(De-)Composing Web Augmenters

Sergio Firmenich[1], Irene Garrigós[2], and Manuel Wimmer[3]

[1] LIFIA, Universidad Nacional de La Plata and CONICET Argentina
sergio.firmenich@lifia.info.unlp.edu.ar
[2] WaKe Research, University of Alicante, Spain
igarrigos@dlsi.ua.es
[3] Business Informatics Group, Vienna University of Technology, Austria
wimmer@big.tuwien.ac.at

Abstract. Immersed in social and mobile Web, users are expecting personalized browsing experiences, based on their needs, goals, and preferences. This may be complex since the users' Web navigations usually imply several (related) Web applications. A very popular technique to tackle this challenge is Web augmentation. Previously, we presented an approach to orchestrate user tasks over multiple websites, creating so-called procedures. However, these procedures are not easily editable, and thus not reusable and maintainable. In this paper, we present a complementary model-based approach, which allows treating procedures as (de)composable activities for improving their maintainability and reusability. For this purpose we introduce a dedicated UML profile for Activity Diagrams (ADs) and translators from procedures to ADs as well as back-translators to execute new compositions of these procedures. By combining benefits of end-user development for creation and model-driven engineering for maintenance, our approach proposes to have the best of both worlds as is demonstrated by a case study for trip planning.

1 Introduction

The evolution of the Web is a complex and constant process. Nowadays, immersed in social and mobile Web, users are expecting a personalized browsing experience, which adapt to their needs, goals, and preferences. One of the main limitations of how to adapt the application to each user is the current use of the Web. When performing a concrete task (e.g., organizing a trip) the user normally exceeds the application's boundaries, visiting several (related) Web applications. In cases like these, the user may feel a loss of context every time she navigates from one application to another, because the new application used has no way of tracking the previous user navigation. This missing integration, and also a lack in customization, has a deep impact in the user's browsing experience.

These limitations motivated the development of mash-ups tools [21] in order to merge a set of resources that are scattered among different websites into specialized applications. One often occurring limitation is that mash-ups are used straightforward when most of the tasks users perform are volatile and do not require the creation of entirely new applications. In the same context of managing existing Web applications, another technique that has emerged is called Web augmentation [3]. Web augmentation

S. Casteleyn, G. Rossi, and M. Winckler (Eds.): ICWE 2014, LNCS 8541, pp. 359–369, 2014.

is the activity of navigating the Web using a "layer" over the visited websites. This layer may manipulate the original UI of existing third-party websites; in this way, users perceive an augmented website instead of the original one. Generally, these augmentations are performed on the client-side, once the content is delivered from the server. Normally, users having some kind of programming skills are the ones who develop the software artifacts that perform these augmentations. Web augmentation as a technique may be applied with different aims; from simple presentation changes to task-based Web integration mechanisms.

In [6] we presented an approach based on Web augmentation to orchestrate user tasks over multiple websites. It supports flexible processes by allowing the users to combine manual and automated tasks from a repertoire of patterns of tasks performed over the Web, creating so-called *procedures*, which are persisted in XML files. Although the tools around our previous approach allow users to record their own *procedures* by-example, and subsequently edit the details; larger editions, such as replacing several tasks with other equivalent ones or building reusable chunks, is challenging. However, this may be often required, since several large tasks such as planning a trip involve several smaller ones (book flights, hotel rooms, cars, etc.) and the requirements involved change, as well as the browsed websites. If the user wants to change a larger part of the procedure, the process order may have to be changed, additional tasks have to be intermingled, or complete procedures have to be substituted or executed in series. The importance of these aspects has been studied before in the field of Web applications [13] [17]. These are also relevant issues in the context of Web Augmentation, not only because the Web changes constantly and consequently the scripts may stop working, but also because the same script could be reused in several Web pages under the same domain [13].

