

Process Flexibility through Customizable Activities: a Mashup-based Approach

Marco Fisichella ^{#1}, Maristella Matera ^{*2}

#L3S Research Center

Appelstrae 9a, 30161 Hannover, Germany

¹fisichella@L3S.de

**Politecnico di Milano, DEI*

P.zza Leonardo Da Vinci, 32, 20133 Milano, Italy

²matera@elet.polimi.it

Abstract—In several contexts, the success of Workflow applications is limited by the excess of rigidity of workflow enactment, which leaves no freedom to end-users to “customize” processes and process activities to their specific needs. In this paper, we capitalize on a reference model for flexible processes definition and execution that effectively supports the dynamic, user-based management of collaboration processes, and propose an extension based on mashup technologies for accommodating the needs of process actors to self-define and customize their process activities. The resulting approach especially aims at supporting the flexibility required by collaborative processes, where the dynamics through which peers collaborate and coordinate can be difficult to predict in advance.

I. INTRODUCTION

Very often, the intrinsic complexity of Workflow Management Systems (WfMSs), and especially the excess of rigidity of workflow enactment leave no freedom to end-users (i.e., the involved process actors) to personalize processes and process activities to their specific needs. This is especially true in all those organizational contexts where collaboration within teams requires people to coordinate by means of light-weight processes that allow them to reach a given goal [1]. Such processes exhibit an explosive number of alternatives that depend on the specific needs of the involved actors and that refer both to the coordination flow of the different activities, as well as to single activities, which actors might want to configure and customize by themselves. In these situations, processes therefore escape the ability of being fully modeled at design time. Besides the intrinsic need of customization, there is also an always increasing trend in democratizing software development, especially in the Web domain, allowing users to become co-creators of their applications [2]. To respond to the previous needs, in this paper, we capitalize on a reference model for flexible processes definition and execution, COOPER [3], which effectively supports the dynamic, user-based management of collaboration processes, and propose new abstractions and mechanisms that further accommodate process flexibility and customization. In this paper we go beyond this flexibility, and propose an approach that promotes the reuse of ready to use Web resources to facilitate: i) at design time, the definition of new activities, and ii) at execution time, the user-based customization of instantiated tasks.

Especially considering that software (and Web) development is increasingly moving toward the possibility offered to users to mash-up their own applications [4], our approach enables a mashup-wise composition of process activities through the reuse of ready to use resources. The same mash-up paradigm is also offered to the end users to customize their activities at execution time. The proposed mechanism for the mash-up of user activities is grounded on our previous experience on the development of mashup environments, and its feasibility is therefore proved through the adoption of a specific tool for mashup creation, Mixup [5]. The approach however is general in its nature, being it aimed especially at opening the COOPER platform toward the integration of external resources and environments for Web resource creation.

A. Motivating Scenario and Paper Organization

To motivate the need of process customization through mashups, let us consider a scenario in which a team leader wants to suggest to team components some events, e.g., conferences to attend. For this purpose he wants to define a process composed by three activities: (1) the selection of the conference and the hotel booking by a team component (*Conference&Hotels activity*), (2) the approval of the choice by the team leader (*ViewChoice activity*), (3) the cash check and the allocation of the requested amount by the treasurer (*Authorize activity*). The first activity is in a sequence with the parallel of the last two ones. In COOPER, the process definition is supported by a Web interface, which asks the designer to fill in forms to configure each activity’s properties and define the process flow. The activity configuration in particular consists of selecting suitable process activity from an activity library and instantiate them by configuring activity properties (e.g., deadline and enrolled users). Suppose that the activity library does not include any activity for the listing of conferences and the retrieval of hotels, and that a mashup integrating ready-to-use services would adequately respond to this need. The definition of the process would therefore succeed if the mashup could be easily integrated within the process task. The mashup would in turn encapsulate four component services: the service *Find a Conference*, supplying

the list of the conferences indexed by DBWorld¹, the service *Hotel Retrieve*², supplying a list of hotels for a given place of interest, the Flickr.NET component, to display the images for a given hotel, and Google Maps to show the map for a given address or point of interest. An example of such a mashup is shown in Fig. 1b; to define it, the process designer can invoke a mashup maker tool. Throughout the paper, we will show for example how the Mixup tool [5] can serve this purpose and how the resulting mashup can be integrated within processes managed through the COOPER platform.

