SFDL: MVC APPLIED TO WORKFLOW DESIGN

Diego Moreno, Emilio García, Sandra Aguirre, Juan Quemada
Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid
Avenida Complutense, 30 – 28040 Madrid – SPAIN
{dmoreno, egarcia, saguirre, jquemada}@dit.upm.es

ABSTRACT

Process management based on workflow systems is a growing trend in collaborative environments. One of the most notorious areas of improvement is that of user interfaces, especially since business process definition languages do not address efficiently the point of contact between workflow engines and human interactions. With that in focus, we propose the MVC pattern design to workflow systems. To accomplish this, we have designed a new dynamic view definition language called SFDL, oriented towards the easy interoperability with the different workflow definition languages, while maintaining enough flexibility to be represented in different formats and being adaptable to several environments. To validate our approach, we have carried out an implementation in a real banking scenario, which has provided continuous feedback and enabled us to refine the proposal. The work is fully based on widely accepted and used web standards (XML, YAML, JSON, Atom and REST). Some guidelines are given to facilitate the adoption of our solution.

KEYWORDS

Model-View-Controller (MVC), Workflow, Open Architecture, Group Management, Collaborative Environment, User Interface, SFDL

1. INTRODUCTION

Collaborative Working Environments play an increasingly important role for successful business, usually involving teams across organizations. Nowadays, people in those teams have frequently to face the challenge of being geographically spread, and they just want to avoid the necessity of physically meeting to carry out their duties. In this context, computer assisted workgroup management becomes notably relevant. Moreover, some specific technologies, such as those supporting a distributed workflow execution, turn out to be crucial: any optimization in those techniques multiplies enormously the benefits obtained by applying them to the collaboration between individuals and workgroups.

Team management exposes the critical nexus between the human web and the web of data and services; it raises the need of open architectures that enable interoperability and interaction among all the elements composing a virtual organization, i.e. the cooperation of individuals belonging to different entities, companies, departments or projects. This is achieved by using open and standard protocols, interfaces, and definition languages.

The work described in this paper proposes the Model-View-Controller architecture (MVC) (Krasner & Pope 1988), as a means of achieving an integrated solution to define a user interface –the view-, ideally web-form based, together with a number of workflow processes that will handle all the logic –the controller-, and providing access to the data –the model-. At this point, two particularities must be taken into account: first of all, workflow engines can be found in quite different environments, ranging from human interaction through web-based environments to machine-to-machine communications using Web Services. Second place, user interfaces should be able to integrate in complex systems, such as workflows. They should have the possibility, for example, of executing functions to obtain information from the workflow engine.

Workflow definition languages –BPEL (Andrews, et al., 2003), XPDL (Workflow Management Coalition 2002)- usually take care of providing the logic behind the processes, lacking the capability of defining user interfaces. This ability is delegated to other languages, such as HTML or XForms in the case of web environments. Having analysed different user interface definition languages which can be used in this MVC approach, our study concluded that none of them was optimised for workflow environments. Therefore, we

studied the creation of a view definition language which allowed for the dynamic generation of user interfaces, while seamlessly integrating in a workflow execution engine. This language, the Simple Form Definition Language (SFDL) (Moreno, et al., 2009) has two main characteristics:

- It has three alternative representations: XML, YAML (Ben-Kiki, et al., 2005) and JSON (Crockford, 2006), which contribute to its adaptability, making it optimum for several scenarios.
- It integrates server-side function execution, enabling interoperation with the workflow engine logic, and adding to the dynamic aspect of the language.

Current article goes beyond the formal specification of the language, proposing an integration path for SFDL into a generic workflow enactment service. As a means of validating the proposal, we have carried the implementation in a particular research project, a software development platform in a banking environment, where a significant number of professionals actively use workflows to coordinate themselves in their day-to-day activities. Thus, this proposal has benefited and been refined from real world feedback; the banking process requirements have established the basis to our work: workflows should be defined nimbly, providing the designers with the control over all the workflow development process, from functionality to end-user interface design. The proposal is completely compatible with the Reference Model (Hollingsworth, 1994) from WfMC, guaranteeing a straightforward integration in compliant systems. It is based on standards such as Atom (Sayre, 2005), REST (Fielding, 2000) and Wf-XML-R (Zukowski, et al., 2008).

