Dealing with Collaborative Tasks in Process Mashups

Victoria Torres
Centro de Investigación en Métodos de Producción de Software
Universitat Politècnica de València
46022 Valencia, Spain
vtorres@pros.upv.es

Jose Manuel Pérez
Instituto Valenciano de Vivienda
Generalitat Valenciana
46001 Valencia, Spain
perez_jmaram@gva.es

Agnes Koschmider
Institute of Applied Informatics and Formal Description Methods
Universität Karlsruhe
Karlsruhe, Germany
agnes.koschmider@aifb.uni-karlsruhe.de

Florian Daniel
University of Trento
38126 Povo (TN), Italy
daniel@disi.unitn.it

ABSTRACT

The potential that mashups can reach in web applications has not yet been exploited in practice. In fact, many of the challenges that introduce some of the most advanced types of mashups are not yet solved and require new mechanisms that allow their proper specification and execution. Among the different types of identified mashups, in this work we focus on *process mashups*, a type of mashups where the integration and coordination of people, tasks, services and UIs is required. Specifically, from the set of characteristics found in this type of mashups we focus on the collaborative aspect found in process tasks. To deal with it we provide a modeling solution that extends a business process modeling language such as BPMN to represent such characteristic. The solution has been defined within the context of OOWS4BP, a model-driven engineering approach to deal with the development of business process-driven web applications.

Categories and Subject Descriptors

D.2.1 [**Requirements/Specifications**]: Languages.

General Terms

Design, Languages

Keywords

Mashups, Collaborative work, Web applications, Model driven engineering.

1. INTRODUCTION

Mashups have gained high popularity in the last years. One reason for this approval is the availability of user-friendly, visual mashup tools for the creation of web applications with low technical and programming knowledge. The simple creation of mashups attracts a high user forum, however, the richness of mashups remains not exploited. In fact, mashups are usually simple applications with one page and a limited navigation. But mashups can be more than a "simple" web application. Also the business domain can gain benefits of advanced features of mashups. Sophisticated mashups allow enterprises to compose heterogeneous resources in a lightweight way.

Recently, the concept of a *process mashups* has emerged as a more sophisticated form of mashups. Daniel et al. [17] describe a *process mashup* through the junction of the three dimensions called *user*, *page* and *workflow*. According to their definition, a *process mashup* should allow concurrent work of multiple users, incorporate multiple pages, and provide workflow support (specifying control and data flow over human tasks). Even though the concept of a *process mashup* shows promise, several challenges remain to be solved in order to offer tool support for them.

In this work, we specifically focus on collaboration in *process mashups*, which requires a broader conception of collaborative work than the one usually conceived. In particular we want to specify and build *process mashups* where different users work together not just at the process level (as common business process modeling languages support) but also at the task level. That is, we strive for an approach in which users share individual tasks and perform them concurrently. Even though all the involved users are required to complete the task, the role played in it by each user can differ significantly (e.g. one role can behave as supervisor while another one can behave as performer). This differentiation in behavior can determine the type of task view required by each user during task execution.

We base our proposal on the model-driven approach developed in [7] and present an extension to it that deals with the modeling requirements that introduce *process mashups*. In the current approach, a *process mashup* is designed starting from a process model, which, after being complemented with other models, is translated into executable code. The main advantage of this model-driven approach is that it allows us to separate the system specification from the technology used to implement such systems. Currently, mashups are built based on specific data and functionality provided by third parties which implies that the system is highly tied to a specific provider. While this approach speeds up the combination of different sources in the short term, evolving such systems becomes difficult since the system is expressed in terms of a specific partner.

However, the suggested model-driven generation of *process mashups* based on process models requires an extension of common, available process modeling languages. In this paper we will show how to apply our approach for the Business Process Modeling Notation (BPMN), which is commonly used. In particular, the BPMN standard lacks of concept support for the coordination of

Mashups 2011, September 2011, Lugano, Switzerland
Copyright 2011 ACM 978-1-4503-0823-6/11/09…$10.00.

actors of shared tasks. Even though some approaches discuss extensions to the notation [10] these just deal with this aspect superficially, focusing mainly on the graphical representation and not on the underlying semantics that entails such collaboration. This means that all aspects of the multi-user dimension (of a *process mashup*) cannot be supported yet. This failing calls for appropriate extensions of BPMN that will be proposed in this paper. In the web engineering field we find different proposals (such as OOHDM, UWE, WebML, OOWS4BP, UWAT+ or MIDAS) that successfully deal with the development of web applications supporting business processes execution. However, none of them provides support to the collaborative work required in the *process mashups* applications we are interested in.

The remainder of the paper is structured as follows. First, in section 2 we present a case study that exemplifies the problem being faced in this work. Then, in section 3 we briefly explain the *process mashup* concept based on the three dimensions: user, page, and workflow. In section 4 we make a revision over related works and tools for *process mashup* construction. In section 5 we provide an overview of the model-driven approach suggested for the construction of *process mashups*. Next, in section 6 we focus on the collaborative aspect of *process mashups* and present the modeling solution proposed in this work. Finally, section 7 provides some conclusions and further work.