Section II describes the COOPER approach for process flexibility. Section III describes the mechanisms to open the COOPER platform towards external services, and in particular toward the instantiation of process activities through mashups. Section IV describes the integration layer in charge of managing the interplay between the process and the mashup engine. Section V reports the results of a preliminary user study. Finally, section VI and VII summarize the main related works and draw our conclusions.

II. THE COOPER REFERENCE MODEL FOR FLEXIBLE PROCESSES

COOPER is a collaborative, open environment on the Web that leverages on the idea of flexible, user-centric process support. Its most salient feature is that it allows cooperating team members (i.e., the platform end-users) to dynamically define collaborative processes, on the basis of the team's preferred procedures, and easily modify, even at execution time, the planned processes. Guiding inexperienced users to define and/or modify processes necessitates that the system be aware of the semantics of the domain where processes must be executed. Such awareness can be achieved through libraries of pre-defined *activity types*, able to support the tasks that users might need to coordinate and execute in a given context.

Activity types represent the definition of process tasks that are regularly performed by users to collaborate. Their definition implies associating the underlying task with a hypertext front-end, which is used as user interface for the execution of the activity. An activity type can be therefore defined as a tuple, $T = \langle Name, HT, P \rangle$, where:

- *Name* is the name of the domain-specific activity type;
- *HT* is the activity types hypertext front-end, which is used as user interface for the execution of the activity, each time an instance of an activity type is executed.
- $P = p_i$ is a set of activity-specific properties, such as description, deadline, etc., to be configured during the process definition when the activity type is instantiated. Each activity type has a very own set of properties.

The definition of a process thus requires the instantiation of a set of activity types by means of user activities. A user activity therefore corresponds to a task to be performed by a user during process execution, and is defined as: $AU = \langle Type, Name, PV, State \rangle$, where:

- *Type* is the name of the instantiated activity type;
- *Name* is the name assigned to the activity during process definition;
- $PV = \langle p_i, v_i \rangle$ are the values v_i for each property p_i ;
- *State* is the state of the activity during process execution, i.e., created, ready, running, blocked, and completed.

The instantiation of a user activity also involves the association with the user(s) that will execute it and with possible resources, i.e., an instance of data (e.g., a database tuple or a file) that can be used by one or more activities for inspection or manipulation. Starting from a library of activity types, the system thus guides the composition of sound, well-structured process models [6]. In [3] the authors show how the modeling constructs offered for process definition also guarantee the semantically correct execution and termination of process instances, and the possibility to easily (flexibly) modify processes even during runtime.

