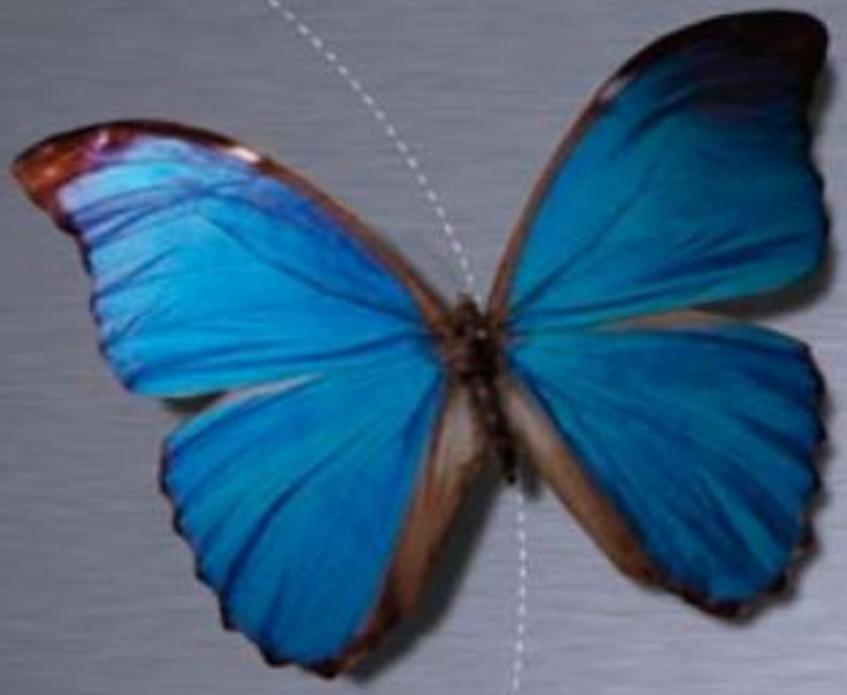
DOCUMENTACIÓN ELECTRÓNICA e INTEROPERABILIDAD de la INFORMACIÓN



Editores

Sergio F. Ochoa
Cecilia Bastarrica
Claudio Gutiérrez

DOCUMENTACIÓN ELECTRÓNICA e INTEROPERABILIDAD de la INFORMACIÓN

Editores

Sergio F. Ochoa
Cecilia Bastarrica
Claudio Gutiérrez

Departamento de Ciencias de la Computación, FCFM, Universidad de Chile
Av. Blanco Encalada 2120, 3er Piso, Santiago Chile.
www.dcc.uchile.cl.

PREFACIO

El libro está destinado a aquellos profesionales que desarrollen sistemas de información y de gestión documental para instituciones del Gobierno de Chile, tanto en forma interna como en forma externa a estas organizaciones, se verán particularmente beneficiados con el programa. Esto, es debido a que el Decreto Supremo Nº 81, con fecha 3 de Junio del 2004, establece al lenguaje XML como norma de representación de información en documentos electrónicos para el Gobierno de Chile. El mismo Decreto establece que "*esta norma se aplica a los documentos electrónicos que se generen, intercambien, transporten o almacenen en o entre los diferentes organismos de la administración del Estado y en las relaciones de éstos con los particulares, cuando éstas tengan lugar utilizando técnicas y medios electrónicos"*. Aunque su entrada en vigencia es gradual, las organizaciones gubernamentales deberán ir evolucionando hasta adherir completamente a esta norma. Esto significa que una gran cantidad de sistemas de información y gestión documental deberán ser adaptados o reconstruidos por profesionales que cuenten con conocimientos técnicos sobre modernas tecnologías de procesamiento documental. Vale aclarar que todas aquellas organizaciones privadas que intercambian documentación digital con Instituciones del Gobierno, también se verán involucradas de una u otra manera con esta norma. Se espera que este libro ayude al lector a:

- Impulsar y administrar proyectos de introducción y renovación de sistemas de información y gestión documental, que adhieran a lo que establece el Decreto Supremo Nº 81.

- Planificar y gestionar los cambios tecnológicos requeridos por cada organización, para adherir a la norma de documentación electrónica.

- Evaluar alternativas tecnológicas para soportar documentación electrónica e interoperabilidad de la información.

ACERCA DE LOS AUTORES

Renzo Angles, es doctor en Ciencias de la Computación de la Universidad de Chile. Obtuvo el grado de Ingeniero de Sistemas, en la Universidad Católica de Santa María, Perú. Sus áreas de interés son: bases de datos, web semántica, ingeniería de software y documentación electrónica. Actualmente se desempeña como profesor asistente en la Universidad de Talca.

María Cecilia Bastarrica, realizó sus estudios de ingeniería en informática en la Universidad Católica del Uruguay. Obtuvo un magister en ciencias de la ingeniería en la Pontificia Universidad Católica de Chile, y un PhD en Computer Science and Engineering en la University of Connecticut. Sus áreas de interés son: ingeniería de software, arquitecturas de software, y desarrollo de líneas de productos de software. Actualmente se desempeña como profesora asistente del Departamento de Ciencias de la Computación de la Universidad de Chile, donde además es directora académica del postítulo de ingeniería y calidad de software.

Alex Bórquez Grimaldi, obtuvo su título de Ingeniero Civil en Computación en la Universidad de Chile. Sus actividades profesionales las ha desempeñado en diversas áreas como por ejemplo documentación electrónica y sistemas de información financieros. Sus áreas de interés son las metodologías ágiles de desarrollo de software y los estándares XML. Actualmente Alex Bórquez Grimaldi es consultor SAP y realiza actividades de docencia para diversas instituciones educacionales.

Cristian Bravo Lillo, es Ingeniero Civil en Computación de la Universidad de Chile, y ha trabajado para el Gobierno de Chile en temas relacionados con firma y documento electrónico. Fue el principal gestor del proyecto de generación del Administrador de Esquemas, un repositorio gubernamental de esquemas XML; y ha colaborado activamente en la generación de las normas que rigen el intercambio de documentos digitales en la Administración Pública.

Philippe Camacho, Philippe Camacho, obtuvo un Master en Criptología y Seguridad Informática en la Universidad de Bordeaux I, Francia. Sus áreas de interés son: criptología, sistemas distribuidos y tecnologías abiertas. Actualmente Philippe Camacho es alumno del Doctorado en Ciencias de la Computación de la Universidad de Chile.

Rodrigo Frez P. es Ingeniero Civil en Computación de la Universidad de Chile. Sus áreas de interés son tecnologías de servicios web, la web semántica y tecnologías de voz sobre IP. Actualmente es gerente de operaciones y sistemas de METS S.A. y es candidato a magister en ciencias de la computación de la universidad de Chile.

Cristán Fuenzalida M., es ingeniero Civil en Computación de la Universidad de Chile. Desarrolló su memoria de Ingeniero sobre "Servicios Web para el Gobierno electrónico en Chile". Sus áreas de interés son Servicios Web, Gobierno electrónico, tecnologías XML, metadatos y Web Semántica. Desde el año 2004 se desempeña como Ingeniero de Desarrollo en Newtenberg Publicaciones Digitales.

Claudio Gutiérrez, realizó su Licenciatura en Matemáticas en la Universidad de Chile, y posteriormente un Magíster en Lógica Matemática en la Pontificia Universidad Católica de

Chile. Obtuvo su Ph.D. en Computer Science en la Wesleyan University, USA. Actualmente Claudio Gutiérrez es profesor asociado en el Departamento de Ciencias de la Computación de la Universidad de Chile Sus áreas de interés son los fundamentos de la Computación; en particular: lógica aplicada a la computación, bases de datos y aspectos semánticos de la Web.

Andrés Neyem, obtuvo su grado de Doctor en Ciencias de la Computación en la Universidad de Chile. Obtuvo su grado de Licenciado en Sistemas de Información en la Universidad Nacional de San Juan (UNSJ), Argentina. Sus áreas de interés son: Ingeniería de Software, Computación Orientada a Servicios, Computación Móvil y Sistemas Colaborativos. Actualmente se desempeña como profesor asistente en el Departamento de Ciencia de la Computación de la Pontificia Universidad Católica de Chile.

Sergio F. Ochoa, obtuvo su grado de Doctor en Ciencias de la Ingeniería, mención Computación, en la Pontificia Universidad Católica de Chile; también obtuvo el grado de Ingeniero de Sistemas de la Universidad del Centro de la Provincia de Buenos Aires (UNICEN), Argentina. Sus áreas de interés son: ingeniería de software, sistemas colaborativos y educación apoyada por tecnología. Actualmente Sergio F. Ochoa es profesor asistente en el Departamento de Ciencias de la Computación de la Universidad de Chile, y asesor tecnológico para organizaciones públicas y privadas. Además es miembro del Joint Sttering Committee de LACCIR y representante chileno ante CLEI.

Daniel Perovich, es Ingeniero en Computación de la Universidad de la República, Uruguay. Obtuvo su grado de magister en computación en dicha institución, y fue reconocido con el primer premio de tesis de maestría otorgado por UNESCO. Sus áreas de interés son arquitectura de software, lenguajes e ingeniería dirigida por modelos. Actualmente es candidato a doctor en ciencias mención computación de la Universidad de Chile.

José Miguel Selman Grez, es Ingeniero Civil en Computación de la Universidad de Chile. Sus áreas de interés son la arquitectura e integración de sistemas empresariales, interoperabilidad de la información, aplicaciones móviles y el correcto apoyo de la tecnología a la generación de nuevos negocios. Actualmente se desempeña como Arquitecto Tecnológico Senior en la empresa BEE Concretorías y Sistemas, donde construye y mantiene, desde el 2002, software para importantes empresas del ámbito financiero tanto en Chile como en el extranjero.

Andrés Vignaga es Ingeniero en Computación y Magíster en Informática de la Universidad de la República, Uruguay. Sus áreas de interés son la ingeniería dirigida por modelos, los métodos formales, y la arquitectura de software. Actualmente es candidato a doctor en ciencias, mención computación, de la Universidad de Chile, y realiza docencia para el Programa de Educación Continua del Departamento de Ciencias de la Computación de la Universidad de Chile.

Agustín Villena M., es Magister en Ciencias, mención Computación de la Universidad de Chile. Sus áreas de interés son las metodologías ágiles de ingeniería de software, la informática educativa y la gestión documental con énfasis en la interoperabilidad, de procesos de negocio y firma electrónica. Actualmente es el Gerente de Consultoría SAP de la empresa NOVIS y desde el 2002 profesor de jornada parcial del Departamento de Ciencias de la Computación de la Universidad de Chile.

AGRADECIMIENTOS

Los editores desean agradecer al Departamento de Ciencias de la Computación (DCC) de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile, por el apoyo recibido durante la elaboración del libro. También quienen agradecer el invaluable aporte de la Sra. Carla Sambrizzi, por el trabajo de revisión de este documento.

INDICE

CAPITULO I: INTRODUCCIÓN A LA GESTION DOCUMENTAL EN CHILE

*Claudio Gutiérrez+, Cristian Fuenzalida+, Cristian Bravo**

+ Departamento de Ciencias de la Computación, Universidad de Chile.
{cgutierr, cfuenzal}@dcc.uchile.cl

* Carnegie Mellon University, USA.
cbravo@cmu.edu

1.1 Introducción

La gestión documental, es un problema recurrente en los procesos de administración tanto pública como privada. Las tendencias actuales nos llevan a desarrollar esta gestión a través de medios digitales[1], es decir, usando las tecnologías de información. En este contexto, aparecen lenguajes de representación de datos y frameworks para apoyar la especificación y administración de la información digital, tales como XML, Arquitecturas Orientadas a Servicios y Servicios Web.

Para abordar la tarea de analizar, diseñar e implementar una solución de gestión documental, se requiere primero comprender qué conceptos están involucrados. Entre los más importantes están: documento electrónico, flujos documentales, descripciones formales de las interacciones entre diversos agentes y su contexto, definiciones de arquitectura, identificación y resolución de los temas de seguridad asociados, y entrega de servicios[2]. Junto a cada uno de estos conceptos, es necesario conocer las tecnologías existentes que permiten llevar a cabo las soluciones propuestas. Los capítulos de este libro corresponden al desarrollo de estos temas.

En este capítulo se presentan los conceptos básicos sobre gestión documental electrónica, los cuales corresponden a las necesidades de información y servicios que tienen los diversos agentes (usuarios), tanto en el Gobierno de Chile como en la industria y en la ciudadanía. Se presenta a continuación el caso del registro de un recién nacido en el Servicio del Registro Civil e Identificación, donde existen diversos requerimientos y problemas asociados con la gestión documental. Luego se presentan las necesidades que existen y cómo ellas evolucionan, qué elementos intervienen, qué decisiones de diseño deben tomarse, cuál es el marco legal vigente en Chile y qué implicancias tiene. Finalmente presentamos los desafíos presentes y futuros a abordar, y una introducción a las tecnologías existentes que permiten dar solución al problema.

1 A lo largo de este texto, hablaremos indistintamente de “electrónico” o “digital”. El segundo es usado ampliamente en el ambiente informático, pero la legislación chilena da preferencia al primero como un concepto genérico que abarca variadas formas, entre ellas, la digital.

2 Cuando hablamos de un “servicio” (con minúscula), nos referimos a una prestación entregada por un ente determinado; en cambio, cuando hablamos de un “Servicio” (con mayúscula), estamos hablando de un organismo de gobierno.

1.1.1 Necesidad de Entregar Servicios e Información Eficientemente

Todas las organizaciones tienen memoria. Éstas representan sistemas complejos, donde su complejidad inherente es usualmente mayor a la suma de sus partes. De alguna manera, podemos pensar en las organizaciones como personas: nacen, aprenden a comunicarse y a interactuar con el resto de las "personas", y (con un poco de suerte) crecen. Independientemente de su orientación o no al lucro, a medida que crecen deben adaptarse al medio y aprender a optimizar sus procesos internos, o perecerán frente a la vorágine de cambios en los que están insertas.

Tanto las personas como las organizaciones aprenden a comunicarse con sus pares a lo largo de su desarrollo. La comunicación se puede modelar basada en algunos elementos fundamentales: los "*hablantes*" o "*comunicantes*" (donde se es alternadamente emisor o receptor de un mensaje), el *mensaje* (lo que se está comunicando, donde está implícita una *lengua* o *idioma*) y el *medio* (a través del cual fluye el mensaje). Entre dos personas, el medio por el cual pueden comunicarse puede ser el aire (en una conversación presencial), una línea telefónica, Internet, etc.

En el contexto de las organizaciones, los mensajes usualmente adquieren la forma de "*documentos"*, y el medio tiene relación con las múltiples maneras posibles de transportar un "*documento"*: por "mano", por correo normal, y más recientemente por fax, email y otros medios digitales. Y son estos últimos los que han producido, en los últimos 20 años, una verdadera revolución en la comunicación entre organizaciones.

En todas las organizaciones, tanto privadas como públicas, existe, por tanto, el problema de gestionar la información que comunican. En ambos casos se requiere manejar documentos y gestionar su almacenamiento, recuperación, comunicación y transformación efectiva y eficientemente. En la comunicación con fines comerciales entre empresas, éstas requieren intercambiar mensajes, interrelacionarse, comunicar los datos estructuradamente usando una "lengua" común, negociar condiciones de colaboración, buscar proveedores de servicios en registros, y definir e implementar sus procesos de negocio. En el caso de las empresas, todo esto ocurre en un marco de necesaria cooperación y acuerdo entre empresas complementarias, pero por otro lado, implica una fuerte competencia entre quienes desean entregar una misma información o servicios [ebxm05].

El fenómeno anterior se da de manera similar en los gobiernos; sin embargo, en este contexto el problema es notablemente más complejo. Si lo consideramos de manera global, un gobierno es normalmente la organización más grande y compleja de un país, porque está formada por un conjunto muy fragmentado de instituciones bastante heterogéneas entre sí, que fueron surgiendo a medida que fueron siendo necesarias, durante un período usualmente largo, y que no necesariamente fueron "pensadas" de manera integrada desde un comienzo. Desde el punto de vista del mercado, un gobierno no tiene "competencia" alguna, y funciona como un monopolio en la entrega de ciertos servicios y productos considerados como "de valor". Por los motivos anteriores, en la prestación de estos servicios y en la producción de estos productos, existen problemas de ineficiencia, duplicidad de funciones, ambiguedad en la asignación de responsabilidades, etc. Adicionalmente, a nivel de país, siempre existen múltiples marcos normativos que restringen de manera compleja el accionar de estas entidades

que forman el gobierno: la constitución, leyes y decretos, reglamentos, etc.

Dado todo lo anterior, existe un sinnúmero de oportunidades de mejora en la gestión de la información comunicada, tanto como en la comunicación misma. A pesar de las diferencias existentes entre el Gobierno y el mundo privado, en general es posible aplicar similares principios de solución en lo referente a las tareas que el Gobierno cumple para con sus "clientes", los ciudadanos, como la entrega de servicios relacionados con las necesidades básicas del ser humano, salud, trabajo, educación, vivienda, protección social, etc.

Podríamos decir que, *para los ciudadanos,* la principal cara del Gobierno son los servicios que éste entrega. Tal como veremos a continuación, estos servicios no siempre son eficientes, e incluyen pasos intermedios donde es posible que la información sea modificada (intencional o casualmente); adicionalmente, no es posible contar con información de control acerca del proceso completo, ni de su estado de completitud.

Estos servicios involucran muchos de los desafíos presentes en la comunicación entre empresas, por lo que su desarrollo e implantación resulta ser un proceso altamente complejo. Dado el deber del Gobierno de realizar su quehacer efectiva y eficientemente, se hace necesario optimizar sus procesos de comunicación. En este documento estudiamos el rol que juegan las TICs en este proyecto.

1.1.2 Ejemplo Real: la Inscripción del Nacimiento de un Hijo

La inscripción de un recién nacido en el Servicio de Registro Civil e Identificación de Chile (en adelante SRCeI), consiste en la unificación de lo que son dos servicios separados: uno resulta en la llamada "partida de nacimiento" (entregada en el Hospital) y otro en el "certificado de nacimiento" (entregado en el SRCeI).

Cuando la futura madre llega al Hospital, su información es ingresada en el sistema de admisión de pacientes, que forma parte del sistema computacional de la Red Informática del Hospital (en adelante RIH). Luego del nacimiento, la madre recibe de parte del Hospital un documento de papel con una "partida de nacimiento" (en adelante PN), la cual es firmada por un médico que certifica que cada recién nacido, es hijo de la madre inscrita en el sistema de admisión (ver Figura 1).

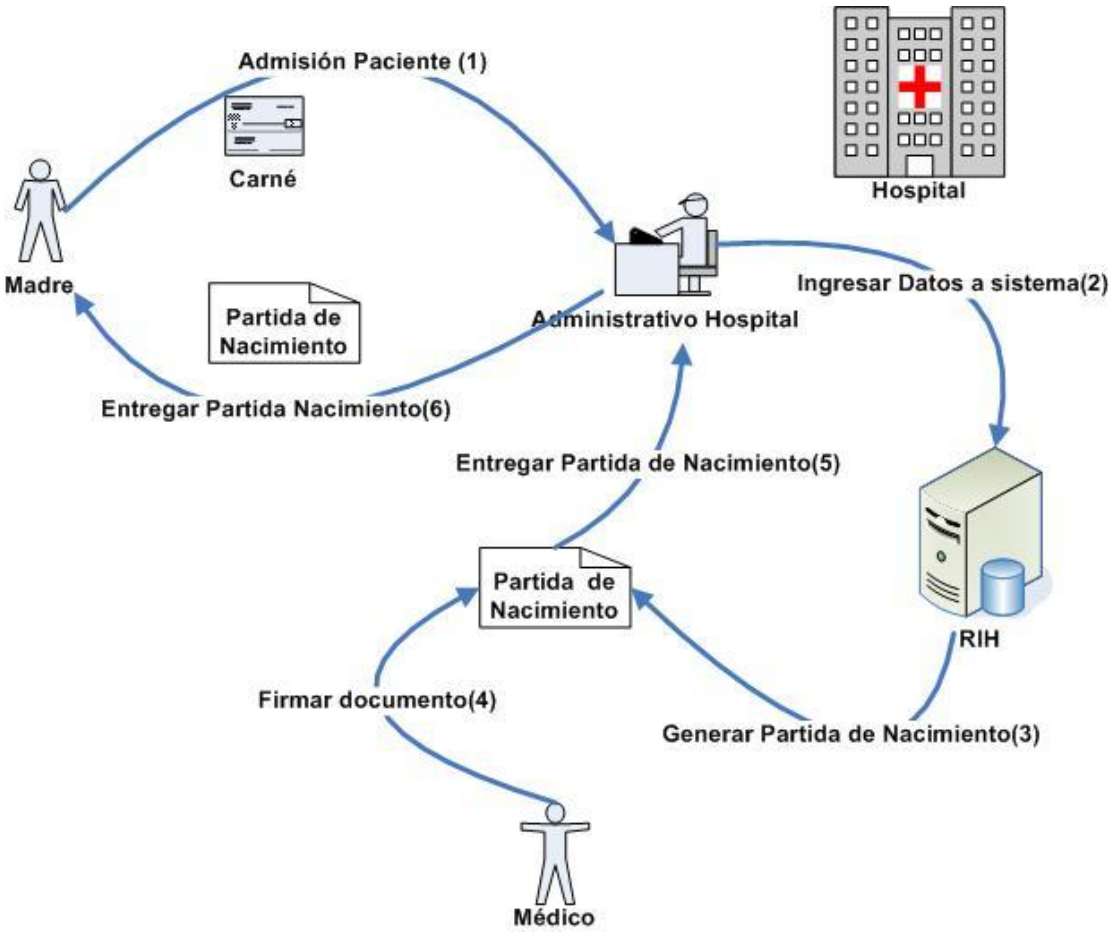


Figura 1: Trámites realizados en el Hospital

Este documento es llevado por los padres al SRCeI, donde ambos presentan sus respectivas cédulas de identidad y la PN entregada por el Hospital (ver Figura 2). Un funcionario del SRCeI certifica que las identidades de los padres y la partida de nacimiento sean válidas, y de ser así, se les entrega el certificado de nacimiento del recién nacido, y su número de cédula de identidad (Rol Único Nacional, o RUN).

Finalmente, como medida para corroborar la información de los nacimientos y para tener información estadística, el Ministerio de Salud debe enviar una vez al año un consolidado al SRCeI con todos los nacimientos de los que se tenga conocimiento en los Hospitales (ver figura 3). Es por esto que se debe incorporar canales de comunicación entre los Hospitales y el Ministerio de Salud, y luego entre el Ministerio de Salud y el SRCeI.

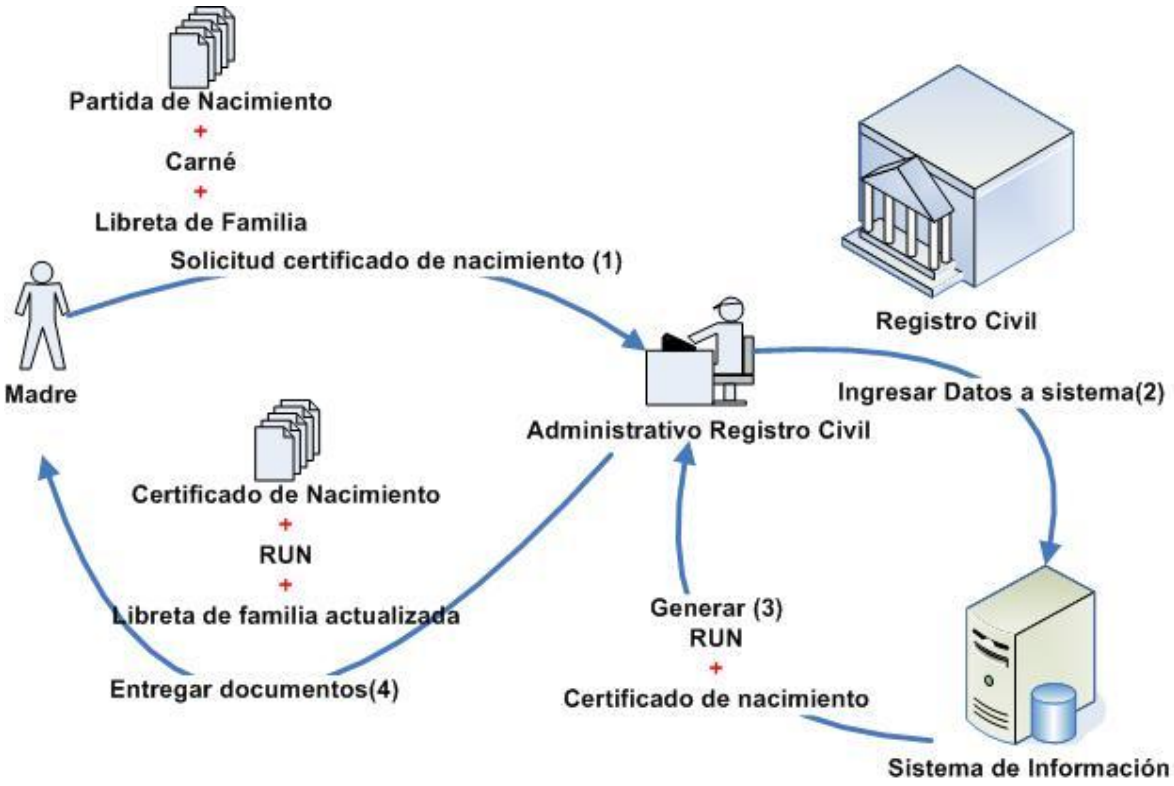


Figura 2: Trámites realizados en el Servicio de Registro Civil e Identificación

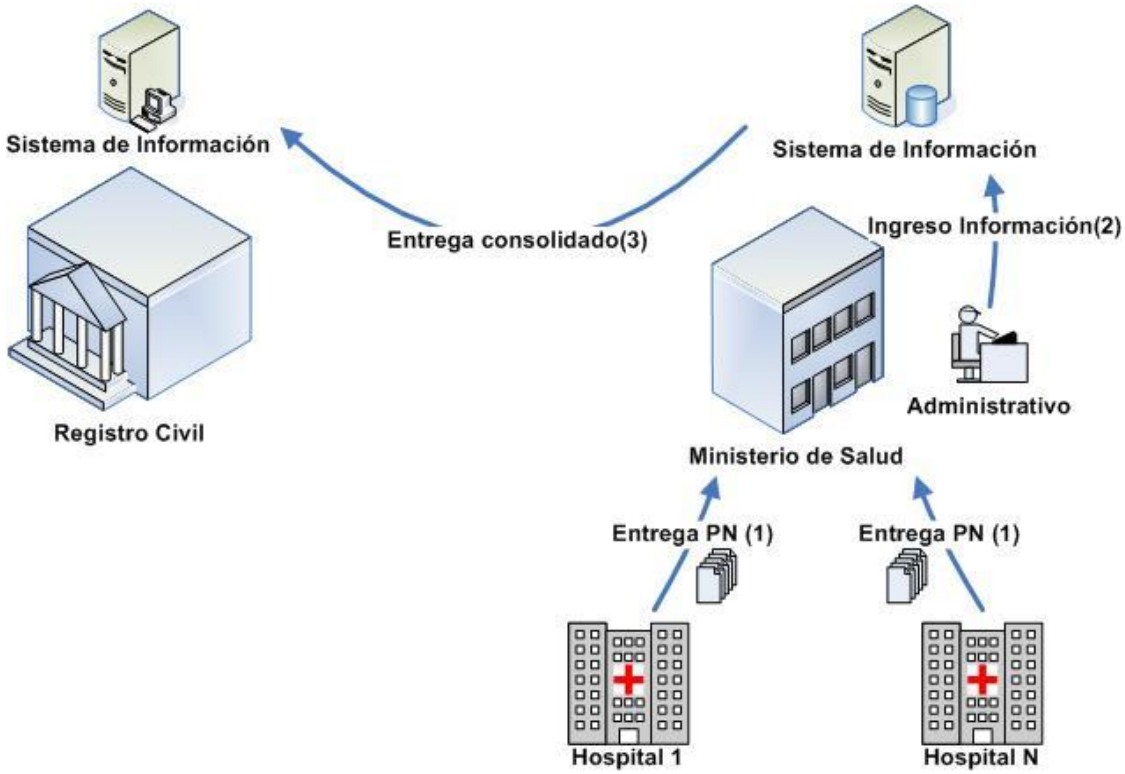


Figura 3: Trámites administrativos internos entre el SRCeI y el Ministerio de Salud

1.1.3 ¿Qué Problemas Hay en este Esquema?

El proceso relacionado con la consecución del objetivo anterior (inscribir un hijo en el SRCeI) posee varios problemas desde el punto de vista del servicio:

1. *El ciudadano ve muchas "islas":* Debido a que el proceso implica interactuar con diversas entidades, el ciudadano debe acudir a más de una para llevar a cabo el trámite completo. Esto es, el ciudadano participa de muchas interacciones, pero no de un servicio[3] .

2. *Documentos innecesarios para ciertos agentes:* Existen documentos intermedios que deberían ser transparentes para ciertos agentes. Un ejemplo de ello es la partida de nacimiento, que sólo tiene sentido para hacer una verificación entre el Hospital y el Registro Civil, no teniendo ninguna utilidad real para el ciudadano.

3. *Interacciones innecesarias entre ciertos agentes*: Existen interacciones que podrían evitarse. Por ejemplo, un ciudadano no debería interactuar con el Servicio del Registro Civil para entregarle la partida de nacimiento, sino que debería ser el Hospital quien lo hace.

4. *Interoperabilidad "a través del ciudadano":* Siendo el Gobierno quien entrega los servicios, es en realidad el ciudadano quien invierte tiempo y dinero en hacer posible el proceso.

5. *Es difícil conocer el estado del trámite:* Como el servicio no depende de una sola entidad , para conocer el estado de un trámite el ciudadano requiere conocer la lógica del servicio (donde "continuar la tramitación"), y por tanto debe interactuar con una o más entidades.

6. *Posibilidad de tener trámites incompletos o inconsistentes:* Nada asegura que una madre inscribirá a su hijo en el SRCeI en un plazo determinado. Es posible entonces que niños nacidos y registrados en el Hospital y también en el Ministerio de Salud no existan para el SRCeI.

1.1.4 ¿En qué Fijarse para Corregir estos Problemas?

Frente a los problemas anteriores, es posible sugerir algunos lineamientos que nos servirán posteriormente para diseñar soluciones satisfactorias. Estos lineamientos son:

1. *Ofrecer un servicio que sea un todo:* Un servicio "*es un recurso que entrega una funcionalidad autocontenida e independiente del contexto*". Esto implica que la solución a un problema debería proveer al ciudadano de una interfaz que "esconda" el proceso interno, y que le haga interactuar con el mínimo posible de interlocutores.

3 Veremos más adelante que un servicio, por definición, debe ser autocontenido.

Claramente la situación actual no considera ésto, lo que provoca algunos de los problemas mencionados en la sección anterior.

2. *Definir con claridad objetivo y alcance del servicio:* El resultado de la entrega de un servicio debe definir claramente qué se busca cumplir con éste, y cuáles son los límites donde el trámite se realiza.

3. *Identificar los agentes participantes y sus funciones:* En ciertas ocasiones, no resulta claro si una repartición o persona forma parte de un servicio. Por ejemplo, no es evidente en principio, que en el proceso de registrar un recién nacido intervengan agentes que no generan información referente al trámite, tales como el Ministerio de Salud. El definir los actores que intervienen en un servicio y sus roles permite luego definir las relaciones e interacciones que hay entre ellos.

4. *Determinar la interacción entre agentes:* Los agentes que participan de un servicio, interactúan siguiendo ciertas reglas. Tanto la lógica de estas interacciones como los flujos deben ser considerados al momento de definir el servicio.

5. *Determinar los factores que influyen sobre la interacción entre los agentes:* Para que los agentes se puedan comunicar, deben existir acuerdos referentes a la estructura y significado de los datos intercambiados, marco legal, políticas de privacidad, etc. Cabe hacer notar que en el ejemplo planteado, aunque existe "interoperabilidad" entre todos los agentes, ella está basada en la intervención humana, lo que explica el por qué son tan costosas algunas interacciones.

6. *Esconder las interacciones que no sean de interés para algunos entes:* Existen diversos tipos de información provistos por los agentes. En el ejemplo, algunos de ellos son: la partida de nacimiento, el certificado del Registro Civil y los reportes que van desde el Hospital al Ministerio de Salud. Una solución debe identificar qué información debe ser provista y recibida por cada agente, cuidando de abstraer a los participantes de aquella información que no deban o necesiten manejar.

En lo que resta del capítulo, intentaremos explicar formalmente lo que implica el proceso de gestión documental electrónica, el contexto legal y normativo en Chile, y los desafíos que deberemos enfrentar a futuro.

1.2 Evolución de las Necesidades en la Administración de la Información

Las necesidades de información y servicios se han hecho más complejas con el correr del tiempo y con la evolución de las TICs. Una aproximación a la evolución de estas necesidades en las organizaciones de gobierno, consta de las siguientes seis etapas (ver por ej. [Vara03]), donde –por la época del reporte--hay un énfasis natural en sitios Web:

1. *Inicial:* Representa el estado en que hay sistemas informáticos en las reparticiones, pero sólo se usan internamente. Asimismo, la comunicación electrónica solamente existe al interior de la repartición.

2. *Información ("Presencia"):* En esta fase los servicios tienen la capacidad de proveer información sobre su acción al ciudadano, es decir, existe información en línea. Sin embargo, el contacto se realiza por teléfono o correo (no e-mail); el sitio Web es muy elemental.

3. *Interacción ("Interacción"):* Considera comunicaciones simples entre el servicio y el ciudadano, y la incorporación de esquemas de búsqueda básica; permite, por ejemplo, la descarga de archivos y formularios, y el uso de correo electrónico para comunicarse con la repartición que ofrece el servicio.

4. *Interacción en dos vías ("Transacción"):* Incluye una interacción electrónica bidireccional entre el ciudadano y el Servicio, en forma alternativa a la atención presencial en las dependencias del organismo. Incluye autentificación, procesamiento de formularios, etc.

