Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 329 (2016) 39–59

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Towards a Generic BPMS User Portal Definition for the Execution of Business Processes

Andrea Delgado[1] Daniel Calegari[2] Andrés Arrigoni[3]

Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Abstract

Business Process Management Systems (BPMS) provide support for the business process (BPs) lifecycle, from modeling to executing and evaluating BPs. Key elements provided within a BPMS are a process engine where BP models are executed and a web portal for user's interaction providing means to manage a work list, taking and completing tasks, among other functionalities. Most BPMS portals provide a set of core features for users to manage their work lists, as well as service-oriented access to their process engine, which is tightly coupled. In this context it seems possible to define a generic BPMS user portal which can be integrated (loosely coupled) with potentially any process engine for the execution of business processes. In this paper we define such generic BPMS user portal based on a unified data model and a generic process engine API. We also show the feasibility of these ideas through the development of a prototype.

Keywords: Business Process Management Systems (BPMS), web user portal, process engine, integration layer

1 Introduction

A key aspect that organizations need to take into account in order to be more efficient and effective is to be able to manage their Business Processes (BPs). Business Process Management (BPM) [18,20,6] provides support to the complete BPs lifecycle, from modeling, through developing, deploying, executing and evaluating their execution, regarding both the organizational and technical environments. In this context, Process Aware Information Systems (PAIS) [7] and in particular, Business Process Management Systems (BPMS) [2] arise as the technology to support the BPs lifecycle.

[1] Email: adelgado@fing.edu.uy
[2] Email: dcalegar@fing.edu.uy
[3] Email: arriori@gmail.com

http://dx.doi.org/10.1016/j.entcs.2016.12.004

Regarding the technical support for managing BPs, there are many BPMS platforms which provide a core set of components to work with BPs and specifically, to allow process execution, such as: a process engine based on some language (BPMN 2.0 [12], XPDL [22], WS-BPEL [9], YAWL [16]) where BPs are executed, and a user web portal which allows them to interact with BPs via managing their worklist by taking, performing, re-assigning and completing tasks, among others. Portals also provide means to initiate cancel or suspend a process, to manage roles, users and permits, among other actions. Both BPs and activities definitions and instances have a lifecycle which defines the possible states and the transitions that are allowed between them, which are the basis for their execution [24]. Most process engines also provide access to its functionalities by means of a native Application Programming Interface (API), which is also exposed as a Web Services API (REST/SOAP) to allow invocations from other systems for integration with already existing organization's software.

In most BPMS the user web portal is tightly coupled with the native process engine API, as the process engine is embedded in the system, in such a way that the only way to use the process engine is via the user web portal that the BPMS provides. This could be the best option in many cases, if the process engine and user web portal integrated in the BPMS fulfill the requirements of the organization. But there are some cases in which it would be desirable for an organization to be able to develop or to integrate their own user web portal with an specific process engine, having some very specific requirements to be fulfilled -not only regarding the changes in look&feel most BPMS provide- but others regarding for example certain design or usability guidelines, specific web technologies, other systems or elements to be integrated in the portal, among others. This, of course, can be easily achieved by invoking the REST/SOAP API provided by the process engine, but although the user web portal will not then embed the process engine directly, it will still be coupled with the specific functionalities the API provides, in terms of operation signatures, parameters and types. Also, some process engines are continuously evolving and also their user web portals to accommodate changes in the underlying engine, so it would be desirable to be able to minimize the impact of these changes in an organization, mostly when the organization's size is considerable and introducing new systems and/or changes in already existing user interfaces (such as the web portal) are difficult to accomplish. Moreover, an organization would want to migrate to a new process engine which support process execution in the same or another language, if the functionalities it needs, are not currently supported.

Based on this analysis, it is possible to define a generic BPMS user portal which can be integrated with potentially any process engine via a normalized API, based on a unified model of the underlying concepts needed for process execution. The independence between the front-end and the process engine provides means to migrate from one process engine to another without affecting the user experience, as well as offer a unified interaction layer for external applications, similarly to the way applications already work nowadays with many database management systems (DBMS). Finally, modeling the specific functionalities of the user web portal, gives

us the possibility to automatically generate the portal for different technologies, generating the invocations to our normalized API in the generation, so the portal can be used with any process engine. Models also help visualizing, analyzing and specifying the concepts and transitions defined for the BPs and activities lifecycle, and for the worklist management, allowing improvements.

In this paper we present the definition of a generic BPMS user portal based on a unified data model and a generic process engine API, which can be integrated with potentially any process engine for the execution of business processes. Its definition was based on a comparative evaluation of three different BPMS (Activiti [4], Bizagi [5] and Bonita [6]) with respect to the functionalities they provide in their user web portals and process engine APIs. Moreover, we modeled our generic BPMS user web portal using the Interaction Flow Modeling Language (IFML) [13], as standard language designed for expressing the content, user interaction and control behavior of the front-end, which allow us to provide a platform-independent front-end model to be used as basis for the development or automatic generation of BPMS user portals.

The rest of this paper is structured as follows. In Section 2 we present related work. In Section 3 we introduce general and key components in a BPMS architecture, as well as a comparative evaluation of the selected BPMS. In Section 4 we detail the architectural definition of the generic BPMS user portal, the underlying unified model and the generic process engine API from which to access any process engine. In Section 5, we introduce the IFML standard and the IFML-based generic user portal we have defined. In Section 6, we show a prototype we have developed as a proof of concept and finally, in Section 7 we present conclusions and future work.

2 Related Work

There have been approaches that tried to unify the concepts of BPM right from the beginning of workflow management as the Workflow Reference Model and the Workflow Management Facility Specification [21,11] in which key functionalities for a workflow engine are defined. In particular, in [21] five functional interface areas are defined: (1) process definition, (2) workflow access from a client, (3) workflow access from other applications, (4) workflow interoperability and (5) management and monitoring, with a common set of API calls and related interchange formats. Although this model defined key functionalities for process engines (which served as a basis for many workflow and BPMS systems) the approach is of a high level of abstraction with no specific operations, which in our approach are defined in detail.

In [8] the authors present a metamodel defining the main concepts of a process model. Besides it is not focused on process execution but on process modeling, the metamodel has some related aspects. In [24] three perspectives for the run-

[4] Activiti BPM Platform. http://www.activiti.org/
[5] Bizagi BPM. http://www.bizagi.com/
[6] Bonita BPM. http://www.bonitasoft.com/

time of a workflow management system are taken: participants, administrators and customers, and some workflow systems functionalities are also analyzed with focus on defining a reference model for the interface (5) of management and monitoring. We also take these three perspectives but our unified data model and API covers a wider set of interfaces from (2) to (5). In [3] the authors describe an effort to define common APIs for structured peer-to-peer overlays and the key abstractions that can be built on them. Although the work is not focused on BP, it also defines a generic API which is validated by implementing some communication protocols. Finally, in [15] the authors define a BP execution needs at least five types of data: (i) business data for the process logic, (ii) BP models, (iii) execution states (and histories), (iv) correlations among BP instances, and (v) resources and their states. They also define a way of expressing this data by means of "self-guided artifacts. Our API handles these data, except for (ii) and (iv) which is not important for the interaction with the process engine, but we need further work in order to abstract data using self-guided artifacts.

Compared to these works, our proposal provides a unified view of concepts and relationships regarding the execution of BPs, along with a highly decoupled view of the main components of a BPMS for process execution i.e. the user portal and the process engine: (i) a generic user web portal specified as an IFML model, with defined functionalities based on the concepts from the unified model; (ii) a unified model which provides an integrated vision of BPs and activities definition and instances concepts along with their relationships as a basis for process execution; and (iii) a generic process engine API which allows managing the concepts defined in the unified model, providing the means to access potentially any process engine. Although the unified model is based on a primary analysis of concepts from three BPMS, we have identified a core set of concepts and relationships based on the concepts analyzed, which provide the basis for process execution and user worklist management, role and users management, among other functionalities. To the best of our knowledge there is no other proposal in this specific direction.

3 BPMS Portal Evaluation

The architecture of BPMS [20,6,7,2] platforms integrates several components to support the complete business process lifecycle [20], based on specified BP models. A key component of a BPMS is the process engine, which is able of executing a model process specified in a language such as BPMN 2.0 [2], XPDL [12], WS-BPEL [22], YAWL [9], among others. Another key component of a BPMS is the user's web portal, which allows users to interact with the process engine in order to manage processes, activities and other BPs execution elements. Other components offer support for process modeling, simulation, monitoring and real-time analysis, among others.

In previous works [4] we have evaluated several open source and proprietary BPMS based on a defined list of key functional characteristics that this kind of tools must provide. such as process model support, process versioning, roles definition,

Table 1
Example of first-level features for BPMS

Category	Type	Process engine language	API	Tool
Open source	Programmer oriented	BPMN 2.0	REST	Activiti jBPM [4]
		XPDL	REST	Joget workflow [5]
			NO	OBE [6] WfMOpen [7] Enhydra Shark [8]
		WS-BPEL	SOAP	Apache ODE [9] Riftsaw [10]
			NO	Orchestra [11] Petals [12]
	User oriented	XPDL	REST	Bonita
Proprietary	Programmer oriented	BPMN 2.0 WS-BPEL	REST	IBM BP Manager [13] Oracle BPM [14]
		WS-BPEL	SOAP	Intalio [15]
	User Oriented	XPDL	SOAP	Bizagi ARIS [16]

workflow patterns [17], among others. However, at that time we did not focused on the explicit identification of operations defined in the process engine API, as we did now. Based on these works, we were able to identify some first level features that help us to shorten the extensive list of existing BPMS [14,23,10], by defining:

(i) categories: open source and proprietary, which mainly determine the availability of the BPMS to users and the free access to the code

(ii) types: programmer or user oriented, which mainly determine the needed programming skills of the developers

(iii) process engine language: which can be BPMN 2.0, XPDL, WS-BPEL, YAWL, among others.

(iv) API publishing: WS interface REST or SOAP, which determines the types of operations and format to invoke the engine from outside the BPMS.

Table 1 shows the classification of some BPMS platforms we have already evaluated in previous works [4].

Also, as defined in [6] three categories of BPMSs can be identified with respect to their support for BPMN process model execution: Pure BPMN, Adapted BPMN and Non BPMN, where in the first case the tools support BPMN natively e.g. Activiti, in the second they use a BPMN skin but use another representation to execute the process model such as XPDL e.g. Bonita and Bizagi, and finally tools which do not support BPMN and use their own proprietary language and semantics e.g. YAWL. For this proposal we focused the evaluation on the first two cases i.e. tools that provide support for BPMN either natively or with another internal representation, being nowadays BPMN the de facto standard for process modeling.

For doing this, we base our selection of BPMS to evaluate on the open source category, combined with programmer and user oriented type, process engine language BPMN 2.0 or XPDL and REST and SOAP publishing API, trying to cover the different existing options by selecting a representative of each equivalence class. We left out WS-BPEL this time since we want to focus on the user portal to manage human tasks and user's worklist, which are not natively managed in WS-BPEL. Also, since we did not count with a previous evaluation of an open source BPMS platform providing SOAP API publishing in BPMN or XPDL, we decided to include the Bizagi BPMS although the execution component is proprietary, to provide an evaluation with SOAP API. Finally, we selected three BPMS from the ones we have already evaluated to seize the knowledge we have gained throughout the years: Bonita, Activiti and Bizagi, to compare the concepts and definitions they manage throughout their user portal and engine API.

3.1 Bonita BPMS

User web Portal. It is organized in three main columns: left, which is used as an information filter; central in which lists of elements such as tasks working lists, cases and process are shown allowing performing actions on them; and right where detail information is shown for the selected elements within the central column, allowing to work on them. The content and structure depends on user's rights.

Data model. It defines concepts such as Profile, Role, Group and Membership of a user, which allows modeling user's rights. The profile is a set of permissions over potential actions that the user can perform on the portal. The concepts of Role, Group and Membership are used to represent the structure of the organiza-

[4] jBPM, http://www.jboss.org/jbpm/
[5] Joget, http://www.joget.org/
[6] OBE, Open Business Engine, http://obe.sourceforge.net/
[7] WfmOpen, http://wfmopen.sourceforge.net/
[8] Together Enhydra Shark, http://www.together.at/prod/workflow
[9] Apache ODE, http://ode.apache.org/
[10] Riftsaw, http://www.jboss.org/riftsaw
[11] Orchestra, http://orchestra.ow2.org/
[12] Petals, http://petals.ow2.org/
[13] IBM BP Manager, http://www-01.ibm.com/software/ integration/business-process-manager/
[14] Oracle 11g, http://www.oracle.com/us/technologies/bpm/suite/overview/
[15] Intalio BPMS, http://www.intalio.com/bpms
[16] ARIS Platform, http://www.softwareag.com/corporate/products/aris

tion. Bonita names a process definition as Application, and process instances as Cases. Applications contain tasks definition, and Cases contain tasks instances. Variables instances can be defined which maintain the task state for a particular case. Variables can be also defined at the application level so variable instances can be also related to cases. Users can add comments to both cases and tasks instances in which they are working.

REST API. It is composed of three sub APIs: Identity API, which groups administration information (users, groups, roles); UserXP API which allows managing the web portal and profiles appearance, configuring actions that a user can perform, and BPM API, which allows administrating elements from a process, e.g. retrieve cases, assign a task, among others. To access any of the defined methods of the API, the user must be first authenticated invoking the "LoginService" which generates a cookie that has to be sent in any subsequent invocation to the server. Depending on the desired resource action (creation, retrieving, updating, deletion and searching) the API uses different HTTP methods (POST, GET, PUT and DELETE) to perform invocations. To assign a Task with identifier "2" to a user with identifier "23" the REST invocation is: `PUT http://<server>/bonita/API/bpm/humanTask/2 Payload {"assigned_id":"23"}`

3.2 Activiti BPMS

User web Portal. It is also organized in three main columns, and a menu above them: left, shows the tasks working list for the logged user; central shows the detail information for the selected elements on the left, and right where events and comments are shown. The menu provides sub-menus to manage tasks, processes, reports, and an option for management, only available for administrators.

Data model. It defines elements such as Group and User to model user's permissions. The Group can be of type "Security-role" or "Assignment" which is a restricted vision of the role concept. Then, specific actions are allowed within the portal e.g. administrator and user in the case of "Security-role", or definition of groups for the assignment of tasks in processes e.g. "Marketing" or "Finance" in the case of "Assignment". The last case represents the same concept represented in Bonita with the Group, Role and Membership combination. Activiti also use Process Definition, Process Instance, Task Definition, Task Instance, Variable Definition and Variable Instances concepts, which are associated to Task or Process Instances. The Events concept is defined for registering a state change of a task.

REST API. Activiti REST API is similar to the one defined by Bonita, using the HTTP methods to perform invocations. It is also composed of several sub APIs named: Repository, Runtime, Identity, History, Forms and Management. Access to any resource needs the user to be authenticated, but opposite to Bonita, Activiti uses the "Basic authentication" as security method, for which in each HTTP invocation the required header has to be sent, including both the user name and password of the user performing the invocation. To obtain a list of available process definitions the REST invocation is: `GET http://<server>/repository/process-definitions`

3.3 Bizagi BPMS

User web Portal. It presents two areas and a menu above them. The left area is the cases area and the right area is the work area. The menu provides options for managing cases, portal administration, reports, among others. The cases area allows the user to see the cases work list, grouped by process definition and ordered according to their state: on time, at risk, expired. The work area is composed of two columns: the left one shows the tasks working list for the user, and the right one shows the detail information for the selected tasks from the left allowing to work on them. As in the previous portals, the options provided correspond to user's rights.
Data model. It defines concepts such as Groups and Roles as in the previous analyzed ones, to model such permissions. Roles can be "Seller" or "Secretary", and Groups can be created combining user characteristics such as skills areas. It also uses the concepts Process and Cases for definitions and instances of processes respectively, an Activity represents a Task instance and Task represents its definition. Regarding comments, it only allows comments on process instances.
SOAP API. Differently to the previous analyzed BPMS, Bizagi provides a SOA layer with SOAP WS to access the process engine. It is organized in four services categories, similar to the sub APIS defined in the others: Workflow Engine, Entity Manager, QuerySOA and RenderSOA. The workflow engine allows managing the control flow of processes executing actions such as start processes or perform activities; the entity manager provides access to the data model allowing adding, retrieving or updating information on entities; the querySOA allows to obtain information regarding cases, tasks or entities; finally, the renderSOA allows querying information regarding the metadata of defined user forms, such as data fields. Each service defines several methods that will be invoked in order to obtain the corresponding data, and for each of them the XML documents which are used as input and output are defined. The definitions in each category can be seen obtaining the corresponding WSDL `http://[Server]:[port]/BizAgi-war/WebServices/[category]?wsdl` where category is one of the sub-APIS defined. The invocation is via a SOAP WS method.

4 Generic BPMS Portal Proposal

Based on the evaluation of BPMS we carried out, we have defined a unified data model, a generic BPMS user portal and a generic process engine API. The generic BPMS portal architecture we have defined follows a layered architectural style, as shown in Figure 1. It is composed of two main layers: the presentation layer where the generic user portal is defined, and the access layer which provides the means to connect the generic portal to the selected process engine, via the generic process engine API which exposes the defined functionalities, and its implementation.

At the Presentation layer level, the BPMS user web portal component based on the unified data model provides the user with the set of functionalities needed to manage BPs and activities cases, worklist, groups and roles, among others. Also, other existing applications can be defined and connected to the generic API in order to interact with the selected process engine.

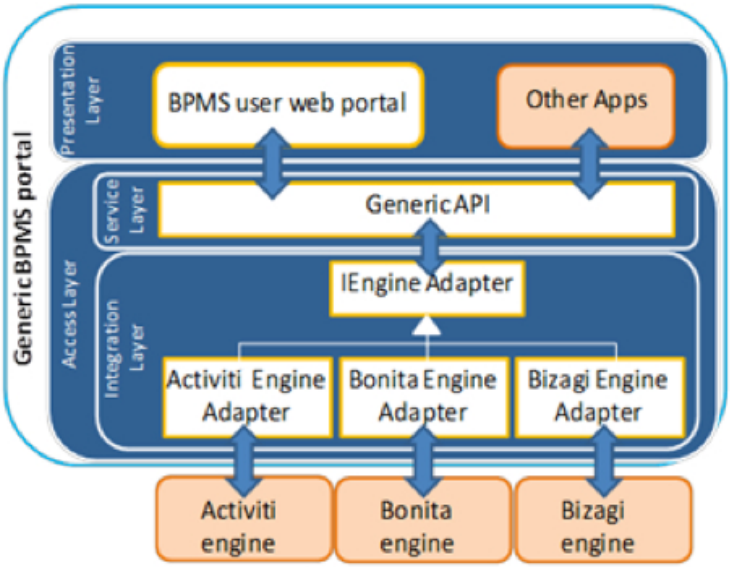

Fig. 1. Generic BPMS portal Architecture diagram

The Access layer is composed of two sub-layers: the Service layer where the generic process engine API is defined exposing functionalities to the user portal, and the Integration layer which provides the means to actually implement the interaction with the selected process engine. Examples of functionalities provided by the generic API to the Portal goes from general services such as logging facilities, to specific services for the portal such as listing pending tasks for a given user. The Integration Layer is responsible for providing the functionalities exposed by the generic process engine API by invoking the specific methods from the selected process engine, since each process engine has different operation signatures and ways of exposing the same functionality. It comprises two main components: (i) the interface IEngineAdapter component which defines the functionalities that have to be provided, and (ii) the specific adapter component such as the ActivitiEngineAdapter which have to be implemented for each selected process engine. Each adapter has the responsibility of providing the functionalities defined in the interface by invoking the corresponding process engine and translating the response to the one expected by the generic API.

This layer decouples the definition of functionalities of the portal from the implementation of these functionalities, which can be provided by any process engine via the Integration layer. In the case that the selected process engine does not provide the selected functionality, it can be disabled in the user portal. In this way, our BPMS generic portal does not provide yet another implementation of a process engine, but allows integrating existing ones through adapters.

4.1 Unified data model

Our unified data model takes into account the concepts defined by each evaluated BPMS, and provides the basis for the functionalities in the generic BMPS portal. Table 2 shows the relation between our concepts and the ones in each portal.

Table 2
Concepts from the unified model and corresponding BPMS

Unified model	Bonita	Activiti	Bizagi
Process	Application	Process definition	Process
Case	Case	Process instance	Case
Task definition	Task definition	Task definition	Task
Task instance	Task instance	Task instance	Activity
Variable definition	Variable definition	Variable definition	Variable definition
Variable instance	Variable instance	Variable instance	Variable instance
User	User	User	User
Group	Group	Group	Group
Role	Role	—	Role

The model is a generalization of the three evaluated BPMS, as are the screens and functionalities offered by the portal which are based on this unified data model. An example is the Process concept which is named Application in Bonita, Process Definition in Activiti and Process in Bizagi, from which we decided to name it only Process as in Bizagi. Also, from the different options analyzed, we decided to use the concepts Role, Group and User separately following the definitions in Bonita and Bizagi, where a User can belong to a Group and can also have a Role. Although the Group and Role concepts can be merged into a single concept as does Activiti where only Groups are used, we decided to separate them since we consider it clearer to provide both concepts. A Process identifies the definition of the BP and a Case refers to each instance of the Process. The User can initiate a Case from a Process. Activity (and Task) definitions are part of the Process definition, and their instances are part of the Cases. Variables of the Process can be defined at the level of both Process and Activity, so their instances are connected to Cases and Activity instances respectively (other systems such as YAWL [9] use the concept of WorkItem which we did not introduce at this time). A User can insert comments to Cases. Figure 2 depicts a conceptual UML models of these concepts and relations.

4.2 Generic process engine API

In Table 3 we provide an example of the kind of operations we have defined in the generic process engine API, regarding the different concepts identified in the unified data model grouped by conceptual category. Then, each Adapter has to implement these operations by invoking the corresponding operation exposed in each BPMS API publishing, as presented in Section 3.

As presented before, the generic process engine API provides a set of functionalities to be offered to the portal and allows invoking any selected process engine, performing the corresponding mapping between operations. These functionalities are accessed from the BPMS user portal, as defined in Section 5.

Table 3
Example of the operations defined in the generic API

Security category
`Login(string user, string password): void`
Allows the user to be identified within the system and access functionalities.
`ChangePassword(string oldPassword, string newPassword): bool`
Allows the user to modify the current password.
Process category
`GetProcessDefinitions (string name, string category, bool active): Enumerable<Process>`
Returns a list with existing process definition by name, category and state.
`GetProcessDefinition(int processId): Process`
Returns the definition of the selected process.
`SuspendProcess(int processId): void`
Suspends process with `processId` so no cases can be generated from it.
Cases category
`CreateCase(int processId): void`
Allows the creation of a case from the process with id `processId`.
`GetAllCases(string name, string creator, string user, string state): IEnumerable<Case>`
Returns a list of cases filter by process name, assignee, creator, or in some state.
`AddCommentToCase(int caseId, string userId, string comment): void`
Allows to add a comment to a case by the selected user.
Tasks category
`GetTasks(string name, bool assignee, string candidateUser): IEnumerable<TaskInstance>`
Returns a list of tasks by name, assigned or not, to be taken by the user.
`TakeTask(string taskId): bool`
Assigns the task to the user performing the invocation.
`GetTaskVariables(string taskId): IEnumerable<VariableInstance>`
Returns the list of variables associated with the selected task
Groups category
`GetGroups(string like): IEnumerable<Group>`
Returns the list of existing groups.
`RemoveUser(string userId, string groupId): void`
Allows deleting the selected user from the group.
Roles category
`GetRoles(string like): IEnumerable<Role>`
Returns a list of existing roles.
`AddUser(string userId, string rolId): void`
Allows to add a role to a user with id `userId`

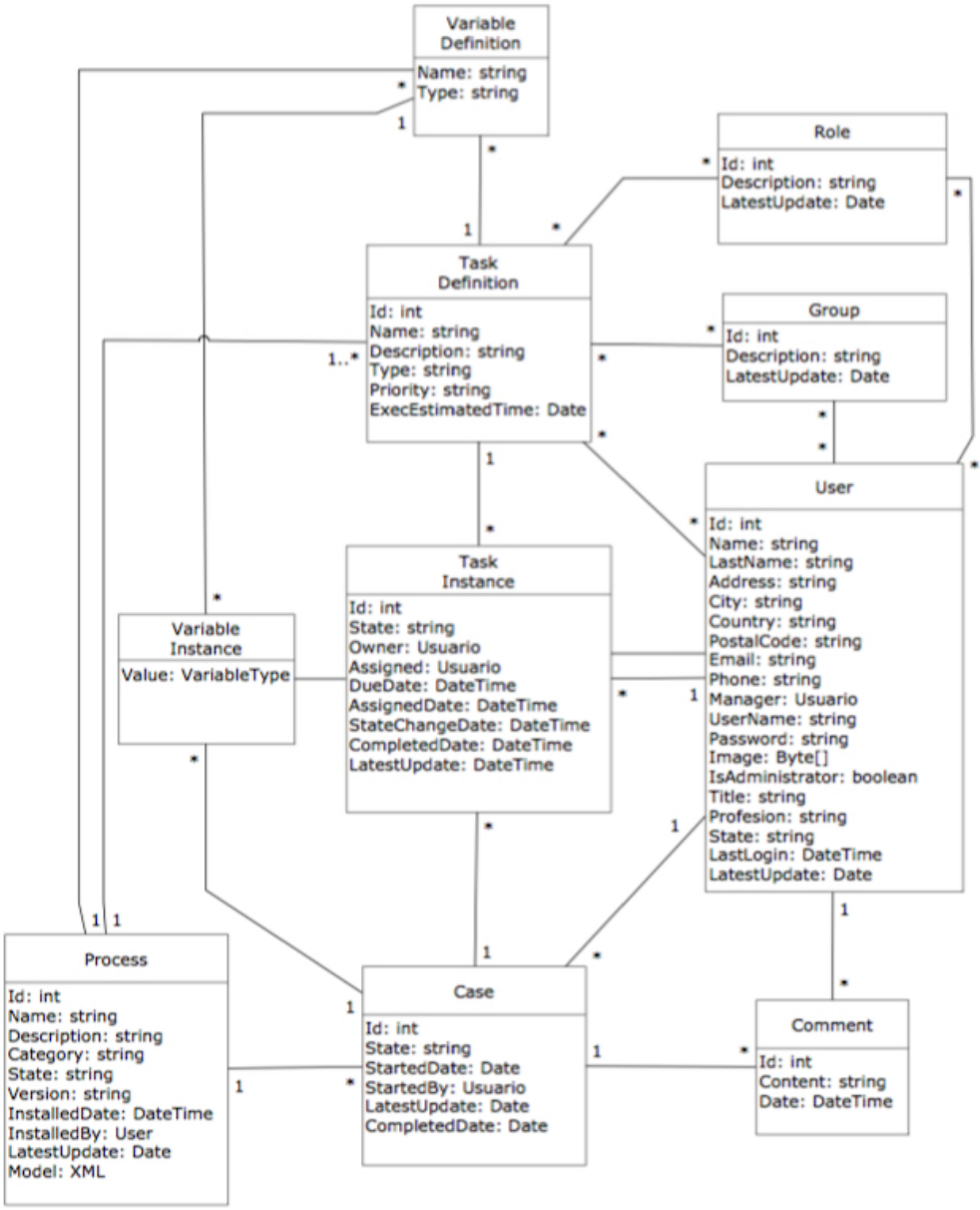

Fig. 2. Unified data model for a BPMS portal in a conceptual UML model

5 IFML-Based Generic BPMS Portal

In this section we describe the IFML models corresponding to the presentation layer of the generic BPMS portal we have defined. The elements provided by IFML allow defining the interaction of a user with any system, in our case we use the WebML extension to model the portal web site, which we present in the following.