The remainder of this paper is organized as follows: Section 2 describes the related works. The methodology followed is introduced in Section 3, along with the requirements that served as a starting point to the approach. Section 4 describes the proposed language, SFDL. Section 5 provides some guidelines for SFDL adoption, before concluding with Section 6.

2. RELATED WORKS

Workflow systems, as expressed by the WfMC Reference Model, are strongly influenced by the languages and protocols chosen for its interfaces. Process definition languages play a role as the common format for the workflow definition interchange between the process definition tools and the runtime workflow management. For this work, we have considered several standards: BPEL is the OASIS proposal for this interface, oriented to Web Services interactions; XPDL is the XML process definition language proposed by the WfMC, and adds both graphics and semantics to the business process representation. Other alternatives include UML, which a more general-purpose orientation, or OpenWFE (Wohed, et al., 2009), which leads among open-source alternatives. It must be noted that the extensions proposed in our work are oriented towards the View component of the suggested MVC approach, therefore being completely valid for any of the definition languages.

Forms are the primary means for creating interactive web-based user interfaces. Different ways to generate Web forms are available, but currently there is a demand for dynamism, which plain static HTML pages lack. XForms (Dubinko, et al., 2003) is a recommendation of the W3C, within XHTML specification, that separates presentation from content, allows reuse, gives strong typing –reducing the number of round-trips to the server- and offers device independence and a reduced need for scripting. Nonetheless, its support in actual web browsers is practically inexistent and, while being based on XML, it needs the support of other languages -CSS for styling, XSLT for dynamic form generation-; its complexity is non-trivial for workflow process designers. HTML 5.0 Forms (Hickson, 2009) has an increasing support in browsers, aiming to reduce the need for other proprietary, but popular solutions, such as MXML(Coenraets, 2003); however, its forms cannot be easily serialized to be processed in other languages, such as JavaScript, and does not provide advanced function support (e.g. does not include nested functions calls).

Our proposal for dynamic view generation must overcome the abovementioned limitations, with a primary focus on the integration with any of the definition languages that a workflow designer might choose.

3. METHODOLOGY

Our scenario is developed into the ITECBAN project, which is aimed at providing the banking core building process with software tools oriented towards the collaborative activities of a Virtual Organization.

ITECBAN must support different collaborative activates such as the software development process inside a banking core, videoconference, content management, etc. Taking into account the target scenario for this project, the workflow management system had to satisfy, at least, the following functional requirements:

- Design of forms which allow the specification of tasks and rules, including the user's roles needed for their execution and the input and output data types.
- User access through any web browser.
- Easy creation of new flows.
- Use of an open source workflow management system.
- Connection with different databases as MySQL, LDAP and CMDB
- Workflow management of incidences, changes, problems and job orders.

In order to identify the variables, policies, roles, process and components that should be taken into account in the workflow management system of this scenario, we analysed the aforementioned functional requirements and workflows depicted as flow charts. Each workflow should be finally deployed as a group of dynamic views orchestrated according to the process logic. Figure 1 shows the incident management flow chat and the dynamic view deployed in the "Incidence Register" process.