5. *Integración Vertical Interna ("Transformación"):* En esta fase el servicio es transaccional, la interacción es personalizada (por ej. decisión, entrega y eventualmente pago).

6. *Gobierno Electrónico Unificado:* Supone el que las redes y/o prestaciones de servicios a disposición en las instituciones públicas y de los ciudadanos, están interconectadas. En particular supone la existencia de una *Ventanilla Única* para la realización de cualquier trámite por parte de los ciudadanos.

1.3 Gestión Documental Electrónica

La gestión documental implica, el manejo y gestión de la documentación relativa a algún proceso (lógica de negocios) para apoyar sus quehaceres. Ello incluye la capacidad de realizar distintas acciones sobre los documentos, tales como: creación, almacenaje, edición, clasificación, recuperación, mantenimiento, soporte, transformación, visualización, archivado (documentos antiguos), foliado y firmado. Junto con ello, una apropiada gestión documental debe proveer mecanismos para coordinar y dar sentido a las relaciones y restricciones que hay sobre los documentos.

Cabe aclarar que el término "*documento*", no se refiere necesariamente a ninguna unidad de información particular. Un documento puede ser un formulario con los datos de un ciudadano, un registro de una base de datos, un expediente u otros. La definición de cuál es la unidad de información apropiada para cada organización y servicio, depende de cada problema en particular, y no debe ser "mapeada" necesariamente a los documentos físicos existentes hoy en día [Terp03].

1.3.1 Gestión documental clásica vs. gestión documental electrónica

La gestión documental digital se preocupa conceptualmente de los mismos problemas que la gestión documental "clásica", pero utilizando documentos- electrónicos en vez de

documentos-papel. La incorporación de TICs permite administrar de manera más efectiva y eficiente la información al interior de las organizaciones. Sin embargo, la incorporación de TICs para apoyar la gestión de la información es un problema mucho más complejo que simplemente encontrar e implementar reemplazos digitales para los documentos-papel existentes. También ha ocurrido que las mismas TICs han creado parte de estos problemas:
"Los enormes monstruos indomables, los mainframes, no sólo cimentaron obsoletas formas de trabajar, sino que agregaron otros niveles de burocracia para su planeación, implementación, operación y control. Durante la década de 1970, esta práctica se difundió a los microcomputadores, y en la década siguiente, a los PC y las LAN. (...) ¿Cuál fue el resultado? Las organizaciones gubernamentales de hoy se encuentran enclaustradas en estructuras y formas de trabajar obsoletas, cada una con sus islas correspondientes de tecnología."[4]

En general, lo anterior se ha producido por una adopción más bien mecánica de las TICs, con falta de perspectiva y de análisis de procesos. Esto puede reflejarse de manera breve en los siguientes "mitos" bastante difundidos:

1. "*La introducción de tecnología resolverá los problemas de gestión*": La tecnología no es una forma de resolver problemas; sólo es una forma de acometerlos más rápido. La tecnología es una herramienta, y no una "bala de plata".

2. "*Mientras más y mejor tecnología, más capacidad de trabajo*": Hay órdenes de magnitud entre el progreso del hardware, del software y de nuestra comprensión de éste y del mundo. En la digitalización de procesos siempre es necesario introducir herramientas de software; pero si las personas no están preparadas o no son capaces de hacer uso de ellas, la capacidad adicional de la tecnología es inútil y contraproducente.

3. "*Las personas asumen rápidamente nuevas herramientas*": Por las mismas razones anteriores, no somos capaces de aprender a utilizar las herramientas a la velocidad necesaria.

4. "El papel no cumplirá ningún rol en el futuro proceso de digitialización documental": Es un error creer que el papel es simplemente un portador de información, cuando en realidad posee características útiles para la organización y la diversidad de las personas:

> *"Por ejemplo, al poner el énfasis en la lógica de la información, le resultó sencillo a Business Week predecir en 1975 que la 'oficina sin papel' estaba cerca. Cinco años después un futurólogo insistía en que 'hacer copias en papel de cualquier cosa era 'primitivo'. Sin embargo, las impresoras y las copiadoras adquirieron mayor velocidad y funcionaron durante toda la década siguiente. Además, a mediados de esa década surgió el fax, que se convirtió en un aparato esencial (basado en el papel) (...) pese a todo, el fax sobrevive. Al igual que el lápiz (cuya partida definitiva fue anunciada por el New York Times en 1938 al producirse el advenimiento de máquinas de escribir más sofisticadas),*

4 [Taps00], pág. 154-155.

el fax, la fotocopiadora y los documentos en papel se rehúsan a desaparecer; la gente los encuentra útiles (...) (La razón es que)… el papel tiene maravillosas propiedades, que se encuentran más allá de la información, porque ayudan a la gente a trabajar, a comunicarse y a pensar en conjunto" [5].

1.3.2 Comparación de gestión documental electrónica y en papel

A pesar de lo anterior, la gestión documental electrónica tiene diversas ventajas sobre la gestión documental usando medios físicos como el papel. Las más importantes se representan en la tabla 1 [Aust95].

Tabla 1: Costo de desarrollo y operación de la gestión documental electrónica y en papel

Proceso	Electrónica	Papel
Recuperación y transformación de información	Moderado	Alto
Administración y gestión de la información	Medio	Alto
Preservación de documentos (archivado)	Moderado	Medio-Alto
Costo de almacenaje, en espacio y tiempo	Bajo	Alto
Manejo del contexto y relaciones en que participan documentos y procesos	Medio	Alto
Seguimiento de procesos por parte del proveedor de servicios	Medio-Alto	Alto
Seguimiento de procesos por parte del cliente	Bajo	Alto
Integración con información legada	Medio-Alto	Bajo
Privacidad y seguridad de la información	Medio	Medio-Alto
Cumplimiento de la legislación vigente	Bajo	Alto

1.4 Documento Electrónico

El bloque constitutivo esencial de la gestión documental electrónica, es el *documento electrónico*. Según la Ley 19.799 de Firma y Documento electrónico, se define documento electrónico, como *toda representación de un hecho, imagen o idea que sea creada, enviada, comunicada o recibida por medios electrónicos y almacenada de un modo idóneo para permitir su uso posterior* [Firm02].

Desde el punto de vista de flujos de información, el documento electrónico es el elemento básico que permite interoperar entre distintos sistemas. Esta interoperabilidad debe ser a nivel de formato, sintaxis, lógica de negocios y leyes, es decir, se requiere una política completa de interoperabilidad. Es importante hacer notar que para contar consta, aunque el documento electrónico es la base, éste no es suficiente. El documento electrónico es sólo el primer paso, que entrega una base mínima de interoperabilidad. Su uso fue pensado para apoyar las etapas siguientes que entregan mayor valor: entrega de servicios y gestión de trámites.

5. Ver [Brow01], pág. 15.

1.4.1 Características deseables en un documento electrónico

El proceso de diseño de un documento electrónico no es una tarea simple. A continuación se presentan las características deseables en un documento electrónico, las que deben ser tomadas en cuenta al momento de diseñarlos y desarrollarlos.

Extensibilidad: Ante el constante cambio que ocurre en procesos, organizaciones y definiciones, es clave que la definición de un documento electrónico pueda ser extensible. Se debe permitir especializar, restringir y enriquecer las definiciones existentes.

Eficiencia: En varias de las operaciones en que intervienen los documentos electrónicos se debe considerar la eficiencia del manejo de ellos. Por ejemplo: el almacenamiento, transmisión o recuperación de un documento, puede ser acelerada mediante el uso de compresión, indexación o uso de caché.

Visualización (diversas representaciones): La separación del contenido de la representación de la información, permite mayor versatilidad a los documentos. En particular, debe permitir definir distintas visualizaciones dependiendo del contexto o dispositivo en el que sea procesado o visualizado.

Procesamiento: Deben existir herramientas, métodos y servicios para el procesamiento de la documentación electrónica, de la manera más automatizada posible, para permitir la interacción entre máquinas, y entre personas y máquinas.

Envío y recepción (transporte): Se debe contar con un buen soporte para el transporte de los documentos electrónicos, e independizarse de tecnologías o protocolos específicos para el envío y recepción de los documentos.

Seguridad: Debido a las operaciones que se realizan con los documentos electrónicos, deben existir maneras de validar y asegurar la integridad, privacidad y validez de la estructura y la información contenida.

Usar tecnología estándar: Los protocolos abiertos y los estándares entregan mejor soporte, flexibilidad de cambio, y una base común para interoperar, lo que resulta clave en un documento electrónico dada su naturaleza (ver: http://books.evc-cit.info/odbook/book.html).

1.4.2 Componentes del documento electrónico

Los documentos, incluso los documentos-papel, siempre han tenido tres elementos constitutivos distintos. En el papel no son distinguibles; sin embargo, en el mundo digital es posible (y muchas veces conveniente) separar estos tres elementos. Ellos son la *estructura*, el *contenido* y la *representación* o *visualización*. Adicionalmente llamaremos *soporte* o *sustrato*[6] a la forma mediante la cual son representados los tres elementos anteriores: papel o digital.

6. *Soporte o codificación:* Son los medios o herramientas que permiten hacer "tangible" el documento. Ejemplos de ello son el papel en que se imprime el certificado de nacimiento, la codificación binaria del certificado de nacimiento digital entregado por el Registro Civil o el timbre de agua del SII en los talonarios clásicos de boletas.

Si tomamos como ejemplo el Certificado de Nacimiento que expende el Servicio de Registro Civil e Identificación, los tres elementos anteriores se ven reflejados de la siguiente forma:

Estructura: La estructura de un documento define cómo se organizan y ordenan sus componentes. La estructura de un documento puede clasificarse según si es intrínseca al documento, lógica o administrativa.

a) La estructura intrínseca del documento está representada por la organización de cada uno de los elementos que forman el contenido. Un ejemplo de ello es la división en títulos, capítulos, lista itemizada y palabras destacadas dentro de un documento.

b) La estructura lógica, se relaciona con la lógica de negocios que tiene asociada un documento. Un ejemplo de ello es el uso de elementos que hacen referencia a conceptos específicos de un servicio, tales como, el valor total como la suma de los valores anteriores.

c) La estructura administrativa, tiene que ver con la información que tiene asociada un documento que no está relacionada con la lógica del negocio, pero sí permite conocer propiedades o realizar operaciones sobre un documento. Ejemplos de ello, son la fecha de modificación e identificadores del documento.

Un componente esencial de la estructura son los *metadatos*. Los metadatos organizan y estructuran los datos del documento. Un ejemplo es el RUT: este metadato indica que el valor del dato asociado, es el Rol Único Tributario de los contribuyentes chilenos. Los metadatos cumplen el rol de unidades básicas para armar la estructura de un documento. En el ejemplo del registro del recién nacido, el documento de certificado de nacimiento está estructurado en secciones tales como: el encabezado con información del Registro Civil, los datos de la persona y una zona de validación del documento con una firma o timbre. Algunos de los metadatos esenciales corresponden a los elementos RUN, nombre y sexo.

Contenido o información: Tiene que ver con los valores de las instancias de los documentos, siguiendo la estructura definida por éste. Por ejemplo, el valor del RUT, nombres y apellidos, son parte del contenido de un certificado de nacimiento.

Representación: Es la definición de las características del soporte del documento que permitirán visualizarlo u operar sobre él. Ejemplos de ello son el tamaño de representación, los colores y el grosor de las líneas que separan cada sección, diseño del logo, etc

1.4.3 Documento Electrónico en XML

Extensible Markup Language (XML) es un formato simple de texto diseñado para la publicación a gran escala. XML describe una clase de objetos llamados *documentos XML*, y especifica en parte el comportamiento de los programas que lo procesan. Ver Capítulo sobre XML.

El uso de XML como documento electrónico no es antojadizo en absoluto, ya que XML cumple con todas las características deseables para un documento electrónico (ver sección 1.6.1 y Tabla 2), y además permite resolver la necesidad de interoperabilidad de documentación electrónica (ver: http://xml.openoffice.org/xml_advocacy.html). Algunas de las características de XML que lo hacen un excelente candidato para manejar documentación electrónica son [Norm05]:

1. Posee una gran flexibilidad para especificar formatos, definir y extender estructura.

2. Facilita la modularidad, composición y reuso.

3. Permite una gran escalabilidad al separar visualización y contenido, y al estructurar los documentos (consideraciones de Bases de Datos).

4. Es un estándar de facto y abierto.

5. Es independiente de plataformas.

6. Posee un buen soporte (apoyo industrial y comercial).

7. Su arquitectura es compatible con futuras extensiones del sistema global de información.

8. La estructura permite definir una granularidad fina para administrar documentos, ya que cada elemento en XML puede identificarse, manipularse y administrarse de manera única.

9. Los metadatos en un documento XML no son sólo para el documento completo, sino que también para unidades de información más pequeñas o grandes, con lo que cada unidad es manejable como un registro de una base de datos.

Tabla 2: XML comparado con otros formatos

Tipo de Documento	Visualización	Contenido	Metadatos
Texto plano	Pobre	Sin estructura	No tiene
DOC	Poco flexible	Mezclada con visualización	Parte de la aplicación
HTML	Flexible	Mezclada con Visualización	Pobre
PDF	No flexible	Mezclada con visualización	Pobre
XML	Flexible	Independiente de visualización y bien estructurado.	Extensible, independiente de la aplicación.

1.5 Más allá del Documento Electrónico

A lo largo de este capítulo hemos mostrado la necesidad y utilidad de la gestión documental, el imperativo de contar con un documento electrónico, y las características que éste debe tener para permitir un buen manejo de los documentos.

Junto con ello, hay que ocuparse de otros temas que aparecen cuando se quiere utilizar el documento electrónico en diversos contextos. Por ejemplo, para cumplir con los quehaceres propios de la gestión documental, tales como almacenaje, edición, recuperación, transformación, envío, validación, entre otros, se deben abordar al menos los siguientes puntos:

- *Diseño de Documentos:* Esto involucra definir: estructura, metadatos, esquemas, relaciones (subclases, reutilización, dependencia), granularidad.

- *Arquitectura:* ¿Qué es y por qué es necesario definir una arquitectura para soportar la gestión documental electrónica? (ver cap. VI).

- *Seguridad:* Validación de documentos y operaciones sobre ellos, evitar intervención de terceras partes.

- *Servicios y Servicios Web:* Transmitir documentos, uso de transacciones, semántica de los servicios, composición y orquestación de servicios para entregar prestaciones de mayor valor y fomentar la reutilización.

Todos estos temas son tratados en profundidad en los siguientes capítulos, lo que nos permitirá revisar las implicancias y sutilezas del desarrollo de soluciones de gestión documental, tanto con conceptos como con ejemplos prácticos.

1.6 Gobierno y Gestión Documental Electrónica en Chile

El *Gobierno electrónico* busca apoyar el quehacer del Gobierno mediante un eficiente y efectivo uso de las TIC. Este quehacer está caracterizado por el manejo de documentos, el procesamiento de requerimientos y la entrega de servicios e información.

1.6.1 Marco Legal

Los cuerpos legales más importantes relacionados con la implementación tecnológica al interior de la Administración Pública Chilena son los siguientes:

Ley Nº 19.799 de Documento y Firma Electrónica: Ley que define la Firma Electrónica, como un conjunto de datos digitales que identifican al firmante de un documento electrónico, y lo vinculan con su identidad. *La firma electrónica tiene el mismo valor legal que una firma en*

papel. Para la administración pública, la ley distingue dos tipos de firmas: electrónica simple y electrónica avanzada. Esta última se impone como obligatoria, sólo para aquellos documentos que revistan naturaleza de instrumento público o que se desea produzcan efectos jurídicos [Firm02].

Ley Nº 19.880 de Procedimiento Administrativo (LPA): Esta ley permite a todo ciudadano que realiza un trámite con el Estado, eximirse de presentar documentos que no correspondan al procedimiento, o que ya se encuentren en poder de la administración pública (en cualquiera de sus reparticiones). Además permite conocer el estado de un trámite en todo momento, y define los plazos máximos que puede durar un trámite [Psee04]. La LPA resulta ser una de las normativas claves, que impulsa al Gobierno electrónico en su conjunto. Por un lado, al definir un plazo máximo a los trámites que se realizan en el Estado, se avanza hacia el objetivo de una mayor eficiencia por parte de los servicios otorgados, siendo un rediseño de procesos, y la automatización de las tareas algo fundamental. Por otra parte, el eximir a ciudadanos de presentar documentos o información que ya tenga el Estado en cualquiera de sus reparticiones, obliga al Gobierno a definir canales de comunicación horizontales, para transmitir tal información si una repartición la necesita y se encuentra en otra, lo que sumado al punto del plazo máximo de los trámites, impulsa al Gobierno a tomar medidas a favor de la interoperabilidad y automatización.

Ley Nº 19.628 de Protección de datos de carácter personal: Esta ley permite identificar el propietario de los datos de un trámite, las restricciones bajo las cuales estarán disponibles los datos de los ciudadanos y la competencia legal de las reparticiones gubernamentales. Es decir, es necesario definir al menos los distintos niveles de restricciones y la compatibilidad de permisos entre las instituciones [Psee04].

Decreto 77 Eficiencia de las comunicaciones electrónicas: Establece que la transmisión o recepción de comunicaciones entre órganos del Estado o con terceras partes deben asegurar: su uso y disponibilidad posterior, ser técnicamente compatibles entre emisor y receptor, contar con medidas de seguridad contra intercepción y alteración, tener una dirección donde recibir dichas comunicaciones, poder utilizarse medios de autentificación de ser necesario, implementar una forma de respuesta a requerimientos de los ciudadanos, registrar respuestas enviadas a las últimas consultas hechas por ciudadanos, y deben existir encargados de derivar a otro encargado las comunicaciones electrónicas que correspondan [Decr04]. Junto con ello, se definen algunos metadatos mínimos para la información almacenada producto de la comunicación, entre ellos están: remitente, destinatario y fecha de la comunicación (envío y llegada).

Decreto 81 Norma técnica sobre documento electrónico: Este decreto define las características mínimas obligatorias sobre interoperabilidad, extensibilidad y mantenibilidad para el documento electrónico. Las etapas de adopción de la norma son 3. En la primera se debe recibir, almacenar, visualizar y reenviar documentos electrónicos. En la segunda, además, se deben generar documentos electrónicos, y en la tercera se incluye su procesamiento [Norm05]. El formato de documentos electrónicos se define como XML, su codificación como UTF-8, y su esquema XML Schema. Los documentos deben ser autocontenidos, referenciar a metadatos (y diccionarios semánticos) asociados al documento y soportar firma digital. Cada repartición debe hacerse cargo de la seguridad y políticas de uso

de esquemas y metadatos de los documentos. Se debe usar XFORMS y XSL para formularios y visualización. Se debe permitir el uso de Web Services para procesamiento de los documentos. Además define dos tipos de documentos electrónicos especiales: *Sobre* y *Expediente* electrónico.

Decreto 83 Seguridad y confidencialidad de los documentos electrónicos: Define las características mínimas de seguridad y confidencialidad, que deben cumplir los documentos electrónicos del Estado. Entre ellas están: la seguridad física y del ambiente donde se encuentran datos y se realizan comunicaciones, respaldo de información, medidas de contingencia, entre otras [Decr04b].

1.6.2 Desafíos

El llevar al Gobierno hacia un nivel de madurez de Gobierno electrónico unificado (Ver sección 1.4), implica un conjunto de desafíos técnicos, legales y organizacionales. A continuación se presentan aquellos más relevantes hoy.

Pautas técnicas: Deben definirse pautas técnicas que guíen el desarrollo de esquemas XML y metadatos, manejo de gestión documental, diseño y metodologías de desarrollo de servicios. Deben complementarse con experiencias y ejemplos concretos.

Migración de Información: Debido a la existencia de sistemas computacionales que no usan XML, y debido a toda la documentación existente sólo en papel, se requiere una forma de migrar esa información y hacerla interoperar con la que se genere a futuro[Norm05].

Herramientas: Se requieren herramientas para apoyar el proceso de gestión documental, tales como: administrar esquemas y documentos, definir flujos documentales y servicios, controlar/fiscalizar operaciones y sistemas, depuración.

Mantener estándares: Las leyes que norman el Gobierno electrónico en Chile pueden requerir ser modificadas, por ejemplo las normas de seguridad o exigencias sobre el documento electrónico. Crear, mantener, actualizar estas definiciones y hacerlas compatibles con las existentes, requiere un esfuerzo del Gobierno, la academia y la industria [Norm05].

Definición formal de servicios: A pesar que esta tarea no es indispensable para un uso funcional de los Servicios, esto sí limita su potencial y dificulta su administración y gestión. Por lo tanto, este problema debería ser abordado primero definiendo guías para la creación, definición y desarrollo de servicios para el Gobierno electrónico, para luego aplicarlo a los servicios disponibles y a la creación de los futuros.

Sistema de autenticación: Hoy en día, existen servicios como el pago de impuestos, que pueden realizarse en línea ingresando con un usuario y clave entregada por la repartición responsable del servicio. El desafío en el corto plazo es contar con una forma de autentificarse ante cada repartición, y en el mediano plazo, es tener una interfaz común de autentificación a todos los servicios.

Coordinación de definiciones de metadatos, esquemas y servicios en el Gobierno: Se debe procurar no repetir servicios, y asegurar que los existentes no se contradigan en su funcionamiento, semántica o sean inconsistentes.

Sumar a todas las reparticiones del Gobierno: El dejar fuera del Gobierno electrónico a ciertas reparticiones de Gobierno, implicaría seguir manejando documentos en papel para todo proceso en que intervenga tal repartición.

La definición de arquitectura, plataforma, software de aplicación, flujos documentales, tecnologías, guías para desarrollo de servicios, metadatos y esquemas, deben ser especificadas para reutilizar el conocimiento adquirido (replicar soluciones), y agilizar el desarrollo del Gobierno electrónico. Para esto, es una buena alternativa realizar experiencias piloto, que permitan deducir las mejores pautas de desarrollo, para lo cual es necesario desarrollar planes acotados que no tengan un alto riesgo debido a las dimensiones del proyecto.

1.7 Resumen del Capítulo

En este capítulo de introducción a la gestión documental en Chile, hemos revisado los diversos aspectos que involucra una solución de gestión documental, la cual involucra el manejo de documentación relativa a algún proceso. En primer lugar explicitamos su necesidad y utilidad; luego, mostramos la necesidad de contar con un buen documento electrónico, y las características que éste debe tener para permitir un correcto manejo de los documentos, usando para ello un ejemplo ilustrativo sobre el registro de un recién nacido en el Servicio de Registro Civil e Identificación, ejemplo que será usado en los siguientes capítulos para aclarar conceptos e ilustrar situaciones. Dada la necesidad de un documento electrónico, XML aparece como una excelente solución debido a su gran flexibilidad, soporte y simplicidad, lo cual lo ha convertido en el estándar de facto para el manejo de documentación electrónica.

Luego, mostramos que el documento electrónico entrega una base mínima de interoperabilidad, lo cual es sólo el primer paso hacia una gestión documental completa, ya que aparecen otros temas como el diseño de documentos, la arquitectura, los servicios y la seguridad asociada a la gestión documental. Finalmente, mostramos el estado del arte de la gestión documental electrónica en Chile, la cual cuenta con una serie de desafíos presentes que todavía deben ser abordados.

Este capítulo busca introducir los conceptos básicos relacionados con la gestión documental, y lo que ella involucra, dando una visión general del problema completo. De esta manera los siguientes capítulos del libro desarrollan en detalle cada uno de estos conceptos, mostrando los puntos más relevantes propios de cada tema.

1.8 Bibliografía y Referencias

[Arab02] Arab Republic of Egypt. E-Government Document Classification and Handling. Recommendations.Ministry of Communication and Information Technology. http://www.mcit.gov.eg/e-Governmnet_Handlingv1_3.pdf. 2002.

[Aust95] Australia Government. IESC's Electronic Data Management Subcommittee. Improving Electronic Document Management: Guidelines For Australian Government Agencies. Australia. 1995. URL: http://web.archive.org/web/20020601155855/http://www.defence.gov.au/imsc/edmsc/iedmtc.htm

[Brow01] Brown, John and Duguid, Paul. La Vida Social de la Información. Prentice Hall. Buenos Aires, Argentina. 2001.

[Decr04] Decreto 77: Norma técnica sobre eficiencia de las comunicaciones electrónicas entre órganos de la administración del Estado y entre estos y los ciudadanos. Junio 2004.

[Decr04b] Decreto 83: Norma técnica para los órganos de la administración del Estado sobre Seguridad y Confidencialidad de los documentos electrónicos. Junio 2004.

[Ebxm05] OASIS. ebXML – Enabling A Global Electronic Market. Noviembre 2005. http://www.ebxml.org

[Firm02] Ley 19.799: Regulación de documentos electrónicos, firma electrónica y servicios de certificación de dicha firma. Marzo 25, 2002. URL: http://www.modernizacion.cl/1350/articles-40703_ley_19799_firma_electronica.pdf

[Mini02] E-Government Document Classification and Handling Recommendations. Ministry of communication and Information Technology. Arab Republic of Egypt. E-Government Workgroup. Septiembre 2002.

[Norm05] Gobierno de Chile. Decreto Nro: 81, 2005. URL: http://www.modernizacion.cl/1350/article-70681.html

[Oasi06] OASIS. OpenDocument Essentials: Using OASIS OpenDocument XML. http://books.evc-cit.info/odbook/book.html. O'Reilly & Associates, Inc. 2005.

[Open06] OpenOffice.org. The StarOffice XML based file format. URL: http://xml.openoffice.org/xml_advocacy.html. Last visit: March, 2006.

[Psee04] Plataforma Integrada de Servicios del Estado: un Proyecto Unificador. Presentación de caso. PRYME. Gobierno de Chile. Thierry de Saint Pierre. 2004. http://www.e2g.gov.cl

[Taps00] Tapscott, Don. La Economía Digital. MacGraw Hill. Santa Fé, Colombia. 2000.

[Terp03] Handbook for Standardization by the Danish XML Committee. Version 1.0. February 2003. http://www.oio.dk/files/Standardization_Handbook_v1_20030203.pdf http://www.nationalarchives.gov.uk/recordsmanagement/ http://www.naa.gov.au/recordkeeping/rkpubs/summary.html

[UKin05] Government of United Kingdom. e-Government Policy Framework for Electronic Records Management. Last visit: May 2006. URL: http://www.nationalarchives.gov.uk/electronicrecords/pdf/egov_framework.pdf http://www.nationalarchives.gov.uk/electronicrecords/pdf/egov_framework.pdf http://www.tomw.net.au/2005/dm/edocmgt.html http://www.tomw.net.au/2001/mdm.html

[Vara03] Varas, Samuel, et al. Gobierno Electrónico en Chile: Estado del Arte. Proyecto de Reforma y Modernización del Estado. Ministerio Secretaría General de la Presidencia. Departamento de Ingeniería Industrial, Universidad de Chile. Abril 2003.

[W3C04] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. Third Edition. W3C Recommendation. 04 February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/

CAPITULO II: XML

Alex Bórquez, Claudio Gutiérrez, Renzo Angles

Departamento de Ciencias de la Computación, Universidad de Chile.
{aborquez, cgutierr, rangles}@dcc.uchile.cl

2.1 Breve Reseña de HTML: Long Time Ago, in a Galaxy Far Far Away…

HTML (*Hyper Text Markup Language*) fue concebido por Tim Berners Lee en 1989 como un lenguaje para el intercambio de documentos científicos y técnicos, adaptado para ser usado por no especialistas en el tratamiento de documentos. HTML es un lenguaje derivado de SGML (*Standard Generalized Markup Language*), un lenguaje para describir la estructura de documentos. HTML consta de un reducido conjunto de etiquetas (o *tags*) para la confección de documentos relativamente simples. Cada etiqueta posee un significado fijo, es decir, no puede ser cambiado. Actualmente HTML, en su versión 4.01 (la versión 5 está en draft), es considerado como una aplicación SGML propiamente tal, conforme al estándar internacional ISO 8879, y es ampliamente aceptado como el lenguaje de publicación estándar de la World Wide Web Consortium[7] (W3C).

> **SGML** es la sigla en inglés para Standard Generalized Markup Language (lenguaje estándar de marcado generalizado). Consiste en un sistema para la organización y etiquetado de documentos. La Organización Internacional de Estándares (ISO) ha normalizado este lenguaje desde 1986. El lenguaje SGML sirve para especificar las reglas de etiquetado de documentos y no impone en sí ningún conjunto de etiquetas en especial. El HTML está definido en términos de SGML. XML es un nuevo estándar con una funcionalidad similar a la del SGML aunque más sencillo, y de creación posterior. La industria de la publicación de documentos constituye uno de los principales usuarios del lenguaje SGML. Empleando este lenguaje se crean y mantienen documentos que luego son llevados a otros formatos finales como HTML, PDF, Postscript, RTF, etc.

2.1.1 Inconvenientes generales de HTML

En poco tiempo, HTML se hizo muy popular y superó largamente los propósitos para los cuales había sido ideado. Desde sus albores, ha habido una sostenida generación de nuevas etiquetas para ser usadas dentro de HTML (como estándar) o para adaptar HTML a mercados verticales, altamente especializados. Esta cantidad de nuevas etiquetas ha llevado a problemas de compatibilidad de los documentos en las distintas plataformas. Entre algunos de los inconvenientes específicos de HTML se encuentran:

- El código HTML puede presentar información directamente a los humanos, y esto sin duda es algo bueno, pero es un lenguaje complicado de procesar para los programas informáticos.

[7] El W3C es un consorcio internacional de industrias. En él participan el MIT (EEUU), INRIA (Francia) y Keio University (Japón). Además, cuenta con el soporte oficial de DARPA (EEUU) y la Comisión Europea. fue creado en 1994, con el objetivo de desarrollar protocolos comunes para la evolución de Internet. http://www.w3c.org

- HTML carece de la posibilidad de añadir semántica adicional a la información que presenta.

- HTML no indica lo que está representando.

- HTML se preocupa principalmente de cómo debe representarse (que algo debe ir en azul, o con cierto tipo de letra), pero no dice que lo mostrado es el título de un libro o el precio de un artículo.

2.1.2 Resumiendo: HTML

Resumiendo, podemos decir que: (a) HTML define más la presentación que el contenido, (b) no es fácilmente procesable por "máquinas", (c) es algo "caótico" en cuanto a su estructura gramatical, (d) es ambiguo respecto a su interpretación según el software utilizado, y (e) se usa principalmente en la Web.

2.2 Introducción al XML

XML es un acrónimo para e**X**tensible **M**arkup **L**anguage (Lenguaje eXtensible de Marcas). XML es un lenguaje abierto[8] que se ha desarrollado desde 1996 como un subconjunto de SGML. En febrero de 1998 fue adoptado como estándar por la W3C. XML permite describir el sentido o la semántica de los datos pues, a diferencia del HTML, separa el contenido de la presentación.

La idea que subyace al XML es la de crear un lenguaje muy general, que sirva para diferentes ámbitos de aplicación. Para ser más precisos, XML es un meta-lenguaje, pues permite la especificación de lenguajes concretos de representación de documentos. El XML describe el contenido de lo que etiqueta o marca. En general, un programa informático puede trabajar más eficientemente sobre documentos XML, que sobre HTML.

> **XML** es un acrónimo para eXtensible Markup Language (lenguaje de marcado extensible o ampliable). XML es un lenguaje abierto que se ha desarrollado como un subconjunto de SGML, estandarizado por la W3C. Al igual que el HTML, se basa en documentos de texto plano en los que se utilizan etiquetas para delimitar los elementos de un documento. Sin embargo, XML define estas etiquetas en función del tipo de datos que está describiendo y no de la apariencia final que tendrán en pantalla o en la copia impresa, además de permitir definir nuevas etiquetas y ampliar las existentes. Son varios los vocabularios desarrollados en XML con el fin de ampliar sus aplicaciones. Podemos considerar fundamentales: XHTML, XSL-FO y XSLT, XLink, XPointer y Schema. Además, existen también versiones para usos específicos, como MathML (fórmulas matemáticas), SVG (gráficos vectoriales), RSS (sindicación de noticias), o XBRL (partes financieros).

[8] Esto quiere decir que para utilizarlo no es necesario pagar por una licencia.

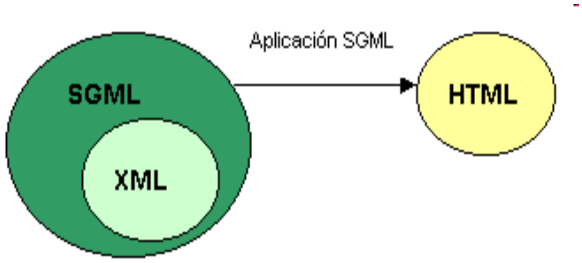


Figura 4: Diferencia entre el SGML, XML y HTML

Tabla 1: ¿Qué no es XML? y ¿Por qué XML?

¿Qué no es XML?	¿Por qué XML?
XML no es una "versión mejorada de HTML".	Es un estándar ampliamente aceptado y difundido.
XML no es un lenguaje para hacer "mejores" páginas Web.	No pertenece a ninguna compañía y su utilización es libre.
XML no es "difícil".	Permite una interacción más completa con la Web.

2.2.1 Características de XML

Algunas de las características más destacables de XML son las siguientes:

- Las etiquetas y sus atributos pueden ser personalizadas.
- La sintaxis es estricta. La especificación XML determina claramente una serie de reglas que especifican cuándo un documento está "bien formado".
- Es posible definir familias de documentos con una estructura que se considerará "válida". Los principales tipos de documentos usados para especificar estructuras son Document Type Definition (DTD) y XML Schema (XSD).

DTD es la sigla en inglés para Document Type Definition. Es un documento SGML o XML que especifica ciertas reglas o restricciones que debe cumplir un documento XML para ser considerado válido. Los DTDs son generalmente empleados para determinar la estructura de un documento XML o SGML. Un DTD, típicamente, describe cada elemento admisible dentro del documento, los atributos posibles y (opcionalmente) los valores de atributo permitidos para cada elemento. Además describe los anidamientos y ocurrencias de elementos.

El DTD puede ser incluido dentro del archivo del documento, pero normalmente se almacena en un archivo de texto separado. La sintaxis de los DTDs para SGML y XML es similar pero no idéntica.