Therefore, one challenging aspect for those approaches that support users tasks based on Web augmentation, is the maintenance of procedures, which has associated two dimensions: *(i)* how to reuse existing augmentation units in order to support complex scenarios (i.e., how to compose them to fulfill a larger goal), *(ii)* how to decompose subtasks and make them reusable chunks. In order to tackle these challenges, this paper extends our previous work with a modeling language based on UML Activity Diagrams (ADs) to represent the procedure's tasks involving dedicated transformations from procedures to activities. Models allow raising the abstraction level and the separation from the applications functional specification [19], which improve the reusability and maintenance of the procedures. In this way, the maintenance of existing procedures as well as the composition of new ones based on existing building blocks is supported by graphical modeling. By having the transformations from activities to procedures, we are able to execute new compositions of Web augmenters. With this approach, we combine the benefits of end-user development for creating procedures based on Web augmentation and model-driven engineering for maintaining Web augmenters to have the best of both worlds as is demonstrated by a case study for trip planning.

The remainder of this paper is as follows: Section 2 briefly summarizes our previous work on Web augmentation and introduces an example used to illustrate our approach. Section 3 elaborates on the proposed model-based approach for representing procedures. Section 4 discusses the state-of-the-art on Web augmentation, and finally, we conclude with pointers to future work in Section 5.

2 Background

Web augmentation is used for improving the user experience in several aspects. In particular, we have previously proposed an approach for supporting Web tasks by supporting users with *procedures* [6]. Procedures are programs focused on executing augmentation tasks when some user interaction is detected. These artifacts support tasks involving more than one application, and also give some mechanisms for moving information from one application to another one. In order to specify *procedures* we have previously designed a DSL based on XML that defines a procedure as a sequence of tasks. This DSL has been improved in the context of this work. The current version of the procedures metamodel is shown in Figure 1.

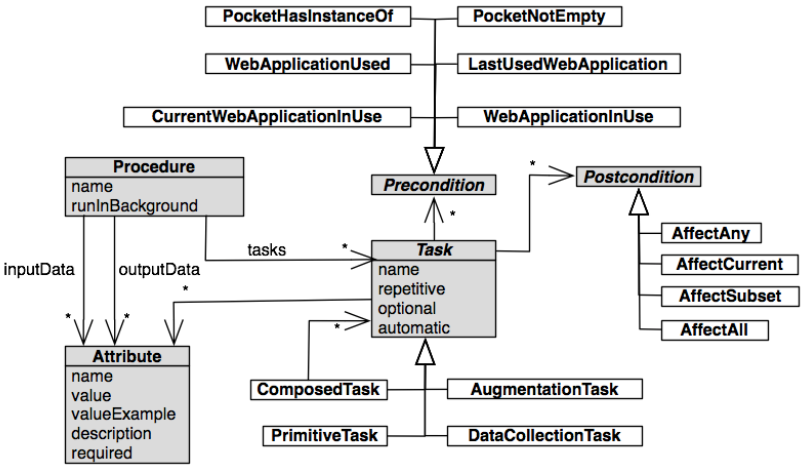

Fig. 1. The metamodel of the Web Augmentation DSL

The main concepts around the DSL are explained in the following.

- There are four types of tasks:
  - *Primitive tasks*: are based on common actions that users perform when navigating the Web (e.g., clicking an anchor).
  - *Augmentation tasks*: are tasks that allow the execution of a specific augmenter developed with our underlying framework for Web augmentation [7].
  - *DataCollection tasks*: this kind of tasks enables procedures to contemplate data collected by users. These tasks are also strongly related to *DataCollectors* and *Pocket*, two tools distributed with the framework supporting the procedures, which allow users to move information among Web applications.
  - *Composed tasks*: these tasks make possible to group other instances of tasks in order to manage them altogether. As an example, imagine the need of executing an augmenter each time that the user collects some information. In this case both tasks may be grouped in order to do repetitive the whole set.
- All tasks have three properties: *(i) repetition* property for specifying if the task may be executed more than once; *(ii) optional* property allows skipping the

execution of the task; *(iii) automatic* property is *true*, then the Web augmentation framework automatically triggers the task.

- Tasks have attributes representing information needed for the execution, e.g., if an augmenter is applied for filling in a form and it is marked as automatic, the augmenter needs to know which form fields are filled with which value.
- Tasks may have preconditions. Preconditions are used to decide if the task will be executed or not according to which information is currently available. There are two main kinds of preconditions: on the one side, *preconditions about collected data,* and on the other side, *preconditions about navigation history*.