2. CASE STUDY

The case study presented here refers to the service provided by the Valencian Regional Ministry of Housing (hereafter MoH) to rent flats to its citizens. The BPMN diagram representing such service is depicted in Figure 1. In it we can see that the process is performed mainly by two different roles, which are the operator (an employee from the MoH) and the citizen. However, one of the tasks included in the process (specifically the *flat booking negotiation* task) is performed collaboratively by these two roles and also, if necessary, by a third one. This particularity is represented in the BPMN diagram by means of an annotation attached to the *flat booking negotiation* task due to the lack of support found in the notation.

The first step in the process refers to the system citizen registration. In it, the citizen provides, by means of the e-government web application, her personal data (i.e. name, birth date, nationality or annual income) as well as information related to flat preferences (i.e. location or size).

Then, the citizen can either perform the search of the flat by s/her own or decide to delegate such search to an operator from the regional ministry. To perform such search the citizen is provided with a UI that integrates data and functionality coming from different sources. For instance, in addition to a flat textual description that is kept in the organization databases, the citizen is provided with (1) a map showing the location of the flat (provided by the Google maps service), (2) a set of pictures from the flat surrounding area (provided by the Flickr service), (3) a translator service to translate the flat description into different languages (e.g. provided by the Google translate service), and (4) a currency converter (e.g. provided by Exchange Rate).

When the flat search and selection is delegated to the operator, s/he is provided with a different UI to perform such task. The type of differences found in this UI refers to the displayed data and also to the provided functionality. On the one hand, the displayed data includes also citizen data (e.g. housing needs or family details) and flat data (e.g. flat owner conditions). On the other hand, the provided functionality includes more advanced filter conditions that allow operators to perform a more detailed flat search.

Once the flat search and selection is completed, the negotiation of the allocation is started. This task is a collaborative task that is performed jointly and at the same time by different users. This means that all the involved users need to participate concurrently in the task to complete it. Usually, the citizen and the operator perform the task jointly. However, when there is a conflict because another citizen has also booked the flat, a third role (the coordinator) comes into play. The collaborative work is achieved by means of UIs that are provided to all the involved users. All these UIs share the same data (all of them visualize the same flat data - pictures and descriptions) and functionality (i.e. a chat service). However, based on the user needs these UIs are adapted by adding extra data and functionality. For instance, the UI provided to the coordinator includes also citizen's sensitive data (e.g. data provided by the Treasury) and functionality (e.g. to reassign the case to another operator) that need to be hidden to both the citizen and the operator.

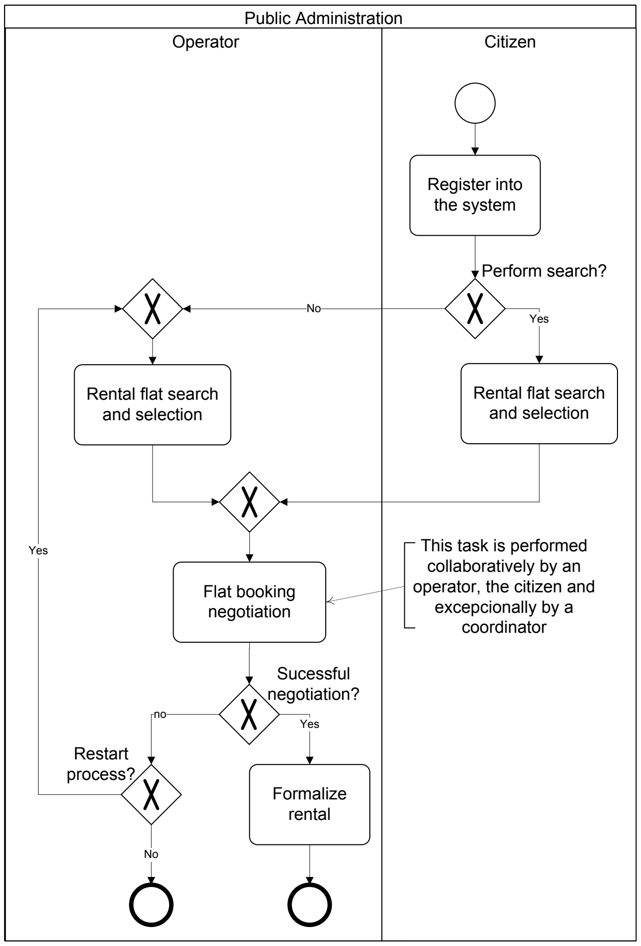

Figure 1 BP diagram for the *flat rental* process

Finally, if the negotiation task is performed successfully the operator finalizes the process by formalizing the rental of the flat by means of a contract. On the contrary, the process can either start again to look for another flat or finish because the coordinator decides that there is no flat availability for the applicant.

After describing the *rental flat* scenario, if we analyze it we observe that to implement the *flat rental* process we need to provide users with UIs that are made up of data and functionality coming from different sources (maps, pictures, text translators, etc.) available on the Web. This is similar to what is done in mashups. In addition, the UIs are driven by a control flow in charge of coordinating the execution of the different steps (tasks) that make up the *flat rental* process. This coordination involves also the different roles taking part in the process and the service invocation that is done to support the process tasks. Another particularity found in the scenario relates to task realization. Specifically, we observe that the *flat booking negotiation* task is performed collaboratively at the same time by more than one role. This collaborative work that can be easily found in many situations in real life cannot be explicitly represented in the system models (we already mentioned that we had to represent such information in the BPMN diagram by means of a task annotation).