XSD es otro nombre para "XML Schema". Se trata de un documento de definición estructural al estilo de los DTD, que además cumple con el estándar XML y permite expresar mayor diversidad de documentos. Los documentos XML Schema (usualmente con extensión XSD) se concibieron como un sustituto de los DTD, teniendo en cuenta los puntos débiles de éstos y buscando mejores capacidades a la hora de definir estructuras para los documentos XML, como la declaración de los tipos de datos.

2.2.2 Documentos bien formados y documentos válidos

Se dice que un documento XML está "bien-formado" (*well-formed*), cuando se trata de un documento XML que cumple con todas las reglas sintácticas que especifica el estándar XML 1.0 del W3C. Se dice que un documento XML es "válido" cuando se comprueba que éste cumple con las especificaciones de un cierto DTD o XSD. Un XML válido para un determinado DTD o XSD, puede no necesariamente serlo para otro.

2.2.3 Resumen de ventajas de XML

Como resumen de las ventajas que aporta XML se pueden mencionar las siguientes: (a) es un lenguaje libre y extensible, (b) fácilmente procesable por humanos y software, (c) involucra una tecnología estándar ya implantada en el mercado, (d) está diseñado para ser utilizado con cualquier codificación estándar (UTF, ISO, etc.), (e) separa el contenido de su representación, (f) tiene un formato ideal para transacciones documentales, (g) permite la aplicación de técnicas de extracción de información y minería de datos, y (h) tiene reglas estrictas para la composición de un documento XML, las cuales permiten su fácil análisis sintáctico.

"Parser" es el término inglés para "**analizador sintáctico**". Un analizador sintáctico es un programa que reconoce si una o varias cadenas de caracteres forman parte de un determinado lenguaje. Los analizadores sintácticos fueron extensivamente estudiados durante los años 70 del siglo XX, detectándose numerosos patrones de funcionamiento en ellos, cosa que permitió la creación de programas generadores de analizadores sintácticos a partir de una especificación de la sintaxis del lenguaje.

2.3 Construcción de documentos XML

Los fundamentos del XML son sencillos y en principio, las únicas herramientas necesarias son: (a) un editor de textos para escribir los documentos XML (el Bloc de Notas (Notepad) de Microsoft Windows o el XEmacs para Linux, por ejemplo) y (b) un procesador o parser XML (Explorer 5.X, Mozilla, XMLWriter).

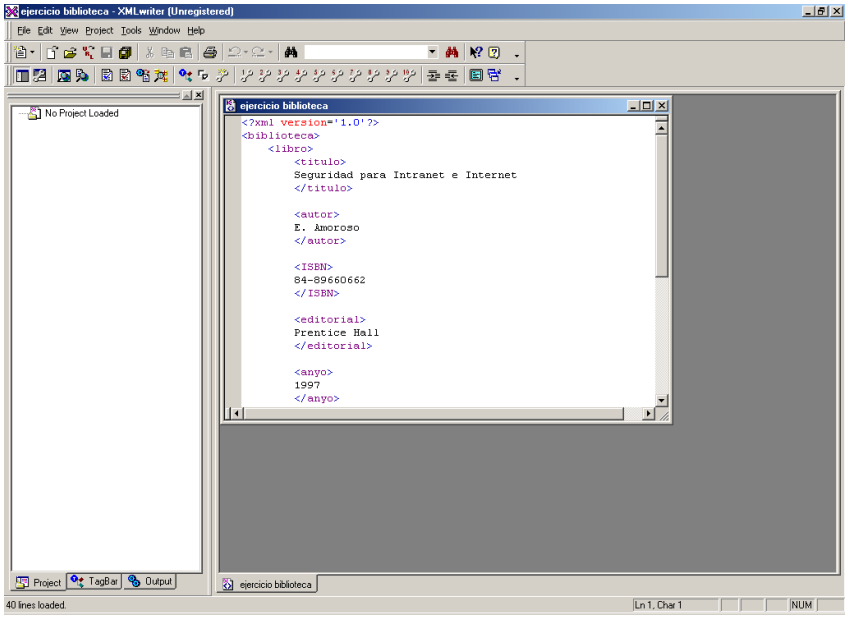


Figura 5: Imagen de editor de XML

2.3.1 Estructura de un documento XML

Aunque a primera vista, un documento XML puede parecer similar a HTML, hay una diferencia principal: un documento XML contiene exclusivamente datos que se autodefinen. En cambio, un documento HTML contiene datos (cuya definición es, al menos, ambigua), mezclados con elementos de formato. En XML se separa el contenido de la presentación de forma total. Una forma rápida de entender la estructura de un documento XML es viendo un ejemplo:

```
<?xml version='1.0'?>
<nacimiento>
      <fecha>03-02-2006</fecha>
      <perfilSanguineo grupo='AB' factorRH='+' />
      <nombre>
            <nombrePropio lugar='primero'>Ivan</nombrePropio>
            <nombrePropio lugar='segundo'>Luis</nombrePropio>
            <apellido parentesco='paterno'>Zamorano</apellido>
            <apellido parentesco='materno'>Albero</apellido>
      </nombre>
</nacimiento>
```

2.3.2 Prólogo

Lo primero que debe observarse es la primera línea. Con ella deben empezar todos los documentos XML, ya que es la que indica que lo que la sigue es efectivamente un documento XML. Puede tener varios atributos, algunos obligatorios y otros no:

- **version:** Indica la versión del estándar XML usada en el documento. Es obligatorio ponerlo, a no ser que sea un documento XML externo a otro que ya incluía esta declaración.

- **encoding:** La forma en que se ha codificado el documento. Se puede poner cualquiera, y depende del analizador sintáctico (*parser*) el entender o no la codificación. Por omisión, se supone que es UTF-8, aunque podrían ponerse otras, como US-ASCII, UTF-16, UCS-2, EUC-JP, Big5, ISO-8859-1 hasta ISO- 8859-7, etc. No es obligatorio, salvo que sea un documento externo a otro principal.

- **standalone:** Indica si el documento va acompañado de un DTD ("no"), o no lo necesita ("yes"); en principio no es obligatorio ponerlo, pues luego se indica el DTD si se necesita.

> Para el uso de lenguajes europeos, incluyendo el juego de caracteres especiales del castellano, en general se usa UTF-7 o ISO-8859-1.

La segunda línea puede contener una "declaración de tipo de documento" DTD, define qué tipo de documento estamos creando para ser procesado correctamente.

Ejemplo

```
<?xml version='1.0' ?>
<?xml version="1.0" encoding="UTF-7" ?>
<?xml version="1.0" encoding=" US-ASCII" ?>
<?xml version="1.0" encoding="UTF-7" ?>
<!DOCTYPE ejemplo SYSTEM "http://www.ejemplos.xml/ejemplo.dtd" >
```

2.3.3 Elementos

Son las partes esenciales de un documento XML. Los elementos XML pueden tener contenido (más elementos anidados, caracteres, o ambos a la vez), o bien ser elementos vacíos. Un elemento debe tener un nombre, y este nombre debe ir entre ciertos caracteres especiales. Estos caracteres dan comienzo y fin a un elemento, definiendo de forma exacta cada una de sus partes.El nombre de un elemento debe ir entre el carácter "<" (menor que) y el carácter ">" (mayor que) de la siguiente forma:

```
<nombre_del_elemento>
```

Esta construcción constituye la etiqueta de comienzo. A continuación puede colocarse texto o más elementos anidados. Se considera como contenido del elemento todo lo que se encuentre entre su etiqueta de comienzo y su etiqueta de fin. La etiqueta de fin debe construirse exactamente igual que la de comienzo, con la única diferencia que después del carácter "<" sigue un carácter "/" (*slash*), como se muestra a continuación:

```
</nombre_del_elemento>
```

Al contrario de lo que ocurre en HTML, en XML siempre debe existir la etiqueta de cierre de un elemento, salvo que éste sea un elemento vacío. Este caso se verá en el punto siguiente. Es importante diferenciar entre *elementos* y *etiquetas*: los elementos son las entidades en sí, lo que tiene contenido, mientras que las etiquetas sólo describen los elementos. Un documento XML está compuesto por elementos, y en su sintaxis, éstos se nombran mediante etiquetas. El símbolo "<" siempre se interpreta como inicio de una etiqueta XML. Más adelante veremos algunas alternativas para escribir estos símbolos en otros contextos.

Ejemplo

```
<fecha> 03-02-2006 </fecha>
<apellido parentesco='paterno'> Zamorano </apellido>
```

2.3.4 Elementos vacíos

Se llama "elemento vacío" a un elemento que no tiene contenido. Como se mencionó anteriormente, todo elemento en XML debe tener etiqueta de cierre. Dado que el elemento vacío carece de contenido, es posible abreviar y evitar escribir dos etiquetas. La manera de cerrar un elemento vacío consiste en agregar un "/" antes del carácter ">", como indica el ejemplo siguiente:

Ejemplo

```
<perfilSanguineo/>
<perfilSanguineo grupo='AB' factorRH='+'/>
```

En el primer elemento del ejemplo anterior, se muestra cómo se cierra un elemento vacío. El segundo elemento del mismo ejemplo *también es un elemento vacío*, con la única diferencia que este elemento posee atributos.

2.3.5 Atributos

Los atributos son componentes opcionales -pero muy poderosos- de los elementos. Es una manera de incorporar características o propiedades a los elementos de un documento XML. Un atributo consta de dos partes: la propiedad del elemento y el valor de la propiedad.

Ejemplo

```
<nombrePropio lugar='primero'>Ivan</nombrePropio>
<apellido parentesco="paterno">Zamorano</apellido>
```

Al igual que en otras cadenas de caracteres en XML, los atributos pueden estar marcados entre comillas verticales simples (') o dobles ("). Cuando se usa un tipo de comillas para delimitar el valor del atributo, el otro tipo de comillas puede usarse dentro. Un elemento con contenido a veces puede modelarse como un elemento vacío con atributos. Más adelante veremos cuándo conviene una u otra versión.

Ejemplo

```
<nacimiento>
      <fecha>03-02-2006</fecha>
      <apellido>Zamorano</apellido>
</nacimiento>


<nacimiento apellido="Zamorano">
      <fecha>03-02-2006</fecha>
</nacimiento>


<nacimiento apellido="Zamorano" fecha="03-02-2006" />
```

A modo de resumen, la Figura 6 muestra un elemento con contenido y con un atributo, indicando el nombre de cada una de sus partes.

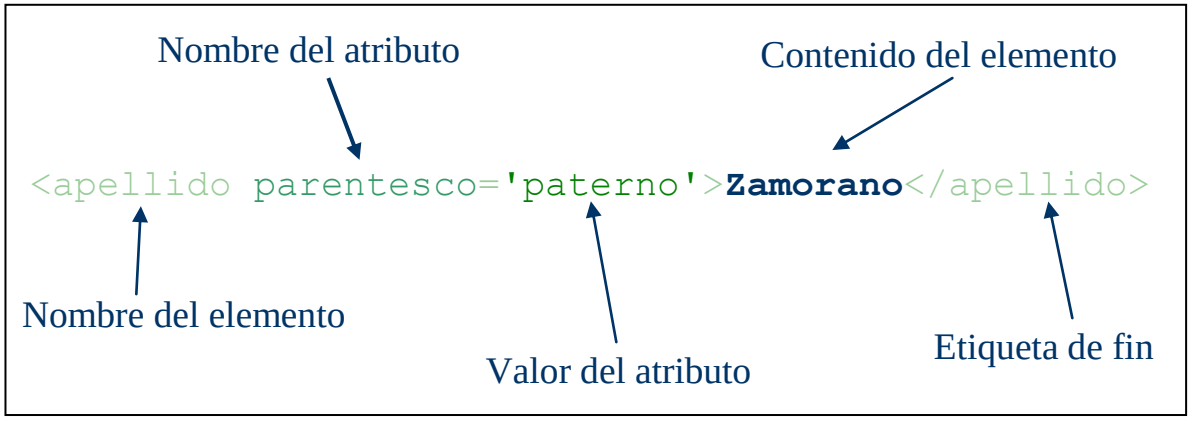


Figura 6: Elemento y atributo

2.3.6 Entidades predefinidas

En XML 1.0 se definen cinco entidades para representar caracteres especiales. Estos caracteres no se interpretan como caracteres de marcado.

Entidad	Carácter
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Figura 7: Entidades predefinidas de XML

Por ejemplo, podemos usar el carácter "<" sin que se interprete como el comienzo de una etiqueta XML.

Ejemplo

```
<?xml version="1.0" encoding="UTF-7" ?>
<ejemplos>
      <descripcion>Lo siguiente es un ejemplo de HTML.</descripcion>
      <ejemplo>
             &lt;HTML&gt;
                    &lt;HEAD&gt;
                           &lt;TITLE&gt;Rock &amp; Roll&lt;/TITLE&gt;
                    &lt;/HEAD&gt;
      </ejemplo>
</ejemplos>
```

2.3.7 Secciones CDATA

Las secciones CDATA (*Character Data*) permiten especificar datos utilizando cualquier carácter (especial o no) sin que se interprete como marcado XML. Un motivo de existencia de estas secciones es que a veces se necesita que un documento XML pueda ser leído fácilmente por un humano. Si un documento tiene muchos códigos de entidades predefinidas, se hace muy difícil su comprensión visual. Las secciones CDATA empiezan por la cadena "<![CDATA" y terminan con la cadena "]]>". Las secciones CDATA no se pueden anidar.

Ejemplo

```
<ejemplo>
      &lt;HTML&gt;
             &lt;HEAD&gt;
                    &lt;TITLE&gt;Rock &amp; Roll&lt;/TITLE&gt;
             &lt;/HEAD&gt;
</ejemplo>


<ejemplo>
<![CDATA[
      <HTML>
```

```
            <HEAD>
                  <TITLE>Rock & Roll</TITLE>
            </HEAD>
]]>
</ejemplo>
```

2.3.8 Comentarios

A veces es conveniente insertar ciertas marcas en un documento XML pero que sean ignoradas por el procesador de la información. Los comentarios tienen el mismo formato que los comentarios de HTML. Es decir, comienzan por la cadena "<!--" y terminan con "-->". Se puede introducir comentarios en cualquier lugar del documento, salvo dentro de las declaraciones del prólogo, dentro de las etiquetas de los elementos, o dentro de otro comentario.

Ejemplo

```
<!-- Un comentario puede ir aqui -->
<?xml version='1.0'?>
<!-- O aquí -->
<nacimiento>
     <!-- Aca tambien -->
     <fecha>03-02-2006</fecha>
     <perfilSanguineo grupo='AB' factorRH='+' />
     <nombre>
            <!-- Incluso aqui -->
            <nombrePropio lugar='primero'>Ivan</nombrePropio>
            <nombrePropio lugar='segundo'>Luis</nombrePropio>
            <apellido parentesco='paterno'>Zamorano</apellido>
            <apellido parentesco='materno'>Albero</apellido>
     </nombre>
     <!-- O aca -->
</nacimiento>
<!-- Y aqui -->
```

2.3.9 Sintaxis de un documento XML

2.3.9.1 Anidación de los elementos

Los elementos deben seguir una estructura de "árbol". Con esto se quiere decir que la estructura del documento (en particular, de sus elementos) debe ser estrictamente jerárquica. Los elementos no pueden superponerse entre ellos. Cuando esto se cumple, se dice que los elementos están "correctamente anidados".

Ejemplo Incorrecto

```
<?xml version='1.0'?>
<nacimiento>
      <nombre>
             <nombrePropio lugar='primero'>Ivan</nombrePropio>
             <apellido parentesco='paterno'>Zamorano</apellido>
</nacimiento>
      </nombre>
```

Ejemplo Correcto

```
<?xml version='1.0'?>
<nacimiento>
      <nombre>
             <nombrePropio lugar='primero'>Ivan</nombrePropio>
             <apellido parentesco='paterno'>Zamorano</apellido>
      </nombre>
</nacimiento>
```

2.3.9.2 Elemento raíz

Debe existir un único elemento “raíz”, en el cual estén contenidos todos los demás elementos.

Ejemplo Incorrecto

```
<?xml version='1.0'?>
      <fecha>03-02-2006</fecha>
      <nombre>
             <nombrePropio lugar='primero'>Ivan</nombrePropio>
             <apellido parentesco='paterno'>Zamorano</apellido>
      </nombre>
```

Ejemplo Correcto

```
<?xml version='1.0'?>
<nacimiento>
      <fecha>03-02-2006</fecha>
      <nombre>
             <nombrePropio lugar='primero'>Ivan</nombrePropio>
             <apellido parentesco='paterno'>Zamorano</apellido>
      </nombre>
</nacimiento>
```

2.3.9.3 Cierre de elementos

Todas las etiquetas que delimitan a un elemento deben que estar correctamente “cerradas”, es decir, con una etiqueta de cierre que se corresponda con la de apertura, como se expuso anteriormente.

Ejemplo Incorrecto

```
<?xml version='1.0'?>
<nacimiento>
      <fecha>03-02-2006
      <nombre>
             <nombrePropio lugar='primero'>Ivan
             <apellido parentesco='paterno'>Zamorano
      </nombre>
</nacimiento>
```

Ejemplo Correcto

```
<?xml version='1.0'?>
<nacimiento>
      <fecha>03-02-2006</fecha>
      <nombre>
             <nombrePropio lugar='primero'>Ivan</nombrePropio>
             <apellido parentesco='paterno'>Zamorano</apellido>
      </nombre>
</nacimiento>
```

Es bueno recordar que los elementos “vacíos” (es decir, sin contenido) tienen una sintaxis especial.

Ejemplo Incorrecto

```
<?xml version='1.0'?>
<nacimiento>
      <perfilSanguineo grupo='AB' factorRH='+'>
</nacimiento>
```

Ejemplo Correcto

```
<?xml version='1.0'?>
<nacimiento>
      <perfilSanguineo grupo='AB' factorRH='+' />
</nacimiento>
```

2.3.9.4 Delimitadores del valor de un atributo

Los valores de los atributos de los elementos siempre deben estar marcados con las comillas dobles (") o sencillas (')

Ejemplo Incorrecto

```
<?xml version=1.0 ?>
<nacimiento>
      <perfilSanguineo grupo=AB factorRH=+ />
      <nombre>
             <nombrePropio lugar=primero>Ivan</nombrePropio>
             <apellido parentesco=paterno>Zamorano</apellido>
      </nombre>
</nacimiento>
```

Ejemplo Correcto

```
<?xml version='1.0' ?>
<nacimiento>
      <perfilSanguineo grupo="AB" factorRH='+' />
      <nombre>
             <nombrePropio lugar="primero">Ivan</nombrePropio>
             <apellido parentesco='paterno'>Zamorano</apellido>
      </nombre>
</nacimiento>
```

2.3.9.5 Nombres de elementos

Según la especificación XML 1.0:

> *Un nombre [empieza] con una letra o, uno o más signos de puntuación, y [continúa] con letras, dígitos, guiones, rayas, dos puntos o puntos, denominados de forma global como caracteres de nombre. Los nombres que empiezan con la cadena "xml" se reservan para la estandarización de ésta o de futuras versiones de esta especificación.*

Un nombre de elemento, atributo, entidad, etc. empieza por una letra, y continúa con letras, dígitos, guiones, rayas, punto o dos puntos. El acrónimo "XML" (o "xml" o "xMl", etc.) *no puede usarse* como caracteres iniciales de un nombre de elemento, atributo, entidad, etc.

2.3.9.6 Mayúsculas, minúsculas y espacios en blanco

El XML es sensible a mayúsculas o minúsculas (*case-sensitive*), es decir, las trata como caracteres diferentes. No es lo mismo **`<automovil>`** que **`<Automovil>`**. La especificación XML 1.0 permite el uso de "espacios en blanco" entre etiquetas (es decir, fuera de los elementos) para hacer más legible el código. En general estos espacios en blanco son

ignorados por los procesadores XML. Sin embargo, resultan muy significativos cuando los espacios se usan para separar las palabras en un texto, que es contenido de un elemento.

2.3.9.7 Marcado y datos

Las construcciones como etiquetas (*tags*), referencias de entidad o declaraciones de prólogo, se denominan "marcas" (*mark-up*). Éstas son las partes del documento que el analizador sintáctico (*parser*) espera comprender. El resto del documento, que se encuentra dentro de las marcas, son los datos que resultan entendibles por las personas. Es sencillo reconocer las marcas en un documento XML: son aquellas cadenas de caracteres que empiezan con "<" y acaban con ">"; en el caso de las referencias de entidad, empiezan por "&" y acaban con ";".

2.3.10 Formación y Validez de un documento XML

Un documento XML tiene dos dimensiones, una "gramatical" y otra "lógica". En su dimensión gramatical, el documento está compuesto por declaraciones, elementos, comentarios, referencias a caracteres e instrucciones de procesamiento, todos ellos están indicados por una marca explícita.

Si un documento XML tiene una gramática correcta, es decir que cumple con las reglas sintácticas anteriormente descritas, puede decirse que el documento está "bien formado" (*well-formed*). Un documento que no está bien formado, no puede ser procesado correctamente por un analizador sintáctico (*parser*), por lo tanto no es un documento XML. En su dimensión lógica, el documento está compuesto por unidades llamadas entidades. Una entidad puede hacer referencia a otra entidad, causando que ésta, se incluya en el documento. Cada documento comienza con una entidad documento, también llamada "raíz".

La correctitud de la dimensión estructural del documento se comprueba a través del procedimiento conocido como "validación". Para que un documento XML sea validado, debe primeramente estar bien formado. Un documento XML es válido si tiene asociado una DTD o un XSD y el documento cumple las restricciones que éstas expresan.

2.4 Modelo conceptual de los documentos XML

Ya se ha mostrado cómo construir un documento XML bien formado. Ahora es el momento de detenerse en el modelo conceptual que hay detrás de un documento XML, el cual es conocido como DOM (*Document Object Model*) y es custodiado por el W3C.

> **DOM** o Modelo de Objetos del Documento (Document Object Model, en inglés) es una forma de representar documentos estructurados (tales como una página Web HTML o un documento XML), que es independiente de cualquier lenguaje. Su finalidad es definir el conjunto de objetos que pueden componer documentos HTML (páginas Web) o XML, así como las estructuras que se definen dentro de él, sus propiedades y sus métodos, independientemente del lenguaje de programación utilizado, con el fin de evitar problemas de compatibilidad entre navegadores. El custodio del DOM es la W3C. En efecto, el DOM es una API para acceder, añadir y cambiar dinámicamente el contenido estructurado en documentos.

El modelo de objetos DOM, es la forma común y estructurada, mediante la cual se representan datos XML en memoria, aunque los datos XML reales se almacenen de forma lineal cuando se encuentran en un archivo. Este último proceso se conoce como "serializar" (*serialize*).

Más formalmente, DOM es una representación interna estándar de la estructura de un documento, y proporciona una interfaz al programador (API) para poder acceder de forma fácil, consistente y homogénea a sus elementos, atributos y estilo. Es un modelo independiente de la plataforma y del lenguaje de programación. La W3C establece varios niveles de actuación:

- **Nivel 1**: Se refiere a la parte interna, y modelos para HTML y XML. Contiene funcionalidades para la navegación y manipulación de documentos. Tiene 2 partes: el *core* o parte básica, referida a documentos XML, y la parte HTML, referida precisamente a los documentos HTML.

- **Nivel 2**: Incluye un modelo de objetos e interfaz de acceso a las características de estilo del documento, definiendo funcionalidades para manipular la información sobre el estilo del documento. También incluirá un modelo de eventos para soportar los *XML namespaces* y consultas enriquecidas.

Posteriores niveles especificarán interfaces a posibles sistemas de ventanas, manipulación de DTD y modelos de seguridad. Por el momento ya se ha sacado el 1er nivel como recomendación (Octubre 1998), y el borrador público (*public draft*) del 2do, es decir, está en etapa de consulta y sujeto a comentarios.

El objetivo es que, cualquier *script* pueda ejecutarse, de forma más o menos homogénea, en cualquier navegador que soporte dicho DOM, y tener una plataforma estándar en la cual poder crear contenidos sin temor a no estar soportado por alguna marca o versión de navegador, que además sea potente y versátil. En la Figura 8 se muestra cómo se estructura la memoria cuando se leen estos datos XML en la estructura DOM.

```
<?xml version="1.0"?>
  <books>
    <book>
        <author>Carson</author>
        <price format="dollar">31.95</price>
        <pubdate>05/01/2001</pubdate>
    </book>
    <pubinfo>
        <publisher>MSPress</publisher>
        <state>WA</state>
    </pubinfo>
  </books>
```

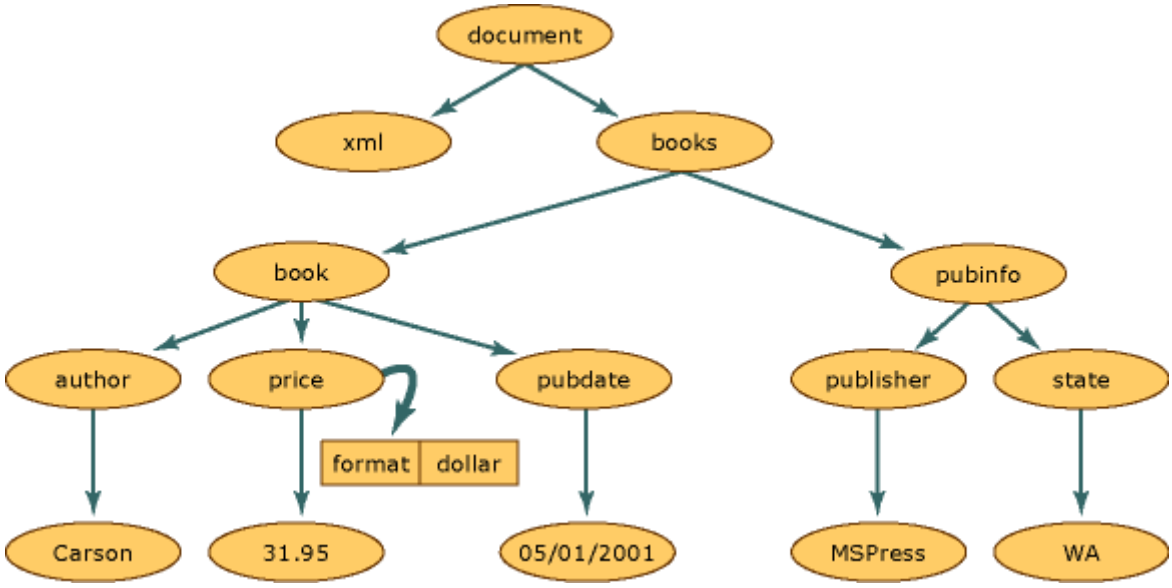


Figura 8: Representación en memoria de árbol XML

Dentro de la estructura de los documentos XML, cada círculo de la Figura 8 representa un "nodo". Los nodos tienen un conjunto de métodos y propiedades, así como características básicas y bien definidas. Algunas de estas características son:

- Un nodo tiene un único nodo primario, que se encuentra directamente encima de él. El único nodo que no tiene un nodo primario es la "raíz" (document), puesto que éste es el nodo de nivel superior y contiene el propio documento y fragmentos de documentos.

- La mayor parte de los nodos pueden tener varios nodos secundarios, que son los que están situados inmediatamente debajo de ellos. Los tipos de nodo que pueden tener nodos secundarios son los siguientes: *Document*, *DocumentFragment*, *EntityReference*, *Element* y *Attribute*. Los nodos *XmlDeclaration*, *Notation*, *Entity*, *CDATASection*, *Text*, *Comment*, *ProcessingInstruction* y *DocumentType* no tienen nodos secundarios.

- Los nodos que se encuentran en el mismo nivel, representados en el diagrama por los nodos "book" y "pubinfo", son nodos relacionados.

La forma de controlar los atributos, es una característica de DOM. Los atributos no son nodos que forman parte de las relaciones entre los nodos primarios y secundarios, o entre nodos relacionados. Los atributos se consideran una propiedad del nodo de elemento y están formados por un par nombre-valor.

Los nodos se crean en la memoria al leer un documento XML. Sin embargo, no todos los nodos son del mismo tipo. Un elemento, en XML, tiene reglas y sintaxis diferentes a las de una instrucción de procesamiento. Por tanto, cuando se leen varios datos, se asigna a cada nodo un tipo; este tipo determina las características y funcionalidad del nodo. En síntesis, DOM provee una interfaz que permite leer datos XML en memoria y cambiar su estructura,

agregar o quitar nodos, o modificar los datos mantenidos en un nodo, como en el texto contenido en un elemento.

2.5 Recapitulación

XML es el formato universal (por ahora) para el intercambio de datos entre sistemas a través Internet. Además, es un estándar para escribir datos estructurados en un archivo de texto. Es similar a HTML pero no es igual; es nuevo, pero no tanto. En general se procesa con computadores, pero puede ser leído directamente por un humano. Consta de una familia de tecnologías relacionadas. Debe escribirse con prolijidad, pero eso no debe suponer un problema. Finalmente, podemos decir que el uso de XML no requiere licencias, es independiente de la plataforma y tiene un amplio soporte.

Ejercicio

Realice un documento XML sobre el nacimiento de una persona en un hospital, que tenga nombres y apellidos del recién nacido, nombres, apellidos y nacionalidad de los padres, fecha de nacimiento, peso, estatura, tipo de sangre y factor RH, equipo médico que participó en el parto, nombre del hospital, comuna y ciudad donde se encuentra (puede llamar las etiquetas como quiera).

2.6 Referencias

- Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation 04 Feb 2004. http://www.w3.org/TR/2004/REC-xml-20040204/
- El lenguaje XML y su aplicación a la construcción de sitios Web. http://wwwdi.ujaen.es/~vrivas/docencia/cursillos/xml_baeza/alumnos/recursos/XML-XML-Alfonso.ppt

CAPITULO III: XML Schema

Alex Bórquez, Claudio Gutiérrez

Departamento de Ciencias de la Computación, Universidad de Chile.
{aborquez, cgutierr}@dcc.uchile.cl

3.1 Introducción a los XML Schemas

En el capítulo de XML se expusieron dos conceptos importantes: formación y validez. Su importancia reside en que XML es usado tanto como medio de intercambio de información, como medio de persistencia. En este contexto es intuitivo pensar que es necesario contar con una forma estándar y eficiente de validar la información, pero eso no es todo. Es igualmente –o más– importante averiguar qué posibilidades nos brinda la tecnología XML en cuanto al modelado de datos.

La información está viva y por tanto, el cambio constante. De ahí la importancia de estructurar la información correctamente y de un modo más intuitivo; o sea, debe estar preparada para el cambio. Esto significa que debe ser extensible y estar jerarquizada. Como propuestas para enfrentar el problema de la estructuración de la información de un documento XML se han definido, al menos tres soluciones que han alcanzado cierto consenso dentro de la comunidad; estas son:

- Document Type Definition (DTD)
- Relax NG (RNG)
- XML Schema (XSD)

Todas ellas comparten la misma aproximación a la solución del problema: definir un objeto externo al documento XML, que permita decidir si el documento en cuestión adhiere a la estructura deseada.

3.1.1 Un poco de historia

Antes de revisar con más detalle los schemas XML, nos detendremos un momento en las otras alternativas. La DTD fue la primera solución en ser utilizada para definir – y por tanto, validar – el vocabulario necesario para identificar los elementos de un documento XML y especificar la estructura que éstos debían seguir. La razón de esta elección es que las DTDs se habían usado con anterioridad en documentos SGML, estándar de cual XML es subconjunto.

Ese aspecto fuerte de las DTDs está en la definición del modelo de contenido de un documento XML, es decir, en qué posición y qué elementos pertenecen a otro elemento de orden superior en la jerarquía del documento. El aspecto débil de las DTDs es su limitada capacidad de especificación del tipo de los elementos. Es importante notar que la sintaxis de

una DTD no sigue las reglas de sintaxis de un documento XML. Dicho más simple: Una DTD no es un documento XML. Otro detalle importante es que la DTD es un estándar de la W3C.

Ejemplo: DTD

```
<!ELEMENT Articulo (Cabecera, Contenido, Final)>
<!ELEMENT Cabecera (Titulo, Autor)>
<!ELEMENT Titulo ANY>
<!ELEMENT Autor ANY>
<!ELEMENT Contenido (Extracto, Cuerpo)>
.................
```

Este aspecto débil en las DTDs propició la aparición de otras alternativas de solución al problema de la estructuración y validación de documentos XML. De estas alternativas, es interesante revisar el caso de RELAX NG. RELAX NG es el resultado de la fusión de otras dos iniciativas que decidieron aunar esfuerzos: TREX y RELAX.

A diferencia de las DTDs, un documento RELAX NG es un documento XML utilizado para especificar a otros documentos XML. La manera que RELAX NG usa para enfrentar el problema de la especificación de un documento XML es la siguiente: un documento XML es la representación de un elemento, pero a su vez, un elemento está formado por una serie de "partes", como son: un nombre, un contexto, un conjunto de atributos y una secuencia ordenada de cero o más hijos, etc. Luego se procede de esta forma con cada una de estas partes.

Es importante destacar que RELAX NG -y sus precursores RELAX y TREX- es un intento de simplificar y potenciar la utilidad de las DTDs. También es importante indicar que esta iniciativa no es oficial de W3C, y aunque goza de cierta reputación, aún no está reconocido por el estándar, ni siquiera como recomendación. El punto más fuerte de RELAX NG respecto a las DTDs y las otras propuestas de solución, es la mayor simplicidad en manejo y aprendizaje que requiere. Como consecuencia de esta simplificación RELAX NG se queda únicamente con el uso de elementos y ciertas etiquetas especiales, como mecanismos para marcar el número requerido de apariciones, los tipos de dato de un elemento, etc.

Ejemplo: RELAX NG

```
<grammar>
<start>
      <element name="Cabecera">
             <ref name="Cabecera.class">
      </element>
      <zeroOrMore>
             <element name="Articulo">
                    <ref name="Articulo.class">
             </element>
      </zeroOrMore>
</start>
<define name="Cabecera.class">
```

```
<element name="Titulo">
.................
```

3.1.2 ¿Por qué XML Schema?