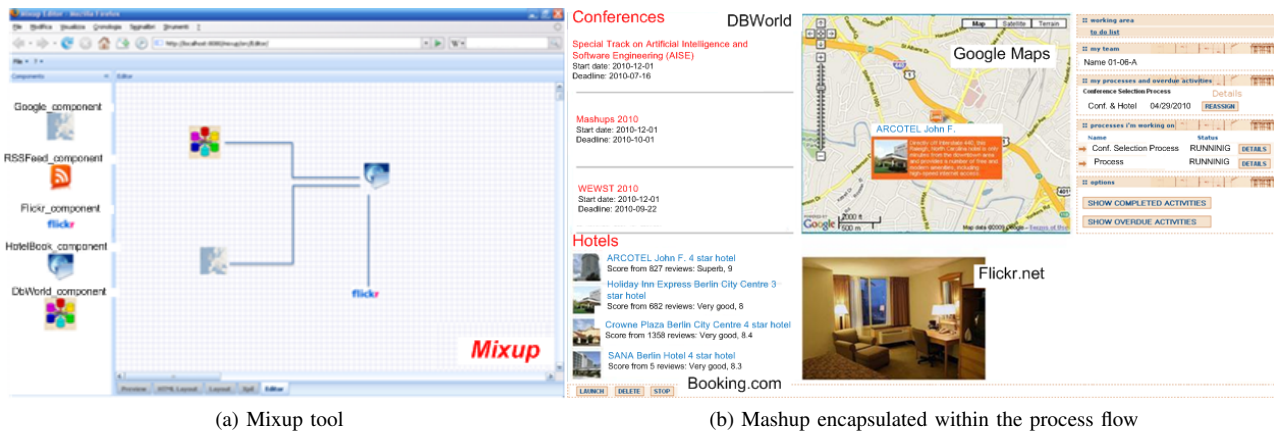
Further support to process definition and modification by the final process actors is given by the availability of process templates, i.e., process models that include the possibility of tagging some of the activities as mandatory. The process designers can define processes starting from templates, by fulfilling the requirement that mandatory activities must be executed in any legal enactment of the process (hence, for example, they cannot be directly or recursively included within disjunctive steps). When a template is copied into a process, the process inherits these constraints. Moreover, if mandatory activities are related by a precedence relationship, then that precedence relationship must be preserved when the process is modified. Setting an activity as mandatory is a choice of the template designer. Fig. 4 illustrates how the process model is represented at data level. The process data, both definition and execution metadata, are: the actors involved into the process possibly clustered in groups (*User Model*); the process composition, as defined by process designer in templates and/or by end-users when extending templates at runtime, plus some data to control the process execution (*Process/template Models*); the activity type library and the resource repository and their relationships with process tasks are as well represented. The figure also represents the metadata extensions needed to manage the integration with mashups, as described in the following sections.

III. MASHING-UP USER ACTIVITIES

Despite the advantages that activity types introduce for the definition of flexible processes, the development of activity types itself, as proposed by the original COOPER approach, is a time-consuming aspect, which requires expert Web developers (e.g., the platform administrators) to produce and integrate the required Web pages within the collaboration environment front-end and release a new version of the platform. Providing libraries of activity types able to cover all the possible (and unforeseen) needs of end-users is also difficult. The need arises for easy mechanisms to extend the library. One possibility is to enable the integration of detached services, to respond to the current trend of involving the users in the construction of

¹<http://zfs.informatik.rth-aachen.de/dbworld>

²<http://www.booking.com>



(a) Mixup tool

(b) Mashup encapsulated within the process flow

Fig. 1: Example of mashup for conference selection defined through the Mixup tool and encapsulated within the process flow.

situational applications by means of mashup technologies [4]. By combining heterogeneous resources, especially if supported by easy-to-use composition mechanisms, the mashup paradigm can indeed favor the on-the-fly construction of applications that quickly allow users to have insights and make decisions based on the resulting combined data. Within the COOPER framework, mashups can certainly facilitate the definition and integration of new activity types by the platform administrator. However, the most tangible advantage is that mashups can enable process designers to create on-the-fly new user activities during process definition, as well as process actors to easily customize already defined activities during process execution, by integrating their preferred services and resources.

A. Container Activities

To achieve the goal of integrating external services, and in particular mashups, for the definition of activity types and user activities, we extend the COOPER activity type library with a new activity type, the container activity (CA), based on two main concepts: (1) as any other activity type in the model, it exposes some properties that are required for its instantiation as a user activity within a process definition (e.g., start and end time, enrolled process actors, etc.); (2) it is a generic container, aimed at encapsulating resources that are defined and managed outside the COOPER collaboration environment. In this paper we particularly concentrate on the integration of composite, mashup-based applications; however, more generally, a container activity provides an execution framework for any external resource that can support the accomplishment of the user task. The *HT* element of a container activity indeed identifies a standard layout template managed by the COOPER process engine that, as can be grasped in Fig. 1b, consists of a *<div>* for the execution of the external resource plus a frame to manage the display of process-oriented commands for the activity suspension or termination (i.e., the Suspend and Finish buttons respectively). While defining a process, CA instantiations can be used when the ready-to-use activity types do not match the requirements of the user tasks and the process designer wants to define a new service and encapsulate it within the container. Different types of resources can be encapsulated by the activity designer and