Therefore, among the characteristics found in the *flat rental* scenario we observe that the system requires the integration and coordination of tasks, services and people to achieve the flat rental goal. A similar approach that combines disparate services coming from different sources is found in mashup applications. However, it is clear that the scenario here presents new challenges that are not covered by typical mashup applications. In fact, to deal with the coordination required in the scenario a more advanced type of mashup is needed. According to the different types of *process mashups* identified by Daniel et al. in [17], the described scenario corresponds to the most advanced type. In the following section we make a revision over the key ingredients that define a *process mashup* and based on the necessities of the current scenario we state the requirements that should be fulfilled to properly address the specification and execution of applications of this type.

3. PROCESS MASHUPS

Process mashups were first introduced by [18] as an advanced type of mashups where the integration does not only refer to the *data* and *presentation layers* but also to the *business process layer*. However, Daniel et al. in [17] went a step further and considered also the integration of other aspects such as *users* and *pages*. To better understand how to overcome the construction of web applications supporting a scenario similar to the one presented previously, in this section we take the *user, pages* and *workflow* dimensions introduced in [17] to explain the *process mashup* concept and contextualize it in the *flat rental* case study. Then, we state the requirements that should be addressed to properly deal with the specification and execution of *process mashup* applications.

3.1 Multiple users

Giving support to multiple users means allowing two or more users to concurrently operate on the same instance of a mashup application. In the *flat rental* scenario presented before we find this type of collaboration in the *flat booking negotiation* task. There, up to three different roles (citizen, operator and coordinator) work concurrently on the same data to perform this task. The set of provided data comes from the combination of different sources (Google Maps, Flickr and a local service developed and maintained by the MoH) providing flat, citizen and operator details. However, the view of the combined sources can differ depending on the type of user accessing to it. The main reason to provide different views over the same mashup would be to adapt the combined data provided to the user according to her/his needs (needs that are directly related to the responsibility of the user over the collaborative task).

3.1.1 Modeling requirements

In order to properly deal with the multiple users feature found in *process mashups* we need modeling mechanisms that allows us to:

- Req1. Identify the different types of roles involved in the process.
- Req2. Identify the type of participation expected from each type of role (if it requires a collaborative participation or not).
- Req3. Identify the mashup up view[1] required by each type of role according to their task responsibilities. The definition of these views would involve not only hiding some of the components that make up the mashup according to the profile of the involved role but also deciding what type of grant access each type of role has over the mashup components.

3.1.2 Execution requirements

Regarding execution requirements, to ensure the correct execution of multiple users in a *process mashup* we need a technological infrastructure that provides:

- Req4. Access mechanisms that allow adapting the system according to the type of user accessing to it.
- Req5. Concurrent access to the same instance of a mashup when we are dealing with collaborative tasks.

3.2 Multiple pages

Providing support to multiple pages involves organizing the different pieces that make up a mashup into one or more pages. This organization is done usually according to conceptual issues (the details of the combined data could be better organized in different pages) or size issues (the UI gets too overloaded due to amount of combined sources). For instance, the *rental flat search and selection* task combines a set of sources which allows enriching the flat data that is provided to the citizen. These sources refer to a map provided by Google maps and pictures provided by Flickr. In this case, the map and the pictures could be organized in separate pages that would be reached by the user through different hyperlinks.

3.2.1 Modeling requirements

In order to properly deal with the multiple pages feature we need modeling mechanisms that allows us to:

- Req6. Organize mashup components in one or multiple pages.
- Req7. Define a navigational structure that allows users to access the different pages that may support the execution of one task. Note that this is different from the traditional control flow structure that supports "navigation" (progress) between tasks.

3.2.2 Execution requirements

The technological infrastructure needed to ensure that mashup components are executed properly independently of their organization are that:

- Req8. All mashup components can be reached either from the same page or by navigating through a hyperlink.

[1] A mashup view can be defined as the way in which a specific user visualizes and/or interacts with the combined sources that make up the mashup.

- Req9. The state of the mashup is kept while the user navigates through its structure (different pages).

3.3 Workflow

The workflow dimension refers to the coordination of the different elements (people, UIs, data and services) that are involved in a *process mashups* to accomplish a specific goal. In the *flat rental* scenario we can see how different roles (citizen, operator and coordinator) participate in the process by taking responsibilities over some of the tasks. To successfully complete their assigned tasks, each user is provided with a UI that includes all the required data and functionality required in each case. The provided data and functionality can come from the MoH system and/or from external sources, which in turn may be combined to help the user during the execution of the corresponding task.

3.3.1 Modeling requirements

In order to properly deal with the workflow dimension we need modeling mechanisms that allows us to:

- Req10. Specify the control flow that establishes the order in which tasks should be executed as well as the conditions that may diverge or join different process paths.
- Req11. Specify the data flow that needs to be propagated not only between connected tasks but also within the same task (to specify how mashed up UI elements and services are connected).
- Req12. Specify when a collaborative task can be stated as completed (this was clear when the task was performed just by one role but when the work is performed collaboratively this needs to be explicitly defined).
- Req13. Specify the different services that are used and that are combined to support a specific process task.