Es natural preguntarse por qué crear un nuevo estándar para realizar la misma función que las DTDs. La pregunta es aún más justificada si se tiene en cuenta que ambas enfrentan el mismo problema, con la misma aproximación: describir la estructura de la información. En los comienzos se usaban los mecanismos que proporcionaba SGML para modelar la información contenida en documentos XML. Aunque no era lo más apropiado, no había más alternativas. El descubrimiento de nuevas aplicaciones de XML, además de su utilización en la estructuración de la información contenida en documentos, obligó a trabajar en alternativas que resolvieran mejor los antiguos asuntos, como también los problemas que planteaban los nuevos usos.

Las DTDs fueron el primer estándar para estructurar documentos SGML. Como toda primera solución aceptable, gana rápidamente una profunda adhesión. Teniendo en cuenta que son las limitaciones de las DTDs las que propician la aparición de XML Schema, pareciera natural desarrollar este capítulo desde allí.

3.1.2.1 De las DTDs al XML Schema

Las DTDs poseen una sintaxis propia, distinta a la de un documento XML. Esto genera problemas, al menos, en dos instancias:

- *En el aprendizaje*: es posible confundir las sintaxis.
- *En el procesamiento del documento:* las herramientas y parsers (o analizadores sintáticos) que se empleen para tratar los documentos de XML deben ser capaces de procesar las DTDs, es decir, deben poder tratar con dos tipos de sintaxis distintas.

Otro problema que presentan las DTDs es que no permiten el uso de *namespaces*. Éstos permiten definir elementos con igual nombre dentro del mismo contexto, siempre y cuando se anteponga un prefijo al nombre del elemento. Las DTDs son muy limitadas a la hora de especificar los tipos de datos de los documentos. Por ejemplo, no permiten definir el tipo de dato de un elemento, por ejemplo, de un tipo numérico o de un tipo de fecha, sólo presenta variaciones limitadas sobre cadenas de caracteres (o *strings*).

El mecanismo de extensión que proveen las DTDs es difícil de comprender y mantener. Está basado en sustituciones sobre cadenas de caracteres. El punto más débil de este mecanismo es que no es suficientemente explícito en la estructuración de las relaciones. Por ejemplo, dos elementos que tienen definido el mismo modelo de contenido, no presentan ninguna relación. Estos aspectos negativos son solucionados en la especificación de XML Schema, permitiendo un lenguaje más expresivo y un intercambio de información más robusto. Además de resolver los problemas anteriormente tratados, XML Schema permite una serie de importantes ventajas adicionales:

- Posee una estructura de tipos mucho más rica. En la segunda parte de la especificación de XML Schema[9] se definen los tipos base que se pueden emplear dentro de esquema de XML, por ejemplo, *integer*, *boolean*, *string*, *date*, etc.

- Permite tipos de datos definidos por el usuario, llamados Arquetipos (*Archetypes*).

- Facilita la estructuración de relaciones permitiendo la agrupación de atributos. Lo anterior posibilita usar el mismo grupo de atributos a varios elementos distintos.

- El uso de *namespaces*[10] está especificado. Teniendo en cuenta la dificultad que implica su utilización, permite la validación de documentos con más de un *namespace*.

Sin embargo, el salto cualitativo de XML Schema sobre las DTDs, lo constituye la posibilidad de extender Arquetipos. De esta forma es posible construir un Schema que extienda a otro, es decir, que permita refinar la especificación de algún elemento, por ejemplo, pero dejando el resto del Schema original tal como estaba. Esta forma de extensión es análoga al concepto de "herencia" de la programación orientada a objetos.

Ejemplo: XML Schema

```
<schema targetNamespace="http://www.in3.cl/Schema_y_DTDs"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:IN3="http://www.in3.cl/Schema_y_DTDs" >
<element name="Articulo" type="IN3:tArticulo" />
<complexType name="tArticulo">
<element name="Cabecera" type="IN3:tCabecera"/>
<element name="Cuerpo" type="IN3:tCuerpo"/>
<element name="Final" type="IN3:tFinal"/>f
      </complezType>
      <complexType name="tCabecera">
      <element name="Titulo" type="string"/>
      <element name="Autor" type="string"/>
      </complexType>
       ..............
```

3.2 Construcción de XML Schemas

3.2.1 El elemento <schema>

El elemento **<schema>** es el elemento raíz de cualquier documento XML Schema.

Ejemplo: El elemento <schema>

[9] XML Schema Part 2: Datatypes http://www.w3.org/TR/xmlschema-2/
[10] XML Schema Part 0: Primer http://www.w3.org/TR/xmlschema-0/#NS

```
<?xml version="1.0"?>
<xs:schema>
...
...
</xs:schema>
```

El elemento **<schema>** puede contener algunos atributos. Una declaración típica es similar a esta:

Ejemplo: Declaración típica del elemento <schema>

```
<?xml version="1.0"?>
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"  targetNamespace="http://www.in3.cl"
xmlns="http://www.in3.cl" elementFormDefault="qualified">
...
...
</xs:schema>
```

Es necesario detenerse un momento en los fragmentos de la declaración. El siguiente fragmento indica que los elementos y tipos de datos usados en el Schema vienen del *namespace* "**http://www.w3.org/2001/XMLSchema**". También especifica que estos elementos deben usar el prefijo "**xs:**".

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

El fragmento que sigue indica que los elementos que van a definirse en el interior del Schema vienen del *namespace* "**http://www.in3.cl**".

```
targetNamespace="http://www.in3.cl"
```

El próximo fragmento de código indica que el *namespace* por defecto es "**http://www.in3.cl**".

```
xmlns="http://www.in3.cl"
```

El próximo fragmento indica que cualquier elemento usado por el documento XML instanciado, que fuese declarado en este Schema, debe ser cualificado por un *namespace*.

```
elementFormDefault="qualified"
```

3.2.2 Referenciando un Schema en un documento XML

En esta sección se revisará la forma típica en que se referencia un XML Schema desde un documento XML. Para esto se utilizará el siguiente documento XML, el cual referencia a un determinado XML Schema.

Ejemplo: Documento XML que referencia un XML Schema

```
<?xml version="1.0"?>
<nota xmlns="http://www.in3.cl"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.in3.cl nota.xsd">
      <para>Juan</para>
      <de>Luisa</de>
      <encabezado>Recordatorio</encabezado>
      <mensaje>No te olvides de estudiar!</mensaje>
</nota>
```

Al igual que en la sección anterior, es necesario revisar este ejemplo por partes. El fragmento siguiente especifica la declaración del espacio de nombres (*namespace*) por defecto. Esta declaración le indica al validador de XML Schema que todos los elementos usados en este documento XML están declarados en el *namespace* "**`http://www.in3.cl`**".

```
xmlns="http://www.in3.cl"
```

Una vez que se tenga disponible el *namespace* de la instancia del XML Schema es posible usar el atributo *schemaLocation*.

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Este atributo tiene dos valores. El primer valor corresponde al *namespace* que vaya a utilizarse. El segundo valor es la ubicación del XML Schema que se usará en ese *namespace*, como se muestra a continuación:

```
xsi:schemaLocation="http://www.in3.cl nota.xsd"
```

3.2.3 Elementos Simples

Un elemento simple es un elemento XML que sólo puede contener texto. No puede contener ningún otro elemento o atributo. Se dice que esta clase de elementos es de "tipo Simple" (*simpleType*). Sin embargo la restricción "solo puede contener texto" puede llevar a confusión. El texto puede ser de muchos tipos diferentes. Puede ser algunos de los tipos incluidos en la definición de XML Schema (tipos de datos estándar o *built-in*), o puede ser uno de tipo

personalizado, definido por el usuario. También se pueden añadir restricciones a los tipos de datos a fin de limitar su contenido. Además se puede requerir que los datos calcen con un patrón definido. La sintaxis de un elemento simple es:

```
<xs:element name="xxx" type="yyy"/>
```

donde “xxx” es el nombre del elemento e “yyy” es el tipo de dato del elemento.

Ejemplo: Elementos Simples

Elementos XML

```
<apellido>Perez</apellido>
<edad>34</edad>
<fechaDeNacimiento>1968-03-27</fechaDeNacimiento>
```

Definiciones de los Elementos Simples en un XML Schema

```
<xs:element name="apellido" type="xs:string"/>
<xs:element name="edad" type="xs:integer"/>
<xs:element name="fechaDeNacimiento" type="xs:date"/>
```

3.2.3.1 Declarando valores fijos y por defecto para Elementos Simples

Los elementos simples pueden tener un valor fijo o por defecto. Se asignará el valor por defecto automáticamente cuando no se le especifica ningún valor. En el ejemplo siguiente el valor por defecto es “rojo”.

Ejemplo: Elemento Simple con Valor por Defecto

```
<xs:element name="color" type="xs:string" default="rojo"/>
```

Por otro lado, un valor fijo es automáticamente asignado al elemento. No es posible especificar otro valor. En el ejemplo siguiente el valor fijo es “azul”.

Ejemplo: Elemento Simple con Valor Fijo

```
<xs:element name="color" type="xs:string" fixed="azul"/>
```

3.2.4 Tipos de datos estándares

Algo se ha adelantado ya, en el transcurso de este documento, sobre la existencia de tipos de datos predefinidos. A éstos se les conocen como "tipos de datos estándar" o "*built-in"*. XML Schema dispone de varios de estos tipos de datos. A continuación se indicarán los más comunes:

- xs:string: Usado para cadenas de caracteres, típicamente alfanuméricas.
- xs:decimal: Para datos numéricos con precisión decimal.
- xs:integer: Para datos numéricos enteros.
- xs:bolean: Para valores de verdad ("*true*" o "*false*").
- xs:date: Para fechas.
- xs:time: Para valores de tiempo.

Ejemplo: Declaración de un Elemento, usando un tipo de dato estándar (*built-in*)

```
<xs:element name="numero" type="xs:Integer"/>
```

3.2.5 Restricciones (Facets)

Es posible definir más detalladamente un elemento, sin salir de la categoría de Tipo Simple (*simpleType*). En esta sección veremos las restricciones disponibles en XML Schema.

3.2.5.1 Restricciones en Valores

El siguiente ejemplo define un elemento llamado "edad" con una restricción: El valor de la edad no puede ser menor que 0 o mayo a 150.

Ejemplo: Restricción de Valores

```
<xs:element name="edad">
       <xs:simpleType>
             <xs:restriction base="xs:integer">
                 <xs:minInclusive value="0"/>
                 <xs:maxInclusive value="150"/>
             </xs:restriction>
       </xs:simpleType>
</xs:element>
```

3.2.5.2 Restricciones en un Conjunto de Valores

Para limitar el contenido de un elemento XML a un conjunto de valores aceptables, se utiliza la restricción de enumeración. El siguiente ejemplo define un elemento llamado "sangre".

Ejemplo: Restricción a un Conjunto de Valores

```
<xs:element name="sangre">
        <xs:simpleType>
              <xs:restriction base="xs:string">
                      <xs:enumeration value="A"/>
                      <xs:enumeration value="B"/>
                      <xs:enumeration value="AB"/>
                      <xs:enumeration value="O"/>
              </xs:restriction>
        </xs:simpleType>
</xs:element>
```

El elemento sangre es un tipo simple con una restricción. Los valores aceptables son: “A”, “B”, “AB”, “O”. El ejemplo anterior pudo haberse escrito también como sigue:

Ejemplo: Restricción a un Conjunto de Valores

```
<xs:element name="sangre" type="tipoSangre"/>
        <xs:simpleType name="tipoSangre">
               <xs:restriction base="xs:string">
                     <xs:enumeration value="A"/>
                     <xs:enumeration value="B"/>
                     <xs:enumeration value="AB"/>
                     <xs:enumeration value="O"/>
               </xs:restriction>
        </xs:simpleType>
</xs:element>
```

Note que en el caso anterior, “tipoSangre” puede ser usado por otros elementos, pues no es parte del elemento “sangre”.

3.2.5.3 Restricción en Series de Valores

Tanto para limitar el contenido de un elemento XML como para definir una serie de números o letras que pueden ser usados, se usa una restricción de patrón. En el siguiente ejemplo se define un elemento llamado “letra”.

Ejemplo: Restricción a una Serie de Valores

```
<xs:element name="letra">
      <xs:simpleType>
          <xs:restriction base="xs:string">
               <xs:pattern value="[a-z]"/>
          </xs:restriction>
      </xs:simpleType>
</xs:element>
```

El elemento “letra” es un tipo simple con una restricción: El único valor aceptable es una de las letras, desde la “a” a la “z”, en minúscula. La sintaxis de los patrones es la típica de expresiones regulares, similar a la usada en el lenguaje de programación Perl[11]. Para más detalles se aconseja ver la recomendación de la W3C sobre este tópico particular[12].

3.2.5.4 Restricciones en los caracteres de espacio en blanco

Existe una restricción para especificar como deben ser manejados los caracteres de espacio en blanco. En el ejemplo siguiente define un elemento llamado dirección.

Ejemplo: Restricción relativa a espacios en blanco

```
<xs:element name="direccion">
     <xs:simpleType>
            <xs:restriction base="xs:string">
                 <xs:whiteSpace value="preserve"/>
            </xs:restriction>
     </xs:simpleType>
</xs:element>
```

El elemento “dirección” es una tipo simple con una restricción. La restricción **`whiteSpace`** se inicializa con el valor **`preserve`**, el cual significa que el procesador XML no removerá ningún carácter de espacio en blanco. Si esta restricción se inicializa con **`replace`**, el procesador XML reemplazará cualquier carácter de espacio en blanco (salto de línea, tabuladores, espacios y retornos de carro) por espacios en blanco simples. Por último, si esta restricción se inicializa con **`collapse`**, el procesador XML reemplazará cualquier carácter de espacio en blanco (salto de línea, tabuladores, espacios y retornos de carro) múltiple o simple, por espacios simples, además de remover todos los caracteres de espacios en blanco, al principio y al final del texto.

3.2.5.5 Restricciones de Longitud

Para limitar la longitud de un valor en un elemento, se usan las restricciones **`length`**, **`maxLength`**, y **`minLength`**. El siguiente ejemplo define un elemento llamado “contraseña”.

Ejemplo: Restricción a la Longitud de los Valores de un Elemento

```
<xs:element name="contraseña">
       <xs:simpleType>
             <xs:restriction base="xs:string">
                    <xs:length value="8"/>
             </xs:restriction>
       </xs:simpleType>
</xs:element>
```

[11] Lenguaje de programación Perl http://www.perl.org/.
[12] Apéndice de la recomendación de XML Schema de la W3C sobre expresiones regulares http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/#regexAppendix.

El elemento “contraseña” es un elemento simple con una restricción. El valor debe ser de exactamente 8 caracteres de largo.

Ejemplo: Restricción a la Longitud de los Valores de un Elemento

```
<xs:element name="contraseña">
     <xs:simpleType>
           <xs:restriction base="xs:string">
                 <xs:minLength value="5"/>
                 <xs:maxLength value="8"/>
           </xs:restriction>
     </xs:simpleType>
</xs:element>
```

En el caso anterior, el elemento contraseña debe de ser de un mínimo de 5 y un máximo de 8 caracteres de largo.

3.2.5.6 Restricciones aplicables a Tipos de Datos

En la tabla siguiente se condensan las restricciones que son aplicables a los tipos de datos.

Restricción	Descripción
Enumeration	Define una lista de valores aceptables.
fractionDigits	Especifica la cantidad máxima de cifras decimales permitidas en un número. Debe ser igual o más grande que cero.
Length	Especifica la cantidad exacta de caracteres o ítems de una lista, permitidos. Debe ser igual o más grande que cero.
maxExclusive	Especifica el límite superior para valores numéricos. El dato debe ser menor que este valor.
maxInclusive	Especifica el límite superior para valores numéricos. El dato debe ser menor o igual que este valor.
maxLength	Especifica la cantidad máxima de caracteres o ítems de una lista, permitidos. Debe ser igual o más grande que cero.
minExclusive	Especifica el límite inferior para valores numéricos. El dato debe ser mayor que este valor.
minInclusive	Especifica el límite inferior para valores numéricos. El dato debe ser mayor o igual que este valor.
minLength	Especifica la cantidad mínima de caracteres o ítems de una lista, permitidos. Debe ser igual o más grande que cero.
Pattern	Define la secuencia exacta de caracteres que son aceptables.

totalDigits	Especifica la cantidad exacta de dígitos permitidos. Debe ser más grande que cero.
whiteSpace	Especifica como deben se manejados los espacios (salto de línea, tabuladores, espacios, retorno de carro).

3.2.6 List y Union

<list> y **<union>** son dos elementos simples derivados de tipos simples. Debido a la utilidad que presentan, se han incluido en este texto, junto a un breve ejemplo para cada una.

3.2.6.1 List

El elemento **<list>** define un elemento de tipo simple, como una lista de valores del tipo de dato especificado. En el documento XML correspondiente, los elementos de la lista se separan con un espacio en blanco. El ejemplo siguiente, muestra un tipo simple que es una lista de números enteros y al correspondiente elemento "valoresEnteros", en un hipotético documento XML.

Ejemplo: List

XML Schema

```
<?xml version="1.0"?>
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
            <xs:element name="valoresEnteros" type="listaDeValores">
                 <xs:simpleType name="listaDeValores">
                       <xs:list itemType="xs:integer"/>
                 </xs:simpleType>
             </xs:element>
      </xs:schema>
```

Documento XML

```
<valoresEnteros>100 34 56 -23 1567</valoresEnteros>
```

3.2.6.2 Union

El elemento <union> define un tipo simple como una colección (unión) de valores, de los tipos de datos simples especificados. En el ejemplo siguiente se muestra a un tipo simple que es una unión de otros dos tipos simples.

Ejemplo: Union

```
<xs:element name="talla">
   <xs:simpleType>
       <xs:union memberTypes="tallaPorNumero tallaPorPalabra" />
```

```
   </xs:simpleType>
</xs:element>
<xs:simpleType name="tallaPorNumero">
   <xs:restriction base="xs:positiveInteger">
          <xs:maxInclusive value="58"/>
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name="tallaPorPalabra">
   <xs:restriction base="xs:string">
         <xs:enumeration value="small"/>
         <xs:enumeration value="medium"/>
         <xs:enumeration value="large"/>
   </xs:restriction>
</xs:simpleType>
```

3.2.7 Atributos

Como se vio en la sección anterior, los Elementos Simples no pueden tener atributos. Si un elemento tiene atributos, es considerado de tipo Complejo (*complexType*). Sin embargo, el atributo mismo es siempre declarado como un Tipo Simple (*simpleType*). La sintaxis para definir un atributo es la siguiente:

```
<xs:attribute name="xxx" type="yyy"/>
```

donde “xxx” es el nombre del atributo e “yyy” es el tipo de dato del atributo.

Ejemplo: Atributos

Elemento XML con atributo

```
<apellido idioma="ES">Perez</apellido>
```

Definición de los elementos con atributo en un XML Schema

```
<xs:attribute name="idioma" type="xs:string"/>
```

3.2.7.1 Declarando valores fijos y por defecto para Atributos

Los atributos, así como los elementos, pueden tener un valor fijo o por defecto. Se asignará el valor por defecto automáticamente cuando no se le especifica ningún valor. En el ejemplo siguiente el valor por defecto es “ES”.

Ejemplo: Atributo con valor por defecto

```
<xs:attribute name="idioma" type="xs:string" default="ES"/>
```

Por otro lado, un valor fijo es automáticamente asignado al elemento. No es posible especificar otro valor. En el ejemplo siguiente el valor fijo es "FR".

Ejemplo: Atributo con valor fijo

```
<xs:attribute name="idioma" type="xs:string" fixed="FR"/>
```

3.2.7.2 Creando atributos opcionales y requeridos

Todos los atributos son opcionales por defecto. Para especificar explícitamente que un atributo es opcional se utiliza el atributo "use".

Ejemplo: Atributo opcional

```
<xs:attribute name="idioma" type="xs:string" use="optional"/>
```

Por otro lado, para hacer requerido a un determinado atributo, se debe modificar como sigue:

Ejemplo: Atributo opcional

```
<xs:attribute name="idioma" type="xs:string" use="required"/>
```

3.2.7.3 Restricciones al contenido

Cuando un elemento o atributo XML tiene un tipo definido, pone una restricción en su contenido. Si un elemento XML es de tipo "xs:date" y contiene un *string* del tipo "Hola mami", el elemento no pasará la validación. Con XML Schema, también se pueden añadir restricciones personalizadas a los elementos y atributos de los documentos XML. Se verá un poco más de ésto, más adelante.

3.2.8 Elementos Complejos

Un Elemento Complejo es un elemento XML que contiene otros elementos (simples o complejos) y/o atributos. Hay cuatro tipos de elementos complejos: (a) elementos vacíos, (b) elementos que contienen otros elementos solamente, (c) elementos que contienen sólo texto y (d) elementos que contienen otros elementos y texto. Por supuesto, cada uno de los anteriores, puede o no, contener atributos. A continuación se mostrarán ejemplos de cada uno de estos tipos.

Ejemplo: Un elemento XML complejo, llamado “producto”, el cual es vacío

```
<producto pid="1345"/>
```

Ejemplo: Un elemento XML complejo, llamado “funcionario”, que contiene otros elementos

```
<funcionario>
     <nombre>Juan</nombre>
     <apellido>Perez</apellido>
</funcionario>
```

Ejemplo: Un elemento XML complejo, llamado “comida”, que contiene solo texto

```
<comida type="postre">Helado</comida>
```

Ejemplo: Un elemento XML complejo, llamado “descripcion”, que contiene otros elementos y texto

```
<descripcion>
    Sucedió un día <fecha idioma="español">25-03-1999</fecha> ....
</descripcion>
```

3.2.8.1 Cómo definir un Elemento Complejo

Hay varias maneras de definir un elemento complejo en un XML Schema. Para eso se utilizarán los ejemplos que siguen.

Ejemplo: Un elemento XML complejo, llamado “funcionario”, que contiene otros elementos

```
<funcionario>
     <nombre>Juan</nombre>
     <apellido>Perez</apellido>
</funcionario>
```

Esta es la primera forma que se revisará. En este caso el elemento “funcionario” puede ser declarado directamente, de la siguiente manera:

```
<xs:element name="funcionario">
     <xs:complexType>
          <xs:sequence>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="apellido" type="xs:string"/>
```

```
            </xs:sequence>
      </xs:complexType>
</xs:element>
```

Si se usa el método descrito arriba, sólo el elemento "funcionario" puede usar el tipo complejo recién especificado. Note que los elementos hijos (*childs*) "nombre" y "apellido", están contenidos en un *tag* (o etiqueta) **<xs:sequence>**. Esto significa que estos elementos hijos deben aparecer en el mismo orden en que han sido declarados: primero "nombre" y después "apellido". El *tag* **<xs:sequence>** pertenece a una clase de *tags* llamada Indicadores (*Indicators*). Veremos más de esta clase de *tags*, más adelante.

Otra forma posible de definir un elemento complejo se muestra a continuación. Ahora, el elemento "funcionario" puede tener un atributo "tipo", que refiera al nombre del tipo complejo (*complexType*) a ser utilizado.

```
<xs:element name="funcionario" type="infoPersona"/>
<xs:complexType name="infoPersona">
     <xs:sequence>
          <xs:element name="nombre" type="xs:string"/>
          <xs:element name="apellido" type="xs:string"/>
     </xs:sequence>
</xs:complexType>
```

Si se utiliza esta última forma, varios elementos pueden referir al mismo tipo complejo, de la siguiente manera:

```
<xs:element name="funcionario" type="infoPersona"/>
<xs:element name="estudiante" type="infoPersona"/>
<xs:element name="socio" type="infoPersona"/>
    <xs:complexType name="infoPersona">
         <xs:sequence>
              <xs:element name="nombre" type="xs:string"/>
              <xs:element name="apellido" type="xs:string"/>
         </xs:sequence>
    </xs:complexType>
```

Otra alternativa es basar un tipo complejo en otro tipo complejo existente, y añadir algunos elementos, de la siguiente forma:

```
<xs:element name="funcionario" type="infoPersonaFull"/>
<xs:complexType name="infoPersona">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellido" type="xs:string"/>
  </xs:sequence>
```

```
</xs:complexType>
<xs:complexType name="infoPersonaFull">
  <xs:complexContent>
    <xs:extension base="infoPersona">
      <xs:sequence>
        <xs:element name="direccion" type="xs:string"/>
        <xs:element name="ciudad" type="xs:string"/>
        <xs:element name="region" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

3.2.9 Indicadores

Existen siete tipos de indicadores: indicadores de orden (**`all`**, **`choice`**, **`sequence`**), indicadores de Ocurrencia (**`maxOccurs`**, **`minOccurs`**) e indicadores de grupo (**`group`**, **`attributeGroup`**).

3.2.9.1 Indicadores de Orden

Los Indicadores de Orden son usados para definir como deben aparecer los elementos. El indicador **`<all>`** especifica, por defecto, cuáles son los elementos hijos que deben aparecer, no imponiendo ninguna restricción sobre el orden en el cual deben aparecer, pero limitando su aparición a una vez cada uno.

Ejemplo: Indicador `<all>`

```
<xs:element name="persona">
  <xs:complexType>
     <xs:all>
          <xs:element name="nombre" type="xs:string"/>
          <xs:element name="apellido" type="xs:string"/>
     </xs:all>
   </xs:complexType>
</xs:element>
```

El indicador **`<choice>`** especifica que sólo uno de los elementos puede aparecer.

Ejemplo: Indicador `<choice>`

```
<xs:element name="persona">
    <xs:complexType>
        <xs:choice>
             <xs:element name="funcionario" type="funcionario"/>
             <xs:element name="socio" type="socio"/>
        </xs:choice>
   </xs:complexType>
</xs:element>
```

El indicador **<sequence>** especifica que los elementos hijos deben aparecer en un orden determinado.

Ejemplo: Indicador <sequence>

```
<xs:element name="persona">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="apellido" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

3.2.9.2 Indicadores de Ocurrencia

Los Indicadores de Ocurrencia son usados para definir cuan seguido un elemento puede aparecer. En particular, el indicador **<maxOccurs>** especifica la cantidad de veces máxima que un elemento puede aparecer.

Ejemplo: Indicador <maxOccurs>

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombreCompleto" type="xs:string"/>
      <xs:element name="nombreHijos" type="xs:string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

El ejemplo de arriba indica que el elemento “nombreHijos” puede aparecer un mínimo de una vez (el valor por defecto de **<maxOccurs>** es uno) y un máximo de diez veces, dentro de un elemento “persona”. Por otro lado, el indicador **<minOccurs>** especifica el número mínimo de veces que un elemento puede aparecer.

Ejemplo: Indicador <minOccurs>

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombreCompleto" type="xs:string"/>
      <xs:element name="nombreHijos" type="xs:string" maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

El ejemplo de arriba indica que el elemento “nombreHijos” puede aparecer un mínimo de cero veces y un máximo de diez veces, dentro de un elemento “persona”. Para permitir que un elemento aparezca un número ilimitado de veces, en **<maxOccurs>** debe asignarse el valor **“unbounded”.**

3.2.9.3 Indicadores de Grupo

Los Indicadores de Grupo son usados para definir conjuntos de elementos relacionados. Los grupos de elementos se definen a través de declaraciones de grupo, como la que muestra el ejemplo siguiente.

Ejemplo: Declaración de Indicador <group>

```
<xs:group name="groupname">
  ...
</xs:group>
```

Se debe definir un elemento **<all>**, **<choice>** o **<sequence>** dentro de la declaración de grupo. El siguiente ejemplo define un grupo llamado “grupoPersona”, que define un conjunto de elementos que debe ocurrir en una secuencia exacta.

Ejemplo: Indicador <group>

```
<xs:group name="grupoPersona">
    <xs:sequence>
         <xs:element name="nombre" type="xs:string"/>
         <xs:element name="apellido" type="xs:string"/>
         <xs:element name="fechaDeNacimiento" type="xs:date"/>
    </xs:sequence>
</xs:group>
```

Después de que se haya definido un grupo, es posible referenciarlo desde otro grupo o desde una definición de tipo complejo como ésta:

Ejemplo: Indicador <group> usado con referencia

```
<xs:group name="grupoPersona">
     <xs:sequence>
          <xs:element name="nombre" type="xs:string"/>
          <xs:element name="apellido" type="xs:string"/>
          <xs:element name="fechaDeNacimiento" type="xs:date"/>
     </xs:sequence>
</xs:group>
<xs:element name="person" type="infoPersona"/>
<xs:complexType name="infoPersona">
```

```
   <xs:sequence>
      <xs:group ref="grupoPersona"/>
      <xs:element name="pais" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Por otro lado, los grupos de atributos son definidos con la declaración **<attributeGroup>**, de la siguiente forma:

Ejemplo: Declaración de Indicador <attributeGroup>

```
<xs:attributeGroup name="nombreGrupo">
  ...
</xs:attributeGroup>
```

El siguiente ejemplo define un grupo de atributos llamado "grupoAtributoPersona".

Ejemplo: Indicador <attributeGroup>

```
<xs:attributeGroup name="grupoAtributoPersona">
  <xs:attribute name="nombre" type="xs:string"/>
  <xs:attribute name="apellido" type="xs:string"/>
  <xs:attribute name="fechaDeNacimiento" type="xs:date"/>
</xs:attributeGroup>
```

Después que se ha definido un grupo de atributos, es posible referenciarlo desde otro grupo o desde una definición de tipo complejo, como muestra el ejemplo siguiente.

Ejemplo: Indicador <maxOccurs>

```
<xs:attributeGroup name="grupoAtributoPersona">
  <xs:attribute name="nombre" type="xs:string"/>
  <xs:attribute name="apellido" type="xs:string"/>
  <xs:attribute name="fechaDeNacimiento" type="xs:date"/>
</xs:attributeGroup>
<xs:element name="persona">
  <xs:complexType>
    <xs:attributeGroup ref="grupoAtributoPersona"/>
  </xs:complexType>
</xs:element>
```

Note que para todos los indicadores de Orden y de Grupo (**<all>**, **<choice>**, **<sequence>** y **<group>**) el valor por defecto de **<maxOccurs>** y **<minOccurs>** es 1.

3.3 Recapitulación

En resumen XML Schema es el sucesor más robusto de los DTD, pues es más potente y versátil. Éste se escribe en XML, por lo que pueden usarse las mismas herramientas que sirven para XML, como analizadores sintácticos (*parsers*), DOM, XSLT, etc. Define los elementos y atributos que pueden aparecer en un determinado documento XML, junto con su jerarquía, orden y cardinalidad. Además define el tipo de datos de los elementos, si pueden ser vacíos o incluir texto, así como sus valores fijos o por defecto.

XML Schema es extensible a futuras adiciones. Por ejemplo reutilizar el Schema en otros Schemas, crear tipos de datos personalizados, a partir de los tipos de datos estándar, referenciar múltiples Schemas desde el mismo documento, etc. Soporta *namespaces*, y en un intercambio de información, asegura que las dos partes tengan las mismas expectativas acerca de la información. Además, permite validar el contenido, más allá de un documento bien formado y es una recomendación de la W3C.

Ejercicio

Realice un documento XML Schema, que norme la construcción de documentos XML que correspondan a la ficha de nacimiento de una persona en un hospital. La ficha debe contener nombres y apellidos del recién nacido, nombres, apellidos y nacionalidad de los padres, fecha de nacimiento, peso, estatura, tipo de sangre y factor RH, equipo médico que participó en el parto, nombre del hospital, comuna y ciudad donde se encuentra (puede llamar las etiquetas como quiera). Luego valide el documento XML construido en el capítulo II, contra su recién construido XML Schema.

3.4 Referencias

- XML Schema Part 0: Primer (Second Edition), W3C Recommendation 28 October 2004
  http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/
- XML Schema 2001 and Relax NG Tutorial
  http://www.zvon.org/xxl/XMLSchemaTutorial/Output/index.html
- XML Schema Tutorial
  http://www.w3schools.com/schema/schema_intro.asp
- ¿Qué es XML Schema?
  http://mat21.etsii.upm.es/mbs/MechXML/que_es_xml_schema.htm

CAPITULO IV: XPATH

Alex Bórquez, Claudio Gutiérrez

Departamento de Ciencias de la Computación, Universidad de Chile.
{aborquez, cgutierr}@dcc.uchile.cl

4.1 Introducción a XPath

El significado de un elemento XML puede depender del contexto; por ejemplo:

```
<libro>
     <titulo>…</titulo>
</libro>
```

```
<profesional>
     <titulo>…</titulo>
</profesional>
```

¿Qué se desea buscar, el título del libro o del profesional? Una solución sería identificar cada elemento con un atributo único; por ejemplo:

```
<libro id='1'>
     <titulo id='2'>…</titulo>
</libro>
```

```
<profesional id='3'>
     <titulo id='4'>…</titulo>
</profesional>
```

El problema es que los identificadores únicos no son suficientes, pues:

- No todos los elementos pueden tener clave.
- Aún existiendo, no siempre se conocerá la clave, por lo que seguirá siendo difícil acceder al elemento buscado.
- Los identificadores no pueden manejar rangos de texto (patrones o expresiones regulares), para facilitar la búsqueda.
- Puede resultar pesado listar una gran cantidad de objetos a través de sus identificadores

XPath explota el contexto secuencial y jerárquico de XML para especificar elementos por su contexto. Por lo tanto los elementos título serian fácilmente distinguibles entre sí, debido al lugar que ocupa cada uno en la jerarquía de sus respectivos documentos. XPath es el resultado de un esfuerzo para proporcionar una sintaxis y semántica comunes, para funcionalidades compartidas, entre XSL Transformations y XPointer. El objetivo principal de XPath es direccionar partes de un documento XML.

XSL (Extensible Stylesheet Language, expresión en inglés traducible como "lenguaje extensible de hojas de estilo") es una familia de lenguajes basados en el estándar XML, que permiten describir cómo la información contenida en un documento XML cualquiera, debe ser transformada o formateada para su presentación en un medio específico. Esta familia está formada por dos lenguajes: XSLT (siglas de Extensible Stylesheet Language Transformations, lenguaje de hojas extensibles de transformación), que permite convertir documentos XML de una sintaxis a otra (por ejemplo, de un XML a otro XML o a un documento HTML); y XSL-FO (lenguaje de hojas extensibles de formateo de objetos), que permite especificar el formato visual con el cual se quiere presentar un documento XML, es usado principalmente para generar documentos PDF. Estas dos especificaciones son recomendaciones oficiales del W3C. En el 2005 ya son soportadas por algunos navegadores, por ejemplo Mozilla o Internet Explorer.

XPointer o Lenguaje de punteros XML, es un estándar del W3C, que proporciona una manera de identificar de forma única, fragmentos de un documento XML con el objeto de realizar vínculos. La especificación XPointer ofrece un mecanismo para direccionamiento de documentos XML en función de su estructura interna, lo que permite examinar su estructura jerárquica al mismo tiempo que se seleccionan sus partes internas, como elementos, valores de atributos, contenido de caracteres y posiciones relativas. La extensión XPointer permite a XPath el seleccionar puntos, intervalos y nodos; utilizar coincidencias de cadenas para buscar información; y utilizar expresiones de direccionamiento en referencias de URI como identificadores de fragmentos, entre otros.

Como soporte para este objetivo principal, también proporciona facilidades básicas para manipulación de cadenas de caracteres, números y booleanos. XPath obtiene su denominación por el uso que hace de una notación de caminos (*paths*), como en los URLs, para navegar a través de la estructura jerárquica de un documento XML.

URL es la sigla en inglés para Uniform Resource Locutor (localizador uniforme de recurso). Una URL es la cadena de caracteres con la cual se asigna una dirección única a cada uno de los recursos de información disponibles en Internet. Existe una URL único para cada página de cada uno de los documentos de la World Wide Web, para todos los elementos del Gopher y todos los grupos de debate USENET, y así sucesivamente. La URL de un recurso de información, es su dirección en Internet, la que permite que el navegador la encuentre y la muestre de forma adecuada. Por ello el URL combina el nombre del ordenador que proporciona la información, el directorio donde se encuentra, el nombre del fichero y el protocolo a usar para recuperar los datos. Además reemplaza la dirección numérica o IP de los servidores, haciendo de esta manera más fácil la navegación; De otra forma se tendría que hacer bajo direcciones del tipo http://148.210.01.7 en vez de http://www.pagina.com. Dado que el concepto de URI engloba al de URL, éste último se encuentra cada vez más en desuso. La especificación detallada se encuentra en la RFC 1738, titulada Uniform Resource Locators.

XPath utiliza una sintaxis compacta y no-XML para facilitar su uso dentro de URI y de valores de atributos XML.

URI es la sigla en inglés para Uniform Resource Identifier (identificador uniforme de recursos). Texto corto que identifica unívocamente cualquier recurso (servicio, página, documento, dirección de correo electrónico, etc.) accesible en una red. Normalmente una URI consta de dos partes:

Identificador del método de acceso (protocolo) al recurso, por ejemplo http:, mailto:, ftp:
Nombre del recurso, por ejemplo "//www.dcc.uchile.cl"

La especificación detallada se encuentra en la RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax.

Además de su uso para direccionar, XPath puede usarse para *patern-matching* (comprobar si un nodo encaja con un patrón o no); este uso de XPath está descrito en XSLT.

4.2 El modelo de datos XPath

XPath opera sobre la estructura lógica abstracta de un documento XML, más que en su sintaxis superficial. XPath modela un documento XML como un árbol de nodos. Hay diferentes tipos de nodos, incluyendo nodos elemento, nodos atributo y nodos texto, entre otros. XPath define un modo de calcular un valor de cadena de caracteres para cada tipo de nodo. Algunos tipos de nodo también tienen nombres.

XPath es totalmente compatible con XMLNamespaces. Así, el nombre de un nodo se modela como un par consistente en una parte local y un (quizá nulo) URI de espacio de nombres; esto se llama un nombre expandido.

> Un espacio de nombres XML (**XML Namespace**) es una colección de nombres, identificados por una referencia URI, que son usados en documentos XML como tipos de elementos y nombres de atributos. Los "espacios de nombres XML" difieren del concepto "espacios de nombres" convencionalmente usado en las disciplinas de las ciencias de la computación, en que la versión de XML tiene una estructura interna y no es, en el sentido matemático, un conjunto. Un ejemplo que describe este concepto: Supongamos una aplicación XML donde un solo documento XML puede contener elementos y atributos (conocido como "vocabulario de marcado") que está definido para ser usado por múltiples módulos de software, una motivación para esta modularidad. Si existiera un vocabulario de marcado, el cual es bien entendido y para el cual hay software útil disponible, es mejor reusar el vocabulario de marcado, que reinventarlo. Tales documentos, conteniendo múltiples vocabularios de marcado, poseen problemas de ambigüedad y colisión. Los módulos de software necesitan poder reconocer las etiquetas y atributos que están diseñados para procesar, incluso enfrentando colisiones ocurrentes cuando la marcación fue ideada para que otro paquete de software usara el mismo tipo de elemento o nombre de atributo. Estas consideraciones requieren que los constructos de los documentos deban tener nombres universales, cuyo alcance se extienda más allá de su documento contenedor. Los "espacios de nombres XML" corresponden a la especificación que permite lograrlo.

4.2.1 Expresiones

La construcción sintáctica básica en XPath es la expresión. Las expresiones son evaluadas para producir un objeto, que tendrá uno de los siguientes cuatro tipos básicos:

- node-set (una colección desordenada de nodos sin duplicados)
- boolean (valores de verdad: verdadero o falso)
- number (un número en punto flotante)
- string (una cadena de caracteres)

La evaluación de expresiones tiene lugar respecto a un contexto. XSLT y XPointer especifican como se determina el contexto para las expresiones XPath usadas en XSLT y XPointer respectivamente. El contexto consiste en:

- un nodo (el nodo contextual)
- un par de enteros positivos no nulos (la posición contextual y el tamaño contextual)
- un conjunto de asignaciones de variables
- una biblioteca de funciones
- el conjunto de declaraciones de espacios de nombres aplicables a la expresión

La posición contextual es siempre menor o igual que el tamaño contextual. Las asignaciones de variables consisten en una correspondencia de nombres de variable a valores de variable. El valor de una variable es un objeto, que puede ser de cualquiera de los tipos posibles para el valor de una expresión, y puede también ser de tipos adicionales no definidos en la especificación de XPath.

Existe una biblioteca básica de funciones, que todas las implementaciones de XPath deben soportar. La biblioteca de funciones consiste en una correspondencia de nombres de funciones a funciones. Cada función toma cero o más argumentos y devuelve un único resultado. Para las funciones de la biblioteca básica, los argumentos y el resultado son de los cuatro tipos básicos. Tanto XSLT como XPointer extienden XPath mediante la definición de funciones adicionales; algunas de esas funciones operan sobre los cuatro tipos básicos; otras operan sobre tipos de datos adicionales definidos por XSLT y XPointer.

Las declaraciones de espacios de nombres consisten en una correspondencia de prefijos a URIs de espacios de nombres. Las asignaciones de variables, biblioteca de funciones y declaraciones de espacios de nombres utilizadas para evaluar una subexpresión son siempre las mismas que las que se emplean para evaluar la expresión que la contiene. El nodo contextual, la posición contextual y el tamaño contextual utilizados para evaluar una subexpresión son a veces diferentes de los que se emplean para evaluar la expresión que la contiene. Varios tipos de expresiones cambian el nodo contextual; sólo los predicados cambian la posición contextual y el tamaño contextual. Al describir la evaluación de un tipo de expresión, siempre se hace constar explícitamente si el nodo contextual, la posición contextual y el tamaño contextual cambian para la evaluación de subexpresiones; si nada se dice sobre el nodo contextual, la posición contextual y el tamaño contextual, permanecerán inalterados en la evaluación de sub-expresiones, de ese tipo de expresión.

Las expresiones XPath a menudo aparecen en atributos XML. La gramática especificada en esta sección se aplica al valor del atributo tras la normalización XML 1.0. Así, por ejemplo, si la gramática usa el caracter "<", éste no debe aparecer en el código fuente XML como "<" sino que debe ser tratado conforme a las reglas de XML 1.0 introduciéndolo, por ejemplo, como "`<`". Dentro de las expresiones, las cadenas literales se delimitan mediante comillas simples o dobles, las cuales se emplean también para delimitar atributos XML. Para evitar que una marca de entrecomillado en una expresión sea interpretada por el procesador XML como terminador del valor del atributo (símbolo de cierre), la marca de entrecomillado, puede introducirse como una referencia de caracter (`&quot;` o `&apos;`). Alternativamente, la expresión puede usar comillas simples si el atributo XML se delimita con comillas dobles o viceversa.

4.2.2 Tipos de Nodos

Debemos distinguir entre los siguientes tipos de nodos: *raíz*, *elemento*, *atributo*, *texto*, *comentario* e *instrucción de procesamiento*. Para cada tipo de nodo, hay una forma de determinar un valor de cadena. Para algunos tipos de nodo, el valor de cadena es parte del nodo; para otros tipos de nodo, el valor de cadena se calcula a partir del valor de cadena de nodos descendientes.

Algunos tipos de nodo tienen también un nombre expandido, que es un par formado por una parte local y un URI de espacio de nombres. La parte local es una cadena de caracteres. EL URI de espacio de nombres es, o bien nulo o bien una cadena de caracteres. Un nodo, "espacio de nombres" especificado en una declaración del espacio de nombres de un documento XML, es una referencia URI. Esto implica que puede tener un identificador de fragmento y puede ser relativo. El componente URI de espacio de nombres de un nombre expandido depende de la implementación, si el nombre expandido es la expansión de un nombre de elemento, cuyo prefijo se declara en una declaración de espacio de nombres con un nombre de espacio de nombres que es un URI relativo (con o sin identificador de fragmento). Una expresión XPath que dependa del valor del componente URI de espacio de nombres de uno de tales nombres expandidos, no es interoperable. Dos nombres expandidos son iguales si tienen la misma parte local, y o bien ambos tienen un URI de espacio de nombres nulo o bien ambos tienen URIs de espacio de nombres no nulos que son iguales.

Hay una orden -el "orden de documento"- definido sobre todos los nodos del documento, correspondiente con el orden en que el primer carácter de la representación XML de cada nodo, aparece en la representación XML del documento, tras la expansión de las entidades generales. Así, el nodo raíz será el primer nodo. Los nodos elemento aparecen antes de sus hijos. Por tanto, el "orden de documento" ordena los nodos elemento en el orden de aparición de sus etiquetas de apertura en el XML (tras la expansión de entidades). Los nodos atributo y los nodos espacio de nombres de un elemento aparecen antes que los hijos del elemento. Los nodos espacio de nombres aparecen por definición, antes que los nodos atributo. El orden relativo de los nodos atributo depende de la implementación. El "orden inverso de documento" es el inverso del orden de documento.

Los nodos "raíz" y los nodos "elemento" tienen una lista ordenada de nodos "hijo". Los nodos nunca comparten un hijo: si un nodo no es el mismo nodo que otro, entonces ninguno de los hijos del primer nodo, será el mismo nodo que ninguno de los hijos del otro nodo. Todos los nodos salvo el nodo raíz tienen exactamente un padre, que es o bien un nodo elemento o bien un nodo raíz. Un nodo raíz o un nodo elemento es el padre de cada uno de sus nodos hijo. Los descendientes de un nodo son los hijos del nodo y los descendientes de los hijos del nodo.

4.2.2.1 Nodo Raíz

El nodo "raíz" es la raíz del árbol. No aparecen nodos "raíz" salvo como raíz del árbol. El nodo "elemento" del elemento que representa al documento, es hijo del nodo "raíz". El nodo "raíz" tiene también como hijos, los nodos "instrucción de procesamiento" y "comentario" correspondientes a las instrucciones de procesamiento y comentarios que aparezcan en el prólogo y tras el fin del elemento que representa al documento.

El valor de cadena de caracteres del nodo "raíz" es la concatenación de los valores de cadena de caracteres de todos los nodos "texto" descendientes del nodo "raíz", en orden de documento. El nodo raíz no tiene nombre expandido.

4.2.2.2 Nodos elemento

Hay un nodo “elemento” por cada elemento en el documento. Los nodos “elemento” tienen un nombre expandido calculado expandiendo el nombre del elemento especificado en la etiqueta. El URI de espacio de nombres del nombre expandido del elemento, será nulo si el nombre del elemento no tiene prefijo y no hay un espacio de nombres aplicable por defecto. Los hijos de un nodo “elemento” son los nodos “elemento”, nodos “comentario”, nodos “instrucción de procesamiento” y nodos “texto” que contiene. Las referencias de entidades, tanto a entidades internas como externas, son expandidas y las referencias de caracteres son resueltas.

El valor de cadena de caracteres de un nodo “elemento” es la concatenación de los valores de cadena de caracteres de todos los nodos “texto” descendientes del nodo “elemento”, en orden de documento.

4.2.2.3 Identificadores únicos

Los nodos “elemento” pueden tener un identificador único (ID). Este es el valor del atributo que se declara en el DTD, como de tipo ID. No puede haber dos elementos en un documento con el mismo ID. Si un procesador XML informa de la existencia de dos elementos de un documento con el mismo ID (lo cual es posible, sólo si el documento no es válido) entonces el segundo elemento en orden de documento debe ser tratado como si no tuviese ID. Si un documento no tiene DTD, entonces ningún elemento del documento tendrá ID.

4.2.2.4 Nodos atributo

Cada nodo “elemento” tiene asociado un conjunto de nodos “atributo”; el elemento es el padre de cada uno de esos nodos “atributo”; sin embargo, los nodos “atributo” no son hijos de su elemento padre. Los elementos nunca comparten nodos “atributo”. Si dos nodos “elemento” son distintos, entonces ninguno de los nodos “atributo” del primer elemento, será el mismo nodo “atributo” del otro nodo “elemento”. El operador “=” comprueba si dos nodos tienen el mismo valor, *no si son el mismo nodo*. Así, atributos de dos elementos diferentes pueden ser comparados como iguales utilizando “=”, a pesar de que no son el mismo nodo.

Algunos atributos, tales como **`xml:lang`** y **`xml:space`**, tienen la semántica de ser aplicables a todos los elementos que sean descendientes del elemento que lleva el atributo, salvo que sean redefinidos por una instancia del mismo atributo, en otro elemento descendiente. Sin embargo, esto no afecta a donde aparecen los nodos “atributo” en el árbol: un elemento tiene nodos “atributo” correspondientes únicamente a atributos explícitamente especificados en la etiqueta de apertura o etiqueta de elemento vacío de dicho elemento o bien que fueron explícitamente declarados en la DTD con un valor por defecto.

Los nodos “atributo” tienen un nombre expandido y un valor de cadena. El nombre expandido se calcula expandiendo el nombre especificado en la etiqueta en el documento

XML. El URI de espacio de nombres, del nombre del atributo, será nulo si el nombre del atributo no tiene prefijo.

Los nodos "atributo" tienen como valor una cadena de caracteres. Esta cadena es el valor normalizado (sin espacios duplicados). Un atributo cuyo valor normalizado es una cadena de longitud cero, no es tratado de una forma especial: da lugar a un nodo "atributo" cuyo valor de cadena es una cadena de longitud cero.

Para atributos por defecto es posible que la declaración tenga lugar en una DTD externa o una entidad de parámetro externa. La Recomendación XML 1.0 no impone a los procesadores XML la lectura de DTDs externas o parámetros externos salvo que el procesador incluya validación. Una hoja de estilo u otro mecanismo que suponga que el árbol XPath contiene valores de atributos por defecto declarados en una DTD externa o entidad de parámetro, podría no funcionar con algunos procesadores XML no validadores. No hay nodos "atributo" correspondientes a atributos que declaran espacios de nombres.

4.2.2.5 Nodos espacio de nombres

Cada elemento tiene un conjunto asociado de nodos "espacio de nombres", uno para cada uno de los distintos prefijos de espacio de nombres con efecto sobre el elemento (incluyendo el prefijo xml, que está implícitamente declarado) y uno para el espacio de nombres por defecto, si hay alguno con efecto sobre el elemento. El elemento es el padre de cada uno de los nodos "espacio de nombres"; sin embargo, los nodos "espacio de nombres" no son hijos de su elemento padre. Los elementos nunca comparten nodos "espacio de nombres": Si dos nodos "elemento" son distintos, entonces ninguno de los nodos "espacio de nombres" del primer elemento será el mismo nodo que ningún nodo "espacio de nombres" del otro nodo "elemento". Esto significa que los elementos tendrán un nodo "espacio de nombres":

- Para cada atributo del elemento cuyo nombre empiece por "**`xmlns:`**".
- Para cada atributo de un elemento ancestro cuyo nombre empiece por "**`xmlns:`**" salvo que el propio elemento o un ancestro más cercano redeclaren el prefijo.
- Para atributos "**`xmlns:`**", si el elemento o alguno de sus ancestros tienen dicho atributo y el valor del atributo en el más cercano de los elementos que lo tienen, es no vacío.

Un atributo **`xmlns=""`** "desdeclara" el espacio de nombres por defecto. Los nodos "espacio de nombres" tienen un nombre expandido: la parte local es el prefijo de espacio de nombres (este es vacío, si el nodo espacio de nombres corresponde al espacio de nombres por defecto); el URI de espacio de nombres es siempre nulo.

El valor de cadena de un nodo espacio de nombres, es el URI de espacio de nombres que se está asociando al prefijo de espacio de nombres; si el nombre de espacio de nombres que aparece en la declaración de espacio de nombres del documento XML es un URI relativo (con o sin identificador de fragmento), entonces el valor de cadena depende de la implementación. Una expresión XPath que dependa del valor de cadena de uno de tales nodos de espacio de nombres, no es interoperable.

4.2.2.6 Nodos instrucción de procesamiento

Hay un nodo "instrucción de procesamiento" para cada instrucción de procesamiento, salvo para aquellas que aparezcan dentro de la declaración de tipo de documento (DTD). Las instrucciones de procesamiento tienen un nombre expandido: la parte local es el destinatario de la instrucción de procesamiento; el URI de espacio de nombres es nulo. El valor de cadena de un nodo "instrucción de procesamiento", es la parte de la instrucción de procesamiento que sigue al destinatario y todo el espacio en blanco. No incluye la terminación "**?>**". Se debe tener presente que la declaración XML no es una instrucción de procesamiento. En consecuencia, no hay nodo "instrucción de procesamiento" correspondiente a ella.

4.2.2.7 Nodos comentario

Hay un nodo comentario para cada comentario, salvo para aquellos que aparezcan dentro de la declaración de tipo de documento. El valor de cadena de los comentarios es el contenido del comentario sin incluir el inicio "**<!--**" ni la terminación "**-->**". Los nodos comentario no tienen nombre expandido.

4.2.2.8 Nodos texto

Los datos de caracter se agrupan en nodos "texto". En cada nodo "texto" se agrupan todos los datos de caracter que sea posible: un nodo "texto" nunca tiene un hermano inmediatamente anterior o siguiente, que sea nodo "texto". El valor de cadena de los nodos "texto" son los datos de caracter. Los nodos "texto" siempre tienen al menos un caracter.

Cada caracter en una sección CDATA se trata como datos de caracter. Así, "**<![CDATA[<]]>**" en el documento fuente será tratado igual que "**<**". Ambos darán lugar a un único caracter "**<**" en un nodo "texto" en el árbol. Por tanto, una sección CDATA se trata como si "**<![CDATA[**" y "**]]>**" se quitasen y cada aparición de "**<**"·y "**&**" fuese reemplazada por "**<**" y "**&**" respectivamente.

Los caracteres dentro de comentarios, instrucciones de procesamiento y valores de atributos no producen nodos "texto". Los finales de línea en entidades externas se normalizan como "**#xA**". El espacio en blanco fuera del elemento de documento, no produce nodos de texto. Los nodos de texto no tienen nombre expandido.

4.3 Location Path

El tipo más importante de expresión es el "Location Path" (camino de localización). Un camino de localización selecciona un conjunto de nodos relativo al nodo de contexto. El resultado de evaluar una expresión que sea un camino de localización, es el conjunto de nodos seleccionados por éste. Los caminos de localización pueden ser absolutos o relativos y son capaces de contener recursivamente otras expresiones utilizadas para filtrar conjuntos de nodos.

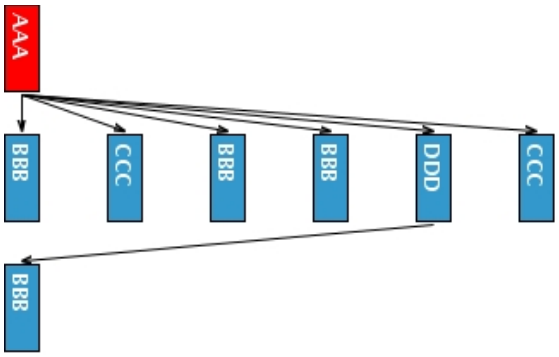
4.3.1 Caminos absolutos

La sintaxis básica de XPath es similar a la del direccionamiento de archivos. Un camino que se inicia con "/", representa un camino absoluto hacia el elemento requerido. La figura siguiente, al igual que muchas de las presentadas en este capítulo, fueron tomadas de su fuente original: ZVON.org - XPath Tutorial (ver sección de referencias de este capítulo).

Ejemplo: Selecciona el elemento raíz `AAA`

```
<xsl:template match="/">
     <xsl:value-of select="/AAA"/>
</xsl:template>
```

```
<AAA>
     <BBB/>
     <CCC/>
     <BBB/>
     <BBB/>
     <DDD>
          <BBB/>
     </DDD>
     <CCC/>
</AAA>
```



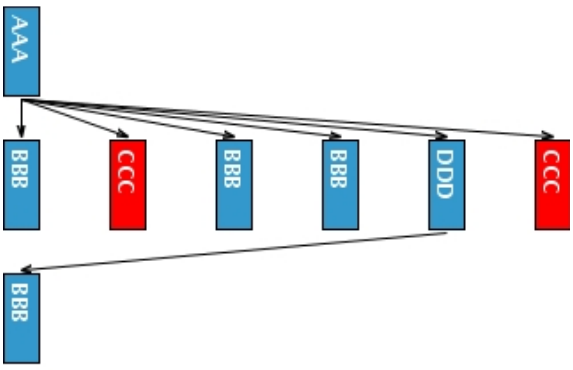
4.3.1.1 Múltiples pasos

Al igual que el direccionamiento de archivos, se puede alcanzar niveles más profundos del árbol XML con solo agregar "/" seguido de cualquier nodo hijo.

Ejemplo: Selecciona todos los elementos `CCC` que están en el nivel de profundidad siguiente del elemento raíz `AAA`

```
<xsl:template match="/">
     <xsl:value-of select="/AAA/CCC"/>
</xsl:template>
```

```
<AAA>
     <BBB/>
     <CCC/>
     <BBB/>
     <BBB/>
     <DDD>
          <BBB/>
     </DDD>
     <CCC/>
</AAA>
```



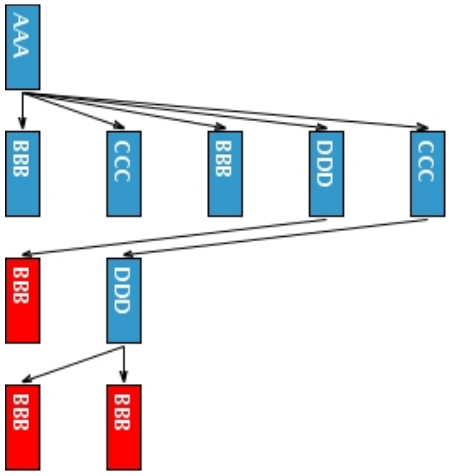
4.3.2 Caminos Relativos

Cuando el camino se inicia con "//" todos los elementos en el documento que cumplen con el criterio que sigue, son seleccionados.

Ejemplo: Selecciona todos los elementos `BBB` que están en el nivel de profundidad siguiente del elemento `DDD`

```
<xsl:template match="/">
     <xsl:value-of select="//DDD/BBB"/>
</xsl:template>
```

```
<AAA>
     <BBB/>
     <CCC/>
     <BBB/>
     <DDD>
          <BBB/>
     </DDD>
     <CCC>
          <DDD>
               <BBB/>
               <BBB/>
          </DDD>
     </CCC>
</AAA>
```



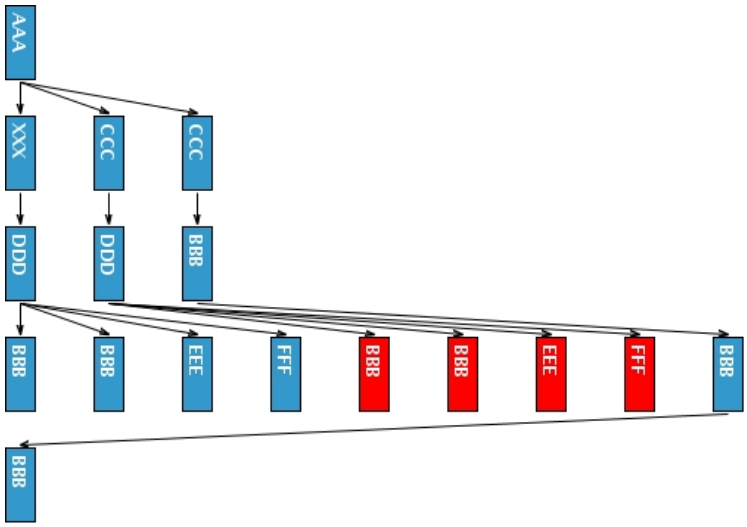
4.3.2.1 Nodos test

A veces, no se conoce el nombre de algún elemento. El asterisco "*" selecciona todos los elementos que cumplen con las restricciones que la expresión XPath imponga.

Ejemplo: Selecciona todos los elementos contenidos en el camino `/AAA/CCC/DDD`

```
<xsl:template match="/">
     <xsl:value-of select="/AAA/CCC/DDD/*"/>
</xsl:template>
```

```
<AAA>
     <XXX>
          <DDD>
               <BBB/>
               <BBB/>
               <EEE/>
               <FFF/>
          </DDD>
     </XXX>
     <CCC>
          <DDD>
               <BBB/>
               <BBB/>
               <EEE/>
               <FFF/>
          </DDD>
     </CCC>
     <CCC>
          <BBB>
               <BBB>
                    <BBB/>
               </BBB>
          </BBB>
     </CCC>
</AAA>
```



Ejercicio: Vea el resultado de aplicar las siguientes expesiones XPath, al XML anterior:

- **`/*/*/*/BBB`**
- **`//*`**

4.3.3 Los ejes (axis)

Un "eje" define un conjunto de nodos relativo al nodo del contexto (nodo actual). Los ejes se utilizan para recorrer la jerarquía de un documento XML (subir, bajar o moverse entre hermanos). En esta sección se revisará brevemente cada uno de los ejes disponibles en XPath.

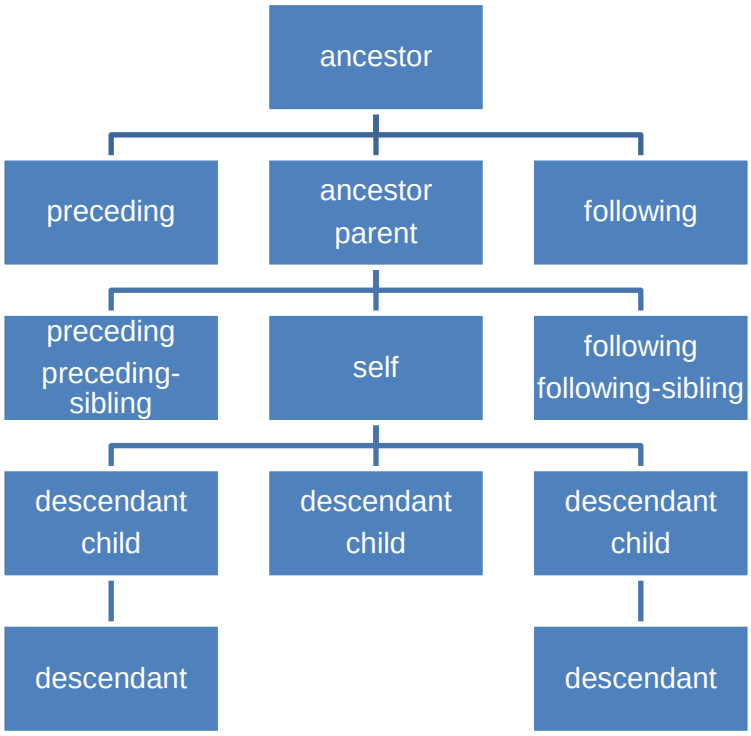


Figura 1: Esquema de los ejes de XPath

4.3.3.1 Hijo

El eje "child" contiene a los hijos del nodo del contexto. Éste eje es seleccionado por defecto y puede ser omitido. La sintaxis de uso es la siguiente: "**`child::`**"

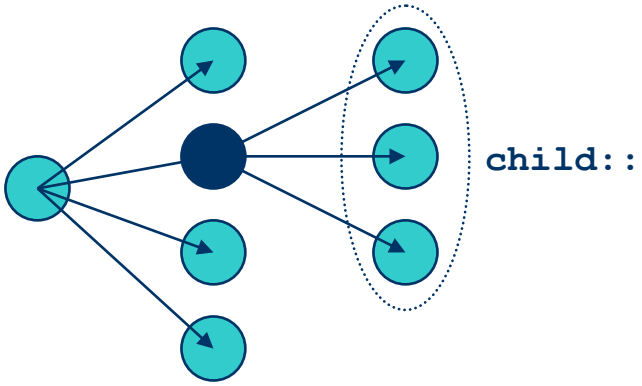
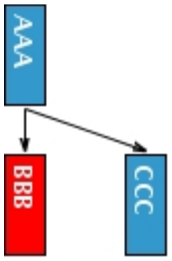


Figura 2: Esquema del eje XPath que permite acceder a los nodos "hijo" del nodo de contexto

Ejemplo: El equivalente a `/AAA/BBB`

```
<xsl:template match="/">
     <xsl:value-of select="/child::AAA/child::BBB"/>
</xsl:template>
```

```
<AAA>
     <BBB/>
     <CCC/>
</AAA>
```



4.3.3.2 Padre

Existen dos formas de referenciar al nodo padre de un cierto nodo. La sintaxis de uso es la siguiente: "**`..`**" y "**`parent::`**".

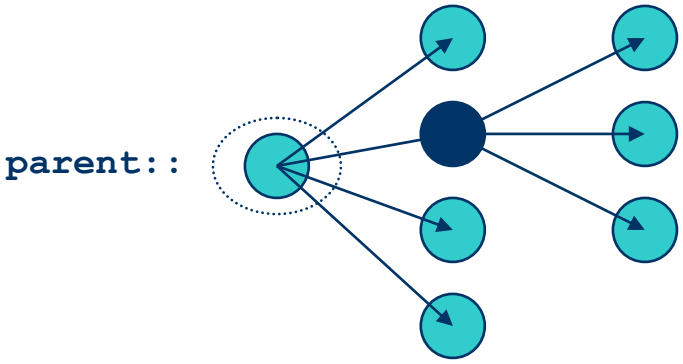
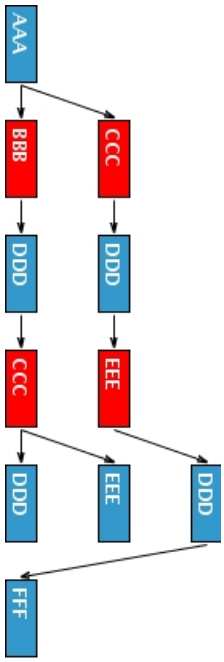


Figura 3: Esquema del eje XPath que permite acceder al nodo "padre" de un cierto nodo

Ejemplo

```
<xsl:template match="/">
     <xsl:value-of select="//DDD/parent::*"/>
</xsl:template>
```

```
<AAA>
     <BBB>
          <DDD>
               <CCC>
                    <DDD/>
                    <EEE/>
               </CCC>
          </DDD>
     </BBB>
     <CCC>
          <DDD>
               <EEE>
                    <DDD>
                         <FFF/>
                    </DDD>
               </EEE>
          </DDD>
     </CCC>
</AAA>
```



4.3.3.3 Ancestros y descendientes del elemento actual

El primer ancestro de un nodo, es su nodo "padre". Además el padre del nodo "padre", es también un ancestro. En verdad, "el padre del padre" va a ser ancestro hasta que se llegue al nodo "raíz". Otra forma de verlo es trazar la línea del árbol, desde un cierto nodo, hasta el nodo raíz. Todos los nodos alcanzados por esta línea serán ancestros.

Por otro lado, podríamos decir que los nodos descendientes son todos los nodos "hijo" de un cierto nodo, y todos sus hijos y los hijos de los hijos. Otra forma de verlo es pensar en todo el subárbol que cuelgue del nodo en cuestión. La sintaxis de uso es la siguiente: "**`ancestor::`**" y "**`descendant::`**".

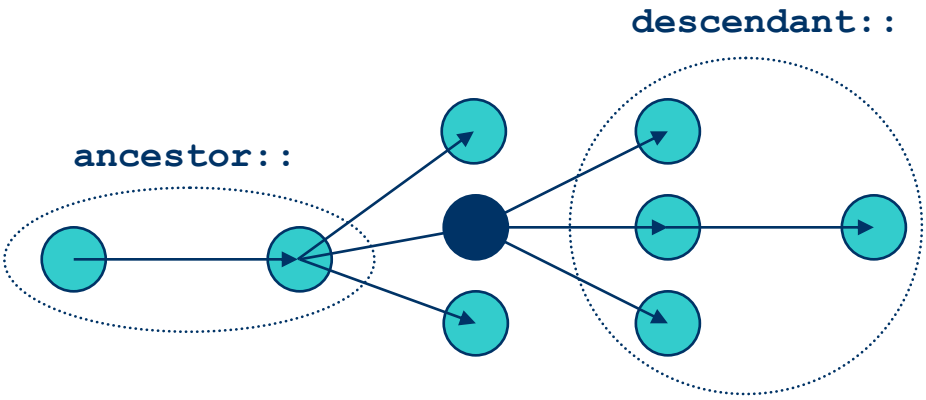
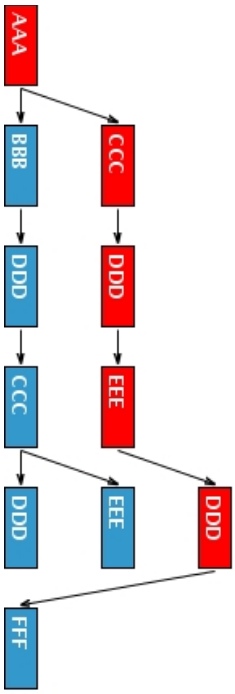


Figura 4: Esquema del eje XPath que permite acceder a los nodos ancestros o descendientes de un cierto nodo

Ejemplo: Selecciona los ancestros de cada elemento `FFF`

