PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 11-05

Enterprise Mashup Markup Language: Análisis de características y perspectivas de extensión

Gabriel López **Raúl Ruggia**

2011

Enterprise Mashup Markup Language: Análisis de características y perspectivas de extensión
López, Gabriel; Ruggia, Raúl.
ISSN 0797-6410
Reporte Técnico RT 11-05
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, mayo de 2011

Enterprise Mashup Markup Language: Análisis de características y perspectivas de extensión

Gabriel Lopez
Facultad de Ingeniería – Universidad de la Republica
Laboratorio de Integración de Sistemas
gabriell@fing.edu.uy

Raul Ruggia
Facultad de Ingeniería – Universidad de la Republica
Laboratorio de Integración de Sistemas
ruggia@fing.edu.uy

Fecha redacción: Mayo 2011

Palabras clave
Mashup, data services, SOA, data remix, EMML.

Resumen
A fines de 2009 se libera la primera versión del *Enterprise Mashup Markup Language* (EMML), un lenguaje especifico de dominio (DSL) para la creación de *mashups* empresariales, producido por la Open Mashup Alliance (OMA).

En este reporte se analizan las principales características de dicho lenguaje en base a su aplicación en un caso de estudio. Como resultado, se presenta un análisis de las fortalezas y debilidades de la versión actual del lenguaje, y se esbozan perspectivas de extensión del mismo.

1. INTRODUCCION

El término *mashup* se utiliza para caracterizar aplicaciones de software que se basan en la integración y composición de datos y servicios de distintos orígenes. Esta integración es en ocasiones llevada adelante por los propios usuarios, que crean una nueva aplicación para satisfacer sus requerimientos [1][2]. Típicamente, esta integración es de carácter liviano, de un bajo nivel de acoplamiento, y no invasiva en relación a las fuentes de datos o servicios que integra ya que se basa en el uso de infraestructura existente.

Se denomina *Mashups Empresariales* a *mashups* concebidos para ser utilizados en ámbitos corporativos. Estos se caracterizan por su capacidad de acceder a orígenes de datos relacionales y Web Services publicados internamente a la organización. El uso de tales fuentes de datos y servicios implica, típicamente, un contexto de seguridad no requerido en los mashups tradicionales (de uso público), lo cual plantea desafíos adicionales.

El Enterprise Mashup Markup Language (EMML) [3] es un lenguaje específico de dominio orientado a la definición de mashups empresariales. El mismo se basa en un markup XML, el cual permite la definición de los flujos de integración de datos y servicios utilizando primitivas de alto nivel. Tales primitivas permiten, entre otros: la creación de invocaciones Web, consultas a base de datos, manipulación de XML mediante XSLT, merge de documentos XML, manejo de bucles, sentencias IF de control condicional, etc. La versión 1.0 del estándar se publicó a fines de setiembre de 2009.

EMML está impulsado y regido por la Open Mashup Alliance (OMA) [4], con el fin de promocionar y estandarizar la implementación de soluciones empresariales basadas en mashups que puedan ser interoperables entre las plataformas ofrecidas por distintos vendedores. La OMA está constituida como un consorcio de empresas, y vendedores de plataformas para la creación de mashups entre los que se encuentra JackBe [5].

En este reporte se analizan fortalezas y debilidades del estándar existente, esbozándose posibles mejoras al mismo.

El resto del reporte se organiza de la siguiente forma. La Sección 2 presenta las características del lenguaje EMML. La Sección 3 analiza fortalezas y debilidades del EMML. La Sección 4 esboza posibles extensiones al EMML, y la Sección 5 plantea conclusiones y trabajos futuros.

2. EMML El Lenguaje

EMML es un dialecto XML, basado en la definición de documentos con un nodo *mashup* como raíz, el cual puede contener como nodos hijos a primitivas del lenguaje. Un *mashup* en EMML, que puede ser parametrizado mediante la declaración de las entradas esperadas, retorna un documento XML como salida de la ejecución del mashup.

Las primitivas del lenguaje permiten expresar la invocación de Web Services, realizar consultas a bases de datos relacionales, así como manipular los resultados obtenidos en formato de documento XML para generar la composición deseada. Algunos ejemplos de las manipulaciones posibles son: filtrar, agrupar, unir, realizar joins de documentos, basándose en XPath como lenguaje de expresiones para identificar subnodos en los documentos manejados.