3.3.2 Execution requirements

Regarding execution requirements, to ensure the correct execution of the workflow in a *process mashup* we need a technological infrastructure that ensures:

- Req14. That all the roles that are required in a collaborative task participate in it.
- Req15. The finalization of a collaborative task.
- Req16. That all the services supporting a specific task are invoked properly during task execution.

To properly deal with the construction of *process mashups* within a model driven approach it is necessary to address all these requirements at both the modeling and execution level. On the one hand at the modeling level by providing the expressiveness required to specify such applications. On the other hand at the execution level to ensure that applications behave as expected. In the following section we focus on the requirements at the modeling level introducing the mechanism that allow specifying such systems.

4. COLLABORATIVE TASKS

In this section we are going to present the mechanisms that have been defined to represent collaborative tasks in *process mashups* at the modeling level. These mechanisms have been defined in the context of OOWS4BP [7], a web engineering approach for the construction of business process-driven web applications. In OOWS4BP, web applications are described by means of a set of models aimed at representing different aspects of the system (e.g. structure, behavior, navigation and presentation). Among these models we find the *business process model* that is used to define all the business processes that need to be supported by the web application. This model is built in terms of a business process modeling language (specifically BPMN) and provides a set of primitives that allows describing processes in terms of tasks, roles in charge of these tasks and finally the sequence that is allowed to complete the process. However, the semantics attached to process tasks refer to work that is performed by a specific type of user (a user that is represented by an organizational role in the BPMN notation). Initially, this expressivity may seem enough. Nevertheless, we can also require, as highlighted in the presented scenario, a more powerful expressiveness to represent that two or more roles participate in the same task at the same time to accomplish the task goal and to progress within the process. Therefore, we need new mechanisms that allow us to represent such collaboration. To deal with the collaborative aspect presented in *process mashups* in this work we introduce a new model, which is the *roles model.* This model allows us to better represent the inner reality of shared tasks identifying not only the set of roles involved in the task but also their different responsibilities.

In addition, the *business process model* is linked to other models to define the behavior and the UI (if required) of each process task. On the one hand, the behavior is defined by associating a process task with an operation that is provided either by the organizational system (represented by an operation defined in the *structural model*) or by an external partner (represented by an operation defined in the *services model*). On the other hand, the UI required by a process task is defined in the *navigational model* associated to the involved role.

Figure 2 depicts the set of models used to represent tasks that are performed jointly by different roles and that required a different type of UI according to their responsibility over the collaborative task.

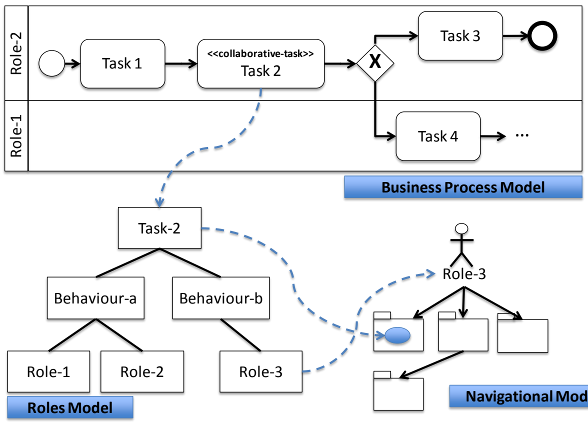

Figure 2 Representing collaborative tasks

In the following subsections we present in more detail each of these models and make explicit how they deal with the modeling requirements stated in section 3.

4.1 Business Process Model

To specify in the *business process model* the existence of tasks that are preformed jointly by two or more roles we added to the task concept a new property named "collaborative". This new property is defined as Boolean and its value determines whether the task is defined or not as collaborative. Graphically, these collaborative tasks can be identified in the diagram by means of the <<collaborative-task>> label that is included in the task graphical element. Even though these tasks can be placed within any lane of the diagram, this will not link them to the corresponding role. Instead, the set of roles in charge of this task are specified separately in the *roles model*, which is explained in the following subsection. Initially, one may think that using directly the expressiveness provided by BPMN would be enough to represent such collaboration (i.e. including a new lane in the diagram that represents all the involved roles and placing the shared task there, replicating the shared task for each of the involved roles or placing the shared task in the diagram in a way that touches all the involved roles). However, these solutions usually introduce complexity into the diagram and also limit us since they do not allow us to specify the additional behavior of the involved roles (which becomes important information to be represented in the model).

Figure 3 shows the business process model for the flat rental case study using the extension previously explained. As this figure shows, the complexity of the diagram has not been increased by the introduced extension. However, it is still very easy to identify which tasks require the collaboration of some roles.

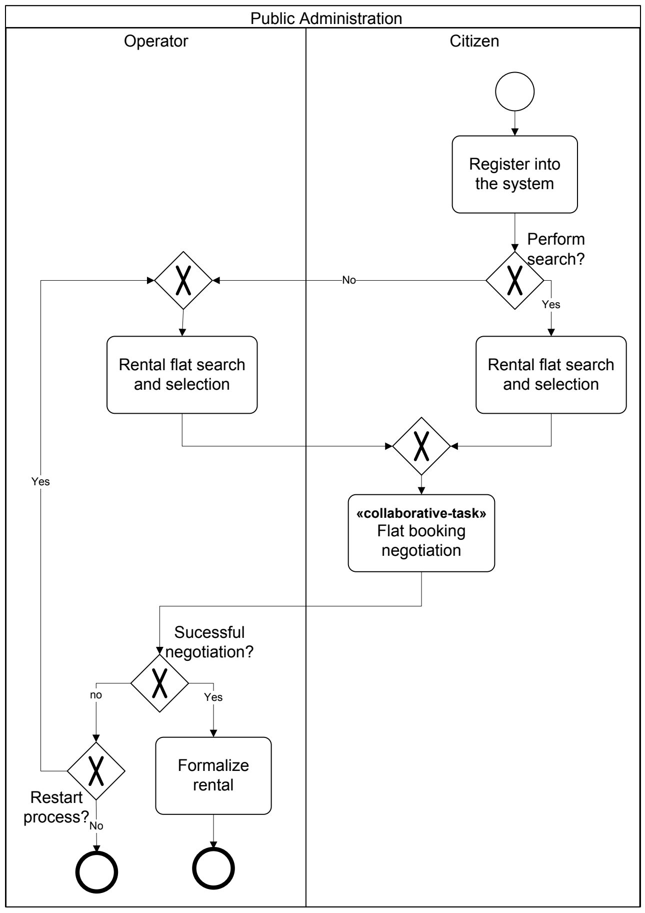

Figure 3 BP diagram for the *flat rental* process

4.1.1 *Support for the modeling process Mashups*

The expressivity provided by the *business process model* allows us to deal with some of the requirements stated in section 3:

- By means of lanes, BPMN allows us to identify the different types of roles involved in a process. Specifically, in this model we specify those roles that do not share responsibilities over individual tasks. The rest of roles that participate collaboratively in some tasks are defined separately in the *roles model* (req.1)
- By means of the flow object elements provided by BPMN (activities, gateways, and events) we can specify the control flow that defines the different paths that can be followed during process execution (req.10)

4.2 Roles Model

With the extension defined in the *business process model* we are only capable of specifying which tasks are going to be performed collaboratively by more than one role. However, we still need to specify which roles are involved in a collaborative task and what type of responsibility they have over it. Therefore, the *roles model* allows us to specify:

(1) The different roles involved in a specific task
(2) The different responsibilities in the task
(3) The optional involvement of a role in a shared task
(4) The role responsible of the task

The *roles model* has been defined as a feature model [19] where the different roles that can participate in a collaborative and shared task are represented as features in the tree. Specifically, the *roles model* has been defined as a three-level tree (see Figure 4) where we specify: the collaborative task (level 1), the different responsibilities identified over the task (level 2), and the set of roles involved in the task and behaving as one of the identified responsibilities (level 3). In addition, associated to the role involvement identified at level 2 we can also specify if such involvements are mandatory or optional. This is depicted graphically with a white circle attached to the association that links such involvement with the task at level 1. The roles model metamodel, both, the *task* and the *role* concepts are imported from the BPMN metamodel. Each task specified in the model as "collaborative" can be associated with one or more responsibilities (cardinality defined between task and task responsibility classes). These responsibilities can be defined either as mandatory or optional. In turn, task responsibilities can be played by one or more different roles. The main reason to introduce the "role involvement" concept at level 2 is because normally, for each different responsibility identified in a task, the associated roles will require a different software support to participate in the task.

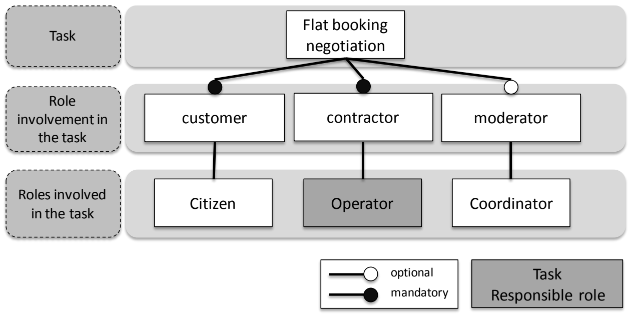

Figure 4 Roles model associated to the *flat booking negotiation* collaborative task

Figure 4 shows the *roles model* for the *flat booking negotiation* collaborative task. In this case, this task is going to be performed by several roles that are going to participate in the task with different responsibilities, as customer, contractor and moderator. In addition, the diagram indicates that while the roles behaving as customer and contractor are mandatorily participating in the task, the roles behaving as moderators are optional in the task. Then, at the bottom level we have three different roles (*citizen*, *operator* and *coordinator*) each one participating with a different involvement (as *customer*, *contractor* and *moderator* respectively). This means that all of them have different responsibilities over the task and different UIs will be required in each case to satisfy their specific needs. Finally, the operator role has been assigned in charge of the task, meaning that this role will be responsible for its completion.

4.2.1 Support for the modeling process mashups

The expressivity provided by the *roles model* addresses some of the requirements stated in section 3 as follows:

- The different roles specified at the bottom level in the tree specify the roles participating collaboratively in a shared task (req.1)
- The different responsibilities identified at level two in the tree allows us to specify the different behavior expected by each of the roles involved in a shared task (req.2)
- By means of the "responsible" attribute attached to the task concept we can specify the role responsible of the task, e.g. the role in charge to complete the task (req.12)

4.3 Navigational Model

Before generating the web application that will give support to the *process mashup*, we still need to specify (1) the UIs that are required by the roles that participate in the process, and (2) the navigation mechanisms that allow users reaching such UIs. In the OOWS4BP approach, this specification is done through the *navigational model.* This model is built in two steps. First of all, an "Authoring-in-the-large" view of the system is constructed (see Figure 6).

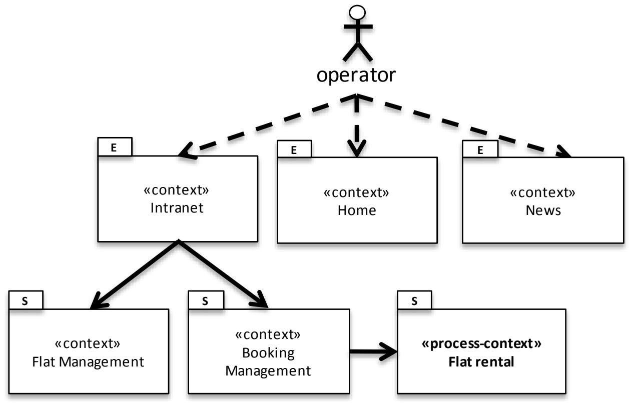

Figure 5 "Authoring-in-the-large" view for the *operator* role

In this view we define part of the navigational structure of the system for each type of role. This navigational structure is made up of interaction units (IUs) that can be defined of two types depending on how these can be reached by the user. On the one hand, exploration IUs (depicted graphically with an "E" label) are always accessible via hyperlinks to the user. On the other hand, sequential IUs (depicted graphically with an "S" label) can only be reached from an exploration IU. Figure 6 shows an excerpt of the navigational model defined for the *operator* role. This model defines direct access to three different IU (intranet, home and news) and sequential access to other three (flat management, booking management and flat rental). One of these sequential IUs (flat rental) represents the entry point to the flat rental process.

Once the "Authoring-in-the-large" view is completed we can start with the "Authoring-in-the-small" view (see Figure 6 and Figure 7), which is used to specify in detail the data and functional view of the system for each type of user. In the OOWS4BP approach we have a specific type of IU to represent the views that relate to process tasks. The particularity of these IUs (which are represented by the "process-context" primitive) is that the navigation followed by users is driven by the process (and not by the own user) to ensure that the user completes her/his assigned tasks. Figure 6 and Figure 7 depict respectively the IUs defined for the *operator* and *citizen* roles for the *flat booking negotiation* task. As these figures show, each IU includes an "activity-container" for each task where the corresponding role participates. The "activity-container" primitive represents the view of the system to accomplish a specific process task. For instance, as Figure 6 and Figure 7 show, these include three different "activity-containers" that refer to the three different tasks where the *operator* and the *citizen* participate in the process (for space constraints only the *flat booking negotiation* activity-container is detailed). In addition, these "activity-containers" are made up of two different types of elements that refer to "main-AIU" or "complementary-AIU". On the one hand, "main-AIUs" are used to include the data and functionality that has to be necessarily included to complete the corresponding task. In collaborative tasks, as it is the case for the *flat booking negotiation* task, only the role defined in the *roles model* as "responsible" of the task will include such "main-AIU" (see Figure 6). On the other hand, "complementary-AIUs" are used to provide roles with data and functionality that is used to facilitate their participation in the task (e.g. providing the *coordinator* with data about citizen house properties or debts).

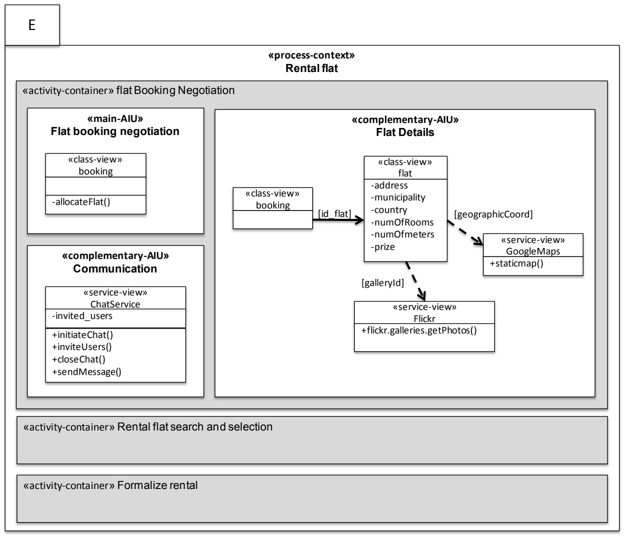

Figure 6 *Operator* role Process-context defined for the *rental flat* process

Then, within a "main-AIU" and "complementary-AIU" we include views over the data and services that have already been defined in the system. This data and functionality can came either from the own system or from external providers. When the data comes from the own system we use the "class-view" primitive which allows us to define the data and functionality associated to a class defined in the local system that we want to make available to the user. On the contrary, when the data or functionality comes from a third party provider we use the "service-view" primitive which allows us defining the data and functionality that we want to use in this specific IU. For instance, in Figure 6 we can see that the "main-AIU" includes a view over the *booking* class. This view makes accessible for the *operator* role the allocateFlat() function which is the functionality that allows indicating that the task has been completed.

The different views that are included either in a "main-AIU" or in a "complementary-AIU" can be connected via relationships which are called "contextual dependency relationships". These can be defined either as "direct" or "indirect". Direct relationships (which are represented by a solid arrow) allow passing data between different views. By doing so, we can retrieve the appropriate data of the connected view. For instance, in Figure 6 there is a direct relationship between the booking "class-view" and the flat "class-view" which is used to pass the identifier of the flat between the connected views. On the other hand, indirect relationships (which are represented by a dotted arrow) specify not only the data passed between views but also that the connected view is reached via a hyperlink. This type of link can be seen in Figure 7. In this case, the photos and the map provided by the external providers Google and Flickr are reached by the user via a hyperlink. By using this type of link we are splitting the mashed up components into different pages that can be reached through the corresponding hyperlink.

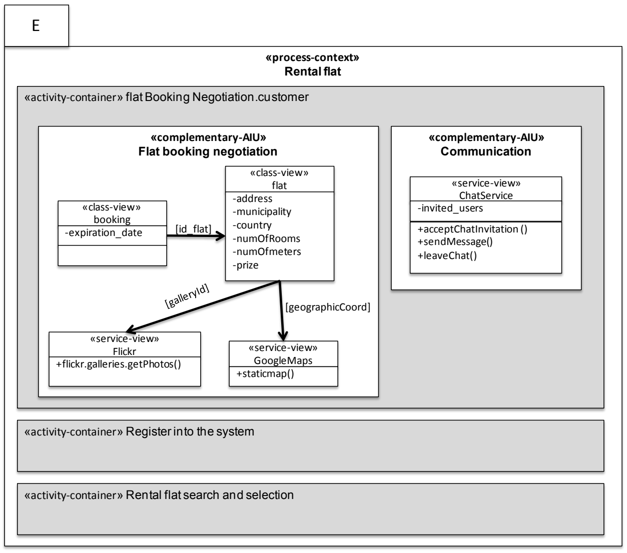

Figure 7 *Citizen* role Process-context defined for the *rental flat* process

4.3.1 *Support for the modeling process Mashups*

The expressivity provided by the *navigational model* addresses some of the requirements stated in section 3 as follows:

- The "complementary-AIU" primitive allows us to personalize the UI required by a specific role according to her/his needs (req.3)
- The organization of Mashup components into one or multiple pages is implicitly defined by the use of direct and indirect relationships. While direct relationships keep all the mashed up components within the same page, the indirect relationship splits the connected components into different pages (one per each relationship) (req.6)
- The navigation that results from the use of indirect relationships constitutes the mechanisms used to navigate through the different pages that contain the mashed up components (req.7)
- The attributes that are associated to direct and indirect relationships allows us to define the data that is passed from one component to another and that serves as input parameter for the connected component. (req.11)
- The "main-AIU" primitive allows us to specify the functionality that is associated to the process task and whose execution drives to the completion of the task (req.13)

5. RELATED WORK

The different challenges that arise for the specification and construction of *process mashups* lead us to consider approaches and tools developed by the web engineering, the mashup, and the business process communities.

Within the web engineering community we find a set of model driven approaches (OOHDM [4], UWE [5], OO-H [5], WebML [3], OOWS [7], UWAT+ [1], [2], Hera [8], MIDAS [6]) that allow the specification and construction of web applications supporting the execution of business processes. These proposals gather in a set of models the different aspects that represent a web application (e.g. structure, behavior, navigation and presentation application). Then, based on the application of model transformations these models are transformed into a ready to use web application that is implemented in a specific technology. Even though these proposals deal with the challenges that introduce the execution of a business processes within the context of a web application (proper navigation between tasks, collaboration of different users to accomplish a specific goal or process state maintenance), none of them provides support to the collaborative work required in the *process mashups* applications we are interested in. This particularity cannot be defined in the models that represent the system and therefore it is not considered during the generation process.

Regarding the mashup community we find a plethora of commercial tools (e.g., Yahoo! Pipes or JackBe Presto) and prototypes designed in research projects (e.g. MashArt or ServFace) to support the mashup creation. Out of these tools only a limited number of tools allows to collaboratively work on a mashup. In fact, most of them are conceived to build mashup pages that are used by a single user not requiring the coordination of UIs and people required in a *process mashup*. Opposite to this we find MarcoFlow [9], a design and execution environment that allows orchestrating distributed UIs based on BPEL4UI, a BPEL extension to deal with UI and user management. MarcoFlow supports the development of mashups that can be concurrently used by multiple users, but it does not do so by systematically distinguishing the roles and responsibilities of the involved actors as proposed in this paper. BPEL4UI is at a much lower level of abstraction.

In the literature we do not find many proposals addressing such collaboration. In fact, we only have found the solution presented by Müller and Rogge-Solti in [10]. In it, the authors propose to represent roles by colors instead of lanes. The idea is associating roles with colors and using these colors in tasks to represent the assignment to a specific role. As a result, when a task is performed by just one role, the task will be colored with the corresponding color. On the contrary when the task is performed by more than one role, the task is represented either as many times as roles are involved in the task (each one with the corresponding color) or as a task with vertical stripes including the colors that represent the involved roles.

Finally, to deal with the access control required in collaborative tasks we need mechanisms to manage the different access rights associated to each involved role. The most popular access right mechanism is the RBAC model [12], which has been applied for business processes [13, 14]. In case that the organizational model changes, access rights need also to be adopted. [15, 16] suggested an approach to cope with changes in organizational models and to propagate these changes to process models.

6. CONCLUSIONS AND FURTHER WORK

In this paper we have stated the different challenges that entail the specification of *process mashups* and, focusing on the collaborative aspect, we have presented the extension designed in the OOWS4BP model-driven approach at the modeling level to deal with it. Introducing such semantics within a BP model turns crucial since we are proposing a development process based on model transformations. This means that all the information that is not represented in the models cannot be used during the transformation process and therefore will require manual changes over the generated artifacts.

The introduction of the roles model to detail the set of roles involved in a shared task allows us to enrich the business process model without increasing its complexity. Even that within a model-driven development process models are used are input artifacts for the generation process, these are also intended for human beings. Therefore, it is very important to maintain the understanding and simplicity of such models.

As future work we plan to address also the challenges that we have identified at the execution level. For such purpose we will explore the most appropriate technological infrastructure to ensure the proper execution of *process mashups*. In addition, we also want to define and implement the model transformations that allow us to generate *process mashups* from the system specification. These model transformations will be implemented within the Bizzy project (http://www.pros.upv.es/labs/projects/bizzy/) as an extension to the OOWS approach to provide the required tool support for the construction of *process mashups*.

7. ACKNOWLEDGEMENTS

This work has been developed with the support of MICINN under the project EVERYWARE TIN2010-18011

8. REFERENCES

[1] Distante, D., Rossi, G., & Canfora, G. 2007. Modeling business processes in web applications: an analysis framework. *In Y. Cho, R. L. Wainwright, H. Haddad, S. Y. Shin, & Y. W. Koo (Eds.) SAC*, 1677–1682.

[2] Distante, D., Rossi, G., Canfora, G., & Tilley, S. R. 2007. A comprehensive design model for integrating business processes in web applications. *Int. J. Web Eng. Technol.,* 3(1), 43–72.

[3] Brambilla, M., Ceri, S., Fraternali, P., & Manolescu, I. 2006. Process modeling in web applications. *ACM Trans. Softw. Eng. Methodol.,* 15(4), 360–409.

[4] Schmid, H. A., & Rossi, G. 2004. Modeling and designing processes in e-commerce applications. *IEEE Internet Computing*, 8(1), 19–27.

[5] Koch, N., Kraus, A., Cachero, C., & Meliá, S. 2004. Integration of business processes in web application models. *J. Web Eng.,* 3(1), 22–49.

[6] Koch, N., Kraus, A., Cachero, C., & Meliá, S. 2004. Integration of business processes in web application models. *J. Web Eng.,* 3(1), 22–49.

[7] Torres, V.; Giner, P. & Pelechano, V. 2010. Developing BP-driven web applications through the use of MDE techniques. *Software and Systems Modeling, Springer Berlin / Heidelberg*

[8] Houben, G.-J., van der Sluijs, K., Barna, P., Broekstra, J., S. C., Fiala, Z., & Frasincar, F. 2008. Web Engineering: Modelling and Implementing Web Applications, chap. HERA, 263–301. *Human-Computer Interaction Series*. Springer London.

[9] Daniel, F.; Soi, S. & Casati, F. 2010: Distributed User Interface Orchestration: On the Composition of Multi-User (Search) Applications. *SeCO Workshop*, 182-191

[10] Müller, R. & Rogge-Solti, A. 2011. BPMN for Healthcare Processes. In Proceedings of the 3rd Central-European Workshop on Services and their Composition*, ZEUS 2011*, Karlsruhe, Germany, CEUR-WS.org, 705, 65-72

[11] OMG: Business Process Model and Notation (BPMN) – Version 2.0 (January 2011)

[12] Sandhu, R., Coyne, E., Feinstein, H. & Youman, C. 1996. Role-Based Access Control Models. *IEEE Computer*, 29(2).

[13] Liu, P. & Chen, Z. 2004. An Extended RBAC Model for Web Services in Business Process. E-Commerce Technology for Dynamic E-Business, *IEEE International Conference on, CEC-East*, IEEE Computer Society, 0, 100-107

[14] Strembeck, M. & Mendling, J. 2010. Generic Algorithms for Consistency Checking of Mutual-Exclusion and Binding Constraints in a Business Process Context. *OTM Conferences* (1), 204-221

[15] Rinderle, S. & Reichert, M. 2005. On the Controlled Evolution of Access Rules in Cooperative Information Systems *OTM Conferences* (1), 238-255

[16] Rinderle-Ma, S. & Reichert, M. 2008. Managing the Life Cycle of Access Rules *in CEOSIS EDOC*, 257-266

[17] Daniel, F., Koschmider, A., Nestler, T., Roy, M. & Namoun, A. 2010. Toward Process Mashups: Key Ingredients and Open Research Challenges. *Proceedings of the Workshop on Web APIs and Service Mashups (Mashups)*, ACM.

[18] Young, G.; Daley, E.; Gualtieri, M.; Lo, H. & Ashour, M. The Mashup Opportunity. Forrester Report

[19] Czarnecki, K. 1998. *Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models*. PhD thesis, Technical University of Ilmenau.