```
<xsl:template match="/">
     <xsl:value-of select="//FFF/ancestor::*"/>
</xsl:template>
```

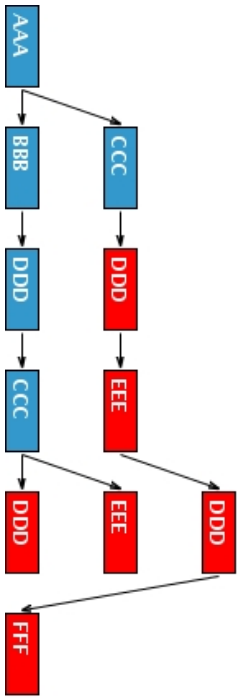
```
<AAA>
     <BBB>
          <DDD>
               <CCC>
                    <DDD/>
                    <EEE/>
               </CCC>
          </DDD>
     </BBB>
     <CCC>
          <DDD>
               <EEE>
                    <DDD>
                         <FFF/>
                    </DDD>
               </EEE>
          </DDD>
     </CCC>
</AAA>
```



Ejemplo: Selecciona todos elementos que son descendientes de un nodo `CCC`

```
<xsl:template match="/">
     <xsl:value-of select="//CCC/descendant::*"/>
</xsl:template>
```

```
<AAA>
     <BBB>
          <DDD>
               <CCC>
                    <DDD/>
                    <EEE/>
               </CCC>
          </DDD>
     </BBB>
     <CCC>
          <DDD>
               <EEE>
                    <DDD>
                         <FFF/>
                    </DDD>
               </EEE>
          </DDD>
     </CCC>
</AAA>
```



Existen variantes para los ejes que acaban de mostrarse. Estos ejes permiten acceder, al mismo tiempo, a los ancestros o descendientes de un cierto nodo, y también al nodo en cuestión. La sintaxis de uso es la siguiente: "**`ancestor-or-self::`**" y "**`descendant-or-self::`**".

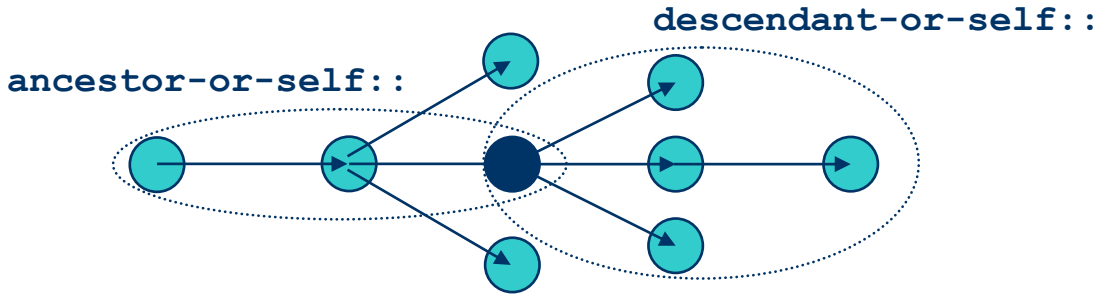
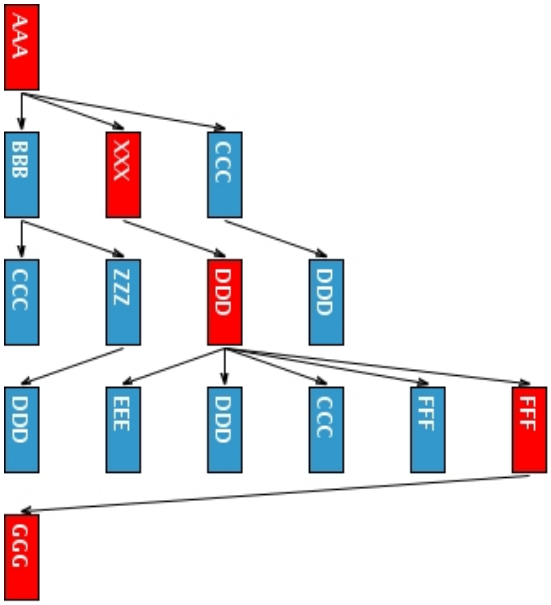


Figura 5: Esquema de los ejes de XPath que permiten acceder a ancestros y descendientes de un nodo, incluyendo al nodo en cuestión

Ejemplo

```
<xsl:template match="/">
     <xsl:value-of select="//GGG/ancestor-or-self::*"/>
</xsl:template>
```

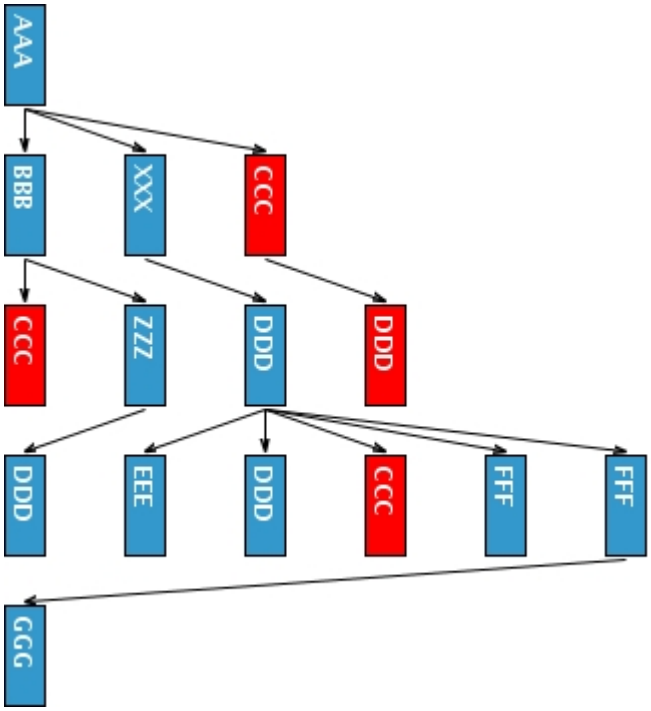
```
<AAA>
     <BBB>
          <CCC/>
          <ZZZ>
               <DDD/>
          </ZZZ>
     </BBB>
     <XXX>
          <DDD>
               <EEE/>
               <DDD/>
               <CCC/>
               <FFF/>
               <FFF>
                    <GGG/>
               </FFF>
          </DDD>
     </XXX>
     <CCC>
          <DDD/>
     </CCC>
</AAA>
```



Ejemplo

```
<xsl:template match="/">
     <xsl:value-of select="//CCC/descendant-or-self::*"/>
</xsl:template>
```

```
<AAA>
     <BBB>
          <CCC/>
          <ZZZ>
               <DDD/>
               </ZZZ>
          </BBB>
          <XXX>
          <DDD>
               <EEE/>
               <DDD/>
               <CCC/>
               <FFF/>
               <FFF>
                    <GGG/>
               </FFF>
          </DDD>
     </XXX>
     <CCC>
          <DDD/>
     </CCC>
</AAA>
```



4.3.3.4 Identidad

Existen dos formas de referenciar sólo al elemento actual. La sintaxis de uso es la siguiente:

- `.`
- **`self::`**

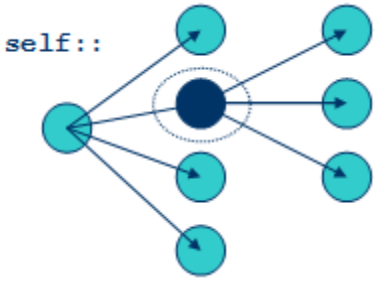


Figura 6: Esquema del eje XPath que permite acceder al nodo de contexto

El uso más habitual de este eje es en la construcción de predicados, lo cuales se verán más adelante.

4.3.3.5 Hermanos del elemento actual

Los hermanos de un cierto nodo, son todos los nodos que también son hijos del nodo "padre" del nodo en cuestión. La sintaxis de uso es la siguiente: "**`preceding-sibling::`**" y "**`following-sibling::`**".

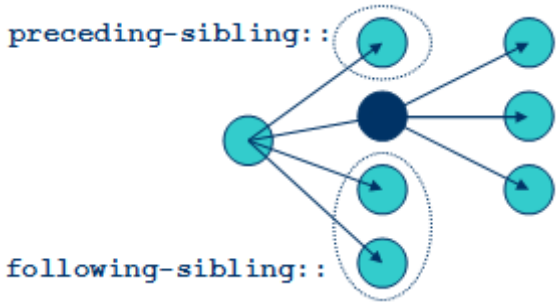
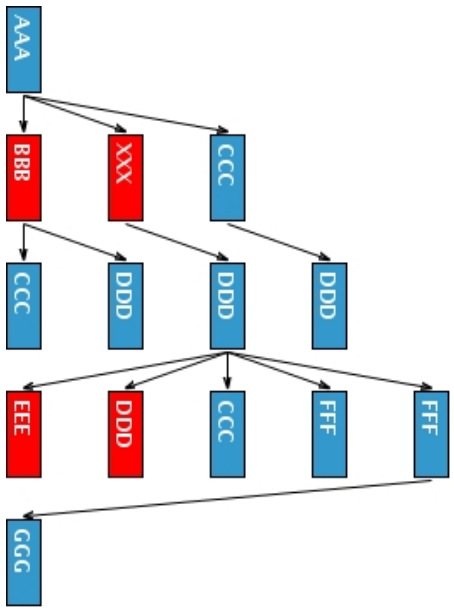


Figura 7: Esquema del eje XPath que permite acceder a todos los otros nodos "hijo", del mismo nodo "padre" de cierto nodo

Ejemplo

```
<xsl:template match="/">
     <xsl:value-of select="//CCC/preceding-sibling::*"/>
</xsl:template>
```

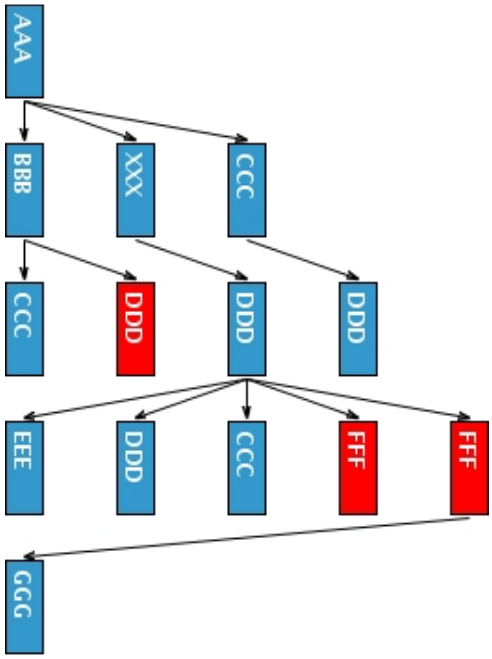
```
<AAA>
     <BBB>
          <CCC/>
          <DDD/>
     </BBB>
     <XXX>
          <DDD>
               <EEE/>
               <DDD/>
               <CCC/>
               <FFF/>
               <FFF>
                    <GGG/>
               </FFF>
          </DDD>
     </XXX>
     <CCC>
          <DDD/>
     </CCC>
</AAA>
```



Ejemplo

```
<xsl:template match="/">
     <xsl:value-of select="//CCC/following-sibling::*"/>
</xsl:template>
```

```
<AAA>
     <BBB>
          <CCC/>
          <DDD/>
     </BBB>
     <XXX>
          <DDD>
               <EEE/>
               <DDD/>
               <CCC/>
               <FFF/>
               <FFF>
                    <GGG/>
               </FFF>
          </DDD>
     </XXX>
     <CCC>
          <DDD/>
     </CCC>
</AAA>
```



4.3.3.6 Acceso a todos los nodos que preceden y siguen al nodo actual

El eje "preceding" contiene todos los nodos del documento que se encuentran antes del nodo del contexto. El eje "following" contiene todos los nodos del documento que se encuentran luego del nodo del contexto. Ambos no incluyen ancestros, descendientes, nodos de atributo ni nodos "namespace". La sintaxis de uso es la siguiente: "**`preceding::`**" y "**`following::`**".

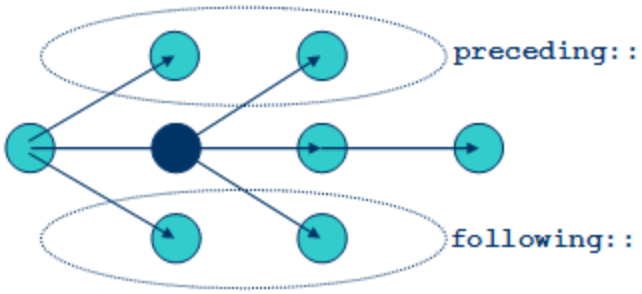
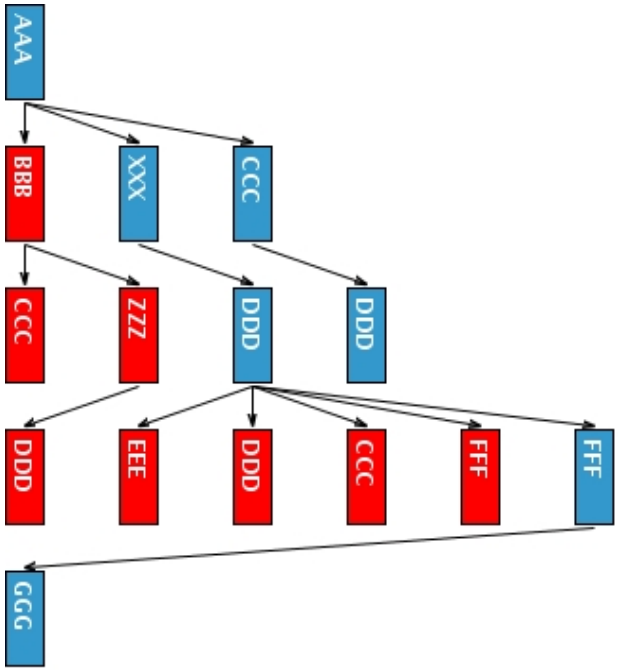


Figura 8: Esquema de los ejes de XPath que permiten acceder a los nodos que preceden o siguen al nodo en cuestión

Ejemplo

```
<xsl:template match="/">
     <xsl:value-of select="//GGG/preceding::*"/>
</xsl:template>
```

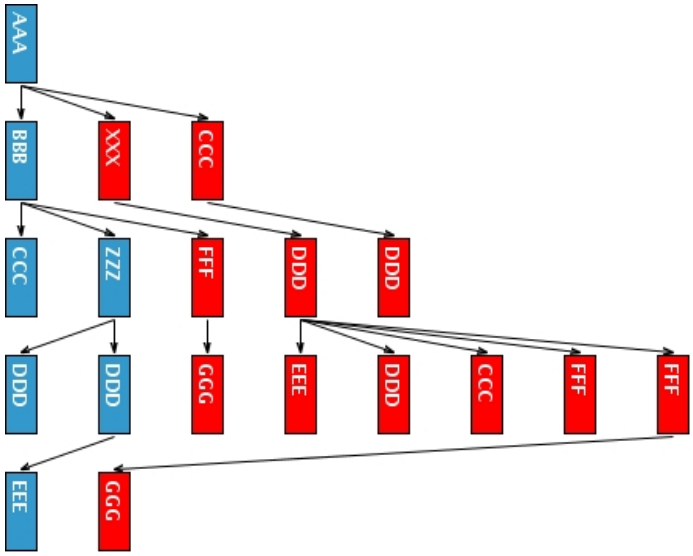
```
<AAA>
     <BBB>
          <CCC/>
          <ZZZ>
               <DDD/>
          </ZZZ>
     </BBB>
     <XXX>
          <DDD>
               <EEE/>
               <DDD/>
               <CCC/>
               <FFF/>
               <FFF>
                    <GGG/>
               </FFF>
          </DDD>
     </XXX>
     <CCC>
          <DDD/>
     </CCC>
</AAA>
```



Ejemplo

```
<xsl:template match="/">
     <xsl:value-of select="//ZZZ/following::*"/>
</xsl:template>
```

```
<AAA>
     <BBB>
          <CCC/>
          <ZZZ>
               <DDD/>
               <DDD>
                    <EEE/>
               </DDD>
          </ZZZ>
          <FFF>
               <GGG/>
          </FFF>
     </BBB>
     <XXX>
          <DDD>
               <EEE/>
               <DDD/>
               <CCC/>
               <FFF/>
               <FFF>
                    <GGG/>
               </FFF>
          </DDD>
     </XXX>
     <CCC>
          <DDD/>
     </CCC>
</AAA>
```



4.3.3.7 Acceso a los atributos

Los atributos se especifican con el símbolo prefijo "@". La sintaxis es la siguiente: "**`attribute::`**".

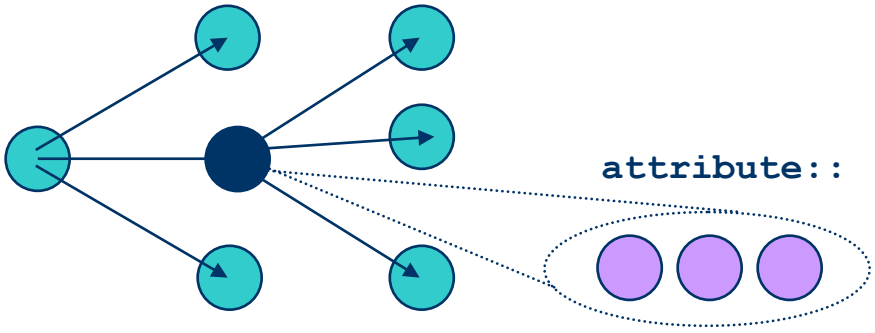
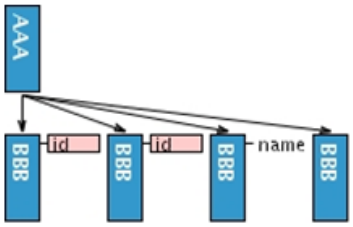


Figura 9: Esquema del eje XPath que permite acceder a los atributos de cierto nodo

Ejemplo: Selecciona todos los atributos "`id`"

```
<xsl:template match="/">
     <xsl:value-of select="//@id"/>
</xsl:template>
```

```
<AAA>
     <BBB id = "b1"/>
     <BBB id = "b2"/>
     <BBB name = "bbb"/>
     <BBB/>
</AAA>
```



4.3.4 Filtros de Predicado

Un "location path" en general devuelve una lista de nodos que cumplen con algún criterio. Los "predicados" permiten filtrar la lista devuelta. Los filtros de predicado se anotan entre "**`[ ]`**".

4.3.4.1 Operadores XPath

Como se observa en la tabla siguiente, existen una serie de símbolos que permiten ciertas operaciones aritméticas entre predicados. Estos símbolos se conocen como "Operadores XPath".

Tabla 1: Lista de los operadores que pueden ser usadas en expresiones XPath

Operador	Descripción	Ejemplo	Valor de Retorno
`+`	Suma	**`6 + 4`**	**`10`**
`-`	Resta	**`6 - 4`**	**`2`**
`*`	Multiplicación	**`6 * 4`**	**`24`**
`div`	División	**`8 div 4`**	**`2`**
`mod`	Módulo (resto de la división)	**`5 mod 2`**	**`1`**

4.3.4.2 Predicados booleanos

Los filtros de predicado se anotan entre "**`[ ]`**" y todos los predicados pueden combinarse con **`or`** o **`and`**. Como se observa en la tabla siguiente, existen una serie de símbolos que permiten ciertas operaciones entre predicados. Estos símbolos se conocen como "Operadores XPath".

Tabla 2: Lista de los operadores que pueden ser usadas en expresiones XPath

Operador	Descripción	Ejemplo	Valor de Retorno
`|`	Computa 2 conjuntos de nodos	`//libro | //disco`	Retorna un conjunto de nodos con todos los elementos "libro" y "disco"
`=`	Igualdad	`precio = 9.8`	verdadero si precio es 9.8 falso si es otra cosa
`!=`	Desigualdad	`precio != 9.8`	falso si precio es 9.8 verdadero si es otra cosa
`<`	Menor que	`precio < 9.8`	verdadero si precio es menor que 9.8 falso si precio es mayor o igual a 9.8
`<=`	Menor o igual que	`precio <= 9.8`	verdadero si precio es menor o igual a 9.8 falso si precio es mayor a 9.8
`>`	Mayor que	`precio > 9.8`	verdadero si precio es mayor que 9.8 falso si precio es menor o igual a 9.8
`>=`	Mayor o igual que	`precio >= 9.8`	verdadero si precio es menor o igual a 9.8 falso si precio es mayor a 9.80
`or`	"o" lógico	`precio = 6 or precio = 9`	verdadero si precio es 6 o 9 falso si es cualquier otra cosa
`and`	"y" lógico	`precio > 6 and precio < 9`	verdadero si precio es mayor que 6 y menor que 9 falso si precio es menor o igual a 6 o mayor o igual a 9
`not`	Negación	`not()`	`1`

4.3.4.3 Predicados de Posición

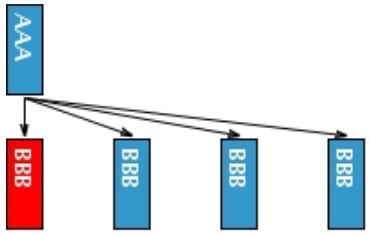
El más simple se construye usando la función "position", que representa la posición del elemento en el conjunto seleccionado. Puede ser abreviado simplemente colocando el número de la posición entre corchetes, como se muestra más abajo.

- **`/AAA/BBB[position()=1]`**
- **`/AAA/BBB[1]`**

Ejemplo: Equivale a `/AAA/BBB[1]`

```
<xsl:template match="/">
     <xsl:value-of select="/AAA/BBB[position()=1]"/>
</xsl:template>
```

```
<AAA>
     <BBB/>
     <BBB/>
     <BBB/>
     <BBB/>
</AAA>
```

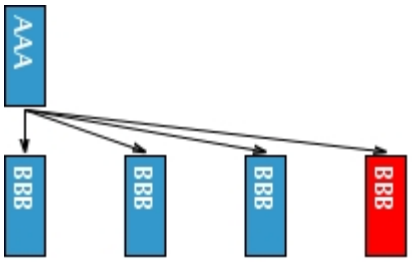


Otra función de posición es "last", que retorna el último elemento en la selección.

Ejemplo: Selecciona el último hijo `BBB` del elemento `AAA`

```
<xsl:template match="/">
     <xsl:value-of select="/AAA/BBB[last()]"/>
</xsl:template>
```

```
<AAA>
     <BBB/>
     <BBB/>
     <BBB/>
     <BBB/>
</AAA>
```

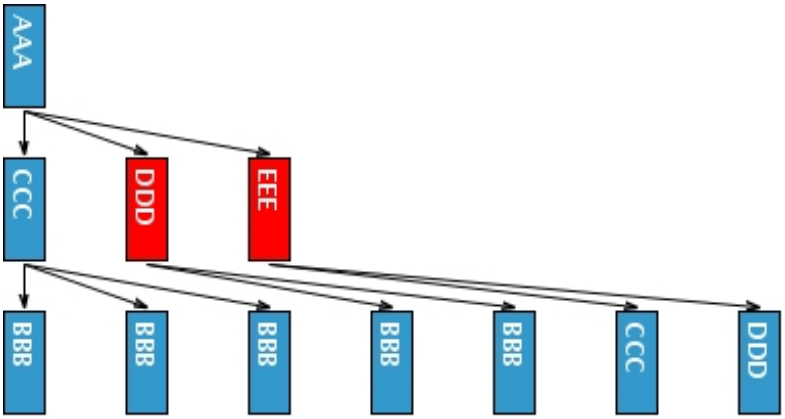


La función “count” evalúa el número de elementos en la selección.

Ejemplo: Selecciona todos los elementos con dos hijos

```
<xsl:template match="/">
     <xsl:value-of select="//*[count(*)=2]"/>
</xsl:template>
```

```
<AAA>
     <CCC>
          <BBB/>
          <BBB/>
          <BBB/>
     </CCC>
     <DDD>
          <BBB/>
          <BBB/>
     </DDD>
     <EEE>
          <CCC/>
          <DDD/>
     </EEE>
</AAA>
```



4.3.4.4 Predicados de contenido

Seleccionar aquellos nodos con un determinado hijo

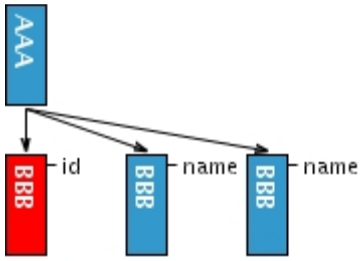
- child::exon[snp]
- child::exon[snp="A30T"]

<exon>
<snp>A30T</snp>
</exon>

Ejemplo: Selecciona el último hijo `BBB` del elemento `AAA`

```
<xsl:template match="/">
     <xsl:value-of select="//BBB[@id='b1']"/>
</xsl:template>
```

```
<AAA>
     <BBB id = "b1"/>
     <BBB name = " bbb "/>
     <BBB name = "bbb"/>
</AAA>
```



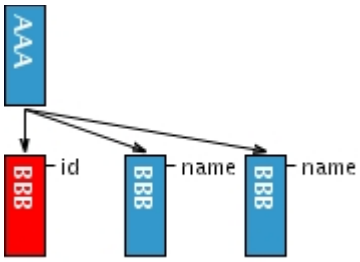
4.3.4.5 Predicados de Atributo

Los valores de los atributos también pueden ser utilizados como criterio de selección.

Ejemplo: Selecciona los elementos `BBB` cuyo atributo "`id`" tiene por valor "`b1`"

```
<xsl:template match="/">
     <xsl:value-of select="//BBB[@id='b1']"/>
</xsl:template>
```

```
<AAA>
     <BBB id = "b1"/>
     <BBB name = " bbb "/>
     <BBB name = "bbb"/>
</AAA>
```



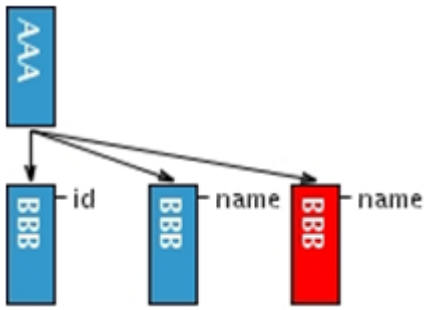
4.3.4.6 Predicados sobre cadenas de caracteres

La función "name" retorna el nombre del elemento seleccionado.

Ejemplo: Selecciona todos los elementos `BBB`, equivalente a `//BBB`

```
<xsl:template match="/">
     <xsl:value-of select="//*[name()='BBB']"/>
</xsl:template>
```

```
<AAA>
     <BCC>
          <BBB/>
          <BBB/>
          <BBB/>
     </BCC>
     <DDB>
          <BBB/>
          <BBB/>
     </DDB>
     <BEC>
          <CCC/>
          <DBD/>
     </BEC>
</AAA>
```

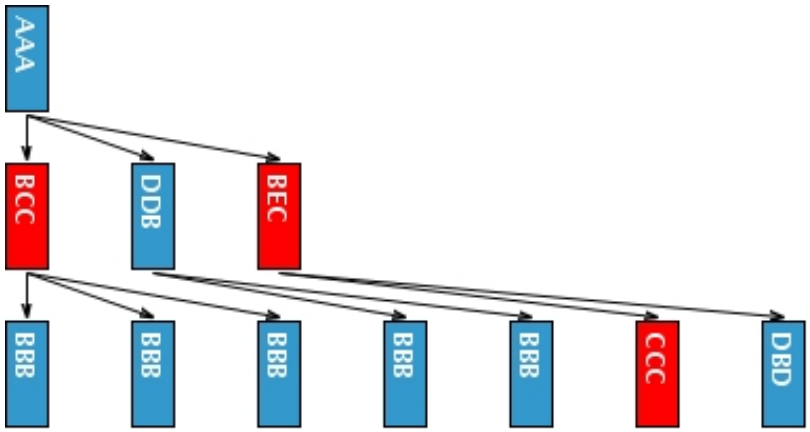


La función “contains” es verdadera cuando la cadena de caracteres del primer argumento, contiene al segundo argumento.

Ejemplo: Selecciona todos los elementos cuyo nombre contenga la letra C

```
<xsl:template match="/">
     <xsl:value-of select="//*[contains(name(),'C')]"/>
</xsl:template>
```

```
<AAA>
     <BCC>
          <BBB/>
          <BBB/>
          <BBB/>
     </BCC>
     <DDB>
          <BBB/>
          <BBB/>
     </DDB>
     <BEC>
          <CCC/>
          <DBD/>
     </BEC>
</AAA>
```

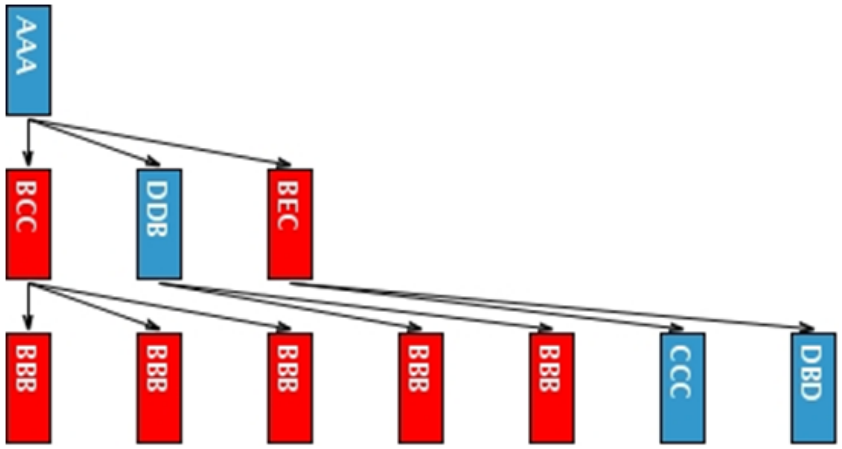


La función “starts-with” es verdadera cuando la cadena de caracteres pasada en primer argumento tiene como prefijo al segundo argumento.

Ejemplo: Selecciona todos los elementos cuyo nombre se inicie con la letra B

```
<xsl:template match="/">
     <xsl:value-of select="//*[starts-with(name(),'B')]"/>
</xsl:template>
```

```
<AAA>
     <BCC>
          <BBB/>
          <BBB/>
          <BBB/>
     </BCC>
     <DDB>
          <BBB/>
          <BBB/>
     </DDB>
     <BEC>
          <CCC/>
          <DBD/>
     </BEC>
</AAA>
```



La función “normalize-space” elimina los espacios al principio y al final, así como también reemplaza las secuencias de blancos, por un solo espacio.

Ejemplo: Selecciona los elementos BBB cuyo atributo `name` tiene por valor `bbb` una vez eliminados los espacios al principio y al final