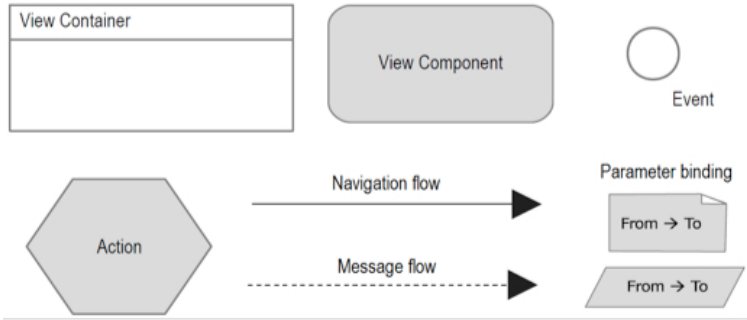

Fig. 3. Main IFML concepts

5.1 IFML standard

The objective of the Interaction Flow Modeling Language (IFML) [13] is to describe the main aspects of an application front-end, basically: the view part of the application together with the data and business logic actions, and the control logic. By using IFML, it is possible then to automatically generate the front-end for different technologies. IFML main elements are shown in Figure 3. An IFML diagram consists of one or more top-level view containers which can be composed by other containers, e.g. a main window with many nested windows. Within a view container, there are view components, i.e. elements of an interface that displays content or accepts input (e.g. a form, a list, etc.)

Events are attached to containers and components (e.g. when an elements of a list is selected) affecting the state of the user interface and possibly causing the triggering of an action (e.g. deletion of the selected element of the list). The state change caused by an event, as well as the effect of an event triggering, is expressed as a navigation flow connecting the event/action to the view container or component affected by the event/action. Data also flows between view elements (view containers and components) or actions. It is possible to define parameters (typed and named values) and bind them to navigation flows. WebML [1] is a domain-specific language for web applications which is defined using the IFML extension mechanism. Figure 4 shows some elements of WebML.

The language defines specific view containers, e.g. site views representing a set of pages and/or areas forming a coherent view of the site such that it can be different site views for different types of users, areas representing a set of pages (possible nested), and pages representing a container of one or more pieces of information shown to the user at the same time. View components are also detailed, e.g. for representing detailed information of an entity, selecting a list of entity keys, representing a form, and expressing database operations for entity creation or deletion.

5.2 Definition of the IFML-based portal

The IFML models we present were specified using the Webratio [19] tool. The web site of the generic BPM portal in IFML begins with the definition of three site views regarding the three different types of users we identified: administrator, common

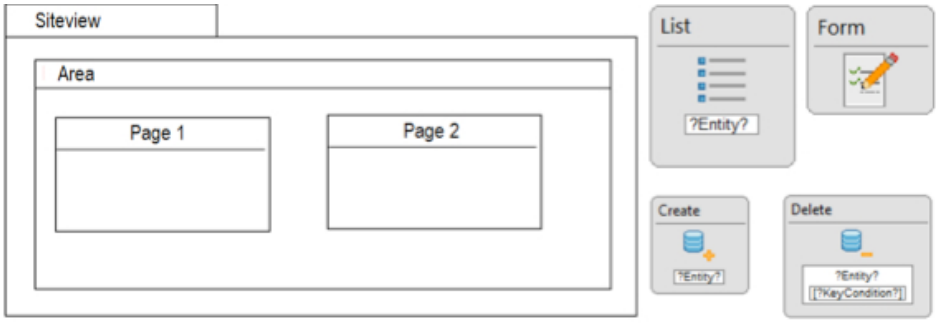

Fig. 4. Main WebML concepts

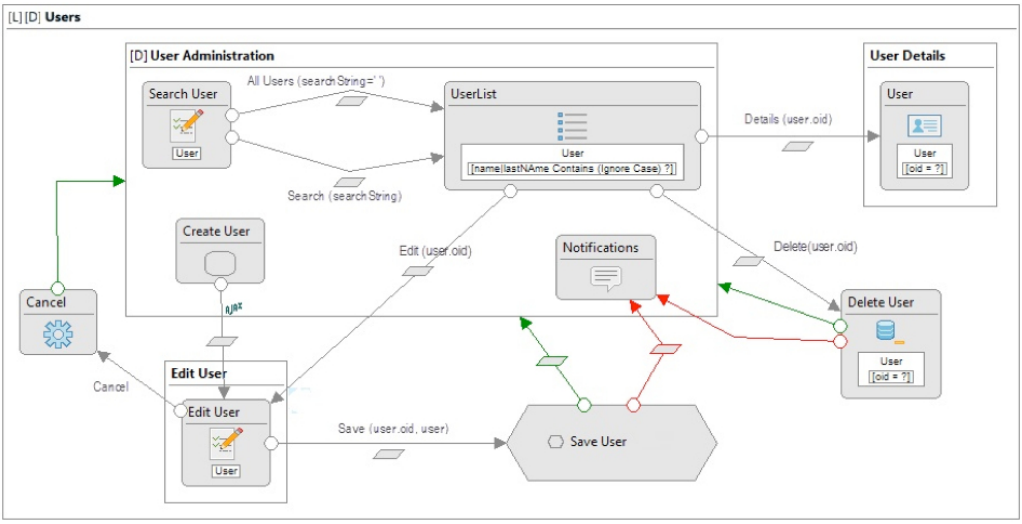

Fig. 5. IFML model for the sub-area users and defined pages of the Administrator site view

user and public. The administrator site is composed of pages to provide functionalities regarding administration of groups, user and processes, among others. To access this site the user must login with valid administrator credentials. The common user site is composed of the pages providing the functionalities for users of the BPMS, e.g. list assigned tasks, take or work on a task, among others. To access this site the user must also login with valid credentials. The public site view is for people not identified within the system, and contains public pages as the login page.

5.2.1 Administrator site view

The Administrator site is divided into two main areas: directory and process management, each of one contains three sub-areas. The first area of directory includes sub-areas for users, roles and groups management, and the second area of process management includes sub-areas for processes, cases and tasks management. As an example we present the IFML modeling of the sub-area user. Figure 5 presents the IFML model, showing the pages defined, along with the components and actions for the navigation between them.

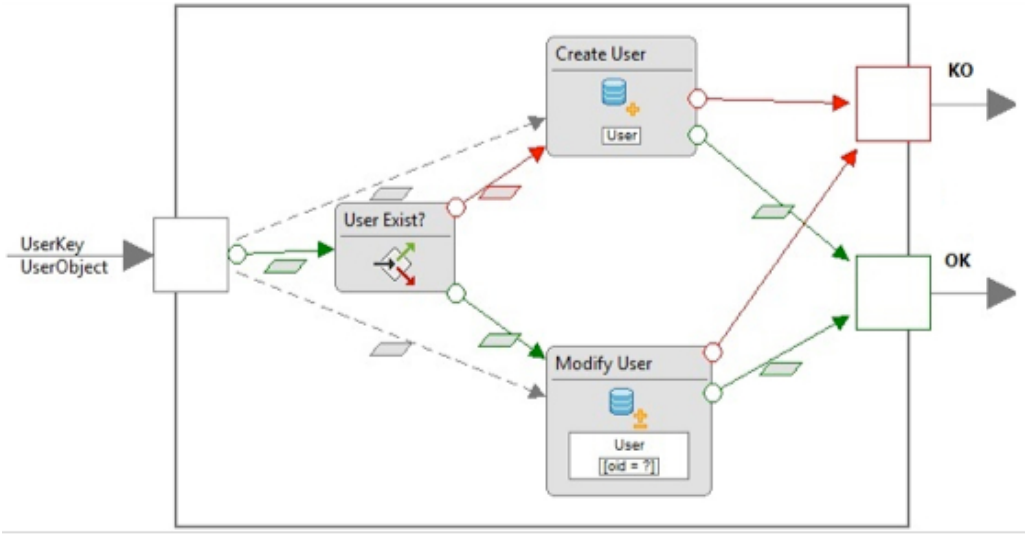

Fig. 6. IFML model for the Save User component

We define three pages named User administration, User details and Edit user, where the first one is marked as default to be showed when the administrator enters the sub-area users. The User Administration page defines the Search user and User list components. Search user is a component of type "Form" which is associated with the user entity, and allows inserting text to search a user whose name contains it. The user list component shows a user list and defines a selector of type "Contains" on the properties "name" and "lastName". This allows defining a link between components Search user and User list sending as parameter the text entered to filter the list. These components are repeated in each diagram that shows a list of elements (entities) and allows filtering the list based on defined parameters.

From the User list component we can navigate to two new pages and to an action on a selected user from the list. The details of a selected user can be seen in a new page named User details, which includes the component of type "Detail". The "Details" link allows navigating to the component which is associated to the user entity, by sending the id of the user as parameter. To edit a selected user we navigate from the user list to the new page Edit user, which includes a component of type "Form". This component will load the user data allowing their modification. Once the edition is completed the option of saving user data is provided, as the action Save user. The results of saving the data can be successful or failed. In the second case a message is sent to the notifications center which is located in the main page of the area. These components are reused to create new users. We define an action named Create User which also invokes the component Edit User but without any user id. Then, the component will not load any user data, allowing data for a new user to be inserted. The same Save user action is used to save the data for the new user. There is also a link "Delete" to a component which executes the deletion of an user. Figure 6 shows the IFML model for the reusable Save User component.

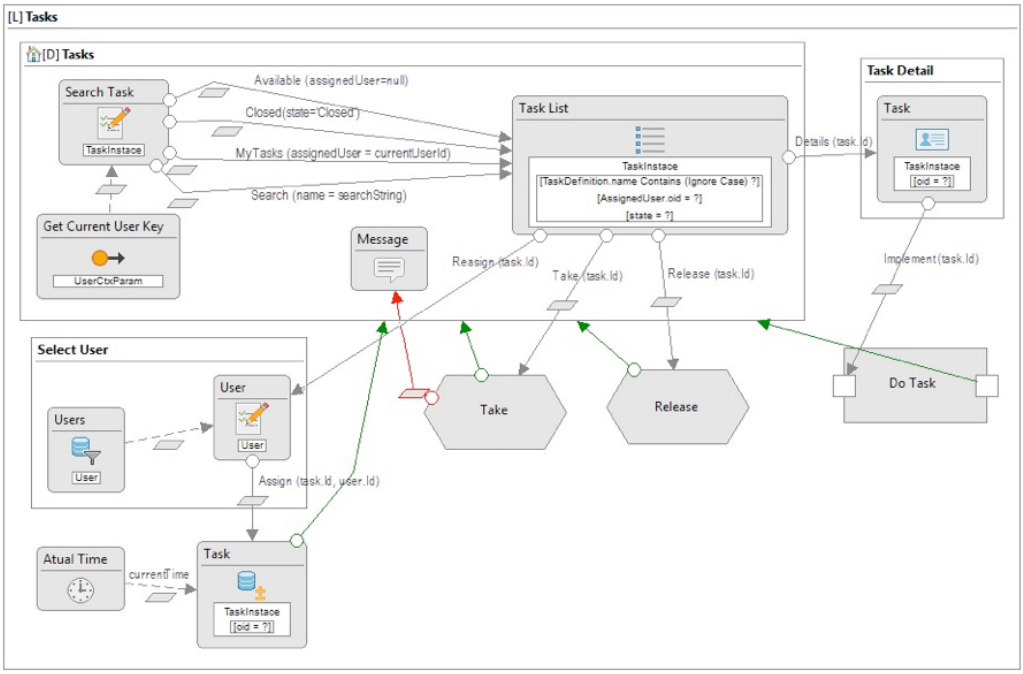

Fig. 7. IFML model for the Task pages of the Common User site view

5.2.2 Common User view

Similarly to the Administrator site, the Common User site is divided into three main areas: tasks, cases and processes, which allow identified users to manage their work within the processes in which they are involved. As an example in Figure 7 we present the IFML modeling of the Tasks area, which contains pages that are related with the task concept, allowing users to see, take, release and execute tasks from process and cases they work. We define three pages named Tasks, Task details and Select User, where the first one is marked as Home and default to show the task list and a "Form" component for searches over the list. The options for listing Tasks are: tasks assigned to the logged user, available tasks or tasks with no assigned user, closed tasks, and filtering tasks by name. As presented before, these options are defined with links over the Search and the Task list components.

Actions that can be performed over a selected task include: take, release, reassign or execute. The take action can be only performed over tasks assigned to the user role. For the reassign process once the task is selected the user to whom it will be assigned has to be selected. To do so, a link is defined between the Task list and a component for the user selection, which is placed in a new page Select User. This component is also linked to the Task component which will be finally updating the task, changing the assigned user. As an example, we describe the options Take task and Execute task.

Figure 8 (a) shows the one corresponding to the Take task option. We define a central component for updating the task, the Take task component which receives three parameters: the task id (which is the input of the action), the user id which

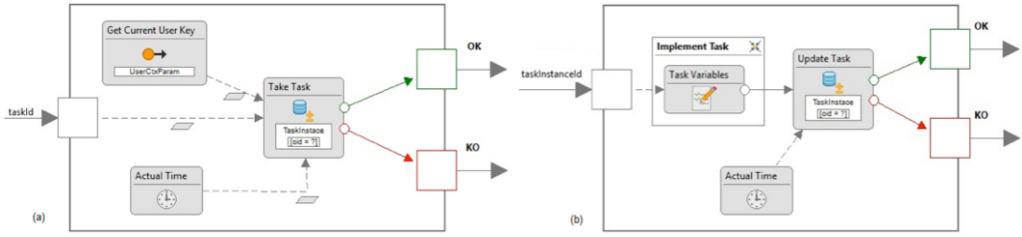

Fig. 8. IFML model for the: (a)Take task option and (b) Execute Task option, of the Common User site view

is provided by the component GetCurrentUserKey, and the actual time (from the system) which will be set as the time on which the task is assigned to the user. The assignment result can be successful or failed, and is represented by the two outputs. Figure 8 (b) shows the Execute task option. In this case we also define a central component for executing the task, but the first thing to be carried out is to allow the user to insert the data corresponding to the data fields defined within the task or globally within the process. This is represented with the new page which contains a "Form" component which will be responsible for getting the data from the task variables. Then the Update task will receive this data as parameters, and the actual time (from the system) which will be set as the time on which the task is completed. The execution result can be successful or failed, and is represented by the two outputs.

6 Proof of Concepts

As a validation of the generic BPMS portal definition given in Section 5, we build a functional prototype and we use it for experimentation through several examples.

6.1 Prototype

We implemented a web application conforming to the IFML-based models. This application is connected with the Activiti process engine by means of the generic process engine API. The general architecture of the solution is based on the generic portal architecture (Figure 1) as presented in Section 4. The presentation layer was implemented as a single page application using HTML5 and Javascript, as well as other frameworks for structuring code (AngularJS) and style sheets and layout definitions (Bootstrap). This kind of client-side application only performs server calls when new data or actions are needed, reducing round tripping and enhancing user experience. Areas within the general IFML-model (tasks, processes and cases) were implemented as separate HTML pages (i.e. segments of a template in AngularJS) together with their own controller. A proxy is used for each controller, resolving the communication with server side components.

The back-end adapts front-end calls into platform-dependent process engine actions. There is an HTTP REST API implemented using Jersey, which provides a concrete instance of our unified process engine API (UI Service Layer in Figure 1). Their methods are organized into three components which use the Activi-

tiEngineAdapter (Integration Layer in Figure 1) for communicating with Activiti's process engine. The data model was implemented using the JavaScript Object Notation (JSON) as a lightweight data-interchange format. There is also a security component encapsulating common functionalities for user authentication.

We develop a partial implementation of the generic process engine API. In particular, we omitted those used for addressing administrative tasks, and we focused on those related with process execution, e.g. list installed processes, init a process case, list pending tasks for a given role, and perform a task.

6.2 Example of application

We use a very basic but complete case study which allows us to exhaustively experiment with our prototype. The process is about handling vacation requests and comes with the Activiti distribution. After logging in a user can see the processes in the menu, and selecting one process he/she can start a new instance (case) of the process. Figure 9 shows the process view tab and the existing processes in the engine, along with the selection of the vacation request process to start a new case.

The site view contains a menu with three main areas: tasks, cases and processes. Each area contains three columns: left to show grouped elements, central to show specific elements and right to show detail information of elements in the central column. The tasks area is the default one and shows the user's tasks list, it provides means to search for a specific task, to take, release, reassign and complete a task. After the process is started, the user must fill in the vacation request data and a manager needs to approve or disprove it. In every case, once the user logs in, which requires the invocation of a service within the common API and the corresponding invocation of the Activiti's login service, the user can access its tasks area, which is shown in Figure 10. This area corresponds to the IFML model depicted in Figure 7. In this case the invocation implemented in the generic API to the activiti REST API is: `http://localhost:8080/activiti-rest/service/runtime/tasks?assignee=user`

In particular, in Figure 10 the manager has already selected a task, thus the variables to execute the task are shown and the manager can save the task for later or complete it after selecting if the request is approved or not. The information related to variables assigned to user tasks are requested to Activiti through the back-end, Activiti sends a JSON object with the requested information, which is then translated into our unified data model and then interpreted by the front-end. Since variables have an associated type, as well as if there are required fields or not, the front-end creates specific controls for each one of them as well as the corresponding validators, e.g. the Approve/Reject combo box in Figure 10.

7 Conclusions and Future Work

In this paper we defined a generic BPMS portal which can be integrated with potentially any process engine via a normalized API. As mentioned in the related work section, we have taken into account several existing standards and definitions for process engines data and functionalities, and addressed a comparative evaluation

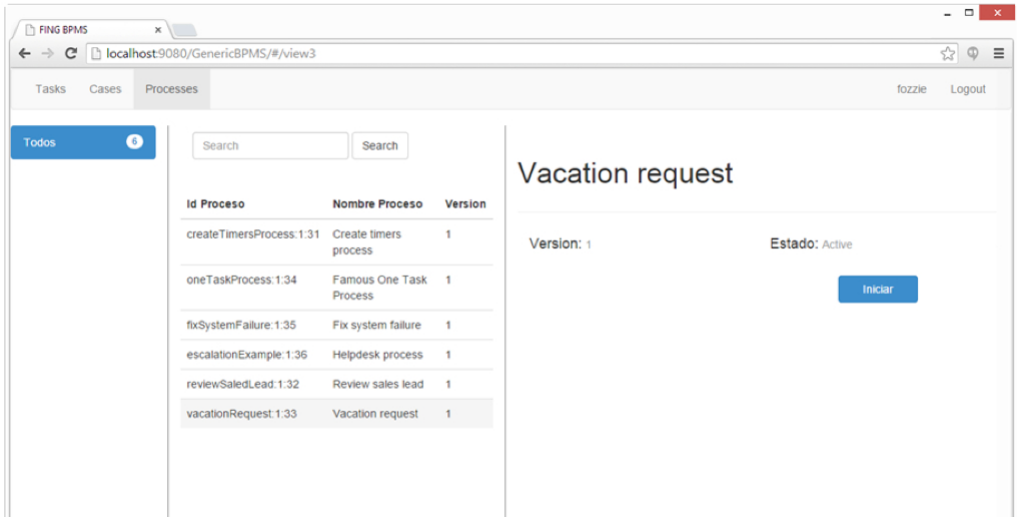

Fig. 9. Processes view of the generic BPMS portal

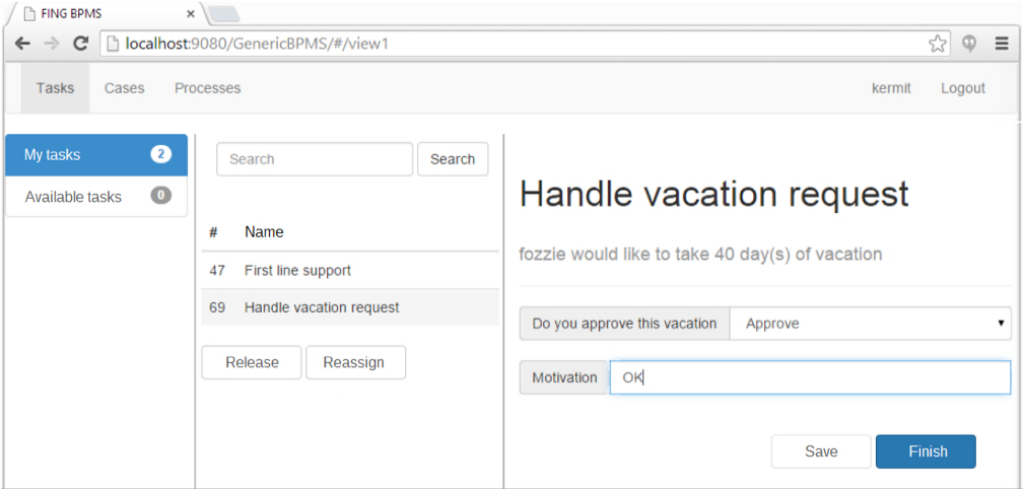

Fig. 10. Tasks view of the generic BPMS portal

of selected BPMS (Activiti, Bizagi and Bonita) with the purpose of unifying the concepts defined within their portals and process engines, based on their being representative of selected categories we have defined. Based on this analysis we have defined a generic BPMS user portal based on a unified data model and a generic process engine API for interoperability between process engines. We found that although these BPMS provides similar concepts, they differ in the way they provide access to the process engine and the functionalities they provide in the APIs.

We also defined a platform-independent front-end model for our BPMS portal using IFML. This model provides a sound basis for the generation of BPMS portals using different technologies. The example of application we presented shows the usefulness of our definitions. Next steps involves the application of model-driven

engineering techniques to generate BPM portals automatically from the IFML models in different technologies, in such a way that usability patterns and best practices are considered in advance. Differently to other proposals such as WebRatio [19], we want to decouple the user interface from the process engine to execute processes, i.e. generate only the web site including the invocations to the generic process engine API we have defined.

The benefits of our proposal are twofold: (i) the generic process engine API can be implemented to invoke different existing process engines from the generic BPMS user portal (i.e. allowing interchanging process engines within the same portal), and (ii) the generic process engine API can be accessed from user's portals implemented in different technologies (i.e. decoupling the user interface from the process engine).

We believe these two levels of decoupling provide the basis for using in any combination selected users portal with selected process engines, based on specific requirements from organizations regarding user portal design and usability, process engine languages and other capacities. For example when a complete BPMS solution does not conform to key normative in the organization, but the process engine fulfills their needs so it can be integrated into existing web portals by accessing its functionalities via the SOAP or REST API they provide, via the solution we provide. If in the future better engines are available, only the process engine can be changed.

We plan to continue improving this model in two main directions: i) comparing our definitions to other existing BPMS such as Camunda, jBPM, Intalio, among others, to enhance the generic API and model, in order to cover as many existing engines as possible, and ii) by considering areas for visualizing other elements such as BPs measures. For the latter, we can extend the unified model adding an engine-independent repository tailored for BPs execution measures analysis, both in real time and post execution. In fact, we can define an extension of the model in order to support the execution measures defined within the BP Continuous Improvement Process methodology (BPCIP) [5].

References

[1] Ceri, S., P. Fraternali, A. Bongio, M. Brambilla, S. Comai and M. Matera, "Designing Data-Intensive Web Applications," Morgan Kaufmann Publishers Inc., 2002.

[2] Chang, J., "Business Process Management Systems: Strategy and Implementation," Auerbach Publications, Taylor & Francis Group, 2005.

[3] Dabek, F., B. Y. Zhao, P. Druschel, J. Kubiatowicz and I. Stoica, *Towards a Common API for Structured Peer-to-Peer Overlays*, Lecture Notes in Computer Science **2735** (2003), pp. 33–44.

[4] Delgado, A., D. Calegari, P. Milanese, R. Falcon and E. Garcia, *A systematic approach for evaluating BPM systems: Case studies on open source and proprietary tools*, IFIP Advances in Information and Communication Technology **451** (2015), pp. 81–90.

[5] Delgado, A., B. Weber, F. Ruiz, I. Garcia-Rodríguez De Guzmán and M. Piattini, *An Integrated Approach Based on Execution Measures for the Continuous Improvement of Business Processes Realized by Services*, Inf. Softw. Technol. **56** (2014), pp. 134–162.

[6] Dumas, M., M. L. Rosa, J. Mendling and H. A. Reijers, "Fundamentals of Business Process Management," Springer, 2013.

[7] Dumas, M., W. M. van der Aalst and A. H. ter Hofstede, "Process-aware Information Systems: Bridging People and Software Through Process Technology," John Wiley & Sons, Inc., 2005.

[8] Kindler, E., B. Axenath and V. Rubin, *AMFIBIA: A Meta-Model for the Integration of Business Process Modelling Aspects*, in: *The Role of Business Processes in Service Oriented Architectures*, 2006.

[9] OASIS, *WS Business Process Execution Language (WS-BPEL)*, Technical report (2007).

[10] OASIS, *WS-BPEL product list*, http://bpel.xml.org/products (2016), [Online; accessed Sep-2016].

[11] OMG, *Workflow Management Facility Specification, v1.2*, Technical report (2000).

[12] OMG, *Business Process Model and Notation (BPMN) Version 2.0*, Technical report (2011).

[13] OMG, *Interaction Flow Modeling Language Specification (IFML) v1.0*, Technical report (2015).

[14] OMG, *BPMN product list*, http://bpm-directory.omg.org/vendor/list.htm (2016), [Online; accessed Sep-2016].

[15] Sun, Y., J. Su and J. Yang, *Separating Execution and Data Management: A Key to Business-Process-as-a-Service (BPaaS)*, Lecture Notes in Computer Science **8659** (2014), pp. 374–382.

[16] van der Aalst, W. M. P. and A. H. M. ter Hofstede, *YAWL: Yet Another Workflow Language*, Information Systems **30** (2005), pp. 245–275.

[17] van Der Aalst, W. M. P., A. H. M. Ter Hofstede, B. Kiepuszewski and A. P. Barros, *Workflow Patterns*, Distrib. Parallel Databases **14** (2003), pp. 5–51.

[18] van der Aalst, W. M. P., A. H. M. ter Hofstede and M. Weske, *Business process management: A survey*, Lecture Notes in Computer Science **2678** (2003), pp. 1–12.

[19] WebML.org, *WebRatio*, http://www.webml.org/webml/page86.do?ctx1=EN (2016), [Online; accessed Sep-2016].

[20] Weske, M., "Business Process Management - Concepts, Languages, Architectures, 2nd Edition," Springer, 2012.

[21] WfMC, *The Workflow Reference Model. Workflow Handbook*, John Wiley & Sons, Inc., 1997 pp. 243–293.

[22] WfMC, *XML Process Definition Language (XPDL)*, Technical report (2008).

[23] WfMC, *XPDL product list*, http://www.wfmc.org/XPDL.htm (2016), [Online; accessed Sep-2016].

[24] zur Muehlen, M., "Workflow-based Process Controlling: Foundation, Design, and Implementation of Workflow-driven Process Information Systems," Advances in Information Systems and Management Science **6**, Logos, 2004.