Our approach gives support to the end-user with visual tools, deployed as Web browser plugins, for creating and executing *procedures*. Figure 2 shows the editor: a sidebar that allows users to specify tasks into the procedure while analyzing websites. The tool provides an assisted mode: users may *record* their interaction with the Web and the corresponding tasks will be added to the procedure automatically. This mode contemplates primitive tasks, augmentation tasks, and data collection tasks. Figure 3 shows how to edit a particular task. It allows users to specify the name, pre and post-conditions as well as values for both properties and attributes. If some sensitive information is saved when recording the interaction, users may remove it by editing the corresponding task.

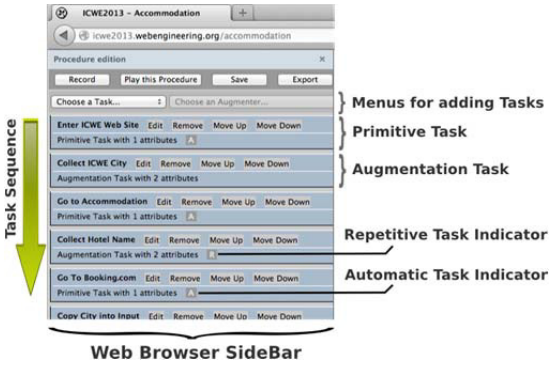

Fig. 2. General view of the tool

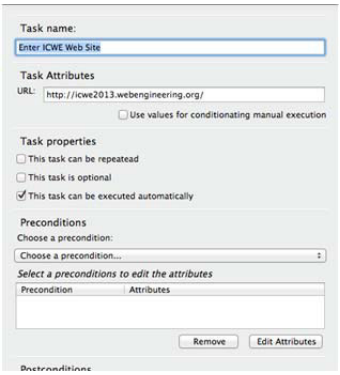

Fig. 3. Edition of a single task

In order to give more insights to the real use of *procedures*, consider the following example (it will be used as a running example during the rest of the paper). The example responds to the following situation:

> *"Peter is going to travel to Paris for vacation. In that context, he has to buy flights from his town to there, and also book a taxi from the airport to downtown. For accomplishing these tasks, he uses different websites, e.g., expedia.com and wecab.com. In each of these two subtasks, Peter has to enter the same information. Besides booking flights and taxi, Peter is also interested in getting touristic information about Paris and nearby areas".*

In scenarios like this, users may take advantage of using Web augmentation approaches, since these may support users on moving relevant information from one application to other while using this information for executing augmenters in the visited websites. It is important to note that not only each augmenter is configurable (even replaceable by other similar ones) but also the subtasks (*book flight* or *book taxi*) may be also reordered and replaced according with the user's interest. Also the information used for performing the tasks (both primitive and augmentation tasks) may vary in distinct executions of the same procedure. It could be achieved by using conceptual tags during data collection tasks. In this way, if different users prefer, for example, different hotel's location or airlines, the procedure can be defined for consuming information through concept names such as "Hotel Location" or "Airline" instead of concrete data.

Although this is a common scenario, the order used for each subtasks may vary for different instantiations of the same scenario, when these are more complex. It may vary even more when Web augmentation is involved, because it is desirable to allow users to vary the augmentations applied in an easy way and to compose different procedures to solve larger examples. Thus, we provide a complementary extension to the end-user based development of Web augmenters, namely a model-based maintenance approach as explained in the next section.

3 (De-)Composing Procedures – A Model-Based Perspective

In order to solve the before mentioned drawbacks, we present a model-based approach, which allows treating procedures as composable activities. For this purpose we introduce transformations from procedures to activities as well as back-transformations to be able to execute new compositions of augmenters. In order to do so, we first need to be able to represent the procedures on the model level. With this goal in mind, we propose to use UML activity diagrams (ADs).

3.1 Model-Based Representation of Procedures

Representing procedures with ADs [16], in particular following the fUML execution semantics proposed by the OMG [15], requires a systematic mapping between our DSL and ADs. Here we follow existing methodologies for deriving UML profiles from DSL metamodels [18,20]. After investigating ADs for the purpose of modeling *procedures*, we identified a high overlap, although the later are, of course, more specific as the former. The following table illustrates the identified mappings between our DSL and ADs from a Web augmentation point of view, i.e., only the AD concepts are shown that are corresponding to the DSL concepts.

In addition to the mappings, to explicitly represent the specifics of Web augmenters (cf. Table 1 – column comments), we introduce a Web Augmentation profile for ADs. By using this profile, we are able to provide information preserving the transformations between the executable procedures expressed as XML files and the corresponding ADs. This property is one of the main building blocks of our approach to allow the continuous development on the front-end side (recording and testing procedures) as well as on the model side (maintaining and composing

Table 1. Mapping of Web Augmentation concepts to UML activity diagrams

Web Augmentation Procedures	UML Activity Diagrams	Comments
Procedure	Activity Diagram	*runInBackground* attribute has no direct mapping to UML, rest has direct mapping to UML
Task	Activity	*Optional*, *automatic*, *repetitive* attributes have no direct mapping to UML, rest has direct mapping to UML
PrimitiveTask	Activity	May be mapped to *Action* metaclass, but to allow for properties, *Activity* is used as metaclass
AugmentationTask	Activity	Same comment as for PrimitiveTask
DataCollectionTask	Activity	Same comment as for PrimitiveTask
ComposedTask	Activity	Activity may contain other activities by using CallBehaviorAction
Attribute	Property	*Value* and *example* attributes have no direct mapping to UML, rest has direct mapping to UML
Precondition	Constraint (LocalPreCondition)	*Precondition* subclasses have no direct mapping to UML
Postcondition	Constraint (LocalPostCondition)	*Postcondition* subclasses have no direct mapping to UML

procedures). Figure 4 shows the introduced stereotypes (for each meta-class we introduce a stereotype) as well as the extended metaclasses of UML. Please note that we only introduce tagged values in the profile for properties of the DSL that miss corresponding properties in the base UML metaclasses. By this, we ensure to reuse as much as possible the UML language and to keep the profile concise and minimal.

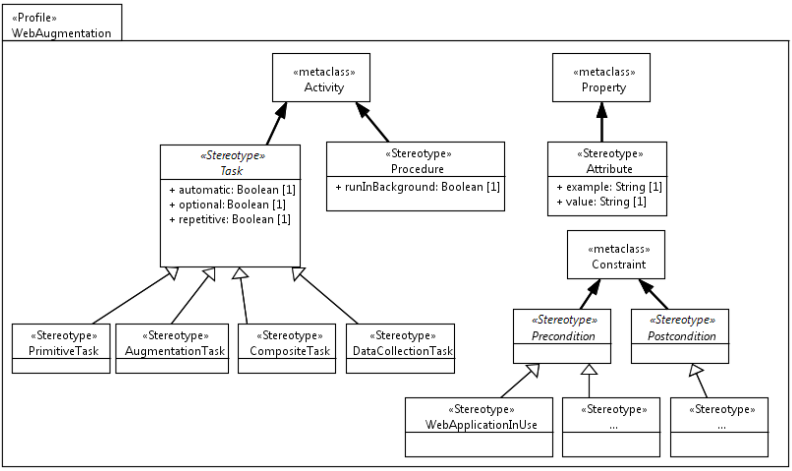

Fig. 4. Web Augmentation Profile – an Extension for UML Activity Diagrams

To summarize the syntax of the developed profile, we use the *Activity* metaclass as the main metaclass for our extension. We map the *Procedure* metaclass to the *Activity* metaclass, because ADs are UML internally represented by a root activity that contains all the elements shown within the ADs. Also the different kinds of *Task*s are mapped to the *Activity* metaclass. Because the *Activity* metaclass inherits from the

Classifier metaclass in UML, activities may contain properties. This is exactly what is required for reflecting the *Attribute* concept of *Tasks*. Besides these aspects, activities may also be nested, by using the *CallBehaviourAction* that is also able to trigger another activity from a context activity. By this, we can simulate the *CompositeTask* concept of the Web Augmentation DSL.

Moreover, we extend the metaclass *Property* with a stereotype to represent the *Attribute* metaclass of the DSL and to introduce additional attributes to allow the definition of an actual *value* and an *example value* for properties. Finally, we extend the *Constraint* metaclass of UML with specific stereotypes to reflect the specific pre- and post-condition types contemplated in our DSL.

Concerning the semantics of ADs, we consider an explicit control flow, normally a sequence of tasks, by defining *control flow links*. This is quite analogue to the sequence of tasks involved in a procedure and the information flow among these. In addition, we also exploit other control structure possibilities of ADs such as parallelization, conditions, etc. However, these constructs are not explicitly available in the current version of the Web augmentation DSL and thus, have to be compiled to a more verbose representation on the execution level. In addition to the control flow, we explicitly model the data flow, i.e., to represent the pocket and data collectors of the Web augmentation DSL, by making use of the *object flow links* supported by UML activity diagrams. By using this type of links, we are able to connect *activity parameter nodes*, i.e., parameters that are set externally before calling a certain activity as well as parameters that provide values after the execution of an activity to its environment, with so called *pins*. Activities may have input and output pins that represent input and output parameters, respectively. Pins are a powerful modeling concept in UML, e.g., by setting the multiplicity of input pins, input pins may be defined as mandatory or optional (i.e., a value is available or not for a given pin) for the execution of an activity. By linking output pins with input pins, data exchange between two activities is defined.

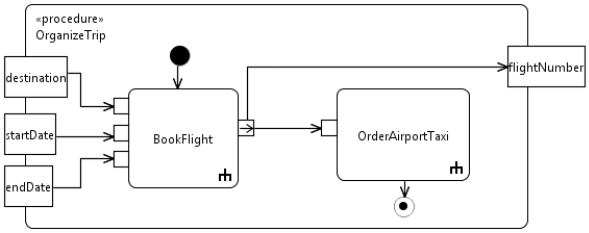

Fig. 5. Procedure OrganizeTrip in UML Activity Diagram Notation

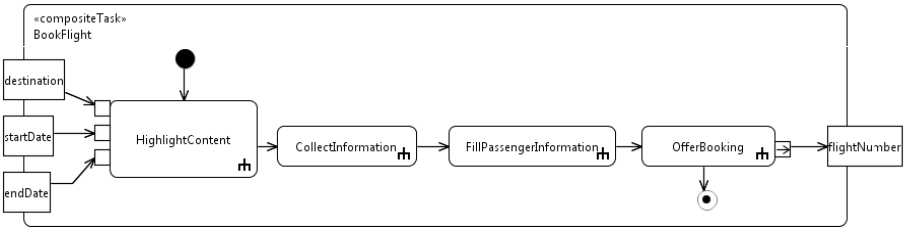

Fig. 6. Composite Task *Book Flight* as UML Activity Diagram

Consider again our main example. If we take only one of the main subtasks, such as *book flights* (a *CompositeTask* called *BookFlight*), an activity diagram with stereotype «*procedure*» is generated (cf. Figure 5) for visualizing the execution of the sequence of *tasks* as illustrated in Figure 6. In this specific case, and for reason of conciseness, we only contemplated *AugmentationTasks*, but the given activity may also include several *PrimitiveTasks* allowing the *procedure* developer to specify specific user interactions. Again we use the *CallBehaviorActions* to call the primitive and augmentation tasks.

3.2 Transformation Chain: Procedures to Activities and Back Again

In order to allow for a transparent transition from Web Augmentation (WA) DSL expressed in XML to UML activity diagrams (ADs) and back again, we implemented a bi-directional transformation chain consisting of a set of transformations as explained in the following paragraphs. More information on the implementation may be found at our project website[1].

Model Injection/Extraction Transformations. We developed an XML 2 WA DSL transformation that parses the XML-based representations and produces models conform to an Ecore-based WA DSL metamodel. In addition, we developed a WA DSL 2 XML transformation for printing models back to executable XML code. These transformations have been implemented in Groovy[2] due to its dynamic programming features and the support by the XmlSlurper and XmlMarkupBuilder APIs.

DSL/UML Integration Transformations. We developed a WA DSL 2 UML AD transformation that produces UML models from WA DSL models and applies automatically the Web augmentation profile to the UML models. In addition, we also developed the inverse transformation that takes a profiled UML model and produces a WA DSL model. These transformations have been implemented in ATL [11] due to its support for EMF models as well as UML models and the possibility to deal with profile information within the transformations.

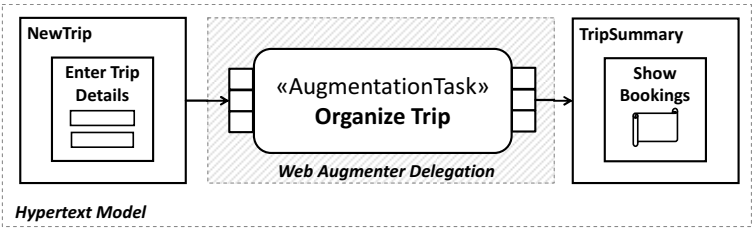

Fig. 7. Composing Web Augmentation Tasks with Hypertext Models

3.3 Composing Web Augmenter Models and Hypertext Models

One additional benefit of having Web augmenters explicitly modeled is the possibility to compose them with traditional Web design models such as supported by WebML,

[1] https://sites.google.com/site/decomposingwebaugmenters
[2] http://groovy.codehaus.org

OO-H [10], or UWE [12]. By this, Web augmentation techniques may be used by Web applications by delegating to pre-defined Web augmenters or Web augmenters may be developed for a specific Web application and integrated in the hypertext models of such applications. Consider the following example. Assume one would like to provide for a Web application that offers specific events the possibility to book a hotel room at an external website. Navigating to the external website with the specific information such as place and time may be provided by the hypertext model. This information may be passed by typical transport links transferring parameters to the Web augmenter activity (as it is done for standard hypertext nodes) and the Web augmenter activity may provide information of the booked hotel room back to the hypertext model again as parameters of a transport link. In Figure 7 we show such a composition of a hypertext model and a Web augmenter activity for the WebML language. We leave as subject for future work the creation of Web augmenter units for WebML based on the WebRatio inherent extension mechanism and the integration of the profile presented in this paper with the UWE profile for modeling hypertext models. We think this is an important line of future work to close the gap between traditional Web modeling and Web augmentation.

4 Related Work

Several approaches for supporting Web user tasks have been created, and different abstraction levels have been used. For example, CoScripter [1] proposes a DSL for supporting recurrent tasks, which may be parameterized in order to alter the data used in each step. The main idea of CoScripter is to automate some tasks by recording the user interactions (based on DOM events) and then the script may reproduce the same steps automatically. A similar approach, ChickenFoot [2], also proposes a DSL that raises the abstraction level of JavaScript programs in order to emulate user behaviour easily. However, although these approaches support slight changes in the task processes, considerable changes over these cannot be contemplated. These tools allow modifying end-user programs to vary the way that tasks are going to be performed, but usually, the *augmentation* effect is limited to a predefined subset of possibilities.

Although we share the philosophy behind these approaches, we think that further efforts should be made for making this kind of tools closer to the actual use of the Web, because users navigate the Web in a volatile way, and some tasks may be achieved in different ways (Web applications involved, data used, navigation) under different circumstances. In previous work we have presented our approach called *procedures*. Although this involves a composition of tasks where each task may be preconditioned and parameterized, the reuse of *parts* of procedures related to a particular subtask is not foreseen. All the mentioned approaches would improve taking into account some aspects from task modelling such as HAMSTERS [14], in which “abstract tasks” may be defined and the execution order may be more flexible.

The most related work in this context is [4], which proposed to model the user navigation using state machines in order to create the so-called webflows. This work defines a DSL, which allows users to specify the navigation flow as well as the data associated with each transition. One of the main differences to our work is the fact that [4] does not foresee the inclusion of third-party augmentations (i.e. developed by

users), which again implies a limitation of augmentation effects. In our approach, this is contemplated by the execution of augmenters [7]. Finally, [9] define a UML profile for data mashups, but the integration with Web augmenters is not considered.

5 Conclusions and Future Work

Web augmentation is an emerging trend that allows users to improve their experiences while navigating the Web. Several approaches have been proposed to improve websites with different goals, from accessibility aspects over data integration to complex user task support, which is the focus of this work.

Although there are currently several works aiming to support specific navigation scenarios, user navigation is not always systematic as current approaches assume. In this way, one of the main challenges in this context is to support users even under volatile requirements. There are several other issues in the middle, such as how easy users may define their own artifacts for these approaches. The key is to find a good trade-off between the expressivity of the approach (what can be specified) and the usability of the tools (how it is specified). Reaching this point is challenging, and in this work, we aim to address a solution of maintaining procedures by using activity diagrams, where each activity represents a relevant subtask in a more general navigation scenario. Of course, the target users of the proposed modeling approach may no longer be end-users, but Web engineers may decompose, recompose, and maintain already existing Web augmenters and integrate these pieces in their developed hypertext models. The next steps imply defining mechanisms for including the transformations developed in this work in our Web augmentation tools and performing experiments with different kinds of users. Since our underlying Web augmentation framework allows tracking the user interaction, we plan to incorporate aspect orientation concepts [8] in order to further (de)compose procedures when cross-cutting concerns occur.

References

1. Bogart, C., Burnett, M., Cypher, A., Scaffidi, C.: End-user programming in the wild: a field study of CoScripter scripts. In: VL/HCC, pp. 39–46 (2008)
2. Bolin, M., Webber, M., Rha, P., Wilson, T.: C. Miller R.: Automation and customization of rendered web pages. In: UIST, pp. 163–172 (2005)
3. Díaz, O.: Understanding Web augmentation. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 79–80. Springer, Heidelberg (2012)
4. Diaz, O., De Sosa, J., Trujillo, S.: Activity fragmentation in the Web: empowering users to support their own webflows. In: Hypertext, pp. 69–78 (2013)
5. Díaz, O., Arellano, C., Iturrioz, J.: Interfaces for Scripting: Making Greasemonkey Scripts Resilient to Website Upgrades. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 233–247. Springer, Heidelberg (2010)
6. Firmenich, S., Rossi, G., Winckler, M.: A Domain Specific Language for Orchestrating User Tasks Whilst Navigation Web Sites. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 224–232. Springer, Heidelberg (2013)

7. Firmenich, S., Winckler, M., Rossi, G., Gordillo, S.: A crowdsourced approach for concern-sensitive integration of information across the web. JWE 10(4), 289–315 (2011)
8. Garrigós, I., Wimmer, M., Mazón, J.-N.: Weaving Aspect-Orientation into Web Modeling Languages. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 117–132. Springer, Heidelberg (2013)
9. Gaubatz, P., Zdun, U.: UML2 Profile and Model-Driven Approach for Supporting System Integration and Adaptation of Web Data Mashups. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 81–92. Springer, Heidelberg (2012)
10. Gómez, J., Cachero, C., Pastor, O.: Extending a Conceptual Modelling Approach to Web Application Design. In: Wangler, B., Bergman, L. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 79–93. Springer, Heidelberg (2000)
11. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Comput. Program. 72(1-2), 31–39 (2008)
12. Koch, N., Kraus, A., Zhang, G., Baumeister, H.: UML-Based Web Engineering - An Approach Based on Standards. In: Web Engineering, pp. 157–191 (2008)
13. Li, J., Gupta, A., Arvid, J., Borretzen, B., Conradi, R.: The empirical studies on quality benefits of reusing software components. In: COMPSAC, pp. 399–402 (2007)
14. Martinie, C., Palanque, P., Winckler, M.: Structuring and composition mechanisms to address scalability issues in task models. In: Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., Winckler, M. (eds.) INTERACT 2011, Part III. LNCS, vol. 6948, pp. 589–609. Springer, Heidelberg (2011)
15. Object Management Group. Unified Modeling Language (UML), Superstructure, Version 2.4.1 (2011), `http://www.omg.org/spec/UML/2.4.1`
16. Object Management Group. Semantics of a Foundational Subset for Executable UML Models (fUML), Version 1.0 (2011), `http://www.omg.org/spec/FUML/1.0`
17. Rossi, G., Schwabe, D., Lyardet, F.: Abstraction and Reuse Mechanisms in Web Application Models. In: Mayr, H.C., Liddle, S.W., Thalheim, B. (eds.) ER Workshops 2000. LNCS, vol. 1921, p. 76. Springer, Heidelberg (2000)
18. Selic, B.: A Systematic Approach to Domain-Specific Language Design Using UML. In: ISORC, pp. 2–9 (2007)
19. Van Deursen, A., Visser, E., Warmer, J.: Model-driven software evolution: A research agenda. In: Workshop on Model-Driven Software Evolution (2007)
20. Wimmer, M.: A semi-automatic approach for bridging DSMLs with UML. IJWIS 5(3), 372–404 (2009)
21. Yu, J., Benatallah, B., Casati, F., Florian, D.: Understanding mashup development. IEEE Internet Computing 12(5), 44–52 (2008)