```
<xsl:template match="/">
     <xsl:value-of select="//BBB[normalize-space(@name)='bbb']"/>
</xsl:template>
```

4.4 XQuery

XPath es usado por otras especificaciones de XML. Una de ellas es “XML Query” o “XQuery” (o simplemente XQL). La especificación de XQuery 1.0 contempla los siguientes ítems:

- Requerimientos de XML Query 1.0.
- Semántica Formal de XQuery 1.0.
- Modelo de Datos de XQuery 1.0 y XPath 2.0.
- Casos de Uso para XML Query.
- Sintaxis XML para XQuery 1.0 (XQueryX).
- Operadores y funciones para XQuery 1.0.

4.4.1 Diseño y concepción de XQuery

XQuery deriva directamente de "Quilt", el que a su vez sintetiza características de:

- XPath: Expresiones de localización.
- XML-QL: La noción de "*binding variable*", las cuales permiten la construcción de nuevas estructuras.
- SQL: El patrón SELECT-FROM-WHERE para reestructurar la información.
- OQL (ODMG): La composición de expresiones en un lenguaje funcional.
- Otras influencias de lenguajes de consulta para documentos estructurados como: "Lorel" y "Yalt".

XQuery es un lenguaje declarativo, capaz de usar cualquier tipo definido en un XML-Schema. Éste trabaja con el concepto de identificador unívoco, de forma análoga al de la clave primaria en bases de datos relacionales. Una consulta puede utilizar referencias internas o externas al documento (incluyendo XLinks). Esto lo convierte en un lenguaje de consulta en grafos.

4.4.2 FLWOR

Pronunciado "*flower*", es un acrónimo para *For-Let-Where-Order-Return*. Las expresiones FLWOR son uno de los dos constructos principales de XQL (el otro constructo principal es XPath). FOR es comparable al SELECT de SQL, proveyendo la capacidad de iterar sobre las secuencias de entrada, LET habilita el "variable-binding", WHERE posibilita definir filtros, mientras que ORDER BY permite ordenar los filtros, y finalmente, RETURN devuelve el resultado.

4.5 Recapitulación

Resumiendo lo expresado en este capítulo podemos decir que XPath provee una sintaxis no-XML para determinar partes de un documento XML. Usa *paths* (rutas) para recorrer un documento XML y contiene una biblioteca de funciones estándar. Además, es un componente muy importante de XSLT y un estándar del W3C.

Ejercicio

Sobre el documento XML que construyó como ejercicio en el capítulo sobre XML, realice las siguientes consultas usando XPath:

- Recupere todos los tipos de sangre y grupo RH distintos, de todos los nacimientos que compartan algún apellido.
- Recupere los apellidos de los padres de los nacimientos en que un mismo miembro de un equipo médico haya estado presente.
- Recupere todos los nacimientos por comuna para un cierto mes que usted escoja.

4.6 Referencias

- XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999 http://www.w3.org/TR/1999/REC-xpath-19991116
- XPath http://www.tejedoresdelweb.com/307/article-55815.html
- XPath – Accediendo a XML http://wwwdi.ujaen.es/~vrivas/docencia/cursillos/xml_baeza/alumnos/recursos/XML-XPath-Fernando.ppt
- ZVON.org - XPath Tutorial: http://www.zvon.org/xxl/XPathTutorial/General/examples.html. Last visit: May 2009.

CAPITULO V: XSLT (eXtensible Stylesheet Language Transformations)

Alex Bórquez, Claudio Gutiérrez

Departamento de Ciencias de la Computación, Universidad de Chile.
{aborquez, cgutierr}@dcc.uchile.cl

5.1 Introducción al XSL

En los capítulos anteriores se ha introducido XML como el lenguaje para la interoperabilidad de la información, tanto en la Web, como entre sistemas. Luego se revisó una forma de validar la información contenida en un documento XML, usando XML Schemas. A continuación se introdujo XPath como un mecanismo para extraer información de un documento XML. La verdad es que en el mundo XML se pueden hacer muchas cosas, pero de ellas hay una que no es posible pasar por alto: Transformar la información contenida en un documento XML.

5.1.1 ¿Para qué transformar datos contenidos en un documento XML?

En realidad, hay demasiadas respuestas a esta pregunta como para incluirlas en esta breve revisión, pero una fácil podría ser "para producir una página HTML para un sitio Web, producir una página WAP y generar un documento PDF, a partir del mismo documento XML". ¿Interesante? Muy bien, pero partamos por el principio.

5.1.2 ¿Qué es XSL?

XSL significa *eXtensible Stylesheet Language*. Una traducción aceptable al español podría ser "Lenguaje eXtensible de Hojas de Estilo". El W3C comenzó a desarrollar XSL porque existía la necesidad de tener un lenguaje de hojas de estilo basado en XML, así como las CSS (*Cascading Style Sheets*) estaban disponibles para el HTML.

Este último lenguaje, HTML, usa *tags* predefinidos y su significado es bien conocido. Por ejemplo el elemento **`<table>`** define una tabla y los navegadores "saben" como desplegarla en pantalla. En la misma línea, añadir estilos a los elementos HTML es simple, pues usando CSS es relativamente sencillo indicarle a un navegador que despliegue en pantalla algún elemento con cierto color, o un texto con determinada fuente.

Como se mencionó anteriormente, la idea fuerza es que el XSL sea el lenguaje para la construcción de las hojas de estilo del XML. Sin embargo, XML no tiene *tags* predefinidos. Más aún, el nombre de los *tags* es completamente personalizable por el usuario, por tanto, su significado puede ser interpretado de forma diferente, dependiendo de quien mire o procese el documento.

En XML, un elemento **<tabla>** puede corresponder al elemento **<table>** de HTML, a un material de construcción u otra cosa, y el navegador no "sabe" como desplegarlo en pantalla. En este ejemplo, es ahí donde entra XSL: describiendo cómo un documento XML debe ser desplegado en pantalla. Al final de esta breve revisión de lo más importante de XSL, el lector debería llegar a la convicción de que XSL tiene mucha más potencialidad de la que se necesita, para ser un simple lenguaje de hojas de estilo.

5.1.3 Transformando XML

XSL consta de tres partes: XSLT: un lenguaje para transformar documentos XML, XPath: un lenguaje para navegar en documentos XML y XSL-FO: un lenguaje para formatear documentos XML. Este capítulo se centrará en XSLT, dado que conforma el subconjunto de XSL más importante.

Pues bien, ¿qué es XSLT? En términos simples, son transformaciones XSL (*XSL Transformation*). Como su nombre sugiere, principalmente es usado para transformar un documento XML en otro documento XML, u otro tipo de documento que los navegadores reconozcan, como HTML y XHTML.

Con XSLT es posible añadir o remover, elementos y atributos, desde o hacia el documento de salida. También es posible reubicar y ordenar los elementos, realizar pruebas condicionales y tomar decisiones acerca de cuales elementos mostrar o esconder, y muchas cosas más. Una forma común de describir el proceso de transformación, es decir que XSLT transforma un "árbol XML fuente" en un "árbol XML resultado".

XSLT usa XPath para encontrar información en un documento XML. Se usa XPath para navegar a través de los elementos y atributos de un documento XML. ¿Cómo lo hace? En el proceso de transformación, XSLT usa XPath para decir qué partes del documento fuente deben calzar con una o más plantillas (*templates*) predefinidas. Cuando se encuentra un calce, XSLT transforma la parte coincidente del documento fuente en el documento resultante.

Otro detalle importante es que XSLT se volvió una Recomendación del W3C, el 16 de Noviembre de 1999. Desde ese momento no ha perdido *momentum*, y no parece tener rivales.

5.2 Procesamiento Básico de Hojas de Estilo

5.2.1 Declaración de Hojas de Estilo

El elemento raíz que declara que un documento es un hoja de estilo XSL es **<xsl:stylesheet>** o **<xsl:transform>**. Estas dos declaraciones son completamente equivalentes y pueden usarse de forma indistinta. La forma correcta de declarar una hoja de estilo XSL de acuerdo a la Recomendación para XSL del W3C es:

Ejemplo: Declaración típica de XSL

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

O bien:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Para tener acceso a los elementos XSLT, atributos y características, primero se debe declarar el espacio de nombres (*namespace*) XSLT, al comienzo del documento. Es necesario detenerse un momento en los fragmentos de la declaración. El siguiente fragmento de código apunta a espacio de nombre XSLT oficial del W3C. Si se usa este espacio de nombres, además debe incluirse el atributo **`versión`**, que en estos momentos corresponde a la 1.0:

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

5.2.2 Creando una hoja de estilos XSL y vinculándola a un documento XML

Supongamos que queremos transformar el siguiente documento XML en XHTML.

Ejemplo: Documento XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nacimientos>
  <nacimiento>
    <nombre>Juan</nombre>
    <apellido>Perez</apellido>
    <tipoDeSangre>A</tipoDeSangre>
    <peso>3</peso>
    <altura>50</altura>
    <fechaDeNacimiento>01-01-2006</fechaDeNacimiento>
  </nacimiento>
  <nacimiento>
    <nombre>Pablo</nombre>
    <apellido>Gonzalez</apellido>
    <tipoDeSangre>B</tipoDeSangre>
    <peso>2</peso>
    <altura>45</altura>
    <fechaDeNacimiento>02-02-2006</fechaDeNacimiento>
  </nacimiento>
.
.
</nacimientos>
```

Entonces se crea la hoja de estilos XSL con una plantilla (*template*) de transformación.

Ejemplo: Hoja de estilos XSL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Nacimientos</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Nombre</th>
            <th>Fecha de Nacimiento</th>
          </tr>
          <tr>
            <td>.</td>
            <td>.</td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Solo queda agregar la referencia a la hoja de estilos del documento XML:

Ejemplo: Documento XML vinculado a una hoja de estilos XSL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="nacimientos.xsl"?>
<nacimientos>
      <nacimiento>
             <nombre>Juan</nombre>
             <apellido>Perez</apellido>
             <tipoDeSangre>A</tipoDeSangre>
             <peso>3</peso>
             <altura>50</altura>
             <fechaDeNacimiento>01-01-2006</fechaDeNacimiento>
      </nacimiento>
.
.
.
</nacimientos>
```

Si se tiene a mano un navegador que sea compatible con la especificación de XSLT, es posible utilizarlo para transformar el documento XML en XHTML.

5.3 Reglas de Plantilla

En la sección anterior se mostraron las declaraciones básicas para vincular un cierto documento XML con una determinada hoja de estilos XSL. En esta sección veremos algunos elementos XSL que nos permiten hacer las transformaciones propiamente tales.

5.3.1 El elemento <xsl:template>

Una hoja de estilos XSL consiste en uno o más conjuntos de reglas, llamadas plantillas (*templates*). Cada plantilla contiene reglas para aplicar cuando un cierto nodo calza con ellas. Es el elemento **`<xsl:template>`** el que se utiliza para construir estas plantillas. El atributo **`match`** puede ser usado, tanto como para asociar una plantilla con solo un elemento XML, como para definir una plantilla para un documento XML completo. El valor del atributo **`match`** es una expresión XPath. Por ejemplo, **`match="/"`** define el documento completo. A continuación se muestra una versión simplificada del documento XSL de la sección anterior.

Ejemplo: Hoja de estilos XSL simplificada

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
.
.
.
</xsl:template>
</xsl:stylesheet>
```

Es necesario detenerse un momento en los fragmentos de éste documento. Dado que una hoja de estilos XSL es también un documento XML, ésta siempre comienza con una declaración XML, en este caso:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

El fragmento siguiente define que el presente documento es un documento hoja de estilo XSLT (junto con el número de versión y el atributo de espacio de nombres XSLT):

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Por otra parte, el fragmento que se muestra a continuación define una plantilla. El atributo **`match="/"`** asocia la plantilla con la raíz del documento XML fuente. El contenido dentro del elemento **`<xsl:template>`** define una parte del HTML a escribir en la salida:

```
<xsl:template match="/">
```

El resultado de la transformación, usando la versión completa de la plantilla, debería verse similar a lo siguiente:

Nacimientos

Nombre	Fecha de Nacimiento
.	.

El resultado del ejemplo anterior puede haber sido algo decepcionante, porque ningún dato fue copiado desde el documento XML hacia la salida. En la próxima sección se verá como resolver este problema.

5.3.2 El elemento <xsl:value-of>

El elemento **`<xsl:value-of>`** puede ser usado para extraer el valor de un elemento XML y agregarlo al resultado de la transformación.

Ejemplo: Elemento `<xsl:value-of>`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Nacimientos</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Nombre</th>
            <th>Fecha de Nacimiento</th>
          </tr>
          <tr>
            <td><xsl:value-of select="nacimientos/nacimiento/nombre"/></td>
            <td><xsl:value-of select="nacimientos/nacimiento/fechaDeNacimiento"/></td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Note que el valor del atributo **`select`** es una expresión XPath. Como se vio en el capítulo respectivo, una expresión XPath funciona similar a como se navega en un sistema de archivos, donde `"/"` sirve para seleccionar subdirectorios. El resultado de la transformación anterior debería verse similar a lo siguiente:

Nacimientos

Nombre	Fecha de Nacimiento
Juan	01-01-2006

El resultado de este nuevo ejemplo puede no haber sido todo lo reconfortante que se hubiera querido, puesto que sólo se copió al resultado una única línea de datos, del documento XML de origen. En la sección siguiente se verá como solucionar este nuevo problema.

5.3.3 El elemento <xsl:for-each>

El elemento **<xsl:for-each>** puede ser usado para seleccionar cada uno de los elementos de un cierto conjunto de nodos (*node-set*).

Ejemplo: Elemento <xsl:for-each>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Nacimientos</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Nombre</th>
            <th>Fecha de Nacimiento</th>
          </tr>
          <xsl:for-each select="nacimientos/nacimiento">
            <tr>
              <td><xsl:value-of select="nombre"/></td>
              <td><xsl:value-of select="fechaDeNacimiento"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

El resultado de la transformación anterior debería verse similar a lo siguiente:

Nacimientos

Nombre	Fecha de Nacimiento
Juan	01-01-2006
Pablo	02-02-2006
...	...

5.3.3.1 Filtrando el resultado de una transformación XSLT

También es posible filtrar el resultado de la transformación XSLT, añadiendo un criterio al atributo **`select`** en el elemento **`<xsl:for-each>`**. Debido a que los las expresiones evaluadas en este atributo pueden ser expresiones XPath, lo más natural es utilizar los filtros de predicado provistos por esa especificación, teniendo en cuenta reemplazar los signos convencionales por los que se muestran a continuación.

- **`<`** en vez de **<** (menor que).
- **`&le;`** en vez de **<=** (menor o igual que).
- **`>`** en vez de **>** (mayor que).
- **`&ge;`** en vez de **>=** (mayor o igual que).

Veamos ahora la hoja de estilos XSL ajustada con un filtro:

Ejemplo: XSL con filtro de predicado

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Nacimientos</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Nombre</th>
            <th>Fecha de Nacimiento</th>
          </tr>
          <xsl:for-each select="nacimientos/nacimiento[apellido='Perez']">
            <tr>
              <td>
                <xsl:value-of select="nombre"/>
              </td>
              <td>
                <xsl:value-of select="fechaDeNacimiento"/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
```

```
    </html>
  </xsl:template>
</xsl:stylesheet>
```

El resultado de la transformación de arriba, debería verse de la siguiente forma:

Nacimientos

Nombre	Fecha de Nacimiento
Juan	01-01-2006

5.3.4 El elemento <xsl:sort>

El elemento **`<xsl:sort>`** se utiliza para ordenar la salida. Para ordenar el resultado, simplemente se debe añadir un elemento **`<xsl:sort>`** dentro del elemento **`<xsl:for-each>`** en el archivo XSL:

Ejemplo: Elemento `<xsl:sort>`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Nacimientos</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Nombre</th>
            <th>Fecha de Nacimiento</th>
          </tr>
          <xsl:for-each select="nacimientos/nacimiento">
            <xsl:sort select="apellido"/>
            <tr>
              <td><xsl:value-of select="nombre"/></td>
              <td><xsl:value-of select="fechaDeNacimiento"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Note que el atributo **`select`** indica que elementos XML serán ordenados. En este caso se ordenará por apellido (el apellido de “Juan” es “Perez” y el apellido de “Pablo” es

"Gonzalez"). El resultado de la transformación de arriba, debería verse de la siguiente forma:

Nacimientos

Nombre	Fecha de Nacimiento
Pablo	02-02-2006
Juan	01-01-2006
...	...

5.3.5 El elemento <xsl:if>

El elemento **<xsl:if>** es usado como prueba condicional contra el contenido del archivo XML. El ejemplo a continuación muestra como es la sintaxis del uso de este elemento.

Ejemplo: Sintaxis de <xsl:if>

```
<xsl:if test="expression">
  ...
  ... Algún resultado si la evaluación de la expresión resulta verdadera…
  ...
</xsl:if>
```

Para añadir una prueba condicional, se debe agregar el elemento **<xsl:if>** dentro del elemento **<xsl:for-each>**, en el archivo XSLT.

Ejemplo: Elemento <xsl:if>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Nacimientos</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Nombre</th>
            <th>Fecha de Nacimiento</th>
          </tr>
          <xsl:for-each select="nacimientos/nacimiento">
           <xsl:if test="peso &lt; 2.5">
                  <tr>
              <td><xsl:value-of select="nombre"/></td>
              <td><xsl:value-of select="fechaDeNacimiento"/></td>
            </tr>
            </xsl:if>
```

```
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Note que el valor del atributo requerido (u obligatorio) **`test`**, contiene la expresión a ser evaluada. El código de más arriba sólo dará como resultado los nacimientos cuyo "peso" sea menor que **2.5** (el "peso" de Juan es "3" y el de Pablo es "2"). El resultado de la transformación de arriba, debería verse de la siguiente forma:

Nacimientos

Nombre	Fecha de Nacimiento
Pablo	02-02-2006

5.3.6 El elemento <xsl:choose>

El elemento **`<xsl:choose>`** es usado en conjunto con los elementos **`<xsl:when>`** y **`<xsl:otherwise>`** para expresar pruebas condicionales múltiples.

Ejemplo: Sintaxis de `<xsl:choose>`

```
<xsl:choose>
  <xsl:when test="expresion">
    ... algún resultado...
  </xsl:when>
  <xsl:otherwise>
    ... otro resultado...
  </xsl:otherwise>
</xsl:choose>
```

Para insertar pruebas condicionales múltiples contra un archivo XML, debe agregarse los elementos **`<xsl:choose>`**, **`<xsl:when>`** y **`<xsl:otherwise>`** al archivo XSL.

Ejemplo: Elemento `<xsl:choose>`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Nacimientos</h2>
        <table border="1">
```

```
          <tr bgcolor="#9acd32">
            <th>Nombre</th>
            <th>Fecha de Nacimiento</th>
          </tr>
          <xsl:for-each select="nacimientos/nacimiento">
            <tr>
              <td>
                <xsl:value-of select="nombre"/>
              </td>
              <xsl:choose>
                <xsl:when test="peso &gt; 2.5">
                  <td bgcolor="#ff00ff">
                    <xsl:value-of select="fechaDeNacimiento"/>
                  </td>
                </xsl:when>
                <xsl:when test="peso &lt; 2.3">
                  <td bgcolor="#cccccc">
                    <xsl:value-of select="fechaDeNacimiento"/>
                  </td>
                </xsl:when>
                <xsl:otherwise>
                  <td>
                    <xsl:value-of select="fechaDeNacimiento"/>
                  </td>
                </xsl:otherwise>
              </xsl:choose>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

El código de más arriba coloreará en negro el fondo de la “fechaDeNacimiento”, cuando el “peso” sea mayor que 2.5. Si el “peso” es menor que 2.3, entonces se coloreará gris oscuro. El resultado debería verse similar a lo siguiente:

Nacimientos

Nombre	Fecha de Nacimiento
Juan	01-01-2006
Pablo	02-02-2006
...	...

5.3.7 El elemento <xsl:apply-templates>

El elemento **<xsl:apply-templates>** aplica una plantilla al elemento actual o a sus nodos hijo (*child nodes*). Si se agrega un atributo **select** al elemento **<xsl:apply-templates>**, éste sólo procesará al elemento hijo que coincida con el valor del atributo. Es posible usar un atributo **select** para especificar el orden en que los nodos hijo serán procesados. Revisemos la hoja de estilos XSL que sigue:

Ejemplo: Elemento <xsl:apply-templates>

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Nacimientos</h2>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="nacimiento">
    <p>
      <xsl:apply-templates select="nombre"/>
      <xsl:apply-templates select="fechaDeNacimiento"/>
    </p>
  </xsl:template>
  <xsl:template match="nombre">
Nombre: <span style="color:#ff0000">
      <xsl:value-of select="."/>
    </span>
    <br/>
  </xsl:template>
  <xsl:template match="fechaDeNacimiento">
Fecha de Nacimiento: <span style="color:#00ff00">
      <xsl:value-of select="."/>
    </span>
    <br/>
  </xsl:template>
</xsl:stylesheet>
```

El resultado de la transformación debería como lo siguiente:

Nacimientos

Nombre: Juan
Fecha de Nacimiento: 01-01-2006

Nombre: Pablo
Fecha de Nacimiento: 02-02-2006

Nombre: ...
Fecha de Nacimiento: ...

5.4 Definición de Variables

En XSL existe la posibilidad de almacenar datos en variables definidas por el usuario. El ejemplo a continuación ilustra la sintaxis de la declaración de variables en XSL.

Ejemplo: Sintaxis de variables en XSL

```
<xsl:variable name="saludo" select="hola" />
```

El atributo **`select`** puede contener cualquier tipo de expresión XSL válida, entre ellas, expresiones XPath. En XSL no existe el concepto de variable global. Una solución para la acumulación de valores es mediante llamadas recursivas a funciones. Se definen dentro de un elemento **`<xsl:template>`** y no conservan el valor asignado, de la llamada de una plantilla a otra.

5.5 Definición de Funciones

En XSL es posible crear funciones añadiendo nuevas plantillas, que en realidad no existen en el archivo XML. El ejemplo a continuación ilustra la sintaxis de la declaración de funciones en XSL.

Ejemplo: Sintaxis de funciones en XSL

```
<xsl:template name="nombre-funcion">
  <xsl:param name="parametro1"/>
  <xsl:param name="parametro2" select="valor"/>
     …
  <!-- Puedo usar $parametro1 o $parametro2 si lo deseo -->
     …
 </xsl:template>
```

Más abajo se ilustra como llamar a una función definida por el usuario. La etiqueta o *tag* **<xsl:param>** permite definir parámetros con los que posteriormente se llamará a la función.

Ejemplo: Sintaxis de la llamada a una función XSL

```
<xsl:call-template name="nombre-funcion">
  <xsl:with-param name="parametro1" select="valor1" />
  <xsl:with-param name="paramatro2" select="valor2" />
</xsl:call-template>
```

Como se señaló en la sección anterior, es posible realizar llamadas recursivas a funciones.

5.6 Recapitulación

Resumiendo, XSLT significa "*XSL Transformations*" (Transformaciones XSL). Éste es el componente más importante de XSL. Una de sus funciones principales es transforma un documento XML en otro documento XML. Usa XPath para recorrer un documento XML y es una Recomendación del W3C.

Ejercicio

Construya una hoja de estilo XSL que transforme el documento XML realizado en el capítulo de XML, generando una página HTML que muestre todos los nacimientos, con su respectiva información, como una tabla, donde los pesos más altos y más bajos, así como las medidas más altas y bajas, estén destacados. Recomendación: Ayúdese con las funciones de XPath. Un resumen de estas funciones están en el siguiente link: http://www.w3schools.com/xpath/xpath_functions.asp.

5.7 Referencias

- XSL Transformations (XSLT) Version 1.0, W3C Recommendation 16 November 1999 http://www.w3.org/TR/1999/REC-xslt-19991116
- XSL Programmers http://www.zvon.org/o_html/group_xsl.html
- XSLT Tutorial http://www.w3schools.com/xsl/xsl_languages.asp
- Generación de páginas Web usando XSLT y XML
- http://geneura.ugr.es/~jmerelo/XSLT/

CAPITULO VI: ARQUITECTURAS ORIENTADAS A SERVICIOS

Cecilia Bastarrica, Sergio F. Ochoa, Daniel Perovich, Andrés Vignaga

Departamento de Ciencias de la Computación, Universidad de Chile.
{cecilia, sochoa, dperovich, avignaga}@dcc.uchile.cl

6.1 Arquitecturas de Software

La arquitectura del software es una primera aproximación al diseño de alto nivel donde se identifican los principales componentes, su interacción y sus dependencias. Según [Bass02], puede decirse que:

> *"La arquitectura de un programa o sistema de cómputo es la estructura o estructuras del sistema, los cuales comprenden los elementos de software, las propiedades externamente visibles de estos elementos, y las relaciones entre ellos".*

Esta definición tiene múltiples implicancias. Primeramente, la arquitectura es una abstracción de un sistema. De hecho, todo sistema tiene una arquitectura, lo cual no significa ni que ésta sea conocida, ni buena, ni apropiada para los propósitos de la aplicación. Si se desea que la arquitectura sea buena para la aplicación que se está desarrollando, es necesario planificarla, dado que el diseño de la arquitectura determinará en gran medida el cumplimiento de los requisitos funcionales, y también y muy especialmente los no funcionales. Existen otras definiciones, entre las cuales destacamos:

> *"La arquitectura del software es una partición prudente del todo en sus partes con una relación específica entre ellas, tal que permite a grupos de personas separadas por fronteras organizacionales, geográficas y/o temporales trabajar en forma cooperativa y productiva juntos para resolver un problema más grande que el que podrían hacer cada uno en forma independiente."*

Esta definición establece además la utilidad de contar con una arquitectura de software conocida que sirva como medio de comunicación entre todas las partes involucradas en el desarrollo de un sistema de software, y también como base para la administración del proceso de desarrollo.

6.1.1 Cualidades del Software

Si sólo la funcionalidad de un sistema de software fuese importante a la hora de hacer su diseño, cualquier sistema monolítico podría servir. Sin embargo, la realidad es otra.

Muchos otros atributos de calidad deben ser tenidos en cuenta para determinar la estructura y el comportamiento del sistema. Hacer un adecuado balance entre mantenibilidad, interoperabilidad, portabilidad, performance, seguridad, disponibilidad y reusabilidad, entre otros atributos, es esencial para obtener un sistema que cumpla las expectativas de las distintas partes interesadas.

Esto es algo más fácil de decir que de llevar a la práctica debido a que cualquier cambio en la arquitectura, de modo de mejorar uno de los atributos, en general estará afectando negativamente otro atributo. Y como si esto no fuese poco, el diseño de una arquitectura que tenga en cuenta los atributos deseados, sólo permitirá que el sistema resultante tenga estas características, pero no lo garantiza.

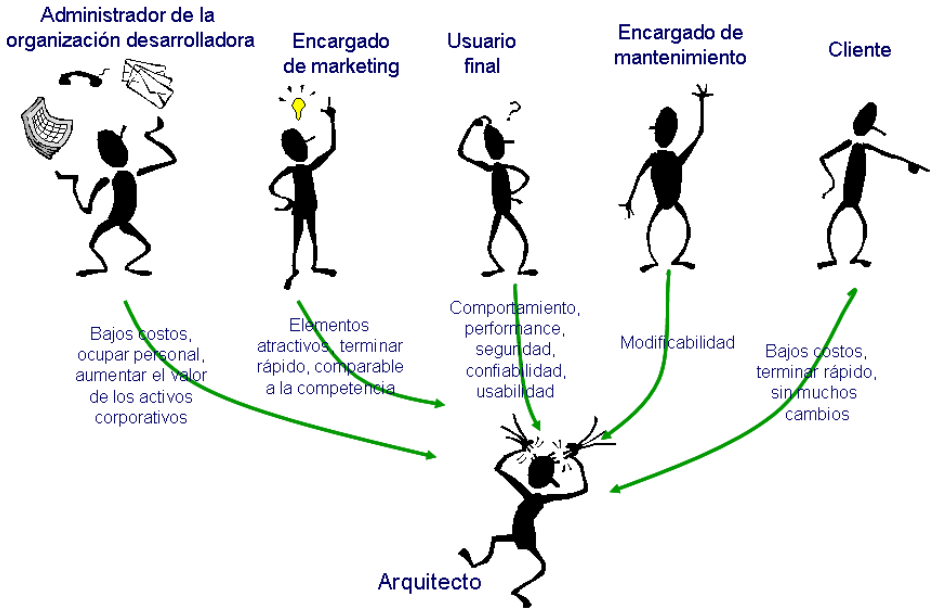


Figura 1: Stakeholders y atributos de calidad

En esta sección se presenta el patrón de arquitectura Orientación a Servicios, el cual propone la definición de servicios independientes que ofrezcan las funcionalidades requeridas en un negocio, con interfaces invocables bien definidas, que pueden ser orquestadas formando procesos de negocio.

Podemos clasificar los atributos de calidad del software en dos grandes grupos: aquellos que pueden comprobarse durante la ejecución del software y aquellos que no pueden comprobarse durante la ejecución. Dentro del primer grupo se encuentra la performance, seguridad, disponibilidad, funcionalidad y usabilidad. Dentro de los segundos, y no menos importantes, están la modificabilidad, portabilidad, reusabilidad, integrabilidad y verificabilidad. La tabla 1 describe cada uno de estos atributos.

Tabla 1: Atributos de Calidad del Software

Atributos observables durante la ejecución	Performance	*Tiempo que requiere el sistema para responder a un evento o estímulo, o bien el número de eventos procesados en un intervalo de tiempo*
	Seguridad	*Medida de la capacidad del sistema para resistir intentos de uso y negación de servicios a usuarios no autorizados sin restar servicios a los usuarios autorizados*
	Disponibilidad	*Proporción del tiempo que el sistema está en ejecución*
	Funcionalidad	*Habilidad de un sistema para hacer la tarea para la cual fue creado*
	Usabilidad	*Aprendibililidad, Eficiencia, Recordabilidad, Propenso a errores, Manejo de errores, Satisfacción*
Atributos no observables durante la ejecución	Modificabilidad	*Habilidad para hacer cambios al sistema de una forma rápida y poco costosa*
	Portabilidad	*Habilidad de un sistema para ejecutar en diferentes ambientes*
	Reusabilidad	*Capacidad que tiene un sistema para que su estructura o alguna de sus componentes puedan ser usadas en futuras aplicaciones*
	Integrabilidad	*Habilidad para hacer que piezas de software desarrolladas separadamente trabajen correctamente juntas*
	Verificabilidad	*Facilidad con la cual el software puede mostrar sus defectos*

Dada la naturaleza diversa de los potenciales atributos de calidad deseados, se ha visto que es necesario contar con distintas "vistas" que pongan de manifiesto las distintas facetas de un sistema de software a la hora de especificar su arquitectura. Es así como la definición de la arquitectura de un software deberá estar constituida por más de una vista que permita analizar tanto sus cualidades dinámicas como las estáticas.

6.1.2 Estilos de Arquitectura

La arquitectura del software es un artefacto complejo que no puede expresarse en una forma simple y unidimensional. Según [Clem02], documentar una arquitectura es documentar las vistas más relevantes y luego agregar la información que se aplica a más de una vista. Una vista es la representación de un conjunto de elementos del sistema y las relaciones asociadas con los mismos. Las vistas usadas dependen de las necesidades y el uso que se daría a la arquitectura, así como también a los atributos de calidad requeridos. Existen tres tipos de vistas: módulos, componentes y conectores, y distribución. El tipo de vista utilizado limita los elementos y relaciones que contiene una vista de ese tipo.

Módulos: Estructura en términos de unidades de implementación. Permiten analizar las cualidades estáticas del software.

Componentes y conectores: Estructura en términos de elementos interactuantes durante su ejecución. Permiten analizar cualidades dinámicas del sistema.

Distribución: Relación entre el sistema de software y las estructuras del ambiente. Permiten analizar tanto cualidades estáticas como dinámicas.

Un estilo de arquitectura es una especialización de un tipo de vista que establece los elementos y las relaciones entre ellos, así como una serie de restricciones de cómo pueden

usarse. Un estilo define una familia de arquitecturas que satisface estas restricciones. La elección de un estilo determina los elementos y las restricciones que habrá que documentar. Los estilos también son denominados *patrones* de arquitectura [Busch96], como analogía con los patrones de diseño [Gamm95].

Distintos estilos de arquitectura promueven distintos atributos de calidad. Es así que, dependiendo de los atributos de calidad identificados, será el estilo de arquitectura que deberá usarse para modelar el sistema. La tabla 2 muestra una clasificación de los estilos, según el tipo de vista al cual pertenecen y los atributos de calidad que promueven.

Tabla 2: Estilos y Atributos de Calidad

Tipos de Vistas	Estilos	Cualidades
Módulos	Capas	Mantenibilidad, reusabilidad, portabilidad
	Uso	Mantenibilidad, interoperabilidad
	Descomposición	Administración del desarrollo, mantenibilidad
Componentes y Conectores	Cliente-servidor	Performance, seguridad, escalabilidad
	Tubos y filtros	Reusabilidad, mantenibilidad
	Repositorio	Integrabilidad, extensibilidad
Allocation	Asignación	Administración de costos, planificación
	Implementación	Gestión de configuración, mantenibilidad
	Distribución	Performance, seguridad, escalabilidad

6.2 Arquitecturas Orientadas a Servicios

Las arquitecturas orientadas a servicios (SOA, Service Oriented Architectures) corresponden a un estilo de componentes y conectores. Los atributos de calidad esenciales que promueven son la interoperabilidad, la flexibilidad, la escalabilidad y la reusabilidad.

Los sistemas de software con arquitecturas de servicios, se estructuran como una serie de aplicaciones que exponen los servicios que proveen, de tal modo que otras aplicaciones puedan usarlos. Es así que distintos procesos se articulan como la composición de invocaciones a servicios disponibles. La reusabilidad de estos sistemas se basa en que los servicios pueden ser implementados con aplicaciones legadas a las cuales se les construyen una nueva interfaz, a través de la cual otras aplicaciones interactúan.

Sin embargo, aplicaciones legadas suelen estar desarrolladas usando tecnologías y paradigmas diversos, lo cual hace necesario estandarizar las interfaces para permitir la interoperabilidad. En sistemas con arquitecturas de servicios, XML se usa como lenguaje básico para la codificación de los mensajes que todas las aplicaciones envían y/o reciben. También se usa XML para la especificación de cada uno de los servicios disponibles y los protocolos de interacción a través de SOAP (Simple Object Access Protocol).

Además de los servicios, existen en los sistemas SOA, unos servicios especializados llamados UDDI (Universal Description Discovery and Integration). En ellos se registran los servicios disponibles en el sistema, así como su interfaz de acceso (WSDL, Web Service Description Language) y su localización. Así, cada vez que una aplicación requiera el uso de un servicio, podrá buscarlo en el UDDI y accederlo sin tener conocimiento anticipado de sus características y localización. Además, con este esquema de acceso, los servicios pueden evolucionar modificando tanto su interfaz como su localización, sin afectar a sus potenciales clientes, promoviendo así la flexibilidad de los sistemas. También, el contar con directorios UDDI permite registrar nuevos servicios durante la ejecución del sistema, que podrán ser utilizados a partir de ese momento.

En esta sección se presenta el patrón de arquitectura Orientación a Servicios, el cual propone la definición de servicios independientes que ofrezcan las funcionalidades requeridas en un negocio, con interfaces invocables bien definidas, que pueden ser orquestados formando procesos de negocio.

6.3 Patrón para Arquitecturas Orientadas a Servicios

El patrón de arquitectura que se presenta en esta sección incluye el contexto, el problema a resolver, la solución propuesta, implantación, un ejemplo y sus usos conocidos. Se espera que este patrón ayude al lector a identificar nichos y variantes para utilizar arquitecturas orientadas a servicios.

6.3.1 Contexto

Definición de un ambiente integrado de múltiples sistemas, existentes o nuevos y potencialmente heterogéneos, con requerimientos funcionales muy cambiantes.

6.3.2 Ejemplo

La línea de negocios de una agencia de viajes se ve potenciada en el contexto de la venta de paquetes turísticos. Dos aspectos importantes rigen a este negocio. Por un lado, los operarios que trabajan junto al cliente, solucionando sus necesidades, ofreciéndoles paquetes turísticos a su medida. Por otro, los operarios encargados de realizar contactos y alianzas de negocio con empresas de servicios locales y extranjeras, con el objeto de conseguir precios o servicios diferenciadores en su mercado. La tecnología de la información (TI) asociada al negocio juega un rol preponderante. Además de funcionalidades básicas de gestión de cuentas de clientes y ventas, se requieren funcionalidades asociadas a las competencias de la agencia. Por tal motivo, la infraestructura tecnológica debe permitir a los operarios que atienden a los clientes el acceso a todas las oportunidades de negocio obtenidas por los otros operarios, las cuales son altamente dinámicas.

El proceso de venta de un paquete turístico a medida inicia cuando un cliente solicita a un operario un destino, fechas y los servicios turísticos de preferencia. El operario consulta la cartera de alianzas de negocio de la agencia de viajes e inicia los contactos para ofrecer

diferentes alternativas al cliente. Para ello, confirma disponibilidad en el sistema de reservas de vuelos en uso en la agencia, y utiliza los sistemas o contacta a las empresas de servicios de hotelería, alquiler de autos, etc., en función de las preferencias del cliente. El operario presenta luego al cliente un conjunto reducido de presupuestos diferentes, basándose en la respuesta obtenida de las empresas de servicios. El cliente opta por uno de ellos y el operario procede a realizar las reservas. Previo a que las reservas caduquen, el cliente confirma la compra del paquete, por lo que el operario pasa la información a la sección de contabilidad, donde el cliente paga por el paquete contratado. La sección de contabilidad corrobora la validez del mecanismo de pago del cliente, como ser la tarjeta de crédito por ejemplo. Una vez consumado el pago, la sección contabilidad informa al operario, el cual confirma las reservas en las empresas de servicios y solicita a la sección de emisión y entrega de la agencia de viajes, que emita la carpeta de documentación y se la haga llegar al cliente. Una vez entregado, la sección de emisión y entrega notifica que la venta del servicio esta cumplida.

La agencia de viajes quiere mejorar el proceso de negocio de venta de un paquete turístico ya que toma un tiempo excesivo en llevarse a cabo. Para ello, requiere que la plataforma tecnológica dé soporte a mejoras de interoperabilidad entre los sistemas internos en funcionamiento en la empresa y con los sistemas de las empresas de servicios turísticos. Asimismo, dado que estas últimas presentan una alta rotatividad, se requiere que la plataforma se adapte a las variantes en las empresas con las que hay alianzas de negocio. Además, la agencia de viajes quiere proveer un portal Web de la empresa, en el cual una mayor cantidad de potenciales clientes consulten por paquetes turísticos.

La división de TI de la agencia de viajes se enfrenta con varios problemas ante este requerimiento del negocio. En primer lugar, las aplicaciones internas a la empresa están pobremente preparadas para ser integradas e interoperar entre sí: están desarrolladas utilizando distintas tecnologías y están corriendo en plataformas diferentes, son aplicaciones monolíticas que en ocasiones ofrecen APIs propietarias para su uso por otros sistemas, están conectadas entre sí por mecanismos de comunicación ad-hoc, y algunas ofrecen funcionalidades repetidas como la gestión de información de clientes. En segundo lugar, la mayoría de las empresas proveedoras de servicios no están alineadas con el objetivo de la automatización de este proceso de negocio: las funcionalidades ofrecidas no son uniformes en cada área de servicios, y los mecanismos de interoperabilidad difieren en la tecnología.

Por otra parte, la incorporación y el reemplazo del acceso a los sistemas de las empresas de servicios es difícil de realizar ya que requiere la adaptación de los sistemas y/o procedimientos de los sistemas de los operarios para que interactúen con los nuevos sistemas. Además, la adecuación del proceso de negocio hacia nuevas oportunidades, como ser por ejemplo la inclusión de nuevos servicios a los paquetes turísticos o la notificación al cliente por medios electrónicos (telefonía movil, PDAs, e-mail), no es ágil ni flexible ya que involucra tareas de mantenimiento adaptativo en algunos de los sistemas de la agencia de viajes.

6.3.3 Problema

En las últimas décadas los sistemas empresariales han experimentado un gran crecimiento en términos de tamaño, complejidad y dificultad tecnológica. A su vez, las organizaciones son cada vez más dependientes de sus sistemas informáticos, los cuales tienen un alto impacto en la operativa del negocio. Asimismo, la Tecnología de la Información (TI) debe estar alineada al negocio, siendo éste su principal influenciador. En el contexto de un negocio, uno de los principales objetivos es aumentar el retorno de la inversión (ROI). A pesar de que la TI naturalmente representa un factor potenciador en este objetivo, la dificultad de adaptación de ésta al contexto fuertemente dinámico de los negocios no está surtiendo el efecto esperado.

La problemática que enfrenta un negocio actualmente continúa siendo la misma que en años anteriores. Su foco está en su propia área de negocios, en lo que lo hace único y exitoso, diferenciándose de la competencia. Por este motivo, la integración con nuevos socios, ya sea mediante alianzas, fusiones o adquisiciones, y con clientes, permite enfocarse en su rol en la línea de negocios. Igualmente, el negocio debe ser receptivo; debe tener la habilidad para responder con agilidad a las cambiantes demandas de los clientes, a las nuevas oportunidades de negocio y a las amenazas externas. Por último, un nuevo influenciador sobre el negocio ha surgido en los últimos años: Internet. El bajo costo de la conexión y el acceso generalizado a nivel global a la red brinda un gran abanico de oportunidades para los negocios, requiriendo así llevar el negocio on-line, tanto para llegar a (más) clientes como para interoperar con (nuevos) proveedores.

El negocio impone a la TI exigencias en diferentes aspectos, de forma que ésta se alinee con los influenciadores del negocio; en otras palabras, el negocio determina las líneas generales de la TI que lo acompaña. De esta manera, es necesario que la TI tenga flexibilidad y agilidad de adaptación a los cambios en los requerimientos. El soporte para nuevos negocios debe brindarse en forma veloz y además la reutilización de los recursos existentes debe aumentarse, reduciendo así el tiempo de puesta en producción de una nueva funcionalidad o de la refactorización de funcionalidades. Así, los costos del desarrollo, tanto en tiempo, horas de desarrollador, y en aspectos tecnológicos, debe reducirse.

Por otra parte, el propio estado del arte de la TI, tanto a nivel mundial como dentro de la propia organización, impone influenciadores adicionales al soporte que el negocio requiere. Primero, las organizaciones cuentan generalmente con gran heterogeneidad en los sistemas existentes, tanto en la finalidad de los mismos como en la tecnología empleada; es necesario manejar esta heterogeneidad. Asimismo, el soporte de TI debe tener la capacidad de adaptarse a cambios en la propia tecnología. Segundo, la gestión de TI debe abarcar ambientes aún más complejos que en el pasado. Los sistemas existentes son independientes entre sí, o están acoplados fuertemente, utilizando en consecuencia múltiples interfaces para su interoperabilidad. Además, deben mantenerse los sistemas legados ya que hay conocimiento y capital invertido en ellos, por lo que no pueden ser simplemente desechados. Tercero y último, no se pueden desaprovechar las oportunidades de crear insumos reusables. El reuso debe ser planificado y no detectado a posteriori. Este último enfoque es el que ha llevado a tener actualmente en las empresas aplicaciones que brindan

funcionalidades redundantes. A su vez, factores tecnológicos o de diseño han evitado que las funcionalidades sean reutilizables.

En este escenario organizacional, el (re) diseño del ambiente integrado o ecosistema debe balancear las siguientes *fuerzas*:

- Encapsular las funcionalidades del negocio.
- En el contexto en que existen diferentes proveedores de una misma funcionalidad, con distintos atributos de QoS, tener la flexibilidad de optar por el más conveniente al momento de requerir la funcionalidad.
- Permitir el acceso a las funcionalidades brindadas por sistemas legados.
- Permitir a cualquier consumidor de funcionalidades el acceso a ellas, salvando los impedimentos tecnológicos que éste presente.
- Permitir el acceso a la declaración de las funcionalidades disponibles y que sea posible determinar los proveedores correspondientes en función de dicha declaración y de los atributos de QoS.
- Lograr transparencia de ubicación de los proveedores de funcionalidades.
- Lograr que los consumidores de funcionalidades estén débilmente acoplados a los proveedores de las mismas.
- Permitir la definición de nuevas funcionalidades, en términos de otras existentes, donde la lógica de las primeras pueda cambiar con gran flexibilidad.
- Eliminar redundancia de funcionalidades.
- Brindar la capacidad de gerenciar las solicitudes de funcionalidades en términos de seguridad, ubicación, registro, auditoria, entre otros.

6.3.4 Solución

Orientación a Servicios es un patrón de arquitectura para la integración de sistemas basado en el concepto de servicio. En este contexto, se entiende por *servicio* a un recurso abstracto que encapsula una funcionalidad reusable de un negocio tal que:

- es definido explícitamente por una contrato independiente de la tecnología que especifica una colección de mensajes (interfaz) y reglas de secuenciamiento de éstos (protocolo) en el uso del servicio ,
- es utilizado por sistemas o componentes consumidores de servicios,
- es realizado por sistemas o componentes proveedores de servicios,
- acopla débilmente a los consumidores con los proveedores, y
- el enlace entre consumidores y proveedores, y las ulteriores invocaciones son realizadas (es enlazado e invocado) a través de protocolos de comunicación que enfatizan la transparencia de ubicación y la interoperabilidad.

Así, el patrón propone entender las funcionalidades de uno o más sistemas en términos de servicios. Se entiende por *consumidor de servicio* a un sistema de software que requiere y utiliza a uno o más servicios para su funcionamiento. Por *proveedor de servicio* se entiende a un sistema de software que ofrece y realiza todas las funcionalidades descritas por un servicio. Un servicio puede ser consumido por uno o más consumidores y realizado por uno o más proveedores. Conceptualmente, un consumidor de servicio descubre el servicio que

describe la funcionalidad que requiere. Localiza luego un proveedor que lo ofrezca y le envía una solicitud de servicio a tal proveedor, el cual envía una respuesta al consumidor. A su vez, los propios proveedores pueden requerir servicios para su funcionamiento, comportándose así también como consumidor de servicios. El consumidor de servicios esta débilmente acoplado al proveedor ya que el proveedor particular y su ubicación no son conocidos previamente por el consumidor, sino descubierta al momento de requerir el servicio.

El mecanismo por el cual un consumidor de servicio descubre un servicio y localiza al proveedor, de forma de mantener bajo el acoplamiento, es mediante un intermediario, que, agregando un nivel de indirección, permite independizar al consumidor del proveedor. El rol de intermediario lo juega el directorio de servicios, el cual sigue la estrategia del patrón de arquitectura Broker. Se entiende por *directorio de servicios* a un componente de software que registra la declaración de los servicios disponibles en un ecosistema de consumidores y proveedores, conoce la información de enlace hacia los proveedores de servicios, y mantiene las propiedades del servicio tanto semánticas como de calidad de servicio. Así, un proveedor de servicios publica los servicios que ofrece en el directorio de servicios, un consumidor de servicios consulta el directorio para solicitar un proveedor que ofrezca el servicio que requiere de forma que cumpla con los aspectos semánticos y de calidad deseados, y es el directorio quién ofrece la información de enlace entre ellos.

Un esquema donde los propios consumidores de servicios sean quienes administran tanto el acceso como las invocaciones a los proveedores de servicios, no es deseable, puesto que impide gerenciar los servicios y mantener aspectos generales del ecosistema de consumidores y proveedores, en términos de seguridad, transparencia de ubicación, registro de actividad, auditoría, entre otros. A tales efectos un *bus de servicios* oficia de intermediario entre consumidores y proveedores, similar a un despachador en el patrón Client-Dispatcher-Server, el cual transporta tanto las solicitudes de servicios como las respuestas al destino correspondiente, teniendo adicionalmente la capacidad de procesarlas, y por tanto, gerenciar los servicios en los términos ya mencionados.

Los servicios pueden componerse, es decir, se pueden combinar de acuerdo a cierta lógica de negocio, lo que permite a un servicio compuesto modelar un proceso de negocio. Se entiende por *proceso de negocio* a una colección de actividades diseñada para producir una salida específica para un determinado cliente. Es un ordenamiento específico de actividades de trabajo a lo largo del espacio y el tiempo, con un comienzo, un fin y entradas y salidas claramente definidas. Dado que un proceso de negocio puede ser entendido y ofrecido como servicio, es posible combinarlo con otros servicios en otro proceso de negocio de granularidad aún mayor. La implementación de la lógica de los procesos de negocios suele aparecer embebida en forma explícita en la propia implementación de los sistemas que los soportan, lo cual resulta por un lado en una pobre trazabilidad entre la implementación del proceso y su especificación, y por otro en rigidez a la hora de efectuar modificaciones al flujo del proceso. En tal sentido, una especificación de la lógica de un proceso que describa el orden o flujo de las actividades necesarias para producir la salida esperada, presentaría trazabilidad directa con la especificación del proceso de negocio. Por otra parte, un *motor de procesos* sería capaz de procesar dinámicamente la definición, invocando los servicios necesarios en el orden adecuado para producir el resultado final. Este esquema resulta

preferible al ya mencionado, puesto que la mejora en la trazabilidad provee flexibilidad, simplificando la modificación de la lógica de los procesos de negocio, incluso dinámicamente.

6.3.4.1 Estructura

En la solución participan cinco tipos de componentes: consumidores de servicios, proveedores de servicios, motor de procesos, directorio de servicios y bus de servicios.

Un *consumidor de servicios* es un componente de software que requiere un servicio. Es la entidad que inicia el descubrimiento del servicio en el registro de servicios, e invoca la función de servicio a través del bus de servicios. El consumidor de servicios interactúa con un proveedor de ese servicio vía el bus enviando mensajes de acuerdo al protocolo del contrato del servicio. En casos en que un consumidor de servicios no tenga la capacidad práctica de consumir servicios es posible crear, en base al patrón de diseño Proxy y en forma provisoria o permanente, un componente de software que proporcione al consumidor un API mediante la cual sea efectivamente capaz de consumir el servicio requerido.

Un *proveedor de servicios* es un componente de software que realiza un servicio, es una entidad ubicable que acepta y ejecuta pedidos de los consumidores. Puede tratarse desde un sistema mainframe, un componente de software, hasta un objeto que ejecuta la solicitud de servicio. Realizar un servicio significa implementar la interfaz especificada por el contrato del servicio, respetando a su vez el protocolo. Un proveedor de servicios publica su contrato de servicio y su dirección física en el directorio de servicios, a través del bus de servicios, con el objetivo de poder ser accedido por los consumidores de servicios, también a través del bus. Simétricamente a los consumidores, es posible que los componentes que dispongan de funcionalidades de relevancia para el negocio no tengan la capacidad de proveerlas en términos de servicios. Este es un caso típico de sistema legado, el cual fue construido eventualmente en forma monolítica, pero de todas formas no fue preparado para ser proveedor de servicios. Usualmente, la reimplementación de las funcionalidades de interés contenidas en el sistema legado no es una opción viable. En esos casos, se construye un componente de software basado en el patrón de diseño Adapter que adapte una API del sistema legado al enfoque de servicios, permitiendo de esta forma que dicho sistema sea efectivamente un proveedor de servicios.

El *motor de procesos* es el componente encargado de la ejecución u orquestación de procesos de negocio. Contiene las definiciones de un conjunto de procesos de negocios que a su vez realizan servicios (por tanto el motor puede ser entendido como un proveedor de servicios, que ofrece los servicios representados por los procesos de negocio que maneja). Cada vez que el servicio asociado a un proceso es requerido, su definición es instanciada y ejecutada por el motor, quien a su vez administra su ciclo de vida. El motor además dirige los mensajes que recibe por parte del bus de servicios hacia la instancia de proceso de negocio adecuada; esto se denomina correlación de mensajes. Por último y como parte de la orquestación de procesos, al llevar a cabo las actividades que componen el proceso, el motor se encarga de realizar los envíos de mensajes para la consumición de los servicios que compongan al proceso en ejecución. Así, el motor de procesos también puede entenderse como un consumidor de servicios.

El *directorio de servicios* es un componente accesible que contiene información relativa a los servicios disponibles. Es una entidad que acepta a través del bus los contratos de servicio de los proveedores, los almacena, y los provee junto a la información de invocación (información de enlace) a los consumidores.

El *bus de servicios* ofrece funcionalidades de comunicación entre los componentes de la arquitectura. Así, permite que un consumidor de servicios solicite una funcionalidad a un proveedor de servicios determinado (eventualmente el motor de procesos), en base a la información de enlace previamente provista por el directorio de servicios. En forma simétrica, hace llegar las respuestas de los proveedores de servicios a los consumidores de servicios. Además, permite que los proveedores de servicios publiquen en el directorio de servicios los contratos de los servicios que proveen. Provee además un API para incorporar definiciones de procesos de negocios en el motor de procesos. Al centralizar y administrar toda la comunicación, el bus de servicios puede administrar los envíos de mensajes, y así proveer funcionalidades adicionales. Entre ellas se encuentran seguridad (autorizar solicitudes, encriptar y desencriptar datos, validar información), movilidad (permitir que quien provee un servicio sea cambiado de ubicación por razones de performance o disponibilidad), registro de actividad (permitir auditorias, monitoreo y uso de métricas), ruteo dinámico de mensajes (ser tolerante a fallas y balancear la carga), calidad de servicio (manejo de transacciones y compensaciones), e integración (permitir integración entre ecosistemas). El diagrama de clases de la Figura 2 muestra la estructura estática de los componentes presentes en la arquitectura. Se puede apreciar que el bus de servicios constituye un mediador entre el resto de los componentes al estar asociado con cada uno de ellos.

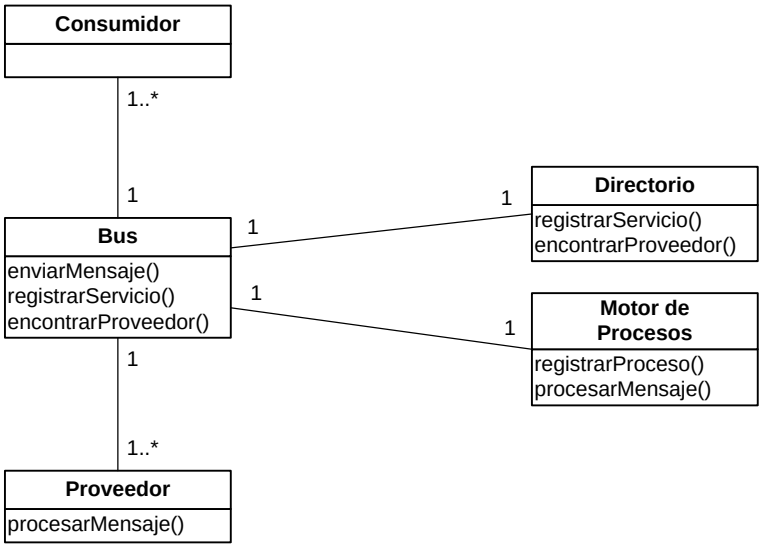


Figura 2: Estructura de la Propuesta

Como observación final, cabe notar que las funcionalidades del directorio de servicios, e incluso los componentes Proxy y Adapter correspondientes a los consumidores y proveedores de servicios respectivamente, podrían ser incorporadas como capacidades del

bus de servicios. Para la especificación del presente patrón de arquitectura se decidió considerar dichas funcionalidades en forma independiente del bus de servicios, ya que en el caso particular del directorio de servicios, se trata de un componente relevante para la solución que a su vez presenta un nivel de abstracción comparable al del resto de los componentes presentados.

6.3.4.2 Dinámica

A continuación se presentan y discuten algunos de los escenarios de funcionamiento de la estructura propuesta como solución. Los escenarios son descritos en términos de interacciones de alto nivel entre instancias de los componentes definidos.

Escenario I. Un proveedor de servicios registra ante el directorio de servicios un servicio que ofrece: registra el contrato del mismo e información del enlace al proveedor. Esta información de enlace permite que al proveedor se le cursen solicitudes de ese servicio. La comunicación entre el proveedor y el directorio se realiza mediante el bus de servicios.

Escenario II. Este caso es análogo al anterior pero referente al registro de una nueva definición de proceso de negocio. Si bien el registro de un nuevo proceso puede entenderse como una solicitud de servicio, en todo caso dicha solicitud será emitida por el administrador del sistema o un desarrollador, y no otra aplicación a nivel del negocio. Por tal razón, típicamente la interacción en este escenario se produce a partir de una aplicación que se comunica directamente con el motor de procesos (e.g. una consola de administración del propio motor), o incluso utilizando el bus de servicios mediante un servicio especial de administración.

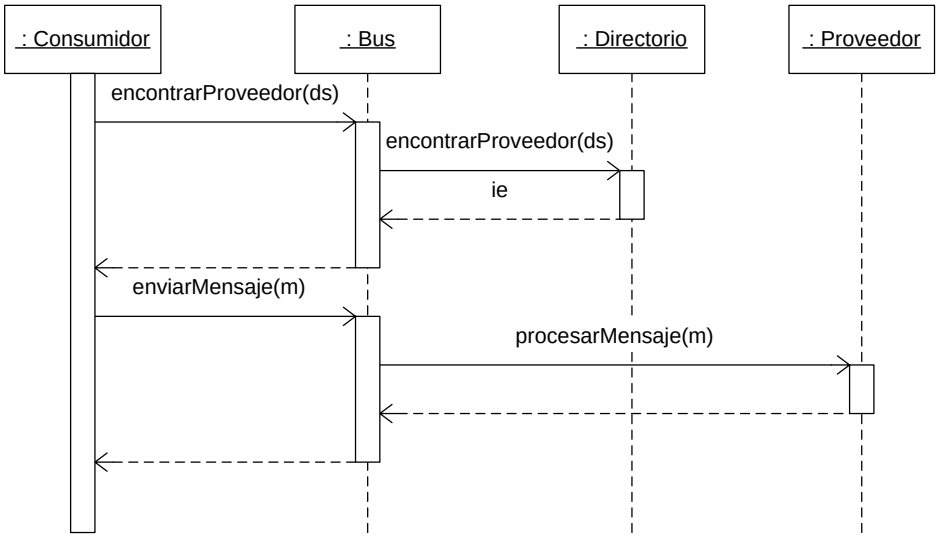


Figura 3: Diagrama de Secuencia de la Estructura Propuesta

Escenario III. Un consumidor de servicios consume un servicio; la interacción entre los elementos se presenta en la Figura 3. El consumidor de servicios primeramente, utilizando la descripción del servicio que desea consumir, procura un proveedor del mismo. Esto lo

hace a través del bus de servicios, el cual se comunica directamente con el directorio de servicios. El directorio típicamente provee al bus la información de enlace mediante la cual se podrá acceder a un proveedor del servicios en cuestión. En el caso en que exista más de un proveedor para dicho servicio, el bus de servicios puede optar por uno de los proveedores (si no es el bus quien tiene esta responsabilidad, la elección la debe realizar el consumidor propiamente). El bus de servicios mantiene la información de enlace entre el consumidor y el proveedor. Luego, el consumidor envía un mensaje al bus de servicios, el cual la retransmitirá al proveedor referido. Notar que el bus de servicios tiene por lo tanto, la responsabilidad de correlacionar los mensajes de un consumidor con los proveedores de cada servicio que esté consumiendo en un determinado instante en el tiempo. Por simplicidad se omiten los eventuales mensajes que siguen al primero. Como ya fuera discutido antes, en caso de que el bus de servicios sea lo suficientemente sofisticado, el mensaje siguiente podría llegar a ser ruteado a otro proveedor del mismo servicio por razones de eficiencia y/o calidad de servicio.

Escenario IV. Este caso corresponde a la consumición de un proceso de negocio. Debido a que un proceso de negocio puede ser entendido como un servicio, y por lo tanto el motor de procesos como un proveedor de servicios, este escenario no presenta grandes diferencias conceptuales respecto al anterior; en lugar de ser un proveedor cualquiera quien procesa los mensajes de solicitud de servicio, es el motor de procesos quién lo hace. La correlación entre mensajes de solicitud de servicio y las diferentes instancias de proceso de negocio es resuelto por el motor de procesos internamente y por lo tanto no es de relevancia en este escenario. Finalmente, al ejecutar un proceso de negocio es necesario consumir servicios, por lo cual intercalados con algunos de los mensajes de solicitud de servicio entrantes, el motor de procesos consume otros servicios. Para dicha parte de la interacción aplica nuevamente el escenario anterior, sin embargo es el motor de procesos quien ocupa el lugar del consumidor de servicios.

6.3.5 Implantación

La siguiente secuencia de pasos describe el proceso de adopción y aplicación de una Arquitectura de Servicios. El proceso aquí presentado asume que se cuenta inicialmente con un conjunto de sistemas independientes o conectados en forma directa entre ellos, y que se quiere construir a partir de ellos una arquitectura que provea los beneficios de la orientación a servicios. Este contexto, que coincide con el motivador del patrón, es el que ocurre con mayor frecuencia. Sin embargo, el proceso no es directamente aplicable a sistemas nuevos que deben ser construidos desde su inicio, sin ningún sistema previo a reaprovechar. Para este tipo de escenarios el estado del arte presenta escasas propuestas, conformándose las mismas bajo el nombre de SOMA (Software-Oriented Modeling and Architecture). Estas propuestas no son atacadas aquí.

El punto de partida es un conjunto de sistemas existentes completamente independientes o pobremente integrados. Por esto último, se entiende que para cada par de sistemas que interoperan se cuenta con las respectivas interfaces donde el mecanismo de comunicación es propietario. Además, estás aplicaciones ofrecen funcionalidades redundantes y en distintos niveles de granularidad.

1. *Ofrecer las funcionalidades legadas de interés como servicios*. Definir servicios para las funcionalidades de interés para el negocio, ofrecidas por los sistemas existentes. Esto se realiza desarrollando adaptadores cuando sea necesario.

2. *Eliminar la funcionalidad redundante del conjunto de servicios ofrecidos*. Para aquellos servicios identificados en el paso 1, identificar servicios redundantes unificando los correspondientes en un único servicio. Mientras los sistemas existentes no sean refactorizados, crear un proveedor de servicios que ofrezca este nuevo servicio y coopere con dichos sistemas (de forma de preservar la integridad).

3. *Permitir a consumidores de interés que soliciten servicios*. Una vez disponible la masa crítica de servicios, los sistemas de interés que requieran funcionalidades de otros sistemas deben solicitar dichas funcionalidades, mediante los servicios. Identificar qué servicio debe consumir cada sistema existente, y reemplazar el mecanismo de solicitud por el consumo de servicios, desarrollando proxies cuando sea necesario.

4. *Dar flexibilidad en la elección del proveedor de servicio*. Para independizar la conexión directa desde los proxies hacia los adaptadores, crear un directorio de servicios bajando el acoplamiento y obteniendo reemplazabilidad.

5. *Proveer un único punto de acceso a los servicios*. El directorio permite independizar a consumidores de proveedores en tiempo de diseño, pero no en ejecución. Para evitar que el consumidor se comunique directamente en tiempo de ejecución, incorporar un integrador de servicios encargado del ruteo de las solicitudes y respuestas de servicio. Asimismo, el integrador puede encargarse de transformaciones en los mensajes para solucionar problemas de incompatibilidad de plataformas.

6. *Proveer funcionalidades de administración de la integración a nivel empresarial*. Reemplazar el integrador por un bus de servicios, el cual, además de cumplir tal función, provee mecanismos de seguridad, registro, políticas, eventos, etc.

7. *Proveer flexibilidad en la realización de los procesos del negocio*. Incorporar un motor de procesos de negocio que permita el registro y ejecución de procesos de negocio.

8. *Refactorizar y consolidar, o incluso reemplazar, sistemas existentes*. La presencia de un motor de procesos de negocio posibilita la definición de funcionalidades de negocio en forma flexible y ágil. En este nivel, los sistemas existentes como tales, han migrado a ser los proveedores de servicios compuestos por los procesos de negocios. Teniendo tal herramienta, se puede refactorizar y consolidar los sistemas existentes de forma tal, que ofrezcan las funcionalidades de negocio esperadas, en forma orquestada con los otros servicios existentes. Los sistemas tienden a ser divididos en varios proveedores de servicio independientes, y algunos de éstos a combinarse para ofrecer nuevos servicios.

En el proceso de adopción del patrón de arquitectura se crean un conjunto de componentes de software que son sugeridos por el patrón y que son independientes del negocio particular, a saber: el directorio de servicios, el motor de procesos y el bus de servicios. Desarrollar en forma propietaria estos componentes tiene un alto grado de dificultad y pone en riesgo la interoperabilidad del ecosistema con otros ecosistemas de interés. Por tal motivo, se recomienda la adopción de productos off-the-shelf que provean las funcionalidades deseadas. Es de esperar, que estos productos estén basados en los estándares de la industria, maximizando así la capacidad de interoperabilidad con otros ecosistemas.

6.3.6 Resolución del ejemplo

La división de TI de la agencia de viajes puede resolver el problema planteado aplicando este patrón de arquitectura. Es importante notar que el proceso de adopción y aplicación del patrón de Orientación a Servicios, como el ya descrito, puede realizarse en forma incremental, pasando por varias configuraciones intermedias con menores capacidades, llegando a la configuración aquí presentada al final de dicho proceso de adopción. La siguiente figura presenta una posible configuración para resolver el problema.

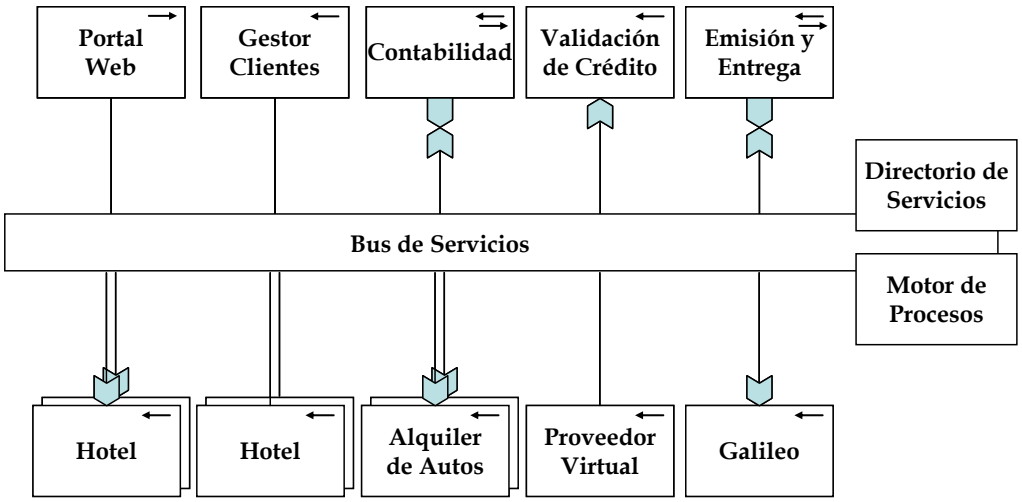


Figura 4: Arquitectura del ecosistema

La notación utilizada en la figura es la siguiente. Se distinguen cuatro tipos de componentes: los de infraestructura, los proveedores de servicios, los consumidores de servicios, y los que juegan estos dos últimos roles. Para distinguir entre ellos, se utiliza en la esquina superior derecha la siguiente notación: la ausencia de símbolo indica que es un componente de infraestructura, una flecha hacia la izquierda indica que es un proveedor de servicios, una flecha hacia la derecha indica que es un consumidor de servicios, y una doble flecha indica que es proveedor y consumidor de servicios. Por otra parte, una flecha gruesa entrante en un componente indica el uso de un adaptador de servicio, mientras que una flecha gruesa saliente indica el uso de un proxy de servicio.

Los componentes presentes en la arquitectura del ecosistema son los siguientes. Primero, se cuenta con *sistemas legados* que deben preservarse: Contabilidad, Validación de Crédito, Emisión y Entregra, y Galileo. Segundo, participan *sistemas externos*. Para tener mayor flexibilidad, si existían definiciones consensuadas de servicios fueron utilizadas, sino se definieron servicios a los cuales deben adaptarse los sistemas externos. Tercero, se define un *proveedor virtual* para aquellas empresas que no tienen sistemas con los cuales interoperar, sino que el contacto se hace en forma personal, por ejemplo empresas de city tours. Se define un servicio y se construye un proveedor virtual que lo realice, el cual resuelve el servicio en forma ad-hoc; por ejemplo agendando una tarea a un empleado de la agencia de viajes. Cuarto, se crearon *nuevos servicios* para ofrecer las nuevas funcionalidades del negocio, a saber el Gestor de Clientes. Asimismo, fue necesario crear un *portal* para brindar el proceso de negocios a través de Internet a los clientes y para su uso por los operarios en la propia agencia de viajes. Finalmente, los componentes de infraestructura también fueron incluidos; éstos son el bus de servicios, el directorio de servicios y el motor de procesos.

Los servicios identificados en el negocio son publicados en el directorio de servicios. Para cada servicio se registra la información de enlace a los distintos proveedores de servicios disponibles en el sistema. Cada proveedor incluye además información de QoS, administrada por el bus de servicios durante la operativa del sistema. El directorio cuenta además con información adicional, publicada por el proveedor de servicios y utilizada por el consumidor de servicios al momento de buscar un proveedor que realice el servicio que necesita. Por ejemplo, para el servicio de hotelería, cada uno de los sistemas hoteleros que lo proveen puede indicar el destino turístico en el que residen. De esta forma, un consumidor de este servicio puede buscar en el directorio un proveedor que se encuentre en tal destino. Esta capacidad del directorio permite mayor automatización en el negocio. Por su parte, se registra en el motor de procesos la descripción del proceso de negocio de venta de un paquete turístico. Esta descripción indica las actividades a seguir en el proceso, los servicios que son requeridos para llevarlo adelante, y los mensajes que son esperados para que el proceso avance.

6.3.7 Usos conocidos

Entre los usos conocidos de este patrón están los siguientes:

Sistemas empresariales. La infraestructura de software de las empresas de mediano y gran porte suelen encontrarse con múltiples sistemas destinados a satisfacer diferentes requerimientos del negocio. Estos sistemas son generalmente de gran tamaño y complejidad, lo cual se debe principalmente a que su evolución estuvo dirigida por exigencias del momento y los mismos no estaban preparados para acompañar tal evolución. Actualmente estas empresas buscan construir un framework común en el cual ejecutar todas sus aplicaciones de negocio, eliminando funcionalidades redundantes que provocan problemas de mantenimiento, y facilitando la definición o adecuación del soporte de TI a los requerimientos del negocio en permanente cambio. Estás empresas se encuentran en el contexto de este patrón y enfrentan los problemas aquí identificados. Existe una tendencia general en este tipo de empresas en adoptar un enfoque de orientación a servicios.

B2B (Business-To-Business). B2B es un caso particular de comercio electrónico en que dos o más empresas realizan un vínculo o relación comercial en forma electrónica. Las empresas han visto en el uso de Internet una posibilidad para mejorar la relación con sus clientes y proveedores, agilizando los procesos de la línea de negocio. Asimismo, nuevas oportunidades de negocio han surgido al ser posible la combinación de servicios ