

Extensiones de Lenguaje de Workflow para la Generación Dinámica de Vistas

Diego Moreno, Emilio García, Sandra Aguirre, Juan Quemada

Departamento de Ingeniería de Sistemas Telemáticos

Universidad Politécnica de Madrid

Madrid, España

{dmoreno, egarcia, saguirre, jquemada}@dit.upm.es

Abstract—Este trabajo está orientado a la mejora de los sistemas de workflow basados en web, partiendo del modelo de referencia de la WfMC. El diseño ha tenido en cuenta los siguientes requisitos: 1) ser abierto y basado en estándares; 2) estar totalmente basado en web; 3) soportar la funcionalidad de un entorno real bancario. La creación de este entorno de *workflows* implica la extensión de los marcos de desarrollo usuales para incluir la definición de interacciones humanas basadas en web, así como la interacción con el modelo de datos del sistema y otras aplicaciones, mediante el desarrollo de un nuevo lenguaje que extiende la funcionalidad de los existentes.

Flujo de trabajo, workflow, arquitectura abierta, gestión de grupos, entorno colaborativo, interfaz de usuario

I. INTRODUCCIÓN

Los Entornos de Trabajo Colaborativos juegan un rol importante dentro de cualquier empresa, generalmente involucrando equipos inter-organizacionales. Tecnologías como la gestión de flujos de trabajo (*workflow*) son esenciales para una colaboración eficaz y eficiente por los beneficios que aportan, aunque a veces son menores de los esperados por las limitaciones propias de las interacciones entre equipos.

Esta propuesta intenta extender los sistemas de *workflow* a partir del modelo de referencia de la Workflow Management Coalition (WfMC) en un entorno web. Como parte de la validación de estas ideas, se ha tomado un escenario real: el proceso de desarrollo software en un núcleo bancario, donde un número significativo de profesionales emplean la gestión de *workflow* para coordinar sus actividades diarias. El proceso revela los requisitos funcionales que sirven de base para este trabajo: es necesario que los *workflows* se desarrollen con agilidad, y que el diseñador tenga control sobre la apariencia y la funcionalidad que verá el usuario final en cada paso.

Las mejoras propuestas avanzan en dos direcciones; hacia la definición de los procesos, primer eslabón de la cadena, y donde el diseñador debería ser capaz de especificar el proceso, apariencia y funcionalidad; y hacia las interacciones del usuario final a través de clientes basados en web, en los cuales las especificaciones del diseñador deben traducirse en interfaces de usuario efectivos y métodos de acceso al modelo de datos. Énfasis especial de este trabajo está puesto en la generación

automática de formularios web desde la definición de los *workflows*: dará poder a los diseñadores para construir fácilmente y bajo demanda interfaces enriquecidas centradas en el usuario, adaptándolas a las necesidades de cada escenario, incrementando la usabilidad del sistema, y permitiendo que el usuario final se centre en la productividad.

Las definiciones de procesos son realizadas mediante un lenguaje común a uno o varios motores, generalmente siguiendo una estructura XML. En este artículo se detalla un nuevo lenguaje extensible, paralelo a los ya existentes, que permita extender su funcionalidad. Pero, al mismo tiempo, manteniendo una generalidad que permita ser compatible con cualquier sistema de proceso de *workflows*, como quedará demostrado. Por otro lado, la interfaz con las aplicaciones cliente debe ser suficientemente genérica para desacoplar el motor de *workflows* de las herramientas de usuario final. Se asume, sin embargo, que la interacción se realizará a través de un navegador web, estándar de facto hoy en día, lo cual garantiza los requisitos de interoperabilidad mencionados antes, y la intención de usar estándares ampliamente adoptados y orientados a la web: Wf-XML-R, REST y Atom.

El artículo está organizado como sigue: la Sección II describe el estado del arte, y el modelo e implementaciones de *workflows*. El escenario se introduce en la Sección III, junto con los requisitos que impone la aproximación. La Sección IV describe la solución adoptada, cambios de la arquitectura propuesta, y algunos detalles acerca del procedimiento de validación hecho para comprobar la idoneidad de las extensiones. Finalmente, la Sección V cierra el artículo, indicando el trabajo futuro y las conclusiones obtenidas.

II. ESTADO DEL ARTE

Se describen aquí los principales elementos de un sistema de *workflow* que son necesarios para soportar la creación y especificación de *workflows* dentro de un entorno corporativo basado en el modelo de referencia WfMC. Nuestro objetivo es desarrollar un sistema web abierto basado en protocolos, interfaces y componentes estandarizados.

A. El Modelo de Referencia para Workflows

El modelo de referencia WfMC [1], mostrado en la Figura 1, representa los principales componentes de un sistema de

gestión de *workflows* y las especificaciones para cada uno de sus principales interfaces.

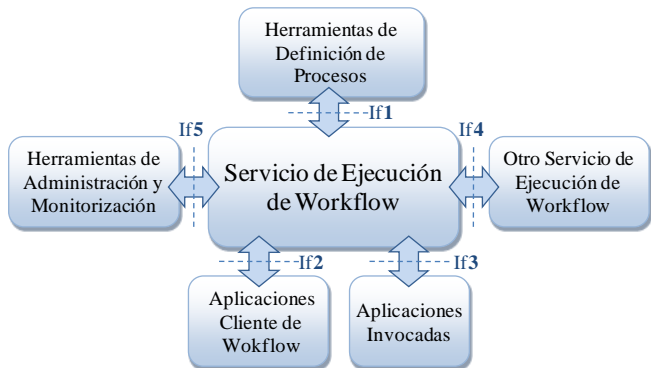


Figura 1. Modelo de Referencia de *Workflow*

Las interfaces definidas entre los distintos componentes son: a) If1 define un formato común para el intercambio de definición de workflows. XPDL es el lenguaje XML de definición de procesos propuesto para esta interfaz; b) If2 proporciona un completo rango de interacciones entre la gestión de workflows en tiempos de ejecución y las aplicaciones cliente de workflow. WAPI (Interfaz de Programación de Aplicaciones de Workflow) es usada en esta interfaz; c) If3 describe como las aplicaciones son invocadas entre la gestión de *workflows* en tiempo de ejecución y las aplicaciones de *workflow* invocadas. WAPI es la API de referencia utilizada en esta interfaz; d) If4 describe las interacciones entre dos servidores de *workflows*; e) If5 especifica una serie de funciones para la administración y monitorización de un servidor de *workflows*.

Durante los últimos años, se han realizado numerosos desarrollos que implementan los elementos de la arquitectura en XML y su comunicación con Web Services [1].

B. Motores de *Workflow*

Un motor de *workflow* es un software que provee el control del entorno de ejecución de una instancia de *workflow* [2].

Puede encontrarse una amplia gama de motores tanto de código abierto como propietarios. Las diferencias entre ellos dependen de los diferentes dominios de aplicación y requerimientos de usuarios hacia los cuales están orientados. Como ejemplos de motores de *workflow* propietarios tenemos: Microsoft's BizTalk Server [3], Microsoft's Windows Workflow Foundation (WF) [4] y Oracle's BPEL Process Manager [5] y a nivel de motores de *workflow* de código abierto tenemos jBPM [6], OpenWFE [7] y Enhydra Shark [8].

OpenWFE ofrece un rango más amplio de características que jBPM y Enhydra Shark. Desde el punto de vista de control de *workflow*, jBPM y Enhydra Shark soportan un conjunto relativamente limitado de operadores de control de *workflow* ofreciendo un bajo soporte para los patrones que se encuentran fuera de la categoría básica de control de *workflow*. OpenWFE provee un mejor rango de facilidades para tareas de concurrencia pero tiene un soporte limitado para el constructor OR-join. Desde el punto de vista de la perspectiva de los datos, jBPM, OpenWFE y Enhydra Shark ofrecen soporte a un

limitado rango de enlaces de elementos de datos y se basan en gran medida en elementos de datos a nivel-case [9].

C. Interfaz 1: Lenguajes de Definición de Procesos

La interfaz entre las herramientas de definición de procesos y el servicio de ejecución de *workflows* se denomina *interfaz de importación/exportación de definiciones de proceso*. Como punto de entrada de las descripciones de los procesos de negocio, sirve a un propósito principal: desacoplar el lenguaje de modelado del motor de procesado.

Entre los lenguajes de definición de procesos, cabe destacar BPEL [10] y XPDL [11]. BPEL es un lenguaje ejecutable con soporte para XML e intercambio de mensajes SOAP para sus operaciones. Su objetivo principal es la orquestación de servicios web, y la secuencia de interacción y flujo de datos. Sin embargo, carece de dos importantes características: soporte gráfico –los flujos no contienen información acerca de los diagramas de definición que los generaron- e interacción humana –que sólo se consigue mediante extensiones-.

XPDL, por el contrario, soporta no sólo la representación gráfica de los procesos, sino que cada paso puede incluir la descripción de la actividad, temporizadores, llamadas a servicios web, o roles para diferentes tipos de participantes (incluyendo humanos). XPDL puede verse como el candidato ideal para esta interfaz, ya que su portabilidad garantiza la fácil conversión a BPEL en aquellos casos en los que no se soporta de manera nativa.

Para el presente trabajo se seleccionó una solución basada en software libre, para asegurar la facilidad y flexibilidad en la modificación de todos los componentes. OpenWFeru [12] (o, simplemente, OpenWFE), el cual sin alcanzar la complejidad de XPDL, provee el lenguaje más rico entre las alternativas de software libre [13], cubriendo prácticamente todos los patrones de control, recursos y datos. Define los *workflows* usando una notación en XML extremadamente simple, y el motor de procesos (el servicio de ejecución) ofrece un API para acceder a los datos del motor.

D. Interfaz 2: API del Cliente de *Workflow*

El objetivo de la Interfaz 2 es la definición de un API [14] para aplicaciones cliente, con el objetivo de solicitar servicios al motor de ejecución, y controlar el progreso del *workflow* – procesos, actividades y *workitems*-. Asumiendo que algunas interacciones necesitan de la colaboración humana, se necesita un mecanismo para exportar datos desde el motor y presentárselos al usuario final. Aparece así el concepto de “lista de trabajo”: una cola de elementos de *workflow* (*workitems*) asignados a un usuario/rol en particular. Esta lista de trabajo es accesible tanto al servicio de ejecución para asignar ítems a los usuarios que deben procesarlos, como a las aplicaciones cliente, de modo que puedan ser adecuadamente presentadas y “consumidas” por sus propietarios. Por esa razón, también las respuestas (acciones) de los usuarios deben comunicarse al motor. Como resultado, la interfaz debe soportar operaciones para: conexión/desconexión, funciones para el estado de procesos y el control de actividades, y comandos para manipulación de listas de trabajo.

Si el *workflow* va a ser integrado en un entorno común adaptado a cada usuario final, de modo que encaje en un sistema de gestión de tareas, surge la necesidad no sólo de soportar la intercomunicación de datos y acciones en la Interfaz 2, sino también de datos relevantes para la presentación. Esta es una de las ideas clave de nuestra propuesta.

E. Interfaz 4: Interoperabilidad

El objetivo de la interfaz 4 es facilitar la comunicación entre dos sistemas de *workflow*. Sistemas de distintos fabricantes deben ser capaces de transferirse información.

Las funciones de interoperabilidad de la WAPI deben soportar el intercambio de información de control y la transferencia de datos relevantes de la aplicación o del *workflow* entre distintos servicios de ejecución de *workflows*. De esta forma, si dos motores de *workflows* dan soporte a un número suficiente de funciones comunes de la WAPI podrán pasarse el control de uno a otro dentro de un mismo *workflow*, además de toda la información necesaria.

1) Wf-XML

La WfMC, con la finalidad de tener un protocolo que permita integrar motores de procesos en tiempo de ejecución a través de Internet, definió Wf-XML, un protocolo basado en XML. Wf-XML permite obtener información y administrar un motor de *workflows* a través de acciones como lanzar procesos, obtener la lista de actividades que el proceso está esperando, conseguir información de la asignación de cada actividad, etc.

En la actualidad, se está trabajando en la especificación de Wf-XML 2.0 [15], tras muchos años de uso de Wf-XML 1.1. La nueva versión es un esfuerzo para definirlo como un servicio web estándar. Ahora, en la definición se emplea el *Web Services Description Language* (WSDL) y para el transporte de mensajes *Simple Object Access Protocol* (SOAP).

2) Wf-XML-R

En los últimos tiempos, el W3C ha añadido la posibilidad de definir servicios web ligeros, conocidos como servicios web REST (*Representational State Transfer*) [16]. En un gran número de casos, el uso de los servicios web tradicionales añade demasiada complejidad. Por esta razón, desde el Open GeoSpatial Consortium se inició un movimiento por la especificación de un nuevo estándar, el Wf-XML-R, basado en servicios web REST. En abril del 2008, Wf-XML-R fue aceptado por la WfMC para desarrollarse como estándar [17].

Para el uso de REST, el servicio de *workflows* ha sido estructurado de acuerdo a la Arquitectura Orientada a Recursos (ROA, *Resource Oriented Architecture*). Así, se han identificado los siguientes recursos del servicio: definiciones, procesos, actividades, trazas, participantes, *workitems*, motor y errores. De esta manera, cada recurso es accesible a través de una *Uniform Resource Locator* (URL) que le identifica.

Para el marcado de la información se mantiene el uso de XML. Además, en Wf-XML-R para estructurar la información que contienen los ficheros XML se recurre al formato Atom. De esta forma, los recursos quedan formateados como *feeds* y *entries*. Se utiliza Atom Syndication Format (Atom) para obtener información de los recursos web y Atom Publishing

Protocol (AtomPub o APP) para la publicación y edición. Para poder tener representaciones completas de los recursos del motor de procesos, Wf-XML-R utiliza dos extensiones de Atom. Por un lado, se emplea la Google Data API (GData), el estándar de Google para leer y escribir datos en la web. Por otro, Wf-XML-R define una extensión de Atom propia, que cubre las particularidades de los recursos de los *workflows* que no abarcan el resto de estándares.

III. ESCENARIO

El proyecto ITECBAN tiene como misión dotar de herramientas de software orientadas a las actividades colaborativas de una organización virtual para la construcción y evolución de los principales sistemas de información de una organización bancaria. ITECBAN dará soporte al desarrollo de diferentes actividades colaborativas tales como, el proceso de desarrollo de software dentro de un *core* bancario, sistema de videoconferencia, gestión de contenidos, etc.

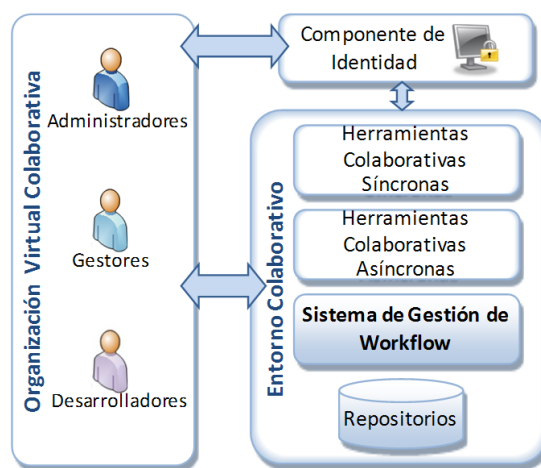


Figura 2. Escenario de ITECBAN

Los componentes principales del escenario ITECBAN - mostrado en la Figura 2- son: a) Organización Virtual Colaborativa: grupo de personas formada por administradores, gestores y desarrolladores. Administradores y gestores son responsables de la especificación de un proceso de *workflow*. Desarrolladores son los usuarios finales del sistema; b) Componente de identidad, el cual es responsable de la autenticación, autorización y políticas de control de acceso; c) Entorno colaborativo formado por las herramientas colaborativas síncronas y asíncronas, el sistema de gestión de *workflows* y los repositorios.

En este escenario, el sistema de gestión de *workflows* debe satisfacer, al menos, los siguientes requerimientos funcionales: a) Diseño de formularios que permiten la especificación de tareas, reglas, roles de usuario y tipos de datos de entrada y salida que son necesarios en la ejecución de un *workflow*; b) Uso de un sistema de gestión de *workflow* de código abierto; c) Actualización dinámica de *workflows*; d) Facilidad en la creación de nuevos *workflows*; e) Uso de un esquema estándar de identidad federada para entornos corporativos (LDAP); f) Acceso del usuario a través de cualquier navegador web; g) Gestión de productos bancarios a través de la CMDb; h) Uso

de un lenguaje de tipado dinámico para la rápida construcción de prototipos.

Considerando el escenario descrito es necesario llevar a cabo una instanciación del modelo de referencia, centrándose en aquellas entidades e interfaces más relevantes para el escenario bancario. Esto es representado en la Figura 3:

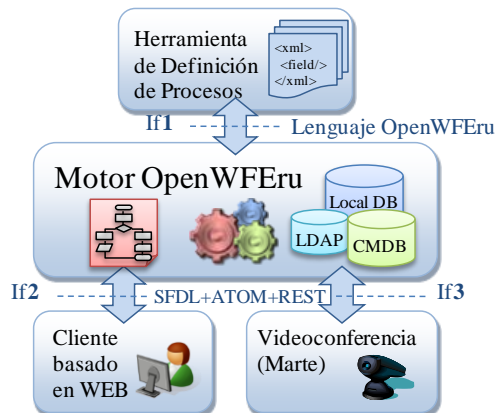


Figura 3. Instanciación del Modelo de Referencia

El trabajo está enfocado a dos puntos principales, como se verá en la próxima sección: las Interfaces 1 y 2. La interfaz 3 también se incluye dado su interés para incluir servicios de videoconferencia avanzados en el escenario –fuera del ámbito de este documento, se deja como trabajo futuro-. Pero las decisiones más importantes están relacionadas con la selección de un lenguaje específico de descripción de *workflows* y su motor correspondiente: OpenWFE. Su naturaleza de código libre y su flexibilidad han contribuido a su selección; pese a todo, debe ser entendido como un medio para validar la aproximación, y no como una restricción de uso de una tecnología concreta. Asociadas al servicio de ejecución de *workflows* hay distintas bases de datos, impuestas por las necesidades de este escenario en particular: LDAP para gestión de usuarios/roles; CMDB para almacenar los productos que se procesan en el *workflow*; y una base de datos local, que almacenará los *workitems*.

Finalmente, un requisito es que toda la interacción de usuario final se haga a través de tecnologías web, para minimizar el impacto de desplegar la solución en un entorno corporativo.

IV. EXTENSIONES Y VALIDACIÓN

La arquitectura presentada en la sección anterior forma los cimientos de los desarrollos de este trabajo, ya que se ve capaz de soportar las extensiones diseñadas, sin que ello implique cambios dramáticos. Esto conlleva un beneficio debido a su naturaleza abierta, y posibilita la interoperabilidad entre la web humana y la web de datos y servicios, todo sobre protocolos estándares como REST.

La propuesta descrita aquí incluye modificaciones –o, más apropiadamente, mejoras– a la arquitectura en los siguientes puntos:

- Interfaz 1: el lenguaje de descripción de procesos será extendido para soportar una definición más amplia de

los *workflows*, incluyendo información para definición de vistas y acceso dinámico a datos.

- Servicio de Ejecución de *Workflows*: el motor sufrirá cambios mínimos para acomodar las nuevas capacidades, especialmente en lo relativo al acceso al modelo de datos.
- Interfaz 2: se propone reutilizar y extender el protocolo de comunicación basado en Atom de la Interfaz 4, para seguir la línea de la Arquitectura de Referencia.

Finalmente, se comentarán algunos de los desarrollos realizados en la parte cliente, necesarios para aprovechar todo lo descrito aquí, y como validación de la aproximación.

A. Interfaz 1: Extensión del Lenguaje

Como ya se ha visto en la Sección II.C, la Interfaz 1 se sitúa entre la definición de procesos –con sus herramientas de modelado– y el motor de *workflows*. La extensión propuesta en este punto cubre dos aspectos principales: la generación de formularios web, para permitir la implementación flexible de vistas desde el propio proceso de diseño, y operaciones básicas desde el lenguaje para acceder al modelo de datos. De este modo, el diseñador puede establecer no sólo el patrón de interacción, sino definir las reglas básicas de la interfaz que manejará el usuario final para disparar dichas interacciones.

Los desarrollos han sido realizados y validados tomando OpenWFE como base; sin embargo, se ha tenido en cuenta el requisito de hacerlos lo más genéricos posible, de cara a que su portabilidad a otros lenguajes de definición de procesos sea lo más directa posible. De acuerdo con [13], todo sistema dispone de algún modo de ejecutar funciones externas, ya sea como código empotrado o como un participante ad-hoc especialmente desarrollado para eso. Esta última es la implementación que se ha elegido aquí: dentro del lenguaje de definición de *workflows* se harán llamadas especiales a una función que cargará datos desde un archivo externo. Los datos de este archivo definirán:

- Pantallas/vistas para el usuario final, entendiendo por ellas los interfaces gráficos de usuario en forma de formularios.
- Las acciones/interacciones que el usuario puede realizar en cada pantalla.
- Y, finalmente, las funciones que deberán ejecutarse para acceder al modelo de datos, con resultados que se mostrarán al usuario, y entradas que se tomarán como consecuencia de sus acciones.

Por todo ello, se ha definido un lenguaje que posibilita todos esos aspectos de una manera clara y sencilla. Adicionalmente, como se verá más adelante, este lenguaje es la base del empleado en la Interfaz 2 del modelo de referencia.

1) SFDL: Lenguaje Sencillo de Definición de Formularios

El Lenguaje Sencillo de Definición de Formularios (*Simple Form Definition Language*, SFDL) es un lenguaje diseñado teniendo en cuenta los requisitos funcionales antes detallados. Concretamente:

- Soporta una amplia variedad de elementos de formularios web: selectores, tablas, desplegables, campos de entrada/salida...
- Es autocontenido: tiene toda la información necesaria – componentes y estilo- en un único fichero.
- Múltiples pantallas por vista: una misma actividad de un *workflow* puede estar formado por distintas pantallas, antes de enviar la información al servidor.
- Permite la expresión en un número de lenguajes de marcado -XML, JSON, YAML- completamente equivalentes entre sí en funcionalidad, pero con particularidades que los hacen más adaptados a distintos entornos. Además, el uso de lenguajes estándares consigue simplificar al máximo el procesamiento de los ficheros.
- Incluye soporte para la ejecución de funciones en servidor, para manejar el modelo de datos desde/hacia las vistas.

Desde el lenguaje de definición de procesos, OpenWFE, los ficheros se cargan a través de un participante especialmente destinado a ello (Figura 4), como se explicará más detalladamente en la Sección IV.B.

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

Figura 4. Carga de un fichero SFDL

Los ficheros definidos en SFDL pueden estarlo en una de tres variantes posibles:

- SFDL-X: utiliza un marcado XML, adecuado por su interoperabilidad entre plataformas, y por ser el mismo formato de la definición de *workflows*, y del lenguaje que se empleará en la Interfaz 2.
- SFDL-J: utiliza el formato JSON, optimizado para Javascript, con una sintaxis que permite un gran ahorro en ancho de banda (hasta el 50% frente a XML), y fácilmente procesable.
- SFDL-Y: definido en YAML, con una sintaxis muy sencilla, basada en indentado, y directamente traducible a JSON.

En el apartado siguiente se muestran algunos ejemplos de la sintaxis. Antes, se discutirá brevemente la conveniencia de utilizar SFDL frente a otras alternativas:

- XForms [18] comparte el objetivo de orientación a diseño de formularios, pero su soporte en los clientes web actuales es prácticamente nulo. Del mismo modo, tiene mayor complejidad –depende de CSS para definir el estilo- y no incorpora una manera estandarizada de ejecutar funciones en el lado servidor.
- HTML 5.0 Forms [19], comienza a estar más soportado en los navegadores, pero sus formularios no son fácilmente serializables para tratarse con Javascript. Además, HTML no incorpora llamadas anidables a funciones en el servidor.

2) Definición de vistas en SFDL

Para la definición de vistas en las actividades de un *workflow* se emplea un sencillo esquema basado en etiquetas para indicar la posición de cada elemento, su tipo, su valor y algunos parámetros que definan el estilo o la funcionalidad de dicho elemento. La Tabla I resume todos los campos:

TABLA I. CAMPOS DE UNA VISTA

Etiqueta	Descripción	Valores
type	Funcionalidad del elemento	Label, input_text, text_area, text_block, selector, table, dynamic_table, link, attach, checkbox
params	Apariencia del elemento	haling, width, height, hint...
value	Valor del elemento	Valores numéricos, alfanuméricos, o funciones
result	Resultado de la interacción del usuario con el elemento	

Cada elemento va precedido de un identificador numérico que indica la posición que va a ocupar en la pantalla. Como ejemplo, se muestra en la Figura 5 la definición en SFDL-Y de un campo etiqueta (label), cuyo valor viene dado por el resultado de una función (user-data). El elemento se posicionará en las coordenadas (04, 30).

```
_v_f0430:
  type: label
  value:
    function-name: user-data
    attribute-name: telephone
  params:
    halign: left
    width: "60"
```

Figura 5. Definición de un campo en SFDL-Y

En este punto conviene incidir en el hecho de que todos los formatos SFDL-* son equivalentes entre sí. La Figura 6 muestra la definición del mismo campo en SFDL-X (XML), con la particularidad de que ese formato, una vez procesada la función, será el que se envíe al cliente a través de la Interfaz 2.

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

Figura 6. Definición de un campo en SFDL-X

Como se ve, el lenguaje SFDL es fácilmente extensible, y permite crear nuevos tipos de elementos con parámetros de manera sencilla.

3) Funciones de acceso al modelo de datos

Uno de los requisitos más importantes de SFDL es que permita el acceso al modelo de datos desde la definición de los formularios, mediante funciones. Surge así una distinción importante según el tiempo en el que se quieran ejecutar dichas funciones:

- En tiempo de ejecución del *workflow*: cuando la definición es procesada por el motor.
- En tiempo de presentación: cuando el formulario es presentado al usuario final.

Para posibilitar ambos comportamientos, se ha diseñado un mecanismo para las llamadas a funciones tanto desde el *workflow* definido en lenguaje OpenWFE, como desde las definiciones en SFDL, siguiendo el modelo funcional.

a) Funciones en tiempo de *workflow*

Las llamadas a funciones desde el lenguaje de definición de procesos se han implementado como referencias a un participante especial, por ser esta la forma más directa dentro de OpenWFE. Sin embargo, la generalidad de esta aproximación queda garantizada en tanto en cuanto todos los lenguajes permiten de un modo u otro el añadir funciones externas. Siguiendo con el ejemplo de la Figura 5, la llamada para obtener el teléfono de un usuario se recoge en la Figura 7.

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

Figura 7. Función definida en el *workflow*

Las funciones implementadas en el motor cubren las operaciones básicas de entrada/salida de datos desde/hacia el modelo y las bases de datos usadas en la arquitectura (LDAP, CMDB): read-attribute, write-attribute, cmdb-out, user-data... aunque por el propio mecanismo de definición de funciones, que se describirá más en detalle en la siguiente sección, es posible ampliar indefinidamente el conjunto de llamadas.

b) Funciones en tiempo de presentación

Las funciones que se deben ejecutar cuando el usuario reciba una vista están definidas en el mismo lenguaje que dichas vistas: SFDL, en cualquiera de sus variantes (-X, -Y o -J). Ejemplos pueden verse en la Figura 5 y la Figura 6. Esta aproximación tiene dos aspectos positivos de gran utilidad:

- Las funciones son anidables y, por tratarse de un lenguaje funcional, puede incorporarse una función en sustitución de un valor en todo punto del SFDL.
- Se trata de una única librería de funciones, conservando los mismos nombres y parámetros (en definitiva, la interfaz de llamada), por lo que las llamadas desde SFDL o desde OpenWFE son idénticas.

B. Adaptación del motor de *workflows*

En todo momento, se ha optimizado la arquitectura para obtener un diseño independiente del motor de *workflows* empleado. No obstante, hay que realizar una serie de pequeñas adaptaciones en el motor de procesos escogido, en nuestro caso fue OpenWFEru.

Las adaptaciones girarán en torno a la ejecución de funciones y la fase de presentación. De esta forma, la adaptación será diferente dependiendo del tipo de función a

ejecutar (en tiempo de ejecución de *workflow* o en tiempo de presentación).

El objetivo será poder ejecutar las mismas funciones independientemente del momento. Así, todas las funciones disponibles se empaquetarán en una librería común.

1) Funciones en tiempo de ejecución del *workflow*

Como se menciona previamente, para este tipo de funciones, en la arquitectura propuesta, se usará la sintaxis que disponga el lenguaje escogido para lanzar una única función "functions" que, a través de un módulo denominado *Function Trigger* (disparador de funciones), será la encargada de realizar las llamadas.

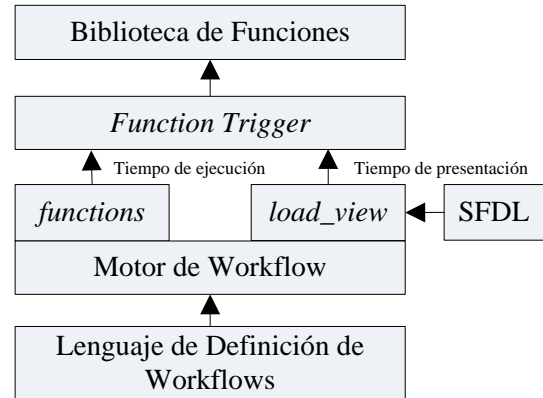


Figura 8. Estructura de las llamadas a funciones

Asimismo, sólo es necesario implementar una función dentro del motor que delegue la ejecución de funciones al *Function Trigger*. Por sencillez, este módulo tendrá una forma común de recibir las llamadas a funciones y sus parámetros, ya sean en tiempo de presentación o de ejecución de *workflows*. Esta forma única de recibir los datos será a través de un único parámetro, que debe ser una estructura de datos tipo *hash*.

De esta forma, para la ejecución de funciones en tiempo de procesado del *workflow* no hay que realizar cambios en ningún motor, simplemente se añadirá la llamada a una función "functions" que delegue el trabajo al módulo *Function Trigger*.

Se puede apreciar en la Figura 8 la estructura que se sugiere para tener una librería de funciones ejecutables independientemente del momento de su ejecución.

2) Funciones en tiempo de presentación

Cuando un usuario solicite la representación de un recurso, será el momento de ejecutar este tipo de funciones. Las funciones en tiempo de presentación serán llamadas desde archivos SFDL que deben ser cargados desde el *workflow*, desde los puntos que se estimen oportunos.

La carga se llevará a cabo con la llamada a una función, denominada "load_view", que se tendrá que implementar de la misma manera que "functions", de la forma que disponga el motor. La función "load_view" asociará el archivo SFDL a la instancia del proceso que se está ejecutando.

Una vez realizada esta asociación cuando el usuario pida la representación del proceso, se incluirá en ella el formulario que se debe cumplimentar para dar por realizada esa actividad. A la

hora de formar la representación, se leerá el archivo SFDL en busca de funciones a ejecutar. Donde se encuentren llamadas a funciones se sustituirán por el resultado que devuelva la ejecución. También en este caso, la ejecución de las funciones será llevada a cabo por el módulo *Function Trigger*.

C. Interfaz 2: Wf-XML-R con extensiones

La interfaz 2 es la interfaz con las aplicaciones cliente. Estas aplicaciones (normalmente un navegador web) podrán comunicarse con el motor de procesos a través de una interfaz REST basada en el protocolo Wf-XML-R.

Wf-XML-R es una adaptación de Wf-XML, diseñada para la Interfaz 4 dentro del Modelo de Referencia de *Workflow* de Trabajo de la WfMC. Pero es posible usar Wf-XML-R en la interfaz 2. Esto es así ya que, según el modelo de referencia, todas las interfaces tienen un conjunto de llamadas comunes dentro de la WAPI, y se diferencian unas de otras en las funciones particulares que añaden cada una. Si usamos funciones dentro del conjunto común no habrá problemas por usar un protocolo de una interfaz en otra.

Wf-XML-R representa los recursos con Atom y AtomPub. El uso de Atom tiene una importante ventaja: admite extensiones fácilmente. De esta manera, la representación de un recurso con Atom puede completarse con extensiones que den cabida a los matices que los recursos de cada servicio web tienen. Es decir, si con la base del protocolo Atom no puede representarse completamente un recurso, se pueden crear extensiones que lo permitan. Este sistema de extensiones se asienta en los espacios de nombres de XML [20]: cada extensión crea un espacio de nombres nuevo, donde se admitirán las nuevas etiquetas necesarias.

La generación dinámica de los formularios de los *workflows* queda fuera del ámbito de Atom y Wf-XML-R. Así, se crea una extensión para el Wf-XML-R basada en una extensión Atom. Esta extensión genera un nuevo espacio de nombres para las vistas.

Actualmente, Wf-XML-R es un borrador de estándar de la WfMC, razón por la cual este protocolo tiene partes sin definir claramente y puede sufrir modificaciones.

A esta representación se le ha añadido la información referente a la vista. La extensión se basará en una etiqueta XML *v:current_view* que contendrá toda la información nueva. Esta etiqueta aceptará los siguientes atributos:

- **Type:** SFDL-X, SFDL-J, HTML, XFORM. Especificará el formato en que está la vista contenida en la etiqueta *current_view*. Se aceptan varios formatos, entre ellos dos de la familia SFDL. No se soporta SFDL-Y, ya que YAML es sensible a los espacios en blanco y XML no, resultando ambos incompatibles e impidiendo así la inclusión de YAML dentro de XML.
- **Screen_id:** contendrá un número entero. Es el identificador de la pantalla de la que se está dando información. Este atributo es la base del soporte de múltiples pantallas.

La Figura 9 es un ejemplo de cómo quedaría la representación con una vista formateada con SFDL-X. Dentro de la etiqueta *v:current_view* se especificarán los campos que componen la vista formateados como indica la Figura 6. De esta manera, se aprecia que el alto grado de integración entre formatos.

```
<entry ...>
...
  <v:current_view type="sfdl-x" screen_id="1">
...
  </v:current_view>
</entry>
```

Figura 9. Representación de un *Workitem* en Wf-XML y SFDL-X.

En nuestro escenario se usó SFDL-X para el envío de las vistas a la aplicación cliente, por su mejor integración con el formato ATOM al ser ambos en XML.

D. Aplicación cliente

En la arquitectura propuesta, al recubrir el motor de procesos con una interfaz estándar como Wf-XML-R (REST + Atom), se facilita la integración con distintos clientes. Pero, en la actualidad, los navegadores web se están convirtiendo en el cliente estándar para el uso de servicios sobre Internet. Así, es lógico crear una aplicación cliente basada en web para acceder al motor de procesos.

El trabajo del navegador dependerá del formato en que se le manden los datos. En nuestro escenario, con una interfaz Wf-XML-R y SFDL-X, se ofrece de forma integrada toda la información de los recursos en XML (datos del *workitem* e información sobre la presentación de las vistas). En el cliente se necesita, por un lado, inteligencia suficiente para interpretar esta información y, por otro, una buena experiencia de usuario.

Con estos requisitos se escogió la tecnología Adobe Flex para implementar el cliente. Esta tecnología ofrece nuevas posibilidades a la hora de generar interfaces de usuario en aplicaciones web, aumentando el número de posibles interacciones (p. ej. *drag&drop*) y mejorando la usabilidad. Además, es posible conseguir útiles y llamativos efectos visuales con facilidad. Esto lo convierte en una potente herramienta para desarrollar aplicaciones web. Por lo tanto, en nuestro proyecto se ha desarrollado un cliente Flex que se comunica con el motor de procesos a través de una interfaz Wf-XML-R y puede presentar dinámicamente las vistas de los formularios.

Cabe mencionar que la arquitectura propuesta daría solución a otros escenarios distintos. Por ejemplo, si el encargado de recibir la información fuese una aplicación cliente basada en JavaScript, sería posible optimizar la transmisión de la información y el tratamiento de la misma, utilizando en la interfaz con el cliente SFDL-J, en lugar de SFDL-X. La razón es que JavaScript puede tratar de forma muy eficiente el formato JSON. Por otro lado, también es posible eliminar toda inteligencia del cliente, y enviar las vistas usando HTML. Sin embargo, esto aumenta la complejidad en el servidor y elimina la posibilidad, dentro de la vista, de ejecutar funciones anidables relacionadas con los *workflows*.

V. CONCLUSIONES Y TRABAJO FUTURO

La arquitectura descrita en este trabajo satisface los objetivos propuestos. Se ha mejorado la comunicación entre los sistemas de *workflow* y sus usuarios basándose en estándares ampliamente aceptados y realizando propuestas en los vacíos detectados. Esto ha sido posible gracias al uso inteligente de los protocolos ya existentes REST, Wf-XML-R y Atom; y a la definición abierta del lenguaje SFDL, basado en XML, YAML y JSON. De esta manera, desde la definición de un *workflow*, el motor que lo procesa puede generar dinámicamente formularios de gran usabilidad, interactuar con servicios web y acceder al modelo que recubre la base de datos. Además, desde el punto de vista del usuario, con sólo un navegador web y a través de los mismos formularios dinámicos es posible realizar la comunicación con el sistema de *workflows*.

La arquitectura propuesta ha recibido una realimentación positiva al ser validada con una implementación completa dentro de un escenario bancario real. En este entorno, se han conseguido importantes mejoras: posibilidad de que los diseñadores de *workflows* puedan especificar la vista que interactúa con el usuario; utilización de un conjunto de métodos estandarizados para acceder al modelo de datos; conservación de una arquitectura suficientemente simple compatible con el Modelo de Referencia (p. ej. reutilizando protocolos como el Wf-XML-R) y sin sacrificar la portabilidad, evitando introducir profundas modificaciones específicas de una plataforma determinada. De esta manera, con una arquitectura y formatos con ventajas probadas, recomendamos su utilización en escenarios con requisitos similares.

Para facilitar y popularizar el uso de los desarrollos propuestos, se desarrollarán una serie de herramientas gráficas. Por un lado, una aplicación para la generación de *workflows* a través de la web, con el objetivo no sólo de cubrir OpenWFE, sino permitir la portabilidad hacia otros lenguajes como XPD L o BPEL. Por otro lado, para el desarrollo de vistas mediante SFDL-*, tanto para su integración dentro de *workflows* como en otros escenarios más generalistas (como aplicaciones web). Esto es posible gracias al gran desacoplamiento entre la definición de procesos y la definición de interfaces de usuario.

Finalmente, se pueden considerar otras interfaces de la arquitectura; principalmente, la interfaz 3, donde es posible integrar aplicaciones de videoconferencia dentro del *workflow*. Estas nuevas funcionalidades implican nuevas extensiones, que presumiblemente validarán el carácter flexible y extensible de la arquitectura propuesta.

AGRADECIMIENTOS

Los autores agradecen al Ministerio de Industria, Turismo y Comercio y a CDTI (Centro para el Desarrollo Tecnológico e Industrial) por el apoyo de la iniciativa ITECBAN. Asimismo, agradecen a INDRA Sistemas, S.A. (<http://www.indra.es/>) su valiosa contribución a este trabajo.

REFERENCIAS

[1] Hollingsworth, D. The Workflow Reference Model: 10 years on. Workflow Handbook 2004. Future Strategies. Lighthouse Point, FL. 2004.

- [2] Hollingsworth, D. The Workflow Reference Model Version 1.1. Winchester, UK: Workflow Management Coalition, WFMC-TC-1003, Ene 1995.
- [3] Microsoft BizTalk Server. Disponible en <http://www.microsoft.com/biztalk/en/us/default.aspx>. Último acceso Mar 09.
- [4] Windows Workflow Foundation. Disponible en <http://msdn.microsoft.com/en-us/netframework/aa663328.aspx>. Último acceso Mar 09.
- [5] Oracle BPEL Process Manager. Disponible en <http://www.oracle.com/technology/products/ias/bpel/index.html>. Último acceso Mar 09.
- [6] JBoss. JBoss jBPM website. Disponible en www.jboss.com/products/jbpm. Último acceso Mar 09.
- [7] OpenWFE. OpenWFE website. Disponible en www.openwfe.org. Último acceso Mar 09.
- [8] Enhydra. Open Source Java XPD L Workflow. <http://www.enhydra.org/workflow/index.html>. Último acceso Mar 09.
- [9] Wohed P., Andersson B., et al., "Patterns-based Evaluation of Open Source BPM Systems: The Cases of jBPM, OpenWFE, and Enhydra Shark". BPM Center Report BPM-07-12, BPMcenter.org, 2007.
- [10] Thatte, S., et al., "Business Process Execution Language for Web Services Version 1.1", BEA, IBM, Microsoft, SAP and Siebel, May 2003.
- [11] "Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface -- XML Process Definition Language (XPD L)" (WFMC-TC-1025). Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 2002 <http://www.wfmc.org/standards/xpdl.htm>. Último acceso Sep 08
- [12] OpenWFEru – open source ruby workflow engine <http://openwferu.rubyforge.org/>
- [13] Van der Aalst, W. et al. "Workflow Patterns Evaluations" <http://www.workflowpatterns.com/evaluations/opensource/openwfe.php>
- [14] Workflow Management Coalition. Programming Interface Specification. Version 2.0. Workflow Management Coalition Specification. Document Number WFMC-TC-1009. Jul 1998.
- [15] Swenson, K., Pradhan, S., Gilger, M., "Wf-XML 2.0, XML Based Protocol for Run-Time Integration of Process Engines". Draft. Oct 2004.
- [16] Fielding, R. T., "Architectural Styles and the Design of Network-based Software Architectures", tesis doctoral, University of California, Irvine, 2000.
- [17] Zukowski, M., Cappelaere, P., Swenson, K., "A RESTful Protocol for Run-Time Integration of Process Engines". Draft 5. Abr 2008.
- [18] W3C XForms 1.0 (Third Edition). Disponible en <http://www.w3.org/TR/xforms/>. Último acceso Mar 09.
- [19] W3C HTML 5. Disponible en <http://dev.w3.org/html5/spec/>. Último acceso Mar 09.
- [20] W3C Recommendation, "Namespaces in XML", T. Bray, D. Hollander, A. Layman, Ene 14, 1999. <http://www.w3.org/TR/1999/REC-xml-names-19990114>

