Model-driven generation of a BPMS portal based on Interaction Flow Modeling Language models

Daniel Calegari, Andrea Delgado
Facultad de Ingeniería, Universidad de la República
11300 Montevideo, Uruguay
Email: {dcalegar,adelgado}@fing.edu.uy

***Abstract*—In any organization, the use of default web applications that come with their software infrastructure, e.g., a Business Process Management Systems (BPMS), is the best option in many cases, if they fulfill the requirements of such organization. However, 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 the legacy infrastructure. In previous work we have proposed a generic BPMS user portal which can be integrated with potentially any process engine for the execution of business processes. The main objective of this paper is to provide an experience report on the use of the Interaction Flow Modeling Language and the Model-Driven-Engineering approach for the generation of such BPMS portal. In particular, we discuss the potential of plain IFML (i.e., the standard definition without platform-specific extensions) for the specification of a front-end, as well as some open issues behind its use for the automatic generation of the BPMS portal in a target platform.**

***Keywords*-Business Process Management Systems; Interaction Flow Modeling Language; Model-Driven-Engineering**

I. Introduction

Business Process Management Systems (BPMS [1]) arise as the technology to support the Business Processes (BPs) lifecycle, from modeling, through developing, deploying, executing and evaluating their execution. Common elements provided by any BPMS are a process engine where BP models are executed, a service-oriented API for accessing its functionalities, and a web portal for user's interaction (e.g., for managing a work list, and performing tasks).

Besides it is possible to begin with a business process specification and to define a front end adapted to the specific process, most BPMS provide process-independent user portals with basically the same set of core features. In Figure 1 there is an example of the Activiti BPMS[1] portal showing the process area, in which deployed processes can be executed, and the task area, in which users can manage their work list (i.e. take, reassign and complete tasks assigned to their roles.

Usually, BPMS portals are tightly coupled with the native process engine API. 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 offers a unified interaction layer for external applications, similarly to the way applications already work nowadays with many database management systems.

[1]Activiti BPM Platform. http://www.activiti.org/

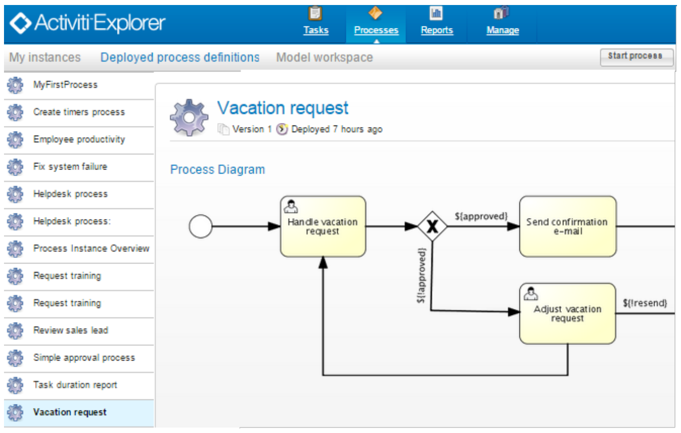
In [2] we have proposed generic (process-independent) BPMS user portal which can be integrated (loosely coupled) with potentially any process engine for the execution of business processes. It is based on a unified data model and a generic process engine API. We have modeled our generic BPMS user portal using WebML [3], a domain-specific language for web applications. WebML can be seen as an extension of the Interaction Flow Modeling Language (IFML [4]), a standard language designed for expressing the content, user interaction and control behavior of the front-end. A platform-independent front-end model gives us the possibility to follow the Model-Driven Engineering (MDE [5]) paradigm for the definition of a Model to Text (M2T) transformation for generating the portal using different technologies.

In such work we faced some difficulties. We specify WebML models using WebRatio[2], the only tool that allows the specifications of these kind of models and also the code generation to some specific platforms. However, we needed to manually implement the prototype since our target technology was not supported. Moreover, although WebRatio provides means for connecting to a BPMS, this connection was restricted to some specific BPMS, and it was not possible to configure every action on the web application to be based on services provided by our generic API.