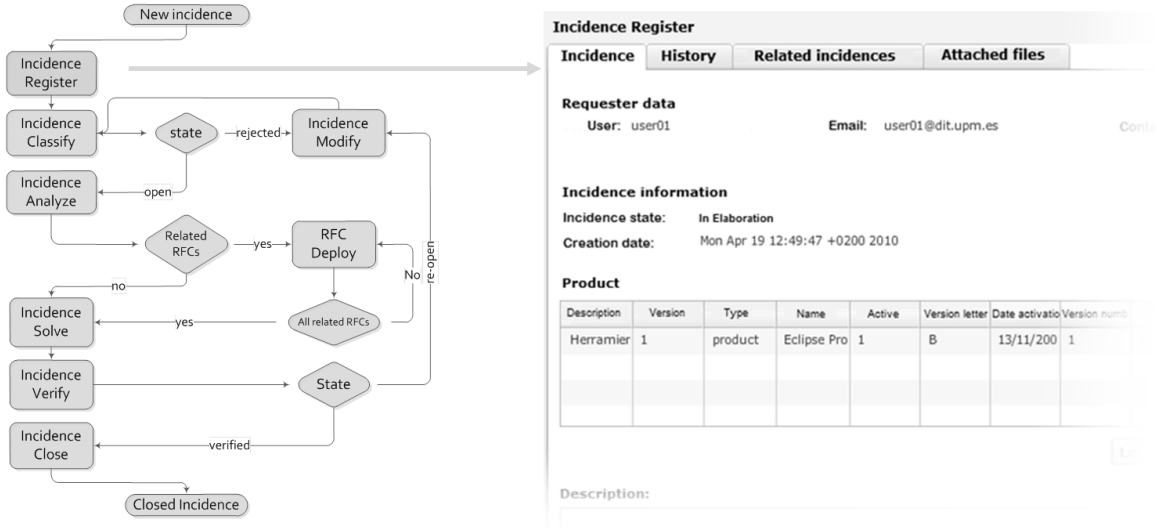

Figure 1. Incident management flow chart and "Incidence Register" view

The components of our workflow system can be defined following the MVC approach: variables and policies represented as objects make up the model; the workflow process becomes the controller; and the view is defined by the user interface elements. A wide range of workflow languages allows defining the Model and Controller components. However, having analysed different view definition languages, we concluded that none of them was optimized for workflow environments. This was the main motivation of our approach, the creation of a view definition language called SFDL, which allows the dynamic generation of user interfaces and can be easily integrated in any workflow execution engine.

Beyond the formal specification of the language SFDL, we propose an integration architecture completely compatible with the WfMC reference model, focusing on those entities and interfaces found more relevant for this scenario. The developments have been made and validated with OpenWFE as a base; nevertheless, there is the strong requirement of guaranteeing a high degree of portability to other process definition languages.

4. SFDL: SIMPLE FORM DEFINITION LANGUAGE

The focus of our work is pointed towards the joint between process definition and modelling tools, and the workflow enactment service. Our proposal covers two main aspects, both related to the MVC pattern:

web form generation, for flexible view generation from the design process itself; and basic operations from the language to access the model. This way, a workflow designer can establish not only the interaction pattern, but also define the basic interfacing rules for the end-user to trigger all the functionality.

The developments have been made and validated with OpenWFE as a base; nevertheless, there is the strong requirement of guaranteeing a high degree of portability to other process definition languages. According to Van Der Aalst (2003), every system has the possibility of executing external functions, be it though embedded code, or calling an ad-hoc participant specially coded for that. The latest is the one chosen here: the workflow definition language will include special calls to a function-participant which will load data from an external file. Data from this file will define:

- Screens/views for the end-user, i.e. the form-based GUI.
- Actions/interactions which the end-user can do in each screen.
- Finally, the functions that will be executed to access the data model, with the results to be presented to the user, and the inputs which will be originated from his actions.

With all this taken into account, a new language has been defined, covering all these aspects in a clear and simple way. Additionally, as it will be seen in the following sections, this language can be the base for others in the Reference Model, maintaining thus the coherence in all the workflow chain: design, implementation and presentation.

4.1 Language definition

Simple Form Definition Language (SFDL) is a special purpose language defined taking into account all the functional requisites formerly detailed. In particular:

- It supports a number of web form elements: selectors, tables, choices, input/output fields…
- It is self-contained: it has all the information needed –components and style- in a single file.
- Multiple screens per view: a single activity in a workflow can be composed of several screens in the client, before sending information back to the server.
- It can be expressed in a number of markup languages –XML, JSON, YAML- which, while completely equivalent in functionality, have their own particularities that make each one of them more adequate for a different environment. Moreover, using standard languages simplifies the processing load on the server.
- Supports server-side function execution, to manage the data model from/to the views.

From the process definition language, OpenWFE, files are loaded with the aid of a special participant (Figure 2).

```
<participant ref="load_sfdl_view"
             external-file="vista01.sfdlx" />
```

Figure 2. Load of an SFDL file

Files can be defined in SFDL in one of three possible variants:

- SFDL-X: with an XML markup, it is adequate for its interoperability among platforms, and keeping the same format used for process definitions, and the language used in Interface 2.
- SFDL-J: using JSON, it is optimized for JavaScript, with a syntax that allows for great bandwidth savings (up to 50% compared to XML), and easily parsed.
- SFDL-Y: defined in YAML, with a very easy indent-based syntax, which can be directly translated into JSON.

4.2 View definition in SFDL

To define views inside the activities of a workflow a simple label-based schema is used in order to indicate the position of each element, its type, value and some parameters which can define more precisely its style or functionality. Table 1 summarizes all the fields.

Table 1. Fields of a view

Tag	Description	Values
type	Functionality of the element	Label, input_text, text_area, text_block, selector, table, dynamic_table, link, attach, checkbox
params	Style of the element	halign, width, height, hint…
value	Element value	Numerical, alphanumerical, functions
result	Result of the user interaction with the element	

Each element is preceded by a numerical identifier which defines the position which it will have in the client screen. As an example, Figure 3 shows the SFDL-Y definition of a label-type field with a value obtained from the result of the *user-data* function. The element will be positioned in coordinates (04, 30).

```
- id: 0430
  type: label
  value:
    function-name: user-data
    attribute-name: telephone
  params:
    halign: left
    width: "60"
```

Figure 3. SFDL-Y field definition

At this point it is important to highlight the fact that all SFDL-* formats are equivalent. Figure 4 shows the definition of the same field in SFDL-X (XML), with the particularity that this format, once the function is processes, will be identical to the one sent to the client through Interface 2.

```
<field>
  <_>
   <id> 0430 </id>
   <type> label </type>
   <value>
    <_>
     <function-name> user-data </function-name>
     <attribute-name> telephone </attribute-name>
    </_>
   </value>
   <params>
    <halign> left </halign>
     <width> 60 </width>
   </params>
  </_>
</field>
```

Figure 4. SFDL-X field definition

It can be seen that SFDL is easily extensible, and the creation of new elements with their parameters is quite straightforward.

4.3 Data model access functions

One of the most important requirements for SFDL is enabling access to the data from form definitions. In our MVC approach, this is done through the controller: functions are invoked from the view, implemented in the controller, and access the model which covers all the databases.

At this point, it is necessary to clarify the two possible ways to execute functions:

- At workflow processing time, when the workflow engine runs the definition
- At presentation time, when the form is presented to the end-user

To handle both behaviours, a mechanism has been defined to call functions both from the OpenWFE language and the SFDL definitions, following a functional model.

4.3.1 Workflow time functions

Function calls from the definition language have been implemented as references to a special participant, being this one the most straightforward method in OpenWFE. Nevertheless, generality of this approach is guaranteed in the sense that every language has some way of adding external functions. Keeping in line with the example from Figure 4, the function call to obtain the phone number from a user would be the one shown in Figure 5:

```
<participant ref="functions"
             function-name="user-data"
             attribute-name="telephone"
             out-field="phone"/>
```

Figure 5. Workflow defined function call

Functions implemented in the engine cover all the basic input/output operations from/to the model and databases used in the architecture (LDAP, CMDB): *read-attribute*, *write-attribute*, *cmdb-out*, *user-data*… however, the function definition mechanism, to be described in next section, allows for a nearly trivial expansion of the call set.

4.3.2 Presentation time functions

Functions to be executed when the user opens a specific view are defined in the same language as that view: SFDL, in any of its variants (-X, -J or -Y). Examples are shown in Figure 3 and Figure 4. This approach has two positive aspects of great utility:

- Functions can be nested and, being a functional language, they can be used in any point of the SFDL view definition in place of a value
- There is a single function library in the system, with a single set of call names and parameters (i.e. a call interface), so that calls from SFDL are identical to those from OpenWFE, bringing consistence to the approach

5. IMPLEMENTATION AND RECOMMENDATIONS

From the beginning, the architecture has been optimized to obtain an independent design from workflow engine selected. This independence is achieved reducing the coupling with the engine, limiting the integration points. Thus, any engine can adopt SFDL with minimal changes. Anyway, in this section we impart some recommendations to make an implementation, using our experience as a base. Recommendations will go in two directions: first, what is the easiest way that a workflow engine adopts SFDL and, secondly, what are the recommended interfaces for interaction with other systems or users.

First, it is desirable to package all the SFDL executable functions in a common library which must also be accessible at workflow process time. Then, it is necessary a module, you can call Function Trigger. Its function will be receiving a call function with SFDL syntax and run that function through the common function library, returning a result. This Function Trigger module should be accessible at workflow process time and presentation time.

Furthermore, the representation of the workflow engine resources can be done under any format you choose, but here there are some guidelines to achieve a standard integration with SFDL.

Nowadays, web browsers are becoming the standard tool for consuming Internet services, so it is logical to create a web-based client application to access to process engine. Use of REST interfaces is recommended for the communications with this type of clients. They use HTTP at transport layer, which is native browser protocol. Also, they are a kind of light web services that give the server scalability and efficiency.

One recommended possibility to workflow engine with SFDL support is a REST interface based on Wf-XML-R. Wf-XML-R is a Wf-XML (Swenson, et al., 2004) adaptation, designed for communication among different workflow engines (interface 4 within the Workflow Reference Model of WfMC). But it is possible to use Wf-XML-R into communication between engine and clients (interface 2). This is because, according to the reference model, all interfaces have a common set of calls within the WAPI, and each one differs from others in the particular functions that it adds. Keeping within the common set of functions, there is no problem in using a protocol from an interface into another.

Wf-XML-R represents the resources with Atom and AtomPub. The use of Atom has an important advantage: it admits extensions easily. Consequently, a resource representation with basic Atom can be complemented with extensions to accommodate the nuances every resource has. That is, if a resource cannot be adequately described with the basic Atom protocol, extensions can be created which allow the complete representation.

The Atom extension system is based on XML namespaces (Bray, et al., 2006). Each extension creates a new namespace, where new tags are admitted. As the representation of the necessary variables for the generation of dynamic forms is beyond the scope of Atom and Wf-XML-R, a new extension, and its related namespace, was created for SFDL support into Atom. With this extension all new information is inside the *v:current_view* tag. This tag accepts two attributes:

- Type: SFDL-X, SFDL-J, HTML, XFORM. This attribute specifies the format of the contained form. It accepts a variety of formats to make more versatile this extension.
- Screen_id: it is the identifier of screen witch belong the contained fields. Thanks to this attribute it is possible to support multiple screens.

Within the label *v:current_view* there will be the elements that make up the form. We recommend the usage of SFDL-X in this scenario because of its perfect integration with XML and Atom (see Figure 6)

```
<entry
   xmlns = "http://www.w3.org/2005/Atom" xmlns:g = "http://geobpms.geobliki.com/1.0"
   xmlns:v = "http://localhost:3000/scheme">
  <id>tag:localhost,2005:WfElement/1</id>
  <link href="http://localhost:3000/workitems/2.atom" type="application/atom+xml" rel="self"/>
  <author><name> bob </name></author>
  <title type="text"> Vacation Request </title>
  <updated>2008-09-01T16:14:20Z</updated>
    <g:item_type> workitems </g:item_type>
  <v:current_view type="sfdl-x" screen_id="1">
    [fields in SFDL-X]
  </v:current_view>
<entry>
```

Figure 6. SFDL-X representation into Wf-XML-R

6. CONCLUSION

The result of our investigation has been the specification of a new dynamic view definition language, SFDL, optimized for workflow engines. Our architectural design, based on the MVC paradigm, has allowed us to fulfil all the initial requirements, and has been validated, leading to significant gains in productivity. Furthermore, it provides an improved communication between workflow systems and users.

