USATASKER: A TASK DEFINITION TOOL FOR SUPPORTING THE USABILITY EVALUATION OF WEB APPLICATIONS

Leandro Guarino de Vasconcelos[1] and Laércio Augusto Baldochi Jr.[2]
[1]*Instituto Nacional de Pesquisas Espaciais (INPE) - Av. dos Astronautas, 1758 – 12227-010 – São José dos Campos/SP Brazil*
[2]*Instituto de Ciências Exatas - Universidade Federal de Itajubá (UNIFEI) - Caixa Postal 50 – 37500-903 – Itajubá/MG Brazil*

ABSTRACT

Everyday tens of new applications are deployed on the Web. In general, most of these applications have rich and complex interfaces, which are bound to present usability problems. However, for modern Web applications, usually developed in "Internet time", regular approaches for the evaluation of usability, based on laboratory tests, tend not to be appropriate, as they demand an amount of effort and time that developers are not willing to spend. In order to tackle this problem, we developed USABILICS, a system targeted for the automatic remote evaluation of usability based on an interface model. The proposed model allows the definition of tasks using a simple and intuitive approach, which can be applied to large and dynamic Web applications. In this paper, we detail this approach, presenting the tool that supports the definition of tasks. Moreover, we show that our approach is effective towards supporting the definition of complex tasks in a myriad of modern Web applications.

KEYWORDS

Remote usability evaluation, web applications, task definition tool, task analysis

1. INTRODUCTION

Developing applications for the fast-paced environment of the Web is a challenging task. Competing on "Internet time" requires speeding up the development cycle in order to deploy applications on the Web as fast as possible. In this scenario, usability principles are rarely considered in the development process, resulting in Web application interfaces that tend to present usability problems.

Using regular approaches based on laboratory tests for evaluating the usability of Web applications may not be appropriate, as these approaches demand an amount of effort and time that developers are not willing to spend. An option to tackle this problem is using remote automatic or semi-automatic usability evaluation tools. These tools allows evaluating a large number of users by a low cost, as users and evaluators may be separated in time and space (Andreasen et al, 2007).

The usual approach for providing remote usability evaluation consists on capturing log information on the client using applications that run in background gathering information about the user's interaction (Ivory and Hearst, 2001). The captured logs are sent to server-side applications, where they may be processed in different ways. An effective way towards identifying usability problems consists in analyzing the captured events according to a task model, which is previously defined for the application under evaluation. The comparison between the sequence of events performed by the end user and the sequence of events defined on the model may indicate usability problems. WebRemUSINE (Paganelli and Paternò, 2002) and AWUSA (Tiedtke et al, 2002) are examples of tools that exploit this approach. These tools, however, use procedures for defining tasks that do not scale well for large and dynamic Web applications, in which tens or even hundreds of tasks need to be defined.

In order to tackle this problem, we proposed an interface model that supports the definition of tasks in a simple and intuitive way (Vasconcelos and Baldochi, 2012). An important aspect of our model is the fact that it considers the similarity among the possible paths of a given task, allowing a generic approach for the

definition of similar paths. This approach considerably shortens the time taken to define tasks, as a group of similar tasks may be represented by a single generic task.

We validated our model by implementing USABILICS, a system that evaluates the execution of tasks by calculating the similarity among the sequence of events produced by users and those previously captured by evaluators. This evaluation provides a metric for the efficiency and for the effectiveness of each evaluated task. We named this metric the *usability index* of a task (Vasconcelos and Baldochi, 2011). USABILICS also provides recommendations detailing actions to be performed in order to solve detected usability problems (Vasconcelos and Baldochi, 2012).

We performed several experiments that showed the effectiveness of USABILICS towards detecting usability issues and recommending fixes to solve these issues. In order to validate our approach for calculating the usability index, we selected tasks from different applications and evaluated them using USABILICS. Following, we applied laboratory-based tests to the same tasks. We noticed an agreement between the results of the lab tests and the values of the usability indexes. We also applied validation tests in order to verify the effectiveness of our recommendations. By following the automatic recommendations provided by USABILICS, developers were able to raise the usability index in most of the tested applications.

By testing USABILICS with more than a dozen of different Web applications, we could notice some limitations in our system, particularly in the definition of tasks. The initial version of USABILICS provided a tool that supported the definition of tasks composed of linear paths, where the beginning and the end of each task was clearly defined. There are, however, tasks that do not present a linear sequence of events. The task *adding a product to a shopping cart*, is an example of such task. In this task, the end user may perform a linear sequence of steps in order to select a product and then add and remove the product to and from the cart several times. Just because the user is not sure about buying the product, it does not mean that she is performing a wrong action. USABILICS was unable to detect this kind of behavior. Issues were also found in the definition of tasks where the user may perform a certain sequence of events in any order – as filling a form, for instance. Finally, defining tasks that present optional events was also not supported.

In order to address these issues, we developed UsaTasker, a task definition tool that supports the management of tasks targeted to be evaluated in a Web application. UsaTasker allows evaluators to define tasks by simply interacting with the application's graphical interface. After recording the task, our tool provides facilities for the management of the captured events, allowing (i) the definition of sequence of events in which each event may occur in any order; (ii) the definition of sequence of events that may be repeated several times; and (iii) the definition of optional events.

The main goal of USABILICS is performing usability evaluation without burdening both the application developer and the end user. In order to accomplish this goal, UsaTasker provides a graphical user interface which presents a task as a sequence of boxes, where each box represents an event of the task. In order to define an event as optional, all the evaluator needs to do is to select the desired box using the mouse and change its property from mandatory to optional. The other functionalities of the tool are equally simple.

This paper presents UsaTasker, showing that it plays an important role towards making USABILICS a complete and robust system for the usability evaluation of modern Web applications. The text is organized as follows. Section 2 presents COP, the interface model that allows the generalization of tasks. Section 3 details how UsaTasker exploits COP in order to define generic tasks. This section also presents the graphical user interface tool that allows the management of captured tasks. Following, Section 4 shows UsaTasker in action, presenting a usage scenario in which the most important features of UsaTasker are exploited. Finally, Section 5 presents our final remarks and discusses future work.

2. THE COP INTERFACE MODEL

The automatic remote usability evaluation based on task analysis is commonly performed using test scenarios, which can be analyzed by specific algorithms, such as those that identify interaction patterns, or those that points out the interactions that do not follow the optimal path of a task. For modern Web applications, the definition of the set of tasks that composes a test scenario may be very time consuming, as there are virtually hundreds of paths that may be used to perform a given task. Therefore, the definition of each possible path for each task of a large application is simply not feasible.

A Web application is composed of a collection of pages, which in turn are composed by elements such as hyperlinks, tables, forms, etc. These elements, specially in dynamic Web applications, are commonly shared among several pages. By exploiting this pattern, we proposed COP, an interface model that aims at facilitating both the definition and the analysis of tasks (Vasconcelos and Baldochi, 2012). The COP model is based on three main concepts: ***C****ontainer*, ***O****bject* and ***P****age*. An *object* is any page element that the user may interact with, such as hyperlinks, text fields, images, buttons, etc. A *container* is any page element that contains one or more objects. Finally, a *page* is an interface that contains one or more containers. According to the 4.01 HTML specification, the attributes *id* and *class* are identifiers for page elements, and the *id* attribute needs to be unique within a page.

According to the COP model, an object may be unique (using its id) or similar to other objects in terms of formatting (i.e. border or font type, color, etc.) and/or in terms of content (i.e. hyperlinks, buttons and images). The same applies to containers: a container may be identified in a unique way, or it may be classified as similar to other containers, but only in terms of formatting. Finally, it is worth noticing that objects and containers may appear in one or more pages of a Web application.

The structure of the COP model is presented in Figure 1. A *unique object* is an object that is identified in a unique way. Unique objects, as well as similar objects in terms of formatting or content may be kept within a single container, which is also identified in a unique way (*unique container*), or in two or more *similar containers*. Figure 1 also shows that a container may belong to a unique page or take part in several pages.

As far as the W3C Recommendation for the construction of hypertext-based documents (Raggett et al, 1999) is concerned, it is possible to say that the concepts of the COP model are sufficient to uniquely identify any element in a Web interface. Therefore, we advocate that our model is adequate for the definition of tasks in Web applications. Defining a task means specifying an optimal path for accomplishing this task. An optimal path for a given task is the sequence that presents the smaller number of required events for performing the task.

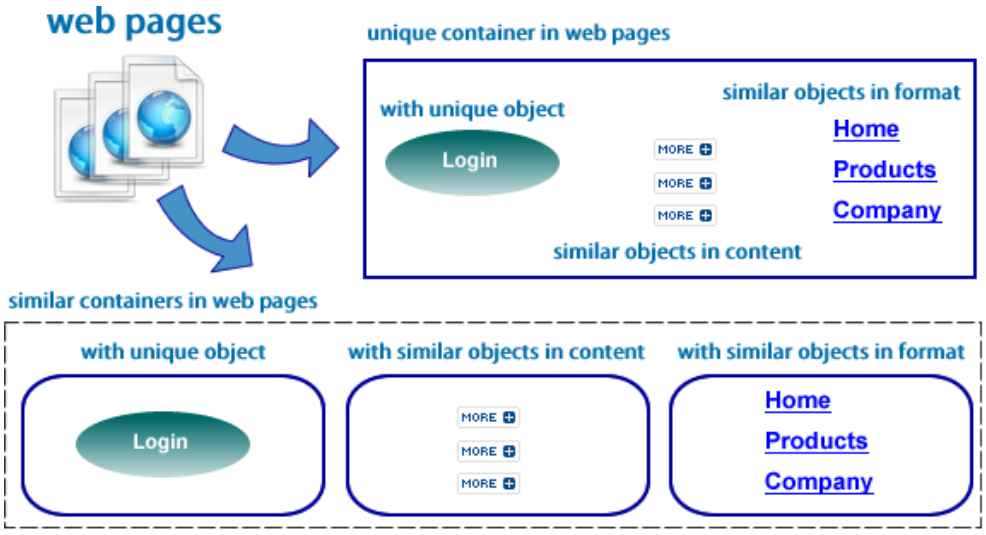

Figure 1. The COP model.

UsaTasker exploits the COP model in order to allow the generalization of events for similar objects and containers. This feature allows to represent a (large) group of similar tasks using a single captured task. In order to make this feature clear, consider the definition of the task *buying a product* in an e-commerce Web site that has 10,000 products for sale. Individually specifying all the possible paths to perform this task is virtually impossible. However, by exploiting the COP concepts, UsaTasker allows the automatic definition of alternative paths as it considers the similarities among objects and containers. As a result, the effort for defining tasks is considerably shortened.

3. TASK DEFINITION

Defining tasks is the starting point for providing remote and automatic usability evaluation. The usual approach for the definition of tasks is based on notations, which are used to specify task models. WebRemUSINE, for instance, exploits the *ConcurTaskTree* notation (Patternò, 2000) for defining tasks.

The use of notations, however, makes the definition of tasks cumbersome. Nielsen (1993) points out that learning complex notations and formal methods may prevent developers from applying usability evaluation in their projects. Moreover, it is also complex to compare the end user logging information to a notation-based task definition. Finally, it is worth noticing that the definition of tasks is specially challenging for today's large Web applications, in which it is possible to perform a given task using different paths in the application's GUI.

Towards making the definition of tasks easier, we developed a tool that allows defining tasks in a simple and intuitive way, by simply interacting with the application's graphical interface. Similar to Google Analytics (www.google.com/analytics) and WELFIT (Santana and Baranauskas, 2010), this tool is a server-side application that allows users to subscribe themselves as evaluators of specific Web applications. Therefore, the evaluator defines a task by simply using the application, in the same way end users are supposed to do.

In order to validate the USABILICS system, we performed several experiments with this tool (Vasconcelos and Baldochi, 2011, 2012). While testing USABILICS, we noticed that our approach for defining tasks as linear paths was not appropriate, as it was not able to support the definition of tasks that present: (i) events that can be performed in any order; (ii) optional events; and (iii) repeated events.

Besides this limitation, we noticed that more than just supporting the definition of tasks, our tool should support the management of tasks, providing CRUD-like operations. Therefore, we built a new version of our task definition tool, which we called UsaTasker.

UsaTasker provides an user-friendly interface for the management of tasks, where the evaluator can **c**reate, view (**r**ead), **u**pdate and **d**elete tasks. For creating a new task, the evaluator logs in the server-side application and fills in the name and a description for the task she wishes to define. Following, the selected application is loaded in a new window, making it possible to start recording the task. While the evaluator surfs the application interface, UsaTasker defines optional paths according to the options of the COP model. Upon finishing the task, the evaluator closes this window, stopping the recording process.

During the task definition, when the evaluator selects and object, such as a button, she is prompted with specialization options associated to this object and to its containers within the page. According to the COP model, containers can be tables, cells, forms and divs. This approach allows selecting any container of an object, making it simple to generalize tasks. As an example of this feature, consider a container A, which contains some links and another container, A1, also containing links. If the evaluator clicks in a link in A and selects the option “consider similar objects”, then that event will be generalized to all links in A and A1, as A contains A1. Therefore, when defining an event within a task, it is possible to consider from a single object of a form to all objects within a page.

When the evaluator finishes the task definition, UsaTasker presents the captured events graphically, as shown in Figure 2. This visual feedback is important towards providing a way to verify if each event that composes a task was correctly recorded. In Figure 2, each box represents an event, and the blue directed edges indicate the order of each event within the task. Besides viewing the details of each event, the evaluator may delete an event, if she considers that this event is irrelevant in the optimal path of the task. To perform the deletion of an event, all the evaluator needs to do is clicking on the *X* icon on the top of the desired box.

The graphical representation of events provided by UsaTasker was specially tailored to address the limitations of our first task definition tool. Therefore, the next subsections shows how UsaTasker allows the definition of tasks that present events that can be performed in any order, optional events and repeated events.

3.1 Ordering of Events

There are tasks in which the order of events is irrelevant. In order words, these tasks do not present a precedence order among its events. The filling of a form is an example: the end user may fill the text fields in any order, if all mandatory fields are filled, the task succeeds.

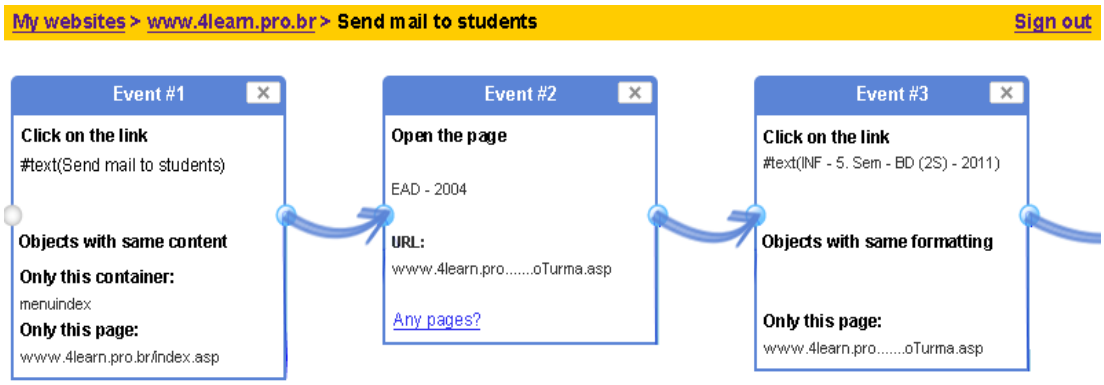

Figure 2. Captured events in UsaTasker.

UsaTasker, by default, defines that the events of a task present a precedence relation, according to the order that the events were recorded during the definition of the task. However, after recording the task, the evaluator may define that an event or a set of events do not present such relation. To perform this action in the UsaTasker GUI, the evaluator selects the events (consecutive boxes) in which she does not wish to apply the precedence relation. Figure 3 presents a sequence of boxes representing events. When an event is marked as without precedence, it appears with a yellow background.

3.2 Optional Events

In order to evaluate a task performed by an end user, USABILICS compares the sequence of events performed by this user to the corresponding sequence recorded by the evaluator in the definition of the task (the optimal path for the task). Events in the user's sequence that do not belong to the optimal path are considered *wrong actions*. The more wrong actions appear in the user's sequence of events, the lower is the usability index for that task.

There are, however, some actions that should not be considered wrong, even if they result in events that do not belong to the optimal path of the task. Consider, for instance, an application that provides a Help button to aid the end user. Clicking this button should not impact the evaluation of the task. Another example is the optional fields of a form.

UsaTasker provides a feature to mark a box as optional, indicating that the corresponding event should not be considered in the analysis of the task. Figure 3 shows an optional event, which is indicated by the dashed line on the border of the event's box. It is worth noticing that an event may be optional and, at the same time, belong to a sequence of events without precedence relation. This leverages the flexibility for the definition of complex tasks.

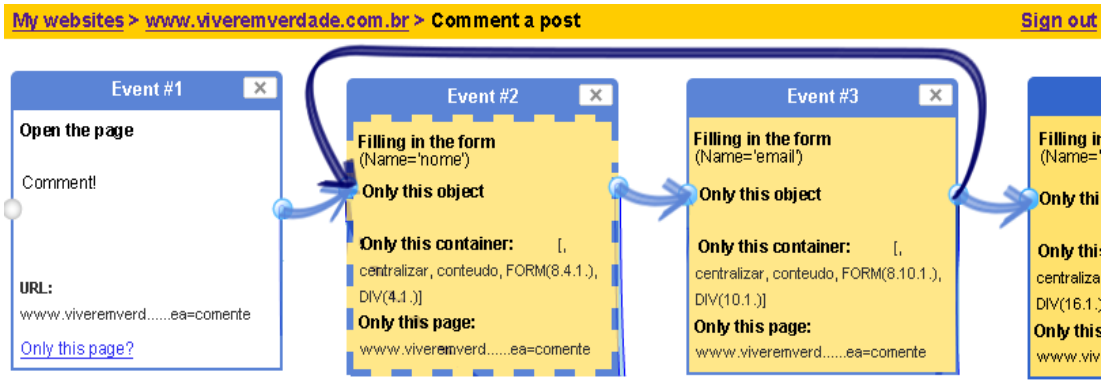

Figure 3. UsaTasker features: optional and repeated events.

3.3 Repeated Events

Modern Web applications present events that could or should be repeated a given number of times within the sequence of events of a task. In an e-commerce application, for instance, the action of selecting a product and putting this product in the shopping cart may be repeated many times, according to the quantity of products the end user wishes to buy.

In order to allow the repetition of events within a task, UsaTasker presents a feature that allows selecting sequences of events that may be repeated. In order to define repeated events, all the evaluator needs to do is to click in the box that represents the first event in the sequence and then click in the last box of the sequence. After this procedure, a dark blue directed edge connecting the first and the last box will appear. Figure 3 shows that *Event 2* and *Event 3* may be repeated.

4. USATASKER IN ACTION

UsaTasker was designed to make the definition of tasks simple and intuitive. It was specially tailored for modern Web applications, which present pages containing tens or even hundreds of components, usually nested inside different containers. In order to illustrate the features of UsaTasker and show its effectiveness towards defining tasks for today's applications, we trace the procedure for defining the task *buying a deal* in Groupon (www.groupon.com), probably the most popular collective buying website.

The first step towards buying a deal, is selecting the deal. Figure 4A illustrates the selection of the first deal on the deal selection page. The blue rectangle highlights the object selected by the evaluator and the red one indicates the container of this object. Using the generalization options from the COP model, it is possible to define this event a single time for all deals in the website. To do this, the evaluator chooses the options *objects with same content*, in *similar containers*, in *this page*. As a result, all buttons with the image *View Deal* inside all containers in the deal selection page will be considered.

When a deal is selected, the page shown in Figure 4B is loaded. This page presents a *Buy* button, which should be clicked to proceed with the purchase. When the evaluator clicks this button, she is prompted again with the options of the COP model. At this time, the selected options are: *objects with same content*, in *similar containers*, in *any pages*. By selecting these options, a general event is created that considers the selection of all *Buy* buttons in all deal pages. Using the feature of repeated events, it is possible to define that this action may be repeated several times, so that the customer can buy several deals before proceeding to checkout. This is not shown here due to space restrictions.

Figure 4C presents the discount options that are shown in some deals when the customer press the *Buy* button. In order to include the discount options as part of the task, the evaluator may click on any of the discount links and choose the following options from the COP model: *objects with same formatting*, in *similar containers*, in *any pages*. Considering that the discount options are only presented in some deals and that some customers are not eligible to discounts, this event should be optional. Therefore, after finishing the recording of the task, the evaluator must select the box associated to this event and mark it as optional.

When proceeding to checkout, the *Sign in* page shown in Figure 4D may be displayed to the customer, in case she has not logged in before. This event may be defined as optional, as it will not happen for all customers.

The blue rectangles on Figure 4E highlights objects that the customer may interact with before finishing the purchase. The customer may update the purchase options, changing the quantity of deals, for instance. She may also change her personal information. These actions are optional, and therefore, the evaluator must mark their corresponding boxes as optional, after the end of the recording procedure.

Finally, in the checkout page shown in Figure 4F, the evaluator fills the text fields highlighted in blue (payment information) and clicks the *Complete Order* button. These events are identical for all purchases, therefore, the selected options of the COP model for the text fields are: *objects with same formatting*, in *similar containers*, in *any pages*. For the button, the options are *objects with same content*, in *similar containers*, in *any pages.* As a result, a general checkout event is created.

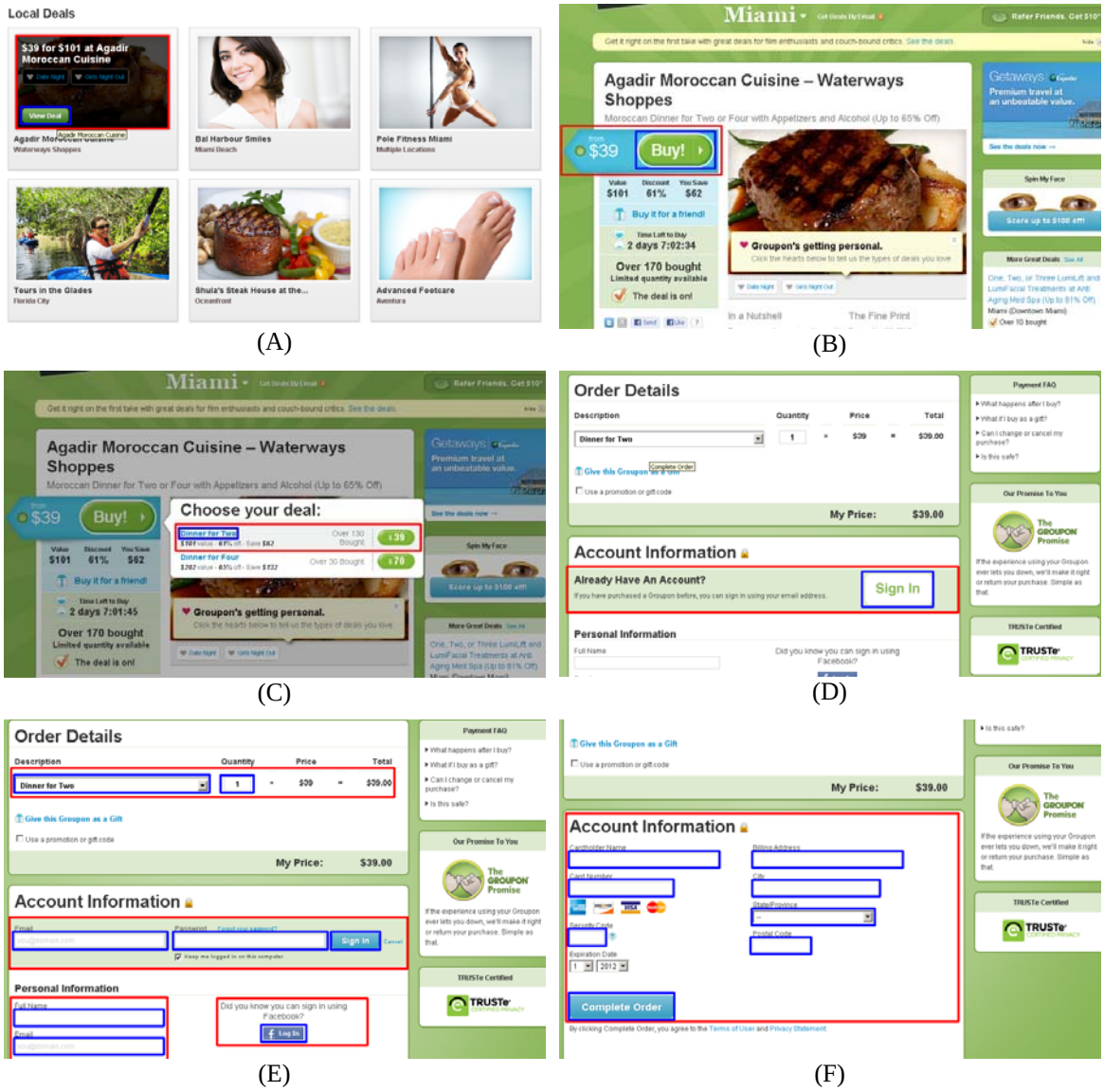

Figure 4. UsaTasker in action - defining a task.

Figure 5 depicts the graphical representation of the recorded task. Boxes with dashed lines represent optional events. Boxes with yellow background shows events that do not present precedence relation, i.e., may occur in any order. Finally, the blue line connects events that may be repeated.

5. CONCLUSIONS AND FUTURE WORK

Performing usability evaluation in large and dynamic Web applications is not a simple issue. At one hand, traditional laboratory-based tests are costly and time-consuming. At the other, existing remote usability evaluation tools are not effective towards evaluating complex modern Web applications. In order to tackle this problem, we developed USABILICS, a system target to provide remote and automatic usability evaluation based on task analysis.

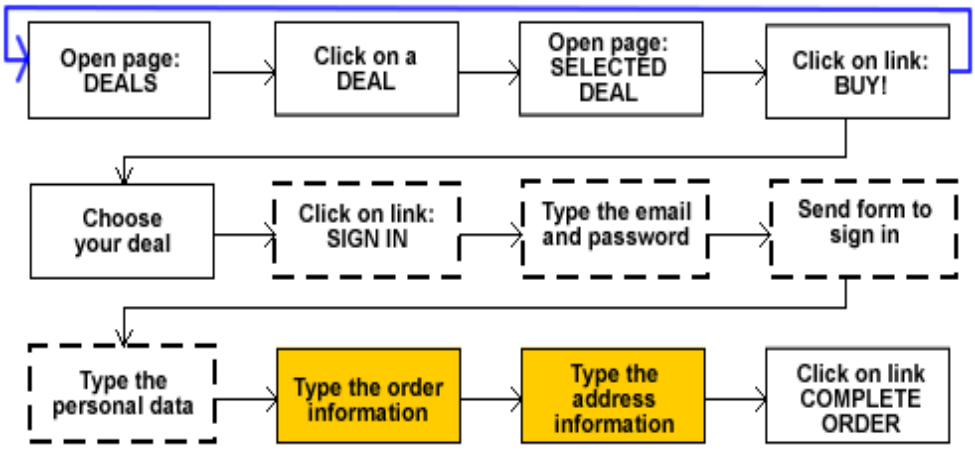

Figure 5. UsaTasker representation for the task *Buying a deal*.

In order to evaluate tasks, the first step is defining them. Existing approaches for defining tasks are too complex, therefore their usage for Web developers is limited. Moreover, these approaches do not provide solutions that scale well for applications that present hundreds of tasks. In order to address this issue, we developed UsaTasker, a task definition tool supported by COP, our interface model. UsaTasker was tailored for today's large Web applications, providing generalization options that make the definition of tasks easier and faster. It provides CRUD-like operations, simplifying the management of tasks. Moreover, it allows viewing tasks as sequences of boxes representing events. Using this interface, evaluators may (i) define an event as optional, (ii) allow the repetition of events and (iii) change the precedence status of an event.

UsaTasker empowers the USABILICS system, making it a robust solution towards providing remote and automatic usability evaluation for Web applications. To the best of our knowledge, no reported tools provide the features found in UsaTasker.

As future work, we plan to improve UsaTasker in order to cluster tasks that present intersections in their sequence of events. If two or more tasks present an identical sequence of events, we may use a single sequence for all these tasks, reducing the amount of information that needs to be stored.

REFERENCES

Andreasen, M. S. et al, 2007. What happened to remote usability testing?: an empirical study of three methods. *Proceedings of the SIGCHI conference on Human factors in computing systems CHI '07*, pp. 1405–1414.

Ivory, M. Y. and Hearst, M. A., 2001. The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys 33*, pp. 470–516.

Nielsen, J., 1993. Usability Engineering. Academic Press.

Paganelli, L. and Paternò, F., 2002. Intelligent analysis of user interactions with web applications. *Proceedings of the 7th international conference on Intelligent user interfaces (IUI '02)*, pp. 111-118.

Paternò, F., 2000. *Model-Based Design and Evaluation of Interactive Applications* . Springer.

Raggett, D. et al, 1999. The global structure of an HTML document. http://www.w3. org/TR/1999/REC-html401-19991224/ struct/global.html.

Santana, V. F. and Baranauskas, M. C. C., 2010 . Summarizing observational client-side data to reveal web usage patterns. *Proceedings of the 25th ACM Symposium on Applied Computing (SAC '10)*, pp. 1219–1223.

Tiedtke, T. et al, 2002. AWUSA: A tool for automated website usability analysis. *PreProceedings of the 9 th International Workshop on the Design, Specification and Verification of Interactive Systems* , pp. 251–266.

Vasconcelos, L. G. and Baldochi, L. A., 2011. USABILICS: remote usability evaluation and metrics based on task analysis (in portuguese). *Proceedings of the 10th Brazilian Symposium on Human Factors in Computer Systems & 5th Latin American Conference on Human-Computer Interaction*, Porto de Galinhas, Brazil, pp. 303–312.

Vasconcelos, L. G. and Baldochi, L. A., 2012. Towards an automatic evaluation of web applications. *Proceedings of the 27th ACM Symposium on Applied Computing (SAC '12)*, Riva del Garda, Italy, pp. 709-716.