A continuación, a los efectos de describir el lenguaje, en primer lugar se van a enumerar y describir brevemente algunas de las primitivas de uso más frecuente del lenguaje. En una segunda instancia se van a repasar ejemplos de escenarios de uso, que ejemplifiquen el uso de las primitivas presentadas previamente.

2.1 Las primitivas

<variable> Permite el almacenamiento de resultados temporales, internos a la ejecución de un mashup.

<input> Se utiliza para especificar los parámetros de entrada de un script de mashup.

<output> Especifica el parámetro que contiene el resultado de la ejecución de un script de mashup.

<directinvoke> Permite la invocación de servicios, o sitios Web, de acceso público en la Web. Los Web Services deben brindar una interfaz REST, SOAP, o RSS/ATOM.

<sql> Permite la ejecución de consultas SQL individuales, así como también procedimientos almacenados.

<filter> Permite filtrar un XML en base al uso de expresiones XPath

<annotate> Permite agregar atributos, o elementos hijos a un nodo seleccionado de una variable, mediante el uso de una expresión XPath.

<script> Permite embeber un bloque de código Javascript, o referenciar un archivo externo, el cual será procesado en tiempo de ejecución.

<xslt> Permite incluir una referencia a una transformación XSLT para que sea procesada en tiempo de ejecución y aplicada a una variable del mashup.

<join> Realiza un join de dos o más servicios basado en una condición de join. Esta primitiva opera como una operación *Join* de base de datos, donde se requiere que las fuentes manejen nodos "clave" que sirvan para determinar cómo hacer el *join* de los datos.

<merge> Es equivalente al operador UNION en base de datos. Realiza la unión de juegos de datos homogéneos.

<appendresult> Realiza un append de una estructura de nodos a una variable del mashup. Típicamente usado en el cuerpo de loops para la construcción de estructuras XML.

<select> Crea una estructura únicamente con los ítems seleccionados en un conjunto de repetido de ítems.

<if> Permite procesamientos condicionales de la forma if-then-else

<foreach> Permite la implementación de bucles iterativos.

2.2 Ejemplos de uso

En esta sección se presentan dos ejemplos EMML orientados a describir el funcionamiento del lenguaje.

2.2.1 Ejemplo 1 - Webclipping

El ejemplo a continuación realiza una búsqueda en google, y retorna los resultados obtenidos en un formato XML.

```
<mashup name="GoogleWebClipping">
    <input name="query" type="string" default="uruguay"></input>
    <output name="result" type="document"/>
    <directinvoke outputvariable = "searchresult1"
endpoint="http://www.google.com/search?q={$query}"/>

    <foreach variable="query"
items="$searchresult1//xhtml:a[@class='l']">
      <appendresult outputvariable="result">
        <itemlink>{$query/@href}</itemlink>
      </appendresult>
    </foreach>
</mashup>
```

Para lograr esto declara los parámetros de entrada y salida. Realiza una invocación a la página de búsqueda de google pasando el parámetro recibido. Finalmente itera sobre la página resultante en formato xhtml, y por cada nodo con un atributo class con valor "l" fuerza la creación de un nuevo nodo itemlink en el documento resultado.

2.2.2 Ejemplo 2 - Join

El siguiente ejemplo realiza un *join* del contenido de dos documentos XML cuya declaración se encuentra embebida en el script EMML.

```
<mashup name="CompanyInfoJoinSample">
    <input name="xigniteProfiles" type="document">
      <profiles>
        <profile>
          <ticker>GOOG</ticker>
          <companyname>Google Inc.</companyname>
          <address>1600 Amphitheatre ParkwayMountain View,
CA94043United States</address>
          <phone> Phone: 650-253-0000 Fax: 650-253-0001 </phone>
          <url>http://www.google.com</url>
          <summary>Google, Inc., a technology company, maintains
index of Web sites and other online content for users, advertisers, Google
network members, and other content providers. Its automated search
technology helps users to obtain instant access to relevant information
from its online index.</summary>
          <source>Yahoo! Finance</source>
        </profile>
        <profile>
          <ticker>JAVA</ticker>
          <companyname>Sun Microsystems Inc.</companyname>
          <address>4150 Network CircleSanta Clara, CA95054United
States</address>
          <phone> Phone: 650-960-1300 Fax: 650-336-0646 </phone>
          <url>http://www.sun.com</url>
          <summary>Sun Microsystems, Inc. develops products and
services for the network economy. It provides network computing
infrastructure product and service solutions worldwide under brands,
including Java technology platform.</summary>
          <source>Yahoo! Finance</source>
        </profile>
      </profiles>
    </input>

    <input name="yahooFinanceQuotes" type="document">
      <GetQuotesResponse>
          <Quote>
            <Symbol>JAVA</Symbol>
            <Name>Sun Microsystems, Inc.</Name>
            <Open>15.12</Open>
            <High>15.51</High>
            <Low>15.11</Low>
            <Last>15.38</Last>
            <Volume>12573587</Volume>
            <Change>0.12</Change>
          </Quote>
          <Quote>
            <Symbol>GOOG</Symbol>
            <Name>Google Inc.</Name>
            <Open>469.41</Open>
            <High>472</High>
            <Low>457.54</Low>
            <Last>464.19</Last>
            <Volume>6048035</Volume>
            <Change>-3.62</Change>
          </Quote>
      </GetQuotesResponse>
    </input>

    <output name="resultDoc" type="document"/>

    <!-- Join multiple datasets using a equivalent data field (e.g. ticker &
Symbol).  Multiple conditions
        may be combined using and / or logical operators.
    -->
    <join outputvariable="resultDoc"
        joincondition="$xigniteProfiles/profiles/profile/ticker =
$yahooFinanceQuotes/GetQuotesResponse/Quote/Symbol"/>
</mashup>
```

3. Análisis

El análisis realizado se basa en el estudio del lenguaje y en su aplicación en el contexto de un caso de estudio.

El escenario elegido para el caso de estudio corresponde a la integración de la información de docentes del InCo, la cual es información pública [07], con la lista de publicaciones de cada docente tomadas del sitio DBLP [08]. De ahora en más denominaremos a dicho escenario como: Publicaciones-Docentes.

A modo de benchmark se realizó una implementación a medida de dicho mashup utilizando Java como lenguaje de programación, y aumentada con información del perfil LinkedIn[09] de cada docente cuando esta existía.

Los resultados del análisis se deben leer a la luz de los escenarios y objetivos para los cuales fue diseñado EMML, los cuales, según la página de la OMA, se orientan a:

- Escenarios empresariales, donde se acceden fuentes internas y externas de datos y servicios
- Lograr el objetivo de estandarización del formato de metadata utilizado para almacenar scripts de mashups entre herramientas de distintos vendedores, reduciendo el riesgo de vendor-lock in para los potenciales compradores.

3.1 Caso de Estudio

En una primera instancia, se intentó el abordaje más simple posible utilizando EMML para la implementación del escenario de prueba Publicaciones-Docentes. El script resultante es el siguiente:

```
<mashup name="PublicacionesDocentes">
    <output name="result" type="document"/>
    <directinvoke
endpoint="http://www.fing.edu.uy/inco/pm/Integrantes/Docentes"
            outputvariable="docentesPage"/>
    <foreach variable="query"
items="$docentesPage//xhtml:a[@class='urllink']">
            <appendresult outputvariable="result">
        <docente>{$query/string()}</docente>
      </appendresult>
      <directinvoke
        endpoint="http://dblp.uni-
trier.de/search/author?author={$query/string()}"
        outputvariable="pubsPage"/>
      <foreach variable="queryPub"
items="$pubsPage//xhtml:a[@class='pubName']">
        <appendresult outputvariable="result">
          <publicacion>{$queryPub/string()}</publicacion>
        </appendresult>
      </foreach>
    </foreach>
</mashup>
```

En el mismo se intenta lanzar una consulta http a la página de docentes, y por cada docente realizar una consulta adicional a la página de DBLP para obtener las publicaciones de dicho docente. Con la información recolectada se crea una estructura XML de docentes y sus publicaciones.

Lamentablemente este abordaje no fue exitoso, en primer lugar por la existencia de bugs en la implementación de referencia que hacen imposible el consumo de fuentes de datos que manejen caracteres de datos no ASCII, como tildes o eñes. En segundo lugar la ejecución de este script sufre de un problema arquitectural: por cada consulta que se dispara en el bucle hay una demora en el orden de varios segundos. Dado que la cantidad de docentes está en el orden de las decenas, el tiempo total requerido para ejecutar el mashup es muy superior al minuto, lo cual es inaceptable como tiempo de respuesta de un Web Service.

De todas formas, en base al ejercicio realizado, se identificaron fortalezas y debilidades, las cuales se describen a continuación

3.2 Fortalezas

3.2.1 Simplicidad

EMML resulta intuitivo y fácil de entender para perfiles técnicos, con experiencia en el uso de lenguajes de programación imperativos, y conocimientos de tecnologías web.

Está acompañado de una implementación de referencia del intérprete del lenguaje cuya instalación y ejecución no es compleja e incluye varios ejemplos. Se distribuye como un archivo WAR, para su ejecución en un servidor web java, tal como el Tomcat de Apache[10].

3.2.2 Componibilidad

El resultado de un mashup definido con EMML está orientado a que sea un XML, el cual puede a su vez ser tomado como entrada en la implementación de otro mashup definido con EMML. Si bien el programador de un script EMML puede elegir salirse de este modelo, generando una salida de texto, o html, se entiende que no es lo que induce la plataforma.

Podemos decir que los mashups EMML tienden a ser servicios de datos componibles, permitiendo su uso y reuso como building blocks dentro de una organización.

3.2.3 Basado en estándares

EMML se basa en estándares como XML, XML Schema & XPath, lo cual facilita su entendimiento si ya se manejan dichas tecnologías, y la aplicación de herramientas de terceras partes para la creación y validación de los contenidos de un script EMML.

3.2.4 Separación entre contenido y presentación

El resultado de un script EMML está orientado a que sea un XML, el cual a priori no incluye información de presentación, a no ser que se decida retornar un XHTML, como resultado, lo cual se puede hacer, o incluir el uso del tag script, para incluir bloques javascript para que sean ejecutados en el navegador cliente, pero estas acciones quedan en la órbita de responsabilidades del programador.

Retornar XML permite que el resultado de la ejecución de un script EMML pueda ser utilizado por cliente móviles, y no solo navegadores web.

3.3 Debilidades

3.3.1 Distribución de los componentes

Los mashups puede ejecutarse como *server-side*, donde todas las composiciones y transformaciones son ejecutadas del lado del servidor, o pueden también como *client–side,* donde mediante lenguajes de scripting se realiza la integración de datos y servicios en el cliente, el cual puede ser un navegador o una aplicación ejecutando en un dispositivo móvil.

El problema de performance que se describe en 3.1, en el cual el tiempo necesario para ejecutar el mashup supera al minuto, es inherente a pretender realizar todas las integraciones en el lado del servidor y en una misma ejecución atómica, sin brindar visibilidad al cliente de lo que está ocurriendo. Un abordaje para solucionarlo consiste en diferir la ejecución de consultas en el tiempo, mediante la inclusión de comunicaciones asíncronas, permitiendo al usuario final ver resultados parciales.

EMML induce el desarrollo de integraciones del lado de servidor, donde cada ejecución es atómica. Su soporte para declarar procesamientos del lado del cliente es muy pobre, y no maneja primitivas para el manejo de asincronismos ni *callbacks*.

Una solución elegante probablemente requeriría de un nivel más rico de expresividad, para permitir declarar que ciertas integraciones se realizan en el lado del cliente, y permitir definir comportamientos asociados a eventos.

3.3.2 Caching

EMML no brinda ningún soporte para el *caching* de datos, lo cual podría ser una solución válida para escenarios como el previamente discutido.

En una forma más amplia observamos que EMML, no contempla en su definición primitivas que permitan definir el comportamiento de *cross cutting concerns* o, intereses ortogonales, típicamente relacionados a la implementación de requerimientos no funcionales necesarios para la puesta en funcionamiento de un mashup, como son: la calidad de servicio (tiempo de respuesta), calidad de los datos, robustez frente a fallos, seguridad, etc.

Quizás los creadores del EMML dejaron esto intencionalmente por fuera del lenguaje, pero esto complica su uso para la creación de mashups correspondientes a casos reales y no tanto de ficticios de muy pequeño porte.

3.3.3 Asincronismo

En gran medida la revolución de sitios Web 2.0 se basa en la capacidad de realizar invocaciones asincrónicas al servidor, mediante el uso de técnicas de scripting AJAX, las cuales permiten refrescar solo la porción afectada por los resultados de una consulta cuando estos son retornados, y no el total de la página. El resultado es una experiencia de usuario mucho más rica, que siga la buena práctica de mostrarle al usuario el estado del sistema, y presentar la información a medida que se va obteniendo. EMML no brinda mecanismos para soportar comunicaciones asíncronas.

3.3.4 Interfaz de usuario

EMML es un lenguaje orientado a la definición de servicios de datos, y se dejó de lado el brindar herramientas que permitan visualizar estos datos. La capacidad de generar componentes de presentación en EMML, está asociada a la generación de XHTML como resultado, y a retornar scripts javascript. Esto tiene la desventaja que acopla la presentación generada a una plataforma, no permitiendo reutilizar la integración definida en otros entornos. Se entiende que existe el potencial para que un DSL para especificar mashups brinde mecanismos para describir componentes de interfaz de usuario con un nivel de abstracción que no lo acople a una plataforma específica.

En el marco de mashups, la capa de presentación desempeña un rol fundamental. Según los estudios realizados[06] cerca de un 50% de los mashups registrados en el sitio *Programmable Web*, corresponden a la visualización de datos geo-referenciados mezclando los servicios de Google Maps, con los de alguna otra fuente de datos. Este tipo de mashups que corresponden aproximadamente a la mitad de mashups existentes, no puede ser definidos con EMML, sin recurrir a hacks javascript.

Los mashups existen para satisfacer alguna necesidad puntual de integración de datos de uno o varios usuarios, y no están completos hasta que brinden una forma para que el usuario final los pueda consumir. Se puede argumentar que un mashup también es la presentación que se realiza de la información.

En este sentido los mashups definidos con EMML no están completos hasta que se especifique de alguna manera cual va a ser la presentación asociada a las integraciones que realiza.

3.3.5 Seguridad

No se encontraron como parte del estudio, primitivas o facilidades a nivel de lenguaje orientados a resolver procesos de autenticación en la invocación de Web Services. Parte del desafío de los mashups empresariales pasa por el acceso a fuentes de datos organizacionales, las cuales requieren un contexto de seguridad.

En el caso de conexiones a bases de datos relacionales las opciones son dos, incluir la información de usuario y password en el script EMML, o referenciar una conexión JNDI. Este soporte es básico ya que o infringe buenas prácticas de seguridad, o acopla el lenguaje a la plataforma JEE de programación.

3.3.6 Flujo de navegación

Así como no se brindan herramientas a nivel de lenguaje para especificar componentes de presentación, no se puede realizar mashups que se basen en el input del usuario como un origen de datos. Los mashups implementados en base a EMML no guardan estado, tienen más en común con un servicio SOA, que con una aplicación que pueda ser consumida por un usuario. Si bien permiten recibir parámetros, están orientados a generar scripts de consulta que retornen datos, sin tener efectos secundarios.

No se brindan mecanismos para establecer una secuencia de interacciones con el fin de recolectar información que pueda posteriormente ser utilizada para almacenar datos. EMML tiene como resultado un script que se comporta como reporte, que puede recibir parámetros, a partir de los cuales realiza una integración, y muestra un resultado. Esto no incluye toda la familia de mashups que

se basan en el input del usuario como un origen de datos, como lo es Wikicrimes [11].

3.3.7 Primitivas para orígenes de datos más populares

El ecosistema de mashups, estudiado a partir de los mashups publicados en el sitio de programable web[12] cumple que un 80% de los mashups se basan en menos de un 20% de las APIs existentes.

Existen APIs y servicios extremadamente populares que concentran la mayoría de mashups, tales como Google Maps, Flickr, Facebook, Twitter, Amazon WS. Se entiende que se podría sacar ventaja de esto, a nivel de lenguaje, definiendo primitivas correspondientes a cada uno de los servicios más utilizados

Cabe la posibilidad de que en el diseño del lenguaje este tipo de aspecto se haya decidido relegar a los entornos de desarrollo propietarios de cada vendedor. Los cuales pueden facilitar este tipo de construcciones aunque por debajo generen un script EMML estándar. En este escenario este tipo de facilidad se brindaría a nivel de herramienta IDE, y no queda capturado a nivel de lenguaje. Resta evaluar las herramientas correspondientes para entender si este es el caso, o directamente este aspecto no está contemplado.

3.3.8 Semántica

Dada la falta de mecanismos para especificar componentes de presentación, una alternativa a explotar podría ser la decoración de los resultados de scripts EMML con semántica, lo cual se entiende facilitaría en gran medida la creación de asistentes que automaticen la generación de interfaces de usuario, para un mashup determinado.

3.4 Debilidades o espacios de mejora en la implementación

Hasta el momento se han enumerado debilidades a nivel de la especificación de lenguaje. A continuación se listan debilidades detectadas a nivel de la implementación brindada de EMML, que se entiende actúan en contra de la adopción del estándar.

3.4.1 Implementación de referencia incompleta

Corregir los bugs y brindar escenarios de uso más complejos como parte de la distribución ayudaría el proceso de adopción de la nueva tecnología. Actualmente los ejemplos brindados son a modo de ejemplificar el uso de alguna primitiva.

3.4.2 Licenciamiento restrictivo

La licencia actualmente es del tipo Creative Commons Attribution Non Derivatives la cual implica que la OMA retiene el control sobre el lenguaje y no se pueden realizar trabajos derivados. Se entiende que las plataformas y herramientas para el desarrollo de mashups están lejos de ser tecnologías maduras, y tienen mucho por recorrer. En este sentido, sería bueno que el estándar promoviera la definición de extensiones y trabajos relacionados, lo cual podría incrementar los niveles de adopción de la tecnología.

4. Perspectivas de extensión.

Del análisis de las debilidades detectadas se desprende que **EMML no brinda mecanismos adecuados para abordar la especificación de comportamientos no funcionales, tales como (asincronismo, caching, distribución, autenticación, presentación) necesarios para la implementación de un mashup**. La implementación de este tipo de aspectos queda en la órbita de los hacks, xslt, o script que puede elegir hacer el programador, pero **no son ciudadanos de primera clase en EMML**.

Muchas de las debilidades detectadas pueden ser categorizadas como aspectos, o cross cutting concerns, (distribución, caching, seguridad, presentación) sobre las cuales EMML no brinda primitivas que permitan predicar sobre estos comportamientos.

EMML brinda propone un lenguaje donde los mashups resultantes que su utilización se asume que no tienen efectos secundarios, su resultado está orientado a ser un XML, lo cual permite que actúen como building blocks y sean componibles. Si bien se entiende que esto representa una fortaleza, al mismo tiempo como ya se repaso previamente el lenguaje por si solo es inadecaudo, o por lo menos incompleto para especificar un mashup en su totalidad, contemplando todos los aspectos de la implementación del mismo, incluyendo su presentación, distribución, performance.

Utilizando esto como base se elaboró un conjunto de extensiones orientadas a subsanar las debilidades encontradas como resultado del ejercicio de poner a prueba el lenguaje en la implementación de un mashup como caso de estudio, buscando dotar mediante estas extensiones al lenguaje de mecanismos para poder predicar sobre aspectos no funcionales *cross cutting concerns*.

4.1 Primitivas propuestas

Se busca dotar al lenguaje de un nivel de expresividad que permita la definición de todos los aspectos relevantes de un mashup, pero no más. Para esto se propone la creación del siguiente conjunto de primitivas:

<dist:server> Declara que la integración que se especifica en el cuerpo de dicho elemento XML debe ser llevada a cabo en el servidor.

<dist:client> Declara que la integración que se especifica en el cuerpo de dicho elemento XML debe ser llevada a cabo en el cliente (web, mobile, etc).

<dist:endpoint> Especifica un servicio de datos que puede ser consumido.

<cache:cacheable> Especifica que un servicio de datos puede ser cacheado, según un campo definido en el atributo onfield, con una expiración definida en el atributo expiration.

<as:asinclnvoke> Invocación asíncrona de un servicio de datos.

<ui:map> Componente de presentación de mapa.

<ui:bindToMap> Posiciona información georeferenciada en un componente de mapa, según un campo el cual tiene las geocoordenadas.

<xforms… Se propone recurrir al uso del estándar xforms para definir formularios, orientados a relevar datos, y componentes de presentación.

<sec:oauth> La integración mashup que se encuentre definida en el cuerpo de este elemento XML solo puede ser accedida luego de una autenticación frente a una autoridad OAuth.

4.2 Caso de Estudio Revisado

La implementación del caso de estudio propuesto utilizando las nuevas primitivas podría expresarse de la siguiente manera:

```
<mashup>
<dist:client type="html/javascript">
<ui:panel>
  <b:directinvoke endpoint="endpoint:docentes"
outputvariable="docentesDOM"/>
  <ui:grid>
    <b:foreach variable="query"
items="$docentesPage//itemlink:a[@class='urllink']">
      <ui:row>
        <b:invokeAsync endpoint="endpoint:pubs"/>
      </ui:row>
    </b:foreach>
  </ui:grid>
</ui:panel>
</dist:client>
<dist:server>
<dist:endpoint name="docentes">
  <output name="result" type="document"/>
  <cache:cacheable onfield="docente"
expiration="DD:1"/>
    <directinvoke
endpoint="http://www.fing.edu.uy/inco/pm/Integrantes/D
ocentes" outputvariable="docentesPage"/>
    <foreach variable="query"
items="$docentesPage//xhtml:a[@class='urllink']">
      <appendresult outputvariable="result">
        <itemlink>{$query/string()}</itemlink>
      </appendresult>
    </foreach>
  </dist:endpoint>

<dist:endpoint name="pubs">
  <cache:cacheable onfield="docente"
expiration="DD:1"/>
    <output name="result" type="document"/>
    <param name="docente" querystring="doc"/>

    <directinvoke endpoint="http://dblp.uni-
trier.de/search/author?author={$docente/string()}"
outputvariable="docentesPage"/>

    <foreach variable="query"
items="$docentesPage//xhtml:a[@class='urllink']">
      <appendresult outputvariable="result">
        <itemlink>{$query/string()}</itemlink>
      </appendresult>
    </foreach>
  </dist:endpoint>
</dist:server>
</mashup>
```

5. Conclusiones y Trabajos Futuros

En este reporte se presentaron resumidamente las características del lenguaje EMML para la definición de mashups, y se hizo un análisis de fortalezas, debilidades y espacios de mejora en base a su aplicación en el desarrollo de un mashup elegido como caso de estudio.

A modo de síntesis, se destacan como fortalezas principales del lenguaje: su simplicidad, componibilidad y el hecho de que se encuentre basado en estándares.

Como resultado del análisis se detectaron varias debilidades, entre ellas: su falta de soporte para realizar aspectos de la integración en el cliente, o especificar a nivel de lenguaje componentes de la presentación necesarios para poder consumir los resultados de un mashup ,y en ocasiones parte fundamental del mashup, como lo es en los casos de integración con Google Maps.

En base a las debilidades detectadas se formuló una propuesta de nuevas primitivas para extender el lenguaje con el fin de dotarlo del nivel de expresividad necesario para que pudiera especificar el mashup planteado en el caso de uso, y mitigar las debilidades detectadas.

Se concluye que EMML aborda una porción de las problemáticas involucradas a la hora de construir mashups, haciendo foco en la integración de datos, para generar resultados de solo lectura. Para poder utilizarlo en la puesta en marcha de mashups se debe complementar con componentes de presentación, y configuraciones a nivel de web server, o extensiones programáticas para abordar aspectos no funcionales como pueden ser la implementación de caching para lograr niveles de performance razonables.

Queda pendiente como trabajo futuro la construcción de una implementación extendida del motor EMML, con soporte para las nuevas primitivas propuestas, validada contra el conjunto de casos de estudio propuestos.

6. REFERENCIAS

[1] T. Fischer, F. Bakalov, and A. Nauerz, “An Overview of Current Approaches to Mashup Generation,” Wissensmanagement, 2009, pp. 254-259.

[2] D. Merril. Mashups: The new breed of Web app. Website 08 2006. http://www.ibm.com/developerworks/library/x-mashups.html [Accessed: April 2010]

[3] Enterprise Mashup Markup Language http://www.openmashup.org/omadocs/v1.0/emml/oma-readme.html [Accessed July 2010]

[4] Open Mashup Alliance http://www.openmashup.org/ [Accessed July 2010]

[5] JackBe Presto. http://www.jackbe.com/products/ [Accessed: April 2010]

[6] Shuli Yu, Jason Woodard, “Innovation in the Programmable Web: Characterizing the Mashup Ecosystem”. Second International Workshop on Web APIs and Service Mashups 2008

[7] Mashup DocentesInco http://docentesinco.appspot.com/docentesinco [Accessed December 2010]

[8] DBLP CompleteSearch http://dblp.mpi-inf.mpg.de/dblp-mirror/index.php [Accessed December 2010]

[9] LinkedIn http://www.linkedin.com/ [Accessed December 2010]

[10] Apache Tomcat http://tomcat.apache.org/ [Accessed December 2010]

[11] Wikicrimes. http://wikicrimes.org/ [Accessed: December 2010]

[12] Programmable Web http://www.programmableweb.com/ [Accessed December 2010