

Una aproximación a la conciliación contexto-versionado

Jorge Martínez Gil
Dept. de Sistemas Informáticos y
Telemáticos
Escuela Politécnica
Universidad de Extremadura
10071 Cáceres
jmargil@unex.es

Antonio Polo Márquez
Dept. de Sistemas Informáticos y
Telemáticos
Escuela Politécnica
Universidad de Extremadura
10071 Cáceres
polo@unex.es

Luis J. Arévalo Rosado
Dept. de Sistemas Informáticos y
Telemáticos
Centro Universitario de Mérida
Universidad de Extremadura
06800 Mérida
ljarevalo@unex.es

Resumen

En el presente artículo se propone una aproximación al problema de la conciliación entre el contexto y el versionado en aras de conseguir sistemas de información que se adapten al usuario. Nuestra propuesta consiste en establecer una correspondencia entre el contexto del usuario y las distintas versiones de la información a la que quiere acceder. Como consecuencia de dicho trabajo se puede extraer que dicha correspondencia no puede hacerse de manera viable si el modelo de almacenamiento de las versiones no ofrece información semántica suplementaria. El objetivo de este trabajo es, pues, la discusión en un foro abierto de las posibilidades que existen para añadir semántica a nuestro esquema de solución.

Palabras clave: Versionado XML, contexto, adaptabilidad, perfiles.

1. Introducción

En este artículo pretendemos abordar la selección de versiones en función de un contexto (asociado o qué define a un usuario). Para ilustrarlo proponemos un ejemplo: Supongamos que administramos una biblioteca digital (BD), con contenidos en formato XML para facilitar que los lectores escojan el formato de la información a la que acceden, y donde hay alojados grandes clásicos de la literatura, por otra parte, supongamos que los usuarios de dicha BD se divide en público infantil y público adulto. A priori, parece una buena idea ofrecer contenidos adecuados para cada perfil. De esta forma no parece razonable introducir a niños en el mundo de los clásicos con la lectura de los originales, no obstante, lo que sí parece adecuado es crear versiones simplificadas que les ayuden a

comprender algunos aspectos clave de dichas obras. No sucedería lo mismo en el caso de un lector adulto, que potencialmente debería tener acceso a las dos versiones de la información. Por un momento supongamos que la división de los lectores de nuestra BD no es tan obvia, y que entre los grupos que demandan las obras que alojamos hay niños, estudiantes de lenguas clásicas, lectores adultos, estudiantes de lenguas extranjeras e investigadores. La labor de asignación de los permisos de acceso a cada una de las versiones se hace más complicada, haciendo prácticamente inviable su asignación manual.

En este ejemplo, cada una de las vistas de una obra forma el conjunto de versiones, mientras el contexto [10,11] sería el perfil (agente si se prefiere), pero no es difícil imaginar situaciones donde el contexto puede ser una agregación de términos que se refieren además del perfil, al espacio y al tiempo (dónde un niño que accede, lo hace además desde Turquía y en el año 2007). La complejidad crece hasta límites exponenciales.

La propuesta que pretendemos transmitir es un mecanismo que resuelva problemas de este género, creemos que se enmarca dentro de lo que en ingeniería se conoce como adaptabilidad, puesto que nuestro fin último es el de conseguir sistemas que “comprendan” al usuario (o a la usuaria), es decir, que identifiquen sus necesidades y se adapten a él (o a ella).

Antes de abordar su resolución, parece sensato enfrentarse al problema desde una perspectiva científica, por ello pretendemos:

- 1) Estudiar los intentos de solución previos que hay recogidos en la literatura científica (Sección 2).
- 2) Buscar un modelo de versionado de la información que facilite la solución (Sección 3).
- 3) Definir claramente qué entendemos por contexto y cómo puede usarse para la selección de la versión adecuada a cada situación. (Sección 4).

4) Analizar los resultados de nuestra propuesta y ofrecer las líneas futuras de investigación (Sección 5).

2. Estado del arte

La primera solución al problema que se nos podría ocurrir es la de mantener el contenido en un fichero independiente por cada versión al que se le asignarían los permisos, por tanto, nuestra propuesta se reduciría al diseño de un sistema de asignación de permisos, sin embargo, esta aproximación presenta dos debilidades; por una parte, existiría una gran cantidad de texto duplicado, lo que es sumamente ineficiente. Por otra parte, a lo largo del tiempo podrían aparecer muchas versiones nuevas, cada una en un fichero independiente, por lo que habría que tratar a los ficheros por separado, con el coste que ello lleva asociado. Por ello, buscamos soluciones basadas en la integración de las distintas versiones.

Si buceamos un poco en la literatura científica, podemos comprobar que el problema ha sido abordado desde un punto de vista que trata de almacenar tipos de marcas diferentes en un mismo documento. Dichas soluciones han sido estudiadas por el Metalanguage Working Group en su Text Encoding Initiative (TEI) [5], basándose en el estándar SGML[6]. Además, en [3] se estudian los diferentes tipos de anidamiento que pueden darse entre dos marcas, haciendo hincapié en resolver las situaciones con solapamiento.

Siguiendo la línea marcada por el TEI, se han llevado a cabo varios trabajos de investigación, entre los que destaca TexMecs [8] que define un nuevo lenguaje sintáctico para dar solución a la ocurrencia de marcas concurrentes o GODDAD [9] en el que se usa un complejo grafo acíclico dirigido para almacenar documentos concurrentes, y que posteriormente puede usarse en un framework para almacenar y consultar documentos XML basándose en el contenido PCDATA.

Otra solución que se aproxima a nuestra idea es [7], donde a través de una técnica de meta marcado (marcas sobre las marcas), se permite transformar cualquier documento XML en otro equivalente, para lo que procede sustituyendo cada marca de inicio y de fin por un elemento vacío que lo representa.

Sin embargo, todas las soluciones se han diseñado teniendo en cuenta problemas de concurrencia, es decir, tratan el problema desde el interior del documento. Nuestra propuesta trata de buscar una solución factible desde el exterior del documento, gracias al contexto externo, y téngase en cuenta que el contexto puede ser desde algo tan sencillo como un tipo de usuario hasta la agregación de múltiples características.

3. En busca de un modelo de versionado satisfactorio

La necesidad de gestionar versiones no es nueva; su necesidad en el mundo de la informática ya se hizo patente a la hora de trabajar en el desarrollo de grandes proyectos de software. El versionado de XML surgió de la necesidad de dar soporte a las bases de datos que tratan a XML de forma nativa o los almacenes de data warehousing. No obstante, cada día es más creciente la demanda de técnicas para tratar la información semiestructurada, y buena prueba de ello la tenemos en que las grandes suites ofimáticas del mercado basan sus nuevos formatos de fichero en XML o el creciente número de aplicaciones que usan XML para almacenar la configuración de usuario. Las propuestas para tratar las versiones XML en la actualidad se pueden reunir en tres grandes grupos:

- Delta XML Management[2,4], que consiste en la aplicación de las técnicas tradicionales adaptadas a XML, es decir, la obtención y el almacenamiento de las diferencias entre dos versiones dadas. Su aplicación ha sido muy estudiada y probada, sin embargo, no permiten la validación de las versiones ni las consultas con lenguajes como XQuery o XPath.
- Timestamp (representación temporal) [3], que usa técnicas derivadas de las bases de datos temporales consistentes en marcar cada etiqueta del documento con información temporal sobre los períodos de tiempos en que dicha etiqueta puede considerarse válida. Su principal problema es que no soporta el versionado ramificado, debido a la naturaleza lineal del tiempo.

- Versionstamp, desarrollada en [1], presenta una técnica que permite solventar las desventajas de las dos técnicas anteriores. Por un lado indica en el documento qué versiones se incluyen y las relaciones entre ellas y por otro lado añade una marca de validez a cada etiqueta en base a esta información.

Puesto que Versionstamp parece completar las carencias de las que adolecen las dos primeras, será la técnica sobre la que vamos a basar nuestro trabajo.

3.1. Breve descripción de Versionstamp

Como venimos diciendo, un documento XML no es estático, pues o bien evoluciona a lo largo del tiempo y es necesario realizar su gestión para permitir consultas sobre las versiones pretéritas o bien evoluciona paralelamente según las premisas de los usuarios que están trabajando con él. Partiendo de un estado inicial del documento que denotaremos como V1, las nuevas versiones se producirán aplicando un número de cambios a la versión actual o en nuestro caso a cualquier versión del documento. Por ejemplo en la Figura 1 se muestra la evolución que ha tenido un documento XML, que almacena información de restaurantes de una determinada ciudad, a lo largo del tiempo.

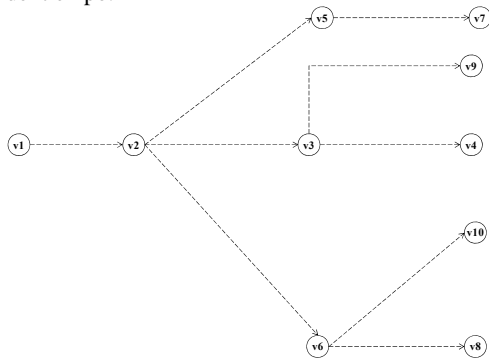


Figura 1. Representación de un esquema de versionado

En esta solución [1] las distintas versiones de este documento son integradas junto con las anteriores versiones en lo que hemos denominado un documento maestro o Vdocument.

El soporte ramificado en nuestra solución se basa en dos ideas: Por un lado indicar en el documento qué versiones se incluyen y las relaciones entre ellas y por otro lado en marcar la validez de cada etiqueta en base a esta información.

Para la primera idea hemos determinado que mediante una estructura arbórea, es decir un documento XML, se puede representar la historia de las versiones, pues cada elemento del árbol representa una versión del documento al que se le está siguiendo la pista. Cada uno de estos elementos tiene un identificador de versión y el anidamiento de estos elementos supone una relación de descendencia, de tal modo que el elemento más externo (llamémoslo Vi) es el más antiguo, y el más interno es el creado más recientemente (Vj).

En segundo lugar para cada etiqueta del documento se ha añadido información que nos indica su validez de versionado, es decir, en que versiones del árbol dicha etiqueta es válida lo que hemos denominado región de versionado. En este trabajo no se va a profundizar sobre esta representación, pero todos los detalles pueden obtenerse en [1].

4. Cómo adaptar el contexto al versionado

Lo que nosotros proponemos resulta de añadir básicamente un multiplexor lógico al esquema de recuperación de versiones de Versionstamp.

La arquitectura de conciliación que proponemos se basa en la Figura 2, donde una consulta efectuada por un usuario entra en el multiplexor lógico. Por otra parte las entradas de selección de dicho multiplexor pueden ser capturadas de diversas formas: cabeceras de un explorador, introducidas por el usuario, negociadas mediante un protocolo de handshake, etc. y conforman el contexto del usuario que efectúa la consulta. La salida de dicho multiplexor es la entrada al servicio de recuperación de una o más versiones de Versionstamp, que devolverá al usuario la versión calculada por el multiplexor. Dicha versión será devuelta en XML, por lo que aún quedará transformarla en un formato legible para el usuario (pdf, html, rtf, etc...).

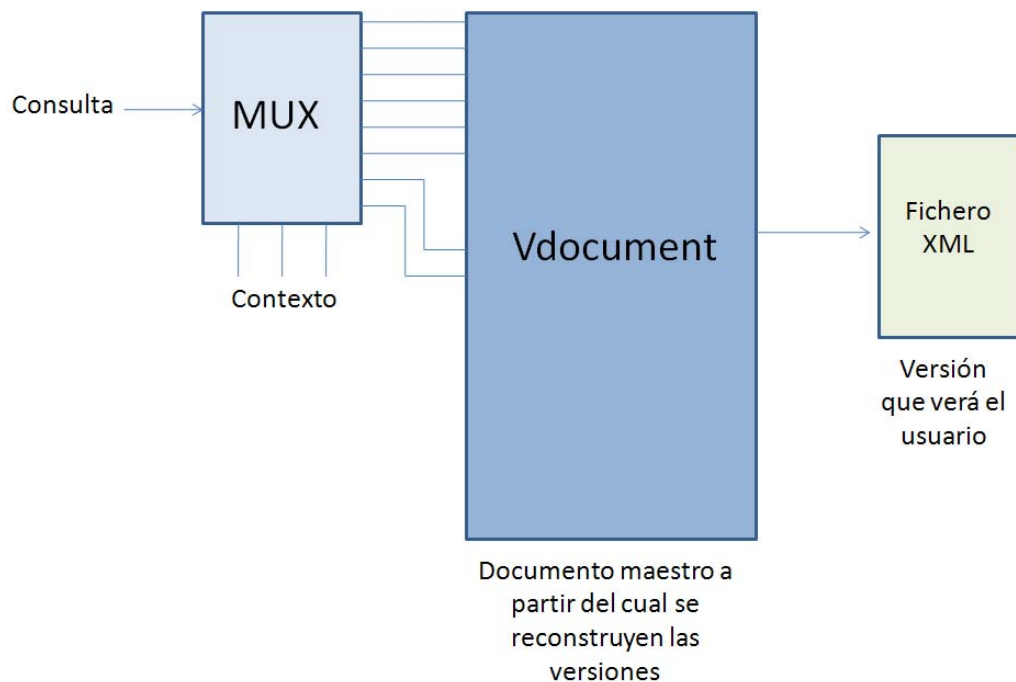


Figura 2. Esquema de conciliación contexto-versionado

Aunque este esquema de conciliación que proponemos es más propio de una arquitectura para sistemas digitales, creemos que es muy intuitiva y que refleja exactamente lo que el sistema (software, por supuesto) debería hacer. La originalidad de nuestro esquema es que rompe con el planteamiento clásico que pretendía la reconstrucción de versiones desde el interior, nosotros proponemos romper con esa línea, y

desarrollar soluciones de esta naturaleza, mucho más modulares y con una implementación más sencilla lo que redundará en ventajas muy positivas para algunas de las fases del ciclo de vida. Además, presenta una ventaja adicional: la reversibilidad, es decir, dada una versión de un documento es posible saber el contexto con el que se encuentra asociado. Este esquema, por tanto, puede seguirse de derecha a izquierda y viceversa.

4.1. Implementación de MUX

Una vez que hemos lanzado nuestra propuesta, vamos a ver cómo se implementaría un multiplexor lógico de ejemplo. Supongamos que el contexto se compone de tres términos: el perfil del usuario invocante, el idioma en que realiza la consulta y la actividad que desarrolla.

Aquí mostramos todos los valores que serían suficientes para un funcionamiento correcto del sistema:

1. Perfil: {Niño, adulto}
2. Idioma: {Castellano, Inglés}
3. Actividad: {Académica, Recreativa}

CONTEXTO			VERSIONES
Perfil	Idioma	Actividad	Obra: Fausto
Niño	Castellano	Recreativa	V1:V3
Niño	Castellano	Académica	V3
Niño	Inglés	Recreativa	V4
Niño	Inglés	Académica	V5
Adulto	Castellano	Recreativa	-
Adulto	Castellano	Académica	V3, V6
Adulto	Inglés	Recreativa	-
Adulto	Inglés	Académica	V5, V8

Tabla 1. Ejemplo de implementación de un MUX

Sea i el número de términos necesario para expresar nuestro contexto. Sea T_i cada uno de los términos que componen el contexto, y sea $T.l$ el número de valores que puede tomar cada término.

Entonces un multiplexor debe incluir:

$$\text{Número de tuplas} = \prod [T_i.l] \quad [1]$$

Para nuestro caso y según [1]:

$$\text{Número de tuplas} = 2 (\text{Perfil}) \times 2 (\text{Idioma}) \times 2 (\text{Actividad}) = 8 \text{ tuplas}$$

Ahora supongamos que añadimos el perfil *estudiante de inglés* y la actividad *investigación*.

En ese caso y según [1]:

$$\text{Número de tuplas} = 3 (\text{Perfil}) \times 2 (\text{Idioma}) \times 3 (\text{Actividad}) = 18 \text{ tuplas}$$

Esta breve demostración evidencia que la generación manual de tuplas no es viable, puesto que cualquier ligero cambio en los requisitos tiene como consecuencia el crecimiento exponencial del número de tuplas a las que se le tiene que asignar una versión de un documento. Se hace necesario por tanto buscar una alternativa automática o semiautomática, donde sea el computador el encargado de implementar el MUX.

Para comenzar a buscar una solución viable parece buena idea fijarnos en el ejemplo de los Servicios Web; con los Servicios Web sucedió que su tratamiento por parte de los computadores, a pesar de que esa era su principal tarea, era prácticamente inviable, puesto que carecían de

información semántica que ayudase a los ordenadores a discernir sobre su función. La solución apareció en forma de evolución: los Servicios Web Semánticos [13], donde ya si se le proporcionaba a un invocante potencial información suplementaria que ayudara a conocer algo más sobre el Servicio.

Siguiendo esta táctica, podemos pensar que, si dotamos a Versionstamp de unos mecanismos para ofrecer semántica, quizás dispongamos de más pistas que ayuden a un futuro proceso software a implementar el multiplexor.

Hemos tomado en consideración dos formas:

1. La primera de ellas sería mediante la adición de semántica explícita mediante una ontología, que nos serviría para describir semánticamente las distintas versiones y las relaciones que existen entre ellas. Además, y puesto que una ontología por documento maestro parece una idea disparatada, dicha ontología debería ser capaz de describir una colección completa de documentos, con las relaciones existentes entre ellos.
2. La segunda idea sería ampliando el esquema de Versionstamp para que los elementos incluyeran un atributo que pudiera contener información semántica complementaria acerca de dicho elemento. En este caso quedaría por determinar la naturaleza de esa "información complementaria"

A partir de ahora, sería razonable determinar mecanismos que permitieran actualizar de manera automática la información semántica una vez que se produjeran cambios.

5. Resultados

A diferencia de soluciones anteriores, que proponían construir el documento adecuado para un lector dado desde dentro, en base a metamarcas; nosotros proponemos realizar la labor desde fuera, con un multiplexor lógico que asocia a cada contexto individual uno o varias versiones de un documento. Consiguiendo ventajas relativas a una mayor posibilidad de expresar las características propias del que accede a la información.

Hemos considerado la posibilidad de que sea un usuario humano, el encargado de establecer las relaciones entre el contexto y el versionado. No obstante, una pequeña prueba nos ha bastado para descubrir que esta idea es totalmente inviable, puesto que pequeños cambios en los valores que puede tomar el contexto suponen grandes cambios con respecto al número asociaciones a administrar. Por tanto, hemos concluido que es necesaria la adición de semántica. Con respecto a este tema, hemos propuesto dos alternativas: la adición de semántica explícita, mediante ontologías, que permitiría describir las relaciones entre las distintas versiones e incluso relaciones entre documentos. Y por otra parte la adición de semántica implícita en forma de atributos en el documento maestro de Versionstamp.

6. Conclusiones y trabajo futuro

En este trabajo se ha presentado una aproximación a la conciliación entre el contexto y el versionado para conseguir sistemas de información adaptables al usuario.

Como uno de los aspectos más importantes de nuestro estudio cabe destacar la importancia que tiene la semántica en el diseño de sistemas adaptables. La semántica es importante no ya desde el punto de vista de la funcionalidad que ofrecemos a los usuarios, sino desde el punto de vista de hacer de esa adaptación una tarea automática, es decir, que pueda llevarse a cabo mediante computadores.

Todos los intentos que siguen esta dirección, es decir, la de sistemas que reconocen las peculiaridades de los individuos se encuadran dentro de lo que se conoce como sistemas de información pragmáticos y que tiene en la Web Pragmática [12] uno de sus puntas de iceberg.

Como trabajo futuro proponemos experimentar con Wordnet algunos pequeños ejemplos de marcado semántico automático de las distintas versiones que componen un documento maestro y que ayuden a implantar un multiplexor lógico por parte de un proceso software.

Referencias

- [1] Luis Jesús Arévalo Rosado, Antonio Polo Márquez and Juan M. Fernández González, Representing versions in XML documents using versionstamp, ECDM 2006. LNCS no 4231.
- [2] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In Proceedings of the 18th International Conference on Data Engineering. 2002.
- [3] Manolis Gergatsoulis and Yannis Stavrakas. Representing changes in XML documents using dimensions. In XSYM 2003.
- [4] F. Wang and C. Zaniolo. XBiT: An XML-based Bitemporal Data Model", in ER 2004.
- [5] TEI. Múltiple Hierarchies. <http://www.tei-c.org/P4X/NH.html>. 2002.
- [6] SGML. <http://www.w3.org/MarkUp/SGML/>
- [7] Multiple Markups in XML documents. Luis Arévalo Rosado, Antonio Polo, Miryam Salas, Juan C. Manzano. ICWE 2003 : International Conference on Web Engineering. LNCS vol. 2722.
- [8] TexMECS. An experimental markup meta-language for complex documents. Claus Huitfeldt, Sperberg-McQueen, M. C. 2001
- [9] GODDAG: A data structure for overlapping hierarchies. Claus Huitfeldt, Sperberg-McQueen, M. C.
- [10] Martínez-Gil, J., Polo-Marquez, A., Arevalo-Rosado, L.J. (2006) Servicios Web basados en el contexto. IV Taller de sistemas hipermedia colaborativos y adaptativos, 32–36, JISBD06, Sitges (Spain)
- [11] Morse, D.R., Armstrong, S., Dey, A.K. (2000) The what, who, where, when, why and how of context-awareness. CHI '00 extended abstracts on Human factors in computing systems, 371–371
- [12] Schoop, M. and de Moor, A. and Dietz, J. (2006) The Pragmatic Web: a manifesto. Commun. ACM, 49(5), 75–76
- [13] Grupo Servicios Web Semánticos del W3C. <http://www.w3.org/2002/ws/swsig/>