Our work is based on widely used standards and open proposals, through the usage of protocols such as REST, Wf-XML-R and Atom, and using XML, JSON and YAML as starting points for the definition of the view in SFDL. On the one side, through workflow definitions –our controller-, a workflow engine can now generate dynamic forms with enhanced usability, and the possibility of interacting with web services and the database model. On the other side, a user with a web browser can interact with the workflow system, and the model, using the dynamic forms.

The proposed design has received much feedback, because it has been validated with a real banking scenario. It has allowed some huge features. Among these, one of the most important is providing workflow designers with the means for including views specification into their process definitions. Also, it has allowed establishing a set of data-access methods into the language to ensure that the workflow is dynamic, accessing to the full data-model offered in the environment. Finally, it keeps it simple enough not to depart with the Reference Architecture (e.g. by reusing protocols such as Wf-XML-R), not sacrificing portability (by not introducing harsh modifications tied to a specific engine); and maintaining the flexibility and extensibility which are essential in this kind of project. This way, we recommend the use of these validated formats and architectures, within scenarios with similar requirements.

Finally, it is important to emphasize that tools are being developed under GPL with the purpose of facilitating and popularizing the use of SFDL. A graphical tool to aid in the design of views with SFDL-* is already available. It can be used for integration into workflows, or for more generalist uses (such as web applications). Furthermore, a Ruby gem called “yaxml” (Moreno, 2009) is also offered, to facilitate the

processing of SFDL-X in this scripting language. These are only the initial steps towards completing a set of tools that facilitate the adoption of the new definition language proposed in this article.

ACKNOWLEDGEMENT

This work has been supported by the ITECBAN project, which is sponsored by the CDTI and the Spanish Ministry of Industry, Tourism and Commerce. The authors would like to express their gratitude to INDRA Sistemas S.A. (http://www.indra.es/) for their invaluable contribution to this work.

REFERENCES

Andrews, T. et al. 2003, "Business process execution language for web services, version 1.1", *Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation*.

Ben-Kiki, O. et al. 2005, *YAML Ain't Markup Language (YAML™) Version 1.1*. Available: http://www.yaml.com/ [2010, January].

Bray, T. et al. 2006, 16 August 2006-last update, *Namespaces in XML 1.1*. Available: http://www.w3.org/TR/xml-names11 [2010, March].

Coenraets, C. 2003, "An overview of MXML: The Flex markup language", *Adobe-Developer Center*.

Crockford, D. 2006, "JSON: The fat-free alternative to XML", *Proceedings of XML*.

Dubinko, M. et al. 2003, "XForms 1.0", *W3C Recommendation*, vol. 14.

Fielding, R.T. 2000, *Architectural styles and the design of network-based software architectures*, University of California, Irvine.

Hickson, I. & Hyatt, D. 2009, , *HTML 5: A vocabulary and associated APIs for HTML and XHTML*. Available: http://dev.w3.org/html5/spec/ [2009, March].

Hollingsworth, D. 1994, *Workflow management coalition: The workflow reference model*, Workflow Management Coalition.

Krasner, G.E. & Pope, S.T. 1988, "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80", *Journal of Object-oriented programming*, vol. 1, no. 3, pp. 49.

Moreno, D. 2009, *The YAXML Module Reference*. Available: http://yaxml.rubyforge.org/ [2010, April].

Moreno, D. et al. 2009, *The SFDL definition*. Available: http://sfdl.dit.upm.es/ [2010, February].

Sayre, R. 2005, "Atom: The standard in syndication", *IEEE Internet Computing*, vol. 9, no. 4, pp. 71-78.

Swenson, K.D. et al. 2004, *Wf-XML 2.0-XML Based Protocol for Run-Time Integration of Process Engines*.

Van Der Aalst, W. et al. 2003, "Workflow patterns", *Distributed and parallel databases*, vol. 14, no. 1, pp. 5-51.

Wohed, P. et al. P. 2009, "Open Source Workflow Systems" in *Modern Business Process Automation* Springer Berlin Heidelberg, pp. 401-434.

Workflow Management Coalition 2002, *XML Process Definition Language (XPDL)*, Lighthouse Point, Florida, USA.

Zukowski, M. et al. 2008, *A RESTFul Protocol for Run-Time Integration of Process Engines*.