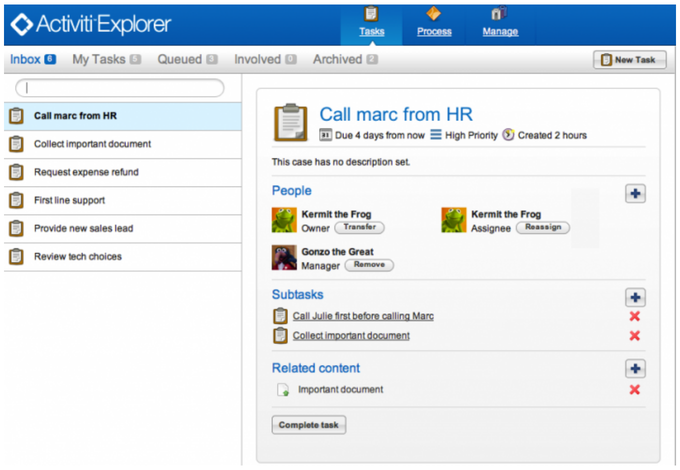
Based on these difficulties, the main objective of this paper is to provide a report on the use of IFML and the MDE approach for the generation of such BPMS portal. In particular, we discuss the potential of plain IFML (i.e., the standard definition without platform-specific extensions) for the specification of a front-end, as well as the complexity behind the definition of M2T transformations to a target platform. Moreover, we use IFML extension mechanism for expressing concrete action elements connecting the BPMS portal with our generic API, but without depending on any specific platform. Finally, we prototype the IFML models with the Eclipse open source editor[3] and define a M2T transformation with Acceleo[4]. Complete code is available here[5].

The rest of this paper is structured as follows. In Section II we review some related work. In Section III we provide a brief introduction to the IFML standard. In Section IV we overview

[2]WebRatio. http://www.webratio.com
[3]IFML editor. https://github.com/ifml
[4]Acceleo. http://www.eclipse.org/acceleo/
[5]Complete code. https://gitlab.fing.edu.uy/open-coal/IFMLModComp



(a) Process area



(b) Task area

Fig. 1. Activiti BPMS process-independent user web portal

the generic BPMS portal we proposed in [2]. In Section V we present the IFML-based specification of such a portal and in Section VI we resume the implementation of a prototype based on a M2T transformation. Then, in Section VII we discuss some aspects of interest, and finally in Section VIII we present conclusions and future work.

II. Related Work

There are several proposals for the front-end development based on the MDE approach, most of them are web engineering approaches as WebML [3] and UML-based Web Engineering (UWE, [6]) method. The IFML [4] raised the level of abstraction and becomes the standard language with a platform-independent approach. In this context, many works arise for the extension of the language, e.g., for mobile applications [7]. However, there is not so much work on the evaluation of the use of plain IFML for the front-end specification and code generation, besides some general mappings defined in the IFML book [4]. In [8] the authors define a virtual machine which translate the IFML models into bytecode that will be interpreted by the Java Virtual Machine.

As far as we know, there is not so much works on the definition of a generic BPMS portal. In [9] the authors propose basic requirements for user interface components in order to make them applicable for SOA based BPM applications, and they exemplify how these ideas can be implemented based on Java Portlets. They do not precisely define the front-end but assume that the execution of a business process can be achieved with many different ones. In contrast, we define a generic user web portal as is commonly defined in most BPMS. As in our work, they propose an architecture with a services layer connecting the front-end with the execution engine. However, they define different usage scenarios for the web services (triggering of events from UI components, synchronization of a response, etc.) and the aspects that must be taken into account, without expressing which specific services they need, as done with the definition of our API. In this sense, their ideas are more abstract than ours, and could be integrated. Finally, WebRatio supports BPM by providing an IFML-based specification of the user portal using a specific web extension of the language. Besides the specification has some similarities with ours, they do not define a generic API (they provide their own process engine) nor allows code generation to multiple platforms.

III. The Interaction Flow Modeling Language

The Interaction Flow Modeling Language (IFML) [4] aims 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. However, it does not allow the definition of graphics and styles, as well as the position and rendering of specific view components.

IFML main elements are shown in Figure 2. An IFML diagram consists of one or more top-level view containers. These containers can be set as landmark containers (tagged with a L near the containers name), i.e. they can be accessed from any other container, e.g. through a link. Moreover, they can be composed by other containers, e.g. a main window with many nested windows. These composition can be set as an alternative (tagged with XOR), i.e. only one nested container is shown at a time. Also, containers can be set as default (tagged with D), i.e. they are shown by default, e.g. the main window. 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.), and view component can have many view component parts (e.g. a simple field of a form or a column in a list). Events are attached to containers and components (e.g. when an element of a list is selected) affecting the state of the user interface and possibly causing the triggering of an action (i.e. a piece of business logic triggered by an event which may reside on the server or on the client side, e.g. deletion of the selected element from the list, or a database update). The state change caused by an event, and 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 flows between view elements and actions bound to

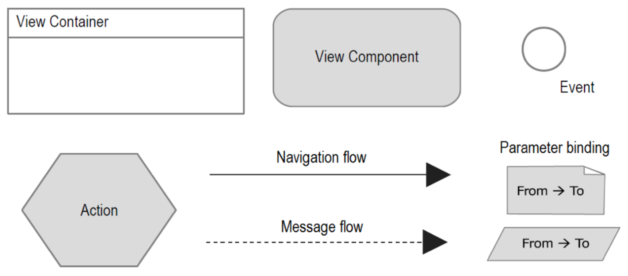


Fig. 2. Main IFML concepts

navigation flows as parameters (typed and named values). View components and navigation flows are connected with a domain model which expresses the domain elements and behaviors for which the IFML model is defined.

WebML [3] is a domain-specific language for web applications which can be defined using the IFML extension mechanism. By using IFML, it is possible then to automatically generate most of the front-end for different technologies. As mentioned before, currently there are only two implementations of IFML: WebRatio which provides a tool suite for WebML modeling and full code generation; and the Eclipse IFML editor providing IFML modeling capabilities only.

IV. A Generic BPMS User Portal

The generic BPMS user portal we proposed in [2] follows the architecture shown in Figure 3.

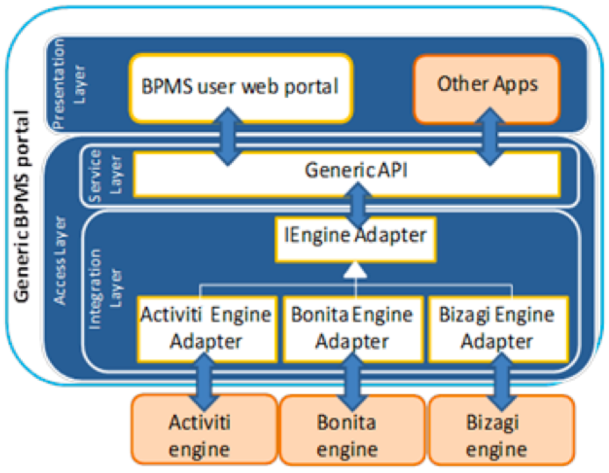


Fig. 3. Generic BPMS portal Architecture diagram

At the Presentation Layer level, the BPMS user web portal provides the user with the set of functionalities needed to manage BPs and activities cases, worklists, 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. 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. Interoperability is achieved based on a unified data model which was defined as an abstraction of concepts from different process engines [2] (shown in Figure 4).

The generic API provides general services such as logging facilities, and specific services 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 which have to be implemented for each selected process engine.

Adapters provide the functionalities defined in the interface by invoking the selected process engine, 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. Functionalities not provided by the selected process engine can be disabled in the user portal. In this way, our BPMS generic portal does not provides yet another process engine implementation, but a clean integration with existing ones.

The Presentation Layer was formerly expressed using WebML and manually implemented using HTML5 and Javascript, as well as other frameworks for structuring code (AngularJS[6]) and style sheets and layout definitions (Bootstrap[7]). We have also implemented a concrete instance of our unified process engine API as a HTTP REST API using Jersey[8], and an adapter for the Activiti process engine. The data model was implemented using the JavaScript Object Notation (JSON[9]) as a lightweight data-interchange format.

V. IFML-based Specification

The IFML models were specified using the open source IFML editor. In what follows we present an excerpt of the models, complete enough for exemplifying the use of the main IFML elements depicted in Figure 2, and for defining the M2T transformation in the next section. A more complete definition of the original WebML models can be found in [2].

The generic BPM portal begins with the definition of a home view which contains a login form which can be used by users to access the site, as shown in Figure 5. The "Login" action has a specific dynamic behavior which allows to use the generic API to login a user to the BPMS. Within IFML actions are defined in an abstract way. However, when the generator detects such behavior, it generates the corresponding logic for login. In general, since we are expressing a presentation layer on a specific and fixed domain (not depending on the specific business process that will be executed), actions are fixed and defined in terms of our generic API.

[6]AngularJS. https://angularjs.org/
[7]Bootstrap. http://getbootstrap.com/
[8]Jersey. https://jersey.java.net/
[9]JSON. http://www.json.org/

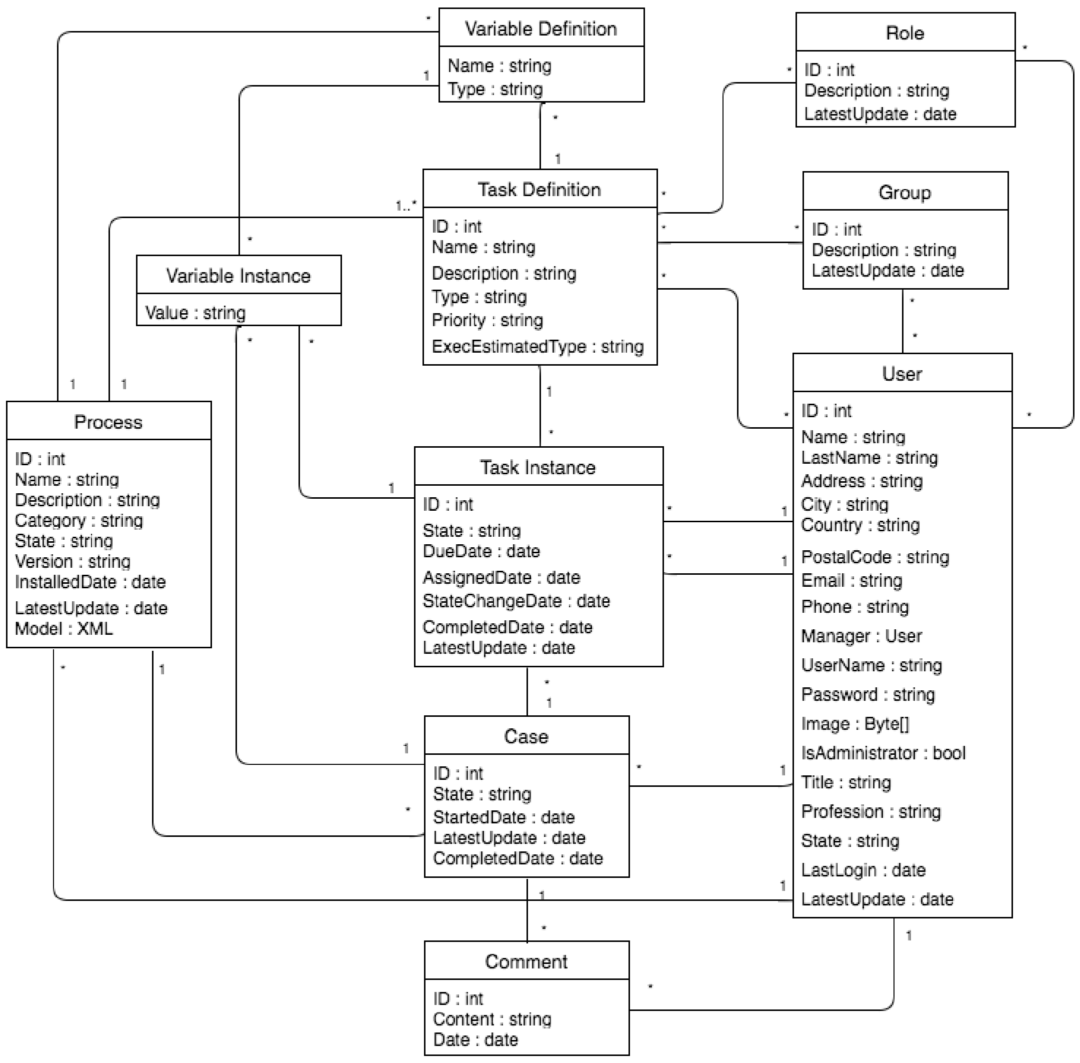


Fig. 4. Unified data model for a BPMS portal in a conceptual UML model

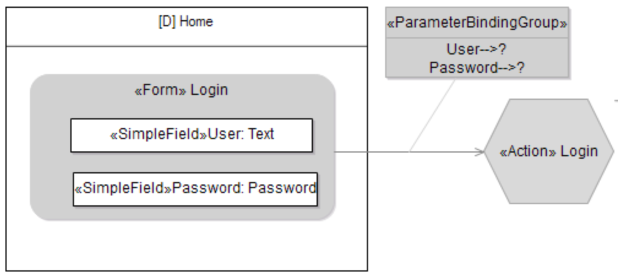


Fig. 5. IFML login model

When a user is logged in, it goes to the Common User view which provides the functionalities for users of the BPMS, e.g., list of assigned tasks, perform a task, among others. It 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 was exemplified for Activiti BPMS in Figure 1. The most important functionality of a web portal for regular users is the work list, in which users can list, take, reassign and complete tasks assigned to their roles (or specifically to them).

In Figure 6 we present the IFML modeling of the processes area, which allows to search for a specific process (represented with a Form component), shows a list of processes (represented as a List component) with their corresponding id, name and version, and allows to initiate an instance of any of them (represented with a navigation flow from the process list to the process form, together with a data binding for identifying the corresponding process instance). Once again, we express a "Start" action in an abstract way, but it has a specific dynamic behavior for starting the process using our generic API.

VI. M2T TRANSFORMATION PROTOTYPE

We defined a M2T transformation from IFML to Java Server Faces (JSF[10]) which allows us to generate a front

[10]Java Server Faces. http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

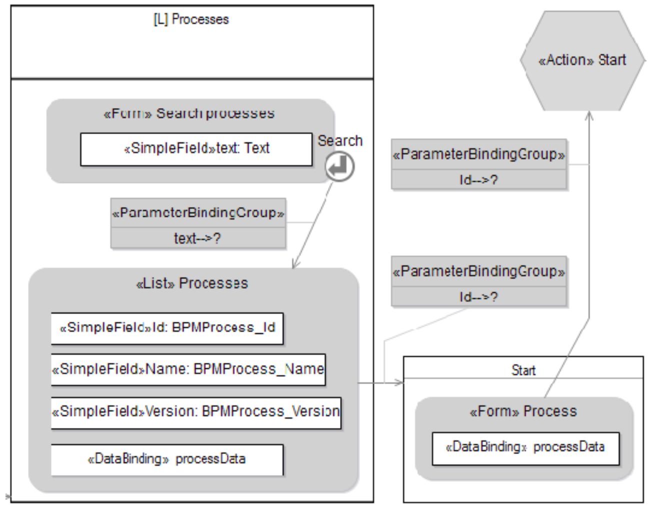


Fig. 6. IFML processes area model

end following the Model View Controller (MVC) pattern. We selected this technology because based on MVC it provides a clear separation between the view (pages) of the presentation and its logic. The business logic of our application is assumed to be provided by the generic API, which can be accessed through web services as in [2], and it is consistent with the data model in Figure 4.

A. M2T transformation

The M2T transformation is basically defined as follows:

- **Windows (View Container)** are translated into web pages. For each window we generate two files: one with the view (XHTML page) and another with the controller (Java code). Since within a window there could be other components, their generated code is included within the same two files. When the XOR property of a window is checked (a disjoint visualization of views), we generate a lateral bar from which each of the XOR views can be selected. Moreover, if the modal property is checked, we generate a pop-up (modal) page. Finally, if the landmark property is checked, meaning that the container must be accessible from any other page, we add a link to this page in a menu bar located in the JSF application header.
- **Form** is a view component inside a view container, e.g., a window. For each form we generate a HTML form with their corresponding fields (i.e., SimpleField in the IFML specification). Each field is bound to a domain element attribute within the domain model, thus we generate the code for binding form fields to Java beans.
- **Action** we use specific actions that are mapped to specific services within our generic API through the dynamic behavior property of the action. In this way, we generate the code for service consumption when for example a button is pressed. We also bind input and output data of the action to the corresponding input and output parameters of the service, respectively. Input and output data is represented with the datatypes generated from the domain model. Since the action has a navigation flow from it to another window, we also generate the automatic flow to the corresponding page (potentially the same).
- **List and Details** are translated into their corresponding JSF components (a list or a table). As in the case of the form, each of their simple fields are bind to some domain concept attribute.
- **Navigation flows** are basically translated into buttons. In the case of a navigation flow from a window, we generate a button in the top right part of the page. In some cases, e.g., when navigating from a list to an action or window, we interpret the navigation flow as a repetitive action, thus we generate a button for each list element.
- **Authentication** there are some aspects of the application which cannot be expressed in plain IFML, as for example authentication controls in each page. Thus, our transformation generates a code fragment for validation which is included within each page and connected with the authentication service of our generic API.

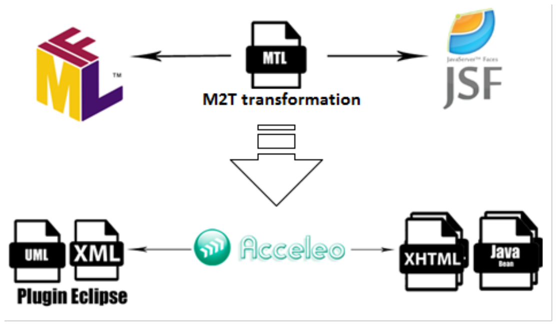


Fig. 7. Prototype generation process

B. Code generation

From the technical perspective, the M2T transformation from IFML to JSF is implemented using Acceleo, an implementation of the MOF Model to Text Language (MTL) standard within the Eclipse environment. As depicted in Figure 7, the generation process takes as input a set of files produced by the Eclipse-based IFML editor representing the IFML metamodel and a specific model, and provides as output a JSF front-end composed by XHTML pages and Java components.

An example transformation rule is depicted in Figure 8. The rule queries the navigation flows defined within a window and, for each one of them, defines a navigation button to its corresponding target flow element (in this case: a window that will be transformed into a web page). We used Bootstrap components and style sheets for adding responsive behavior.

After the M2T is performed, a web project is automatically generated with the web pages and resources for executing the generic BPM portal. By setting defined configuration files we can indicate the concrete BPMS process engine that will be used, since the generated beans invoke the operations defined in the generic API. We encapsulated the invocations in a set of utility classes that are used throughout the site beans, to obtain data from the process engine.

```
[comment Links to other pages in the upper right corner /]
[if (element.oclAsType(IFMLWindow).eContents(NavigationFlow)->size()>0)]
<h:form class="row">
<div class="pull-right btn-group" role="group" aria-label="...">
   [for (element2 : InteractionFlowModelElement | element.eContents())]
      [if (element2.oclIsTypeOf(NavigationFlow))]
         [if (element2.oclAsType(NavigationFlow).targetFlowElement.oclIsTypeOf(IFMLWindow))]
         <h:commandButton class="btn btn-default" value="..." action=""/>
         [/if]
      [/if]
   [/for]
</div>
</h:form>
[/if]
```

Fig. 8. Example of a transformation rule using Acceleo

An example of the JSF front-end generated from the IFML model in Figure 6, is the one depicted in Figure 9. The processes area is a landmark page which can be accessed from the menu bar. There is a logout button in the top right part of the page representing a navigation flow between the processes window to the logout window (not shown in the IFML models). The search form allows updating the list of processes once the button is pressed. Within the processes list there is a repetitive action for starting a process, represented as a button on the left side of each element.

VII. Discussion

In what follows we discuss several aspects that arise from our experience and describe many open research problems.

IFML has a great potential and WebRatio is a powerful tool for supporting the whole life cycle of IFML based applications. Unfortunately it is the only one, and from a research perspective it has many drawbacks: it is not open source, their metamodels are not available, and there is no information about how the M2T transformations are defined.

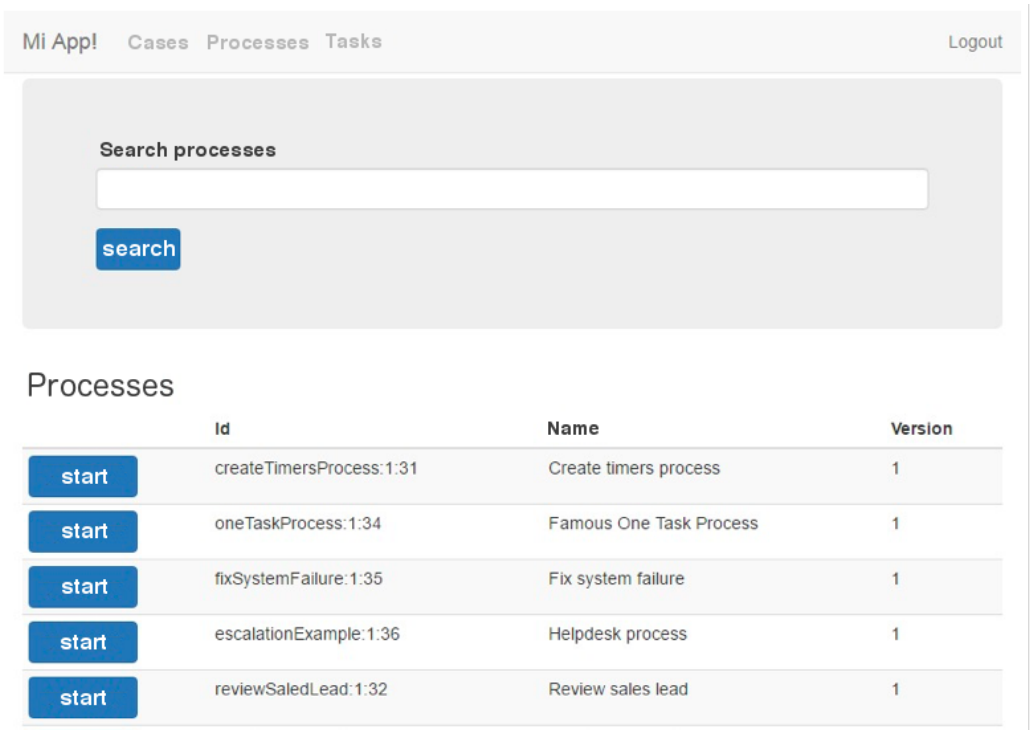
One of the open issues (resolved somehow by WebRatio) is the connection between IFML and other front-end aspects as graphics, styles, position and rendering of components. In our experience, the main problem when using plain IFML or one of their extensions for code generation is that many implicit decisions have to be taken, or postponed for a future step, e.g. adding styles in the resulting web application. However, it could be beneficial to have these aspects modeled and connected to the IFML model, but not as an extension of it (e.g. as a complementary model). In fact, the extension mechanism provided for IFML has some drawbacks from our point of view, as we will discuss next.

Plain IFML, i.e. the standard definition without any extension, is enough for expressing many things, but as mentioned, it is also restrictive. There is a need for lowering the abstraction level in order to express platform-specific aspects, such as: first, the technology to be used, and second, the kind of environment in which the front-end will be executed. With respect to the second one, IFML extensions add concrete actions and components for expressing mobile or web aspects, independent of any given technology. However, once a model is built for a given environment, it must be rebuilt for another, e.g. the use of specific view components, although many of the aspects already expressed do not change, e.g. the navigation between containers. In some approaches, e.g. UWE [6], the presentation and the navigation models are distinguished, whilst in IFML are tightly coupled. In this sense, it is an open problem how to adopt a multi-modeling approach for expressing platform-specific aspects together with the other front-end elements discussed before.

Based on the existent IFML extension mechanisms, it is possible to come up with an IFML extension for modeling BP execution, in connection with the generic process engine API defined in [2]. In this way, we allow the integration of any IFML front-end with potentially any process engine, not necessarily using our BPMS portal definition. For this we need to extend actions with their respective data bindings (input and outputs) for login, creation of cases and tasks, assignation and completion of tasks, among other operations. We can also define some actions and view components for common queries, e.g. for querying the list of pending tasks for a user. Moreover, it could be useful to encapsulate some parts of the BPMS portal into so called portlets, e.g. the task list, in such a way that they can be reused in any IFML application, as proposed in [9] with respect to the definition of GUI components for SOA based BPM applications.

With respect to the definition of M2T transformations, Acceleo is a very powerful tool but his template-based approach can be tricky if there is no previous intention of reuse (e.g. lack of explicit modularization). In particular, it is desirable to define a library for querying the source model, i.e. in the case of IFML, for retrieving every container, every navigation flow from a given component, and so on, which can be reused no matter the target of the transformation. If not, templates result in a mixed combination of explicit model queries and generated text which is hard to read and does not allows reuse.

Finally, from a component perspective, the BPMS world provides an example of componentization in which process engine functionalities can be abstracted and the user interface can be reused in new contexts. The generic API is the most



Processes

	Id	Name	Version
start	createTimersProcess:1:31	Create timers process	1
start	oneTaskProcess:1:34	Famous One Task Process	1
start	fixSystemFailure:1:35	Fix system failure	1
start	escalationExample:1:36	Helpdesk process	1
start	reviewSaledLead:1:32	Review sales lead	1

Fig. 9. Generated JSF processes area

valuable component within the architecture in Figure 3. In order to enable different BPMS providers we can follow an ontology-based approach for mapping their engine services against a set of ontologies in order to support the discovery and invocation of services. This is intended as future work.

VIII. Conclusions and Future Work

In this paper we presented an experience report on the use of IFML and the MDE approach for the generation of a generic BPMS user portal which can be integrated (loosely coupled) with potentially any process engine for the execution of business processes. We provided a plain IFML specification of such a portal and a the definition of an Acceleo M2T transformation for the generation of a JSF-based front-end.

We found that plain IFML has great potential for the generation of a front-end, but there are many issues that must be resolved for improving such front-end. In this sense, we discusses several open problems related with the inclusion of complementary metamodels (e.g. for styles) and the use of a multi-modeling approach for expressing different abstraction levels. We also discuss the possibility of defining an IFML extension for modeling BP execution in an abstract way.

Since IFML is a high-level language with a visual representation, it brings UI experts closer to non-technical stakeholders. However, it is possible to go one step further considering a mockup-driven approach in which IFML can be considered an intermediate model for representing certain aspects of a front-end. In this approach, model-to-model transformations could be defined from some mockup language to IFML and other languages in the multi-modeling approach we discussed.

IX. Acknowledgements

We would like to thank the students who worked in the project regarding the prototype implementation: Adrian Dos Santos, Patricia Rolandi and Guillermo Serratto.

References

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

[2] A. Delgado, D. Calegari, and A. Arrigoni, "Towards a generic BPMS user portal definition for the execution of business processes," *Electr. Notes Theor. Comput. Sci.*, vol. 329, pp. 39–59, 2016.

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

[4] OMG, "Interaction Flow Modeling Language Specification (IFML) v1.0," Tech. Rep., 2015.

[5] S. Kent, "Model driven engineering," ser. Proceedings of Integrated Formal Methods 2002, 2002, pp. 286–298.

[6] N. Koch, A. Knapp, G. Zhang, and H. Baumeister, "Uml-based web engineering - an approach based on standards," in *Web Engineering: Modelling and Implementing Web Applications*, ser. Human-Computer Interaction Series, G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, Eds. Springer, 2008, pp. 157–191.

[7] M. Brambilla, A. Mauri, and E. Umuhoza, "Extending the interaction flow modeling language (IFML) for model driven development of mobile applications front end," in *Mobile Web Information Systems - Proc. of 11th Intl. Conference, MobiWIS 2014*, ser. Lecture Notes in Computer Science, I. Awan, M. Younas, X. Franch, and C. Quer, Eds., vol. 8640. Springer, 2014, pp. 176–191.

[8] S. Gotti and S. Mbarki, "Toward IFVM virtual machine: A model driven IFML interpretation," in *Proc. of the 11th Intl. Joint Conference on Software Technologies (ICSOFT 2016)*, L. A. Maciaszek, J. S. Cardoso, A. Ludwig, M. van Sinderen, and E. Cabello, Eds. SciTePress, 2016, pp. 220–225.

[9] J. Hohwiller and D. Schlegel, "Integration of UI services into SOA based BPM applications," *Lecture Notes in Business Information Processing*, vol. 97 LNBIP, pp. 53–64, 2011.