their execution might require different mechanisms, which are reflected by the three optional properties: (1) the integration of a Web page that shows useful contents. He therefore specifies a *URI* that will be then used at execution time to fill the *<div>* space with the corresponding page. This case also covers, for example, the inclusion of mashups created through Yahoo! Pipes, which in the end generates HTML pages that do not require specific execution environments; (2) the integration some external service/API whose execution within the CA layout requires wrappers (for example a .js script); he therefore instantiates the *Wrapper* property; (3) the definition a composite mashup. In this case, he composes the mashup using Mixup; the *Descriptors* attribute will therefore point to the mashup descriptors that, as explained in the following section, are generated by the Mixup editor and used by the Mixup engine to manage the mashup execution.

The CA instance can be therefore characterized as $MCA = \langle Name, PV, Wrapper, URI, Descriptors, State \rangle$, where:

- *Name* is the name of the mashup;
- $PV = \langle p_i, v_i \rangle$ are the values v_i for each property p_i ;
- *URI* (optional) specifies the location of the executable code of an external resource;
- *Wrapper* (optional) specifies the location of a wrapper needed for the execution of the external service;
- *Descriptors* (optional) specifies pointers to the descriptors of an internal mashup to be executed with Mixup;
- *State* activity status during process execution.

Leveraging the flexibility of COOPER processes, the encapsulation of the resources supporting the activity execution can even be delayed at run-time: the process designer just configures an MCA template, setting the properties that are necessary for the activity enactment, e.g., the enrolled user, the start time and so on, and also encapsulating resources. The process actor can then modify such a template at execution time, adding new resources or substituting existing ones.

B. Mashup Definition

For the definition of mashups, we adopt Mixup [5], a mashup tool that supports integration at the presentation level, that is, components integration by combining their presentation front-ends, rather than their application logic or

data. According to this integration logic, communication and synchronization among component services mainly consists of event notifications by one or more components (*publishers*), which trigger operations into other components (*subscribers*), causing a change in their state. Each component is therefore characterized by a model that describes events and operations.

The publisher/subscriber relationships binding events and operations are specified via event listeners in a composition model. Each listener defines an event publisher, event type, event subscriber, and an operation of the subscribing component. A visual editor (see Fig. 1a) supports the specification of listeners by simply dragging&dropping components in a canvas, and by drawing visual connections among components to specify the UI integration logic. The visual schema is then translated into an XML-based descriptor. Fig. 2 show the composition model of our example. We want that when the user selects a conference from the conference listing component, the hotel listing component displays all the hotels nearby the conference venue, the image displayer shows an image of one selected hotel, while the map will display its location. As specified in Fig. 2 this corresponds to define three event listeners: the first one connects the *ConferenceSelectionChanged* event from *conferenceListing* to the search operation in *hotelListing*; the second one links the *HotelSelectionChanged* event from the *hotelListing* component to the the search operation in the *imageDisplayer* component, and the last one links the *HotelSelectionChanged* event from the *hotelListing* component to the *showPOI* operation of Google Maps.

The component and composition models are used by the Mixup engine (a client-side JavaScript module) to manage the mashup execution. The use of description models in Mixup is in line with the self-describing style of COOPER processes, because from descriptors it is easy to extract metadata useful for the integration of mashups within processes. Descriptors also give us the advantage of easily managing, by means of model extensions (i.e., using the new attributes *<output>* and *<policy>* later described), some additional features that are required for integrating mashups as process activities.

C. Process-Oriented Extensions of the Mashup Model

1) **Management of process resources:** in order to ensure the flow of data and resources, a new attribute *<output>* is associated with listeners to specify the variables of the publisher component that must be monitored as producers of process resources for the next activities. For example, in Fig. 2 the selected conference and hotel are marked as output variables: they are indeed to be passed to the next activities, *ViewChoice* and *Authorize*. The addition of the *output* attribute implied the extension of the Mixup execution engine with wrapping mechanisms able to capture from the mashup execution such process variables and store their values into a buffer shared with the process execution engine. In particular, when a listener event is recognized, the mashup engine monitors modifications of the output variables and stores their values in local data structures that are then synchronized with the process metadata once the activity is completed.



Fig. 2: Composition model for the reference example depicted in 1b. Extensions for flexible process management are highlighted by rectangles.

2) **Modification policies:** to further enhance flexibility, we devised a mechanism so that at runtime enrolled actors can modify an MCA. In line with the abstractions already defined in COOPER for dynamic process modification, in this case the mashup definition bound to the MCA works as a template, holding the indication about what can be changed and what must be kept unvaried. This gives some guarantees about the validity of the process (e.g., concerning the production of data and resources needed for the process to advance), even when the end-user introduces some changes. The composition model is thus enriched with modification policy attributes that specify if listeners are optional or mandatory. Through them, the mashup designer has the possibility to specify which listeners must be kept unvaried for the process activity to be executed correctly in the workflow. This case is expressed by the clause *mandatory* on the listener involving such elements. All the components associated to a listener tagged as *optional* can be instead omitted or modified, including the publisher or the subscriber service, allowing the process actor to replace optional services with his preferred ones. The Mixup editor has been properly extended to interpreting and managing such modification policies. Fig. 2 shows an example of use of modification policies. For example the listeners *conferenceChangedHotelListener* and *hotelChangedMapListener* are mandatory. The involved components, their synchronization and the output they are able to produce are indeed essentials for finalizing the conference selection activity, so that the treasurer can issue a financial authorization; therefore they cannot be altered by any modification. The *HotelChangedImgListener* is instead optional: the process actors executing the activity can therefore modify the mashup. In particular, he can drop or replace the *imageDisplayer* component. The logic under an *optional* policy attribute is that the subscriber component can

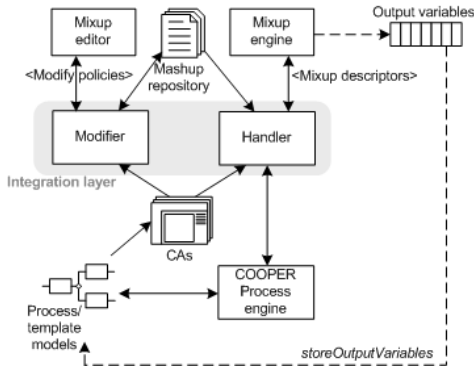


Fig. 3: The integrated framework.

be eliminated. As far as the publisher is concerned, it depends on the existence of other listeners involving it: the existence of any other mandatory listener inhibits its elimination or modification.

3) **Embedding mashups within the process visual environment:** the composition model itself does not define any layout mechanism, but supports the notion of external layout managers, so that existing layouts can be easily reused. Layout information may be specified in a *<layout>* element (see Fig. 2). When the mashup is executed within a process, the layout tag is not interpreted directly by the mashup engine; instead it is passed at runtime to a specific module situated in a middleware layer in charge of managing the integration of mashup layout within the CA HT. The next section describes such layer.

IV. INTEGRATION LAYER

The interplay between the process execution engine and the mashup engine is achieved thanks to an integration layer, in charge of activating: (i) the Mixup editor, when an MCA mashup must be defined during process design or customized during process execution, and (ii) the Mixup execution engine, when an MCA mashup must be executed within a user activity. This integration is based on two modules, *Handler* and *Modifier*. As shown in Fig. 3), when an MCA occurs, the Handler module (a Java servlet) queries the internal repository looking for the mashup associated with the current MCA. In particular, the Handler checks the current activity and retrieves the elements required for executing the associated resources. In case of external resources, whose execution is based on a *URI* or a *Wrapper*, the Handler embeds such elements within the *<div>* of the CA HT dedicated to the execution and display of such resources. In case of a Mixup-based mashup, the Handler retrieves the *Descriptors* from the mashup repository and passes them to the Mixup engine for execution.

Handler manages the construction of the composite layout given by the integration of the CA HT plus the mashup presentation layer. Handler also manages the storing of process variables and resources that are manipulated by the MCA and that need to be preserved for successive activities; this action is indicated in Fig. 3) with a dashed arrow starting from the Handler and terminating into the process model. For example, it provides support to the application client for all the operations for saving and updating the process resources tagged as *output* in the composition model and maintained

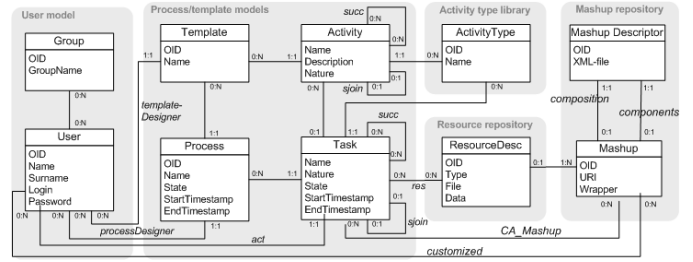


Fig. 4: Schema of process metadata.

in the resource buffer local to the mashup engine. When the process actor terminates the activity, the values of process variables are stored as process resources within the process engine. Finally, it also updates the activity status.

Modifier is then in charge of invoking the Mixup editor when a new mashup has to be defined within a CA, or when the user wants to modify a mashup during process execution. In order to interpret and manage the modification policies, the Mixup editor has been purposely extended. When the mashup model is finalized, *Modifier* then uploads or updates the mashup descriptors into the process metadata, and notifies the *Handler* about the change. *Handler* will then take care of reloading the new descriptor, thus providing the users with their own customized version of the mashup.

The process metadata presented in Fig. 4, allowed us to keep the self-describing nature of processes, even for representing flexibility and customization requirements at the level of container activities: in the *Process Model* a CA is represented as an activity type; a *Mashup Repository* maintains the mashups associated with the MCAs included within processes (*Mashup* entity), as well as their descriptors (*Mashup Descriptor* entity) connected to the *Mashup* entity through *components* and *composition* relationships; customized mashups are associated with the users that defined the customization during process execution (*customized* relationship); the relationship between *Mashup* and *Resource Repository* identifies possible resources to be produced as activity outputs.

V. VALIDATION

We conducted a preliminary user-based evaluation for assessing the effectiveness of the COOPER reference model. Results showed that users get benefit from process flexibility [3]. We then performed a further user study to assess the acceptability by users of the new mashup-based functionalities. Our assessment was conducted within the L3S. Several research projects require L3S people to work within geographically distributed teams. A prototype of our system was deployed at the institute, and 50 people (Ph.D. students and Post Docs) agreed to use it to organize their collaboration activities. The knowledge level of the involved users about mashups and groupware usage was medium. A tutorial on how to use the extended COOPER platform was published on line. Users were allowed to use the prototype for one month; we in particular asked them to try to define processes. Then, a questionnaire was submitted to the users after this phase, to collect their feedbacks about the productivity impact, the ease of use and the overall satisfaction. The usage logs show that users were able to define processes and mashup-

based activities. They were also able to propose different process instantiations. In particular, really attractive was a design process for paper submission as an extension of the process presented in our scenario. The extension was possible thanks to a new MCA integrating: (1) a list of conferences of interest; (2) the own collection of papers using a CVS. This process instantiation is really interesting, since it shows that the mashup approach facilitates a new form of collaborative coordinated content syndication, easing the integration of Web resources into document-driven open collaboration processes.

The users feedback collected are also very encouraging:

- 48 users out of 50 answered that our platform would enable a high productivity improvement.
- 84% of the users rated the difficulty on using such a system from very low to medium, with a pick of 52% on low;
- 80% of the respondents were highly or very highly satisfied; the other users expressed a medium satisfaction level. Further the reuse of activities was considered as strongly positive by the majority of them.

VI. RELATED WORK

The most notable industrial efforts so far devoted to enhance process flexibility have been done in the field of groupware. Such applications provide a set of hardwired collaboration activities, and the extension of this set to respond to unanticipated collaboration needs is difficult or even impossible [7].

Some research approaches have so far dealt with the management of dynamic and flexible processes, focusing on the ability of the process to be (partially) sketched at design time, not completely specified until runtime, and modified during its execution [8], [7], [9]. Based on workflow technologies, on one hand such approaches ensure flexibility; on the other hand they require users to master concepts related to process design. Also, flexibility covers limited features, for example the relaxation of time constraints, and does not address at all the needs of process designers and, especially, of final process actors to self define or customize the process tasks. The possibility to define process tasks by exploiting ready to use services, and customize such tasks even at run-time is to our knowledge still unexplored. A preliminary attempt of enabling process actors to integrate mashup for the execution of process tasks is described in [10]. This work proposes LISL, a design language for creating personal learning environments through the composition of Web tools services into mashups. While the aim of the work is similar to ours, the context is quite different, since the resulting personalized activities are not integrated within processes. Also, the mashup definition is based on a scripting language, which in our opinion cannot be easily managed by end users and in some cases can prevent users from defining mashups. Our notion of flexibility and customization is broader with respect to the previous works. First of all, the notion of activity types enables users to play the role of process designers, allowing them to modify at runtime pre-defined templates and also to define completely new processes. Furthermore, the integration with mashups introduces flexibility and customization also at the level of

user activities. The approach proposed in this paper also tries to respond to the recent challenge of enabling end users, not directly interested in software development, to modify and extend the systems they use according to their needs [6].

VII. DISCUSSION AND OUTLOOK

The framework illustrated in this paper is characterized by the extensive use of description models, both on the process and the mashup side, which greatly facilitated the integration between the process and the mashup environment, also allowing for a fine granularity (at service level) of possible user-based customizations. Nevertheless, we also experimented the integration between the process platform and other environments for mashup definition and execution. The integration of mashups gathered from an external composition tool (as for example Yahoo! Pipes) is trivial: as for any external Web resource - even a Web page, it consists of linking a CA instance with the external *URI* of the resource. For such external resources, we however experienced some difficulties in the run-time customization of activities: the customization granularity indeed decreases, at least until we do not want the activity designer to manipulate its code (where possible). AS future work we plan to further improve the mashup definition, for example through the integration of a new visual editor we are working on, which greatly facilitates the mashup development, thanks to a WYSIWIG paradigm sustaining end-user programming and some mechanisms for automatic generation of listeners. Besides that, our future work will be devoted to improve the sharing of mashup-based process activities, also through the collaborative definition of activity types. Another direction is towards mechanisms for easing the definition of new mashup components.

ACKNOWLEDGMENT

This research has been co-funded by the European Commission within the eContentplus targeted project OpenScout, grant ECP 2008 EDU 428016.

REFERENCES

- [1] T. Nodenot, C. Marquesuzaà, P. Laforcade, and C. Sallaberry, "Model based engineering of learning situations for adaptive web based educational systems," in *WWW*, 2004, pp. 94–103.
- [2] J. Cullen, "Democratizing innovation," *JASIST*, vol. 57, 2006.
- [3] S. Ceri, F. Daniel, M. Matera, and A. Raffio, "Providing flexible process support to project-centered learning," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 6, pp. 894–909, 2009.
- [4] A. Jhingran, "Enterprise information mashups: Integrating information, simply," in *VLDB*, 2006, pp. 3–4.
- [5] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera, "A framework for rapid integration of presentation components," in *WWW*, 2007, pp. 923–932.
- [6] G. Fischer, "End-user development and meta-design: Foundations for cultures of participation," in *IS-EUD*, 2009, pp. 3–14.
- [7] F. Charoy, A. Guabtni, and M. V. Faura, "A dynamic workflow management system for coordination of cooperative activities," in *Business Process Management Workshops*, 2006, pp. 205–216.
- [8] S. W. Sadiq and M. E. Orlowska, "Pockets of flexibility in workflow specification," in *ER*, ser. Lecture Notes in Computer Science, H. S. Kunii, S. Jajodia, and A. Sølvberg, Eds., vol. 2224. Springer, 2001.
- [9] C. W. Günther, M. Reichert, and W. M. P. van der Aalst, "Supporting flexible processes with adaptive workflow and case handling," in *WETICE*. IEEE Computer Society, 2008, pp. 229–234.
- [10] F. Mödrtscher and F. Wild, "Sharing good practice through mash-up personal learning environments," in *ICWL*, 2009, pp. 245–254.