page

Legend: –: task can be performed from any Web site; ∗: task can be performed one or more times.

Average = 27.64, SD = 3.79) were recruited at the University of La Plata, Argentina. Among the participants we counted two PhD, 7 PhD students; the rest of them were undergraduate students. All participants were experienced Web users (i.e. > 5 years using the Web) who browse the Web as a part of their daily activities. The pool of users was selected by convenience and it is not representative of the population of Web users. However, each user performed the manual task and used the

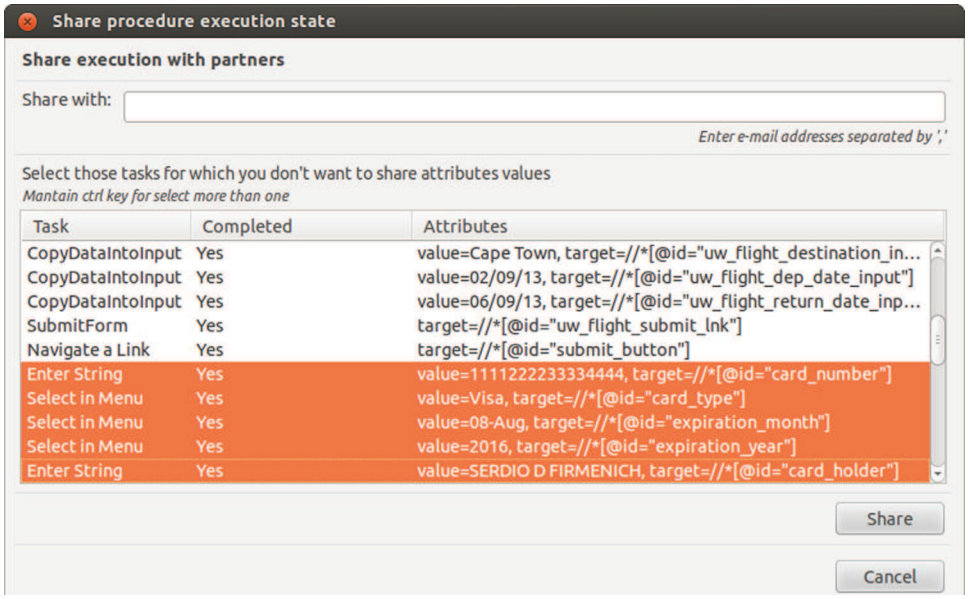

Fig. 20. Sharing a procedure with other users.

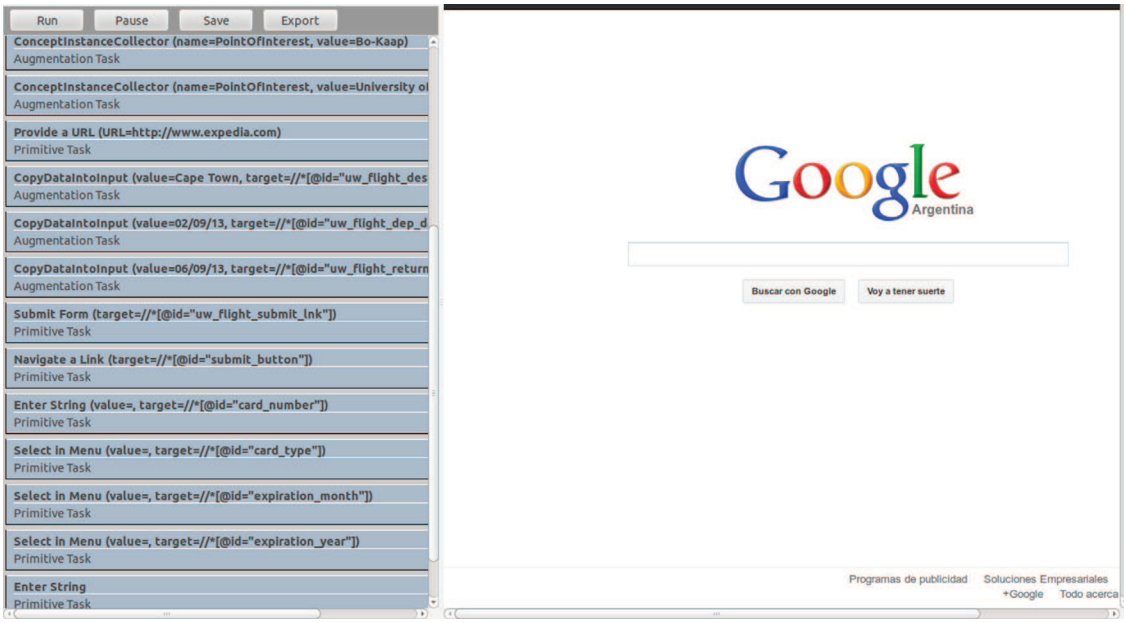

Fig. 21. Visualization of the procedure in the execution mode.

procedure player, so it is still possible to compare the relative user performance.

All participants in the study completed both manual tasks and used the procedure player. Overall the users judged the tasks with the procedure player easy to follow. In a scale from 1 to 5 (1 is very easy, 5 very difficult), the average score for tasks were: Task 0=2.6, Task 1=1.4, Task 2=1.6, and Task 3=1.18. As indicate in Fig. 23 the task performance (except of the task 0) was improved using our tools.

All participants reported positive comments about the tools. The overall the results provided by SUS were also positive with an average SUS score of 75.2. However, three participants given score below 70 points to the SUS (i.e. 50, 55 and 67.5) and a single user gave 100 points to the procedure paper. Moreover, participants provided several suggestions for improving the usability of the tool such as providing explicitly mark for informing when the set of tasks has been finished. In general participants think that the tools help to navigate among the diverse site Web and they appreciate the automation of tasks. Despite the fact that the user input was distributed between the procedure player and the Web sites, none of the participants raised comments about the fact that these two ways of interacting with Web sites would be conflicting. However, some of them (N=5) mentioned they would be appreciate it if the tools could automate the tasks without asking users to follow the list.

These results are rather preliminary. They do not take into account a detailed analysis of the synchronization of tasks between the users. Moreover, it only covers the execution of procedures. Yet it shows some evidence of use for the tools and an initial positive feedback. Indeed, the study shows that users can very easily perform collaborative tasks in distributed Web sites. Users indeed appreciated the support for planning tasks in an integrated manner. User performance was greatly improved using the approach. However, further studies will be required to investigate the usability of the tool with a larger population.

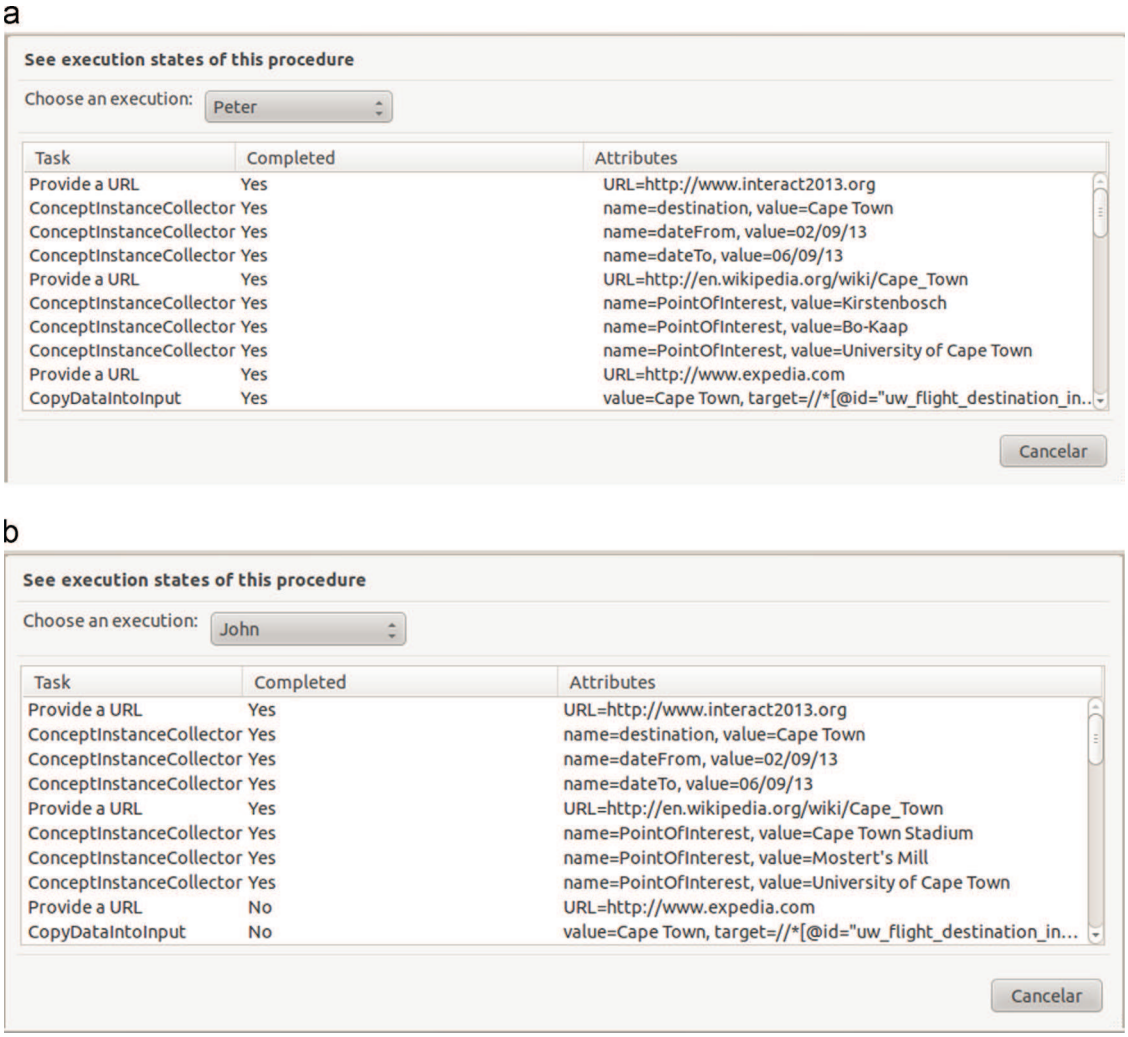

Fig. 22. Visualization of values used in procedure. By selecting the user in combo box, Peter can compare his values with the values used John. (a) Values used by Peter to accomplish the procedure. (b) Values used by John to accomplish the procedure.

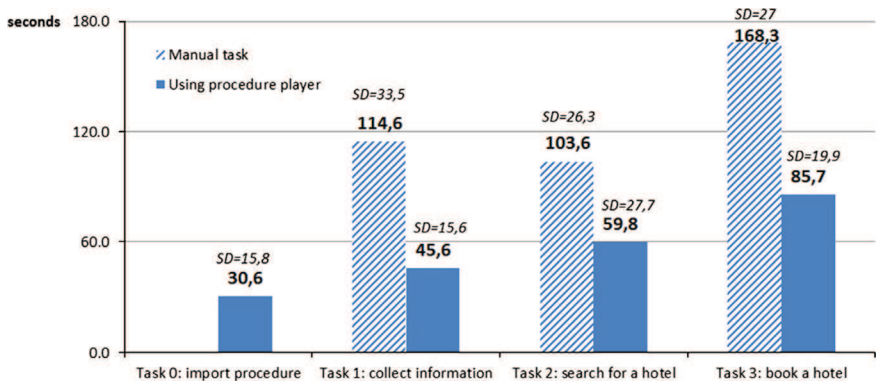

Fig. 23. Average user performance for accomplishing tasks manually and with the procedure player.

7. Conclusions, lessons learned and future work

In this paper we have presented a novel approach for building Distributed User Interfaces (DUIs) aimed at helping users to collaborate whilst navigating multiple Web sites. These DUIs feature a list of tasks that are constructed by assembling building blocks from a repertoire of manual Web tasks (called primitive tasks here) and a set of components called Web augmenters that automate user tasks. By assembling these tasks, users can create their own procedure. The composition mechanism can be described by a dedicated DSL and supported by a tool featuring a plugin. The distributed aspect of this interface is duly discussed in Section 7.

The key aspect of this contribution is the fact that our approach is able to include small pieces of software components, called Web augmenters to support tasks users performing over the Web. By assembling these elements in a list, we argue that we can create a dedicated procedure featuring a distribute user interface that helps users to accomplish a given goal. It is important to notice that, to reach some goals, users still have to act on the Web browser. For that, the approach allows the inclusion of primitive tasks which are used in the composition to provide a kind of contextual help, so that users will know what to do whilst navigating a particular Web site. The DSL that is delivered as part of our approach allows formalizing the task composition. It is noteworthy that this is only possible because the repertoire of tasks users can perform are limited by the number of actions users can do on a Web browser. The elementary user tasks of a Web browser are simple and very well known in the literature (Byrne et al., 1999; Heath, 2010). Web augmenters can automate some of these tasks and perform adaptations on Web pages to be visualized in the Web browser. These new components let users perform additional tasks that go beyond of traditional Web users. Nonetheless, these are extensions that should be coded by developers. The impact of the developer community in developing scripts is known in the GreaseMonkey community (GreaseMonkey, 2012) and motivates this kind of research.

Some of the aspects of procedure modeling and execution discussed in this paper might resemble some techniques for end-user programming (Lieberman et al., 2006; Lin et al., 2009). In the past, experiments with mashups tools for end-user programming (Lin et al., 2009) have demonstrated that some visual tools such as CoScript (Bogart et al., 2008) could be used to create scripts integrating data from many Web sites. Nonetheless, we consider that a direct comparison would be unfair as the skill required to compose procedures with Web augmenters is more close to what we expect from Web developers than from ordinary Web users.

The current implementation is limited to the information exchange between users and the tasks they have performed. In the section case study, we have shown how two users can report to each other the values they have used to accomplish a trip planning. However, it would be possible to envisage a more fine-grained collaboration on which users could distribute the tasks to be performed in a procedure. We are currently working on our tool to increase the level of interaction between tasks performed by users. One of the aspects that we are investigating is to merge procedures developed by two (or more) users, where each procedure contains tasks that are performed by individuals in order to reach a high level goal that is represented by the combination of several procedures.

Given the fact that this approach is very new, the tools have only been used by a limited number of users that were specially recruited to an early trial. Nonetheless they seem enough to prove the overall concept. Future work will include user testing of the tools with a larger population. Such empirical study should investigate in detail the usability of the tool in creation mode and the synchronization aspects of tasks between users. Moreover, we are planning to investigate how users perceive the introduction of new Web augmenters into the plugin.

In parallel we are working on the development of new augmenters that could be supporting a higher level of automation of user tasks. In the near future we are planning to make the tools accessible for the public so that we can start investigating the usability and the user experience with our tools. We are also planning to integrate more function in our tools for supporting a better communication between the users during the execution of distributed procedure. In a long term run we want to investigate the use of procedure descriptions as a support to the analysis of the user activity over multiple Web sites; by doing so we hope that we could optimize the processes used by users to accomplish their tasks on multiple Web sites.

Rather than a definitive solution to the problem of integrating data among multiple Web sites and supporting users collaboration whilst navigating Web application, this work proposes new challenges for the development of Distributed User Interfaces over the Web. For example, how to model and describe user tasks that can be scattered in multiple Web sites? How to help users perform these tasks efficiently? How to automate user tasks over the Web? How to build interfaces that help users to share information about their activity over the Web with their friends and colleagues? How many pieces of the user interface of Web sites can be extracted and then rearranged to feature a new sequence of tasks? How to deal with Web technology to provide better support to users' tasks on multiple Web sites? Certainly much work remains to be done but the results that we obtained with our approach are promising for investigating these questions.

Annex. XSD Specification of the DSL

```
< ?xml version="1.0"003F >
< xs:schema xmlns:xs="http://www.w3.org/2001/
 XMLSchema"
    ttargetNamespace="http://www.lifia.info.unlp.edu.ar"
    xmlns="http://www.lifia.info.unlp.edu.ar" >
  < xs:element name="procedure" >
    < xs:complexType >
    < xs:sequence minOccurs="1" maxOccurs="1" >
      < !- SYNCHRONIZATION!- >
        < xs:element name="configuration" >
        < xs:complexType >
          < xs:element name="server" type="xs:string"/ >
          < xs:element name="executionId" type="xs:
string"/ >
          < /xs:complexType >
        < /xs:element >
        < !- TASK DEFINITION!- >
        < xs:element name="tasks" >
        < xs:complexType >
          < xs:sequence minOccurs="1"
maxOccurs="unbounded" >
              < xs:element name="tasks" >
              < xs:complexType >
                < xs:all minOccurs="1" >
                  < xs:element name="primitiveTask"
minOccurs="0" maxOccurs="*"/ >
                    < xs:complexType >
                      < xs:group ref="taskDefinition"/ >
                      < /xs:complexType >
                  < /xs:element >
                  < xs:element ref="augmentationtask"
minOccurs="0" maxOccurs="*"/ >
                    < xs:complexType >
                      < xs:group ref="taskDefinition"/ >
                      < /xs:complexType >
                  < /xs:element >
                  < /xs:all >
                < /xs:complexType >
              < /xs:element >
            < /xs:sequence >
          < /xs:complexType >
        < /xs:element >
      < /xs:sequence >
    < /xs:complexType >
  < /xs:element >
```

```
< xs:group name = “taskDefinition” >
  < xs:sequence >
    < !– PROPERTIES!– >
    < xs:attribute name = “id” type = “xs:string”/ >
    < xs:attribute name = “repetitive” type = “xs:string”/ >
    < xs:attribute name = “optional” type = “xs:string”/ >
    < xs:attribute name = “automatic” type = “xs:string”/ >
    < xs:attribute name = “synchronize” type = “xs:string”/ >

    < !– ATTRIBUTES!– >
    < xs:element name = “attributes” minOccurs = “0”/ >
      < xs:complexType >
        < xs:all minOccurs = “0” maxOccurs = “*” >
          < xs:element name = “attribute” >
            < xs:complexType >
              < xs:attribute name = “id” type = “xs:string”/ >
              < xs:sequence >
                < xs:element name = “name” type = “xs:
string”/ >
                < xs:element name = “type” type = “xs:
string”/ >
                < xs:element name = “value” type = “xs:
string”/ >
                < xs:element name = “valueExample”
type = “xs:string”/ >
              < /xs:sequence >
            < /xs:complexType >
          < /xs:element >
        < /xs:all >
      < /xs:complexType >
    < /xs:element >
    < !– PRECONDITIONS!– >
    < xs:element name = “preconditions” minOccurs = “0”/ >
      < xs:complexType >
        < xs:all minOccurs = “0” maxOccurs = “*” >
          < xs:element name = “precondition” >
            < xs:complexType >
              < xs:element name = “type” type = “xs:
string” >
                < xs:simpleType >
                  < xs:restriction base = “xs:string” >
                    < xs:enumeration
value = “LastUsedWebApplicationIs”/ >
                    < xs:enumeration
value = “WebApplicationInUse”/ >
                    < xs:enumeration
value = “WebApplicationUsed”/ >
                    < xs:enumeration
value = “PocketHasInstanceOf”/ >
                    < xs:enumeration
value = “PocketIsNotEmpty”/ >
                  < /xs:restriction >
                < /xs:simpleType >
              < /xs:element >
              < xs:sequence minOccurs = “1”
maxOccurs = “*” >
                < xs:element name = “name” type = “xs:
string”/ >
                < xs:element name = “type” type = “xs:
string”/ >
                < xs:element name = “value” type = “xs:
string”/ >
              < /xs:sequence >
            < /xs:complexType >
          < /xs:element >
        < /xs:all >
      < /xs:complexType >
    < /xs:element >
    < !– POSTCONDITIONS!– >
    < xs:element name = “postconditions” minOccurs = “0”/ >
      < xs:complexType >
        < xs:all minOccurs = “0” maxOccurs = “*” >
          < xs:element name = “postcondition” >
            < xs:complexType >
              < xs:element name = “type” type = “xs:
string” >
                < xs:simpleType >
                  < xs:restriction base = “xs:string” >
                    < xs:enumeration
value = “AffectCurrent”/ >
                    < xs:enumeration value = “AffectAny”/ >
                    < xs:enumeration
value = “AffectSubset”/ >
                    < xs:enumeration value = “AffectAll”/ >
                  < /xs:restriction >
                < /xs:simpleType >
              < /xs:element >
              < xs:sequence minOccurs = “1”
maxOccurs = “*“ >
                < xs:element name = “name” type = “xs:
string”/ >
                < xs:element name = “type” type = “xs:
string”/ >
                < xs:element name = “value” type = “xs:
string”/ >
              < /xs:sequence >
            < /xs:complexType >
          < /xs:element >
        < /xs:all >
      < /xs:complexType >
    < /xs:element >
  < /xs:sequence >
< /xs:group >
< /xs:schema >
```

References

Amershi, S., Morris, M.R., 2008. CoSearch: a system for co-located collaborative web search. In: Proceedings of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems (CHI '08). ACM, New York, NY, USA, pp. 1647–1656. doi:10.1145/1357054.1357311. ⟨http://doi.acm.org/10.1145/1357054.1357311⟩.

Bogart, C., Burnett, M., Cypher, A., Scaffidi, C., 2008. End-user programming in the wild: a field study of CoScripter scripts. In: Proceeding of EEE Symposium on Visual Languages and Human-Centric Computing. Germany, pp. 39–46.

Bolin Michael, Webber Matthew, Rha Philip, Wilson Tom, C. Miller Robert, 2005. Automation and customization of rendered web pages. In: Proceedings of the Eighteenth annual ACM Symposium on User Interface Software and Technology (UIST '05). ACM, New York, NY, USA, pp. 163–172. http://dx.doi.org/10.1145/1095034.1095062.

Bouvin, N.O., 1999. Unifying strategies for web augmentation. In: Proceedings of the 10th ACM Conference on Hypertext and Hypermedia.

Brooke, J., 1996. SUS: A 'Quick and Dirty' Usability Scale. In: Usability Evaluation in Industry. Taylor and Francis, London.

Brusilovsky, P., 2007. Adaptive navigation support. The adaptive web: Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl (Eds.). Lecture Notes In Computer Science,vol. 4321, Springer-Verlag, Berlin, Heidelberg, pp. 263–290.

Brusilovsky P., Kobsa A., Nejdl W., 2007. The Adaptive Web: Methods and Strategies of Web Personalization, vol. 4321, Springer-Verlag LNCS, Berlin, Heidelberg.

Byrne, M.D., John, B., Wehrle, N., Crow, D., 1999. The tangled Web we wove: a taskonomy of WWW use. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: the CHI is the Limit (CHI '99). ACM, New York, NY, USA, pp. 544–551. doi:10.1145/302979.303154. ⟨http://doi.acm.org/10.1145/302979.303154⟩.

Card, S., Moran, T., Newell, A., 1983. The Psychology of Human–Computer Interaction. Lawrence Erlbaum Associates, Hillsdale, NJ. (448 pp).

Daniel, F., Casati, F., Soi, S., Fox, J., Zancarli, D., Shan, M., 2009. Hosted universal integration on the web: the mashArt platform. In: Proceedings of ICSOC/ ServiceWave. Stockholm, pp. 647–648.
Demeure, A., Sottet, J.S., Calvary, G., Coutaz, J., Ganneau, V., Vanderdonckt, J., 2008. The 4C reference model for distributed user interfaces. In: Proceedings of the International Conference on Autonomic and Autonomous Systems, IEEE Explore, Piscataway, pp. 61–69.
Diaper, D., Stanton, N.A. (Eds.), 2004. The Handbook of Task Analysis for Human–Computer Interaction. Lawrence Erlbaum Associates, Mahwah, New Jersey, USA.
Diaz, O., Arellano, C., Iturrioz, J., 2010. Interfaces for scripting: making greasemonkey scripts resilient to website upgrades. In: Proceeding of ICWE2010. Springer, Vienna, pp. 233–247.
Elmqvist, N., 2011. Distributed user interfaces: state of the art. In: Gallud, J.A., et al. (Eds.), Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human–Computer Interaction Series. Springer-Verlag, Berlin, Heidelberg, pp. 2011–2012.
Firmenich, S., Rossi, G., Urbieta, M., Gordillo, S., Challiol, C., Nanard, J., Nanard, M., Araujo, J., 2010. Engineering concern-sensitive navigation structures. Concepts, tools and examples. Journal of Web Engineering 9 (2), 157–185.
Firmenich, S., Winckler, M., Rossi, G., Gordillo, S., 2011. A crowdsourced approach for concern-sensitive integration of information across the web. Journal of Web Engineering (JWE) 10 (4), 289–315.
Gallud, J.A., Tesoriero, R., Penichet, V.R.M. (Eds.), 2011. Distributed User Interfaces Designing Interfaces for the Distributed Ecosystem. Human-Computer Interaction Series. Springer, Berlin, Heidelberg.
GreaseMonkey, 2012. ⟨http://www.greasespot.net/⟩ (last accessed 13.07.12).
Han, H., Tokuda, T., 2008. A method for integration of web applications based on information extraction. In: Proceedings of ICWE. Springer, New York, pp. 189–195.
Laurillau, Y., Nigay, L., 2002. Clover architecture for groupware. In: Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work (CSCW '02). ACM, New York, NY, USA, pp. 236–245.
Little, G., Lau, T., Cypher, A., Lin, J., Haber, E., Kandogan, E., 2007. Koala: capture, share, automate, personalize business processes on the web. In: Proceedings of the Conference on Human Factors in Computing Systems, CHI. ACM, San Jose, California, April 2007, pp. 943–946.
Limbourg, Q., Pribeanu, C., Vanderdonckt, J., 2001. Towards uniformed task models in a model-based approach. In: Johnson, Chris (Ed.), Proccedings of the 8th Workshop on Interactive Systems: Design, Specification, and Verification (DSV-IS '01). Springer-Verlag, London, UK, pp. 164–182.
Luyten, K., Coninx, K., 2005. Distributed user interface elements to support smart interaction spaces. IEEE Symposium on Multimedia. Irvine, California, USA, December 12-14, 2005.
Lieberman, Paternò, Wulf, (Eds.), 2006. End User Development. Series: Human–Computer Interaction Series, vol. 9, XVI, Springer, 492 Berlin, Heidelberg.
Lin, J., Wong, J., Nichols, J., Cypher, A., Lau, T.A., 2009. End-user programming of mashups with vegemite. In: Proceedings of the 14th International Conference on Intelligent User Interfaces (IUI '09). ACM, New York, NY, USA, pp. 97–106.
Manca, M., Paterno, F., 2011. Distributed user interfaces with Maria. In: Gallud, J.A., Tesoriero, R., Penichet, V.R.M. (Eds.), Distributed User Interfaces Designing Interfaces for the Distributed Ecosystem. Human-Computer Interaction Series. Springer, Berlin, Heidelberg, pp. 33–40.
Martinie, C., Palanque, P., Winckler, M., 2011. Structuring and composition mechanisms to address scalability issues in task models. In: Proceedings of INTERACT (3). Springer LNCS, pp. 589–609.
Melchior, J., Vanderdonckt, J., Van Roy, P., 2011. A model-based approach for distributed user interfaces. In: Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '11). ACM, New York, NY, USA, pp. 11–20. http://dx.doi.org/10.1145/1996461.1996488.
Morris, M.R., Horvitz, E., 2007. Search together: an interface for collaborative web search. In: Proceedings of the UIST '07, October 7–10, 2007, ACM Press, Newport, Rhode Island, USA, New York, USA. pp. 3–12.
Morris, M.R., 2008. A survey of collaborative web search practices. In: Proceedings of the 26th SIGCHI Conference on Human Factors in Computing Systems (CHI '08). ACM Press, pp. 1657–1660. doi:10.1145/1357054.1357312. ⟨http://doi.acm.org/10.1145/1357054.1357312⟩.
MozillaUbiquity, 2012. At: ⟨http://mozillalabs.com/ubiquity/⟩ (last accessed 13.07.12).
Operator, 2012. ⟨https://addons.mozilla.org/en-US/firefox/addon/operator/⟩ (last accessed 13.07.12).
Paternò, F., Mancini, C., Meniconi, S., 1997. Concur task trees: a diagrammatic notation for specifying task models. In: Proceedings of Interact'97. Chapman & Hall, Sydney, Australia, pp. 362–369.
Paternò, F., Zini, E., 2004. Applying information visualization techniques to visual representations of task models. In: Proceedings of TAMODIA '04. ACM, New York, NY, USA, pp. 105–111.
Paul, S.A., Morris, M.R., 2009. CoSense: enhancing sensemaking for collaborative web search. In: Proceedings of the 27th International Conference on Human Factors in Computing Systems (CHI '09). ACM, New York, NY, USA, pp. 1771–1780. http://dx.doi.org/10.1145/1518701.1518974.
Pilgrim, M., 2005. Greasemonkey Hacks – Tips and Tools for Remixing the Web with Firefox. O'Reilly.
Rädle, R., Jetter, H.-C., Reiterer, H., 2012. TwisterSeacrh: a distributed user interfac fro collaborative Web search. In: Proceedings of the 2nd Workshop on Distrubuted User Interfaces (in conjunction with CHI 2012). Austin, Texas, USA, May 5th 2012.
Selenium, 2012. Available at: ⟨http://jroller.com/selenium/⟩ (last accessed 14.07.12).
Schmid, O., Masson, A.L., Hirsbrunner, B., 2012. Collaborative web browsing: multiple users, multiple pages, concurrent access, one display. In: Proceedings of the fourth ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '12). ACM Press, New York, USA, pp. 141–150.doi:10.1145/2305484.2305508. < ⟨http://doi.acm.org/10.1145/2305484.2305508 > .
Scriptish, 2012. ⟨http://scriptish.org/⟩ (last accessed 14.07.12).
Tom, Heath, 2010. A Taskonomy for the Semantic Web. Semant. web 1, 1,2 (April 2010), pp. 75–81.
Vanderdonckt, J., 2010. Distributed user interfaces: how to distribute user interface elements across users, platforms, and environments. In: Garrido, J.L., Paterno, F., Panach, J., Benghazi, K., Aquino, N. (Eds.)Proceedings of XIth Congreso Internacional de Interacción Persona-Ordenador Interacción 2010 (Valencia, 7–10 September 2010). AIPO, Valencia, 2010, pp. 3–14, Keynote address.
Wong, J., Hong, J.I., 2007. Making mashups with marmite: towards end-user programming for the web. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07); ACM, New York, NY, USA, pp. 1435–1444.
Yu, J., Benatallah, B., Casati, F., Daniel, F., 2008. Understanding Mashup Development. IEEE Internet Computing 12, 44–52.
Yahoo Pipes!, 2012. ⟨http://pipes.yahoo.com/⟩ (last accessed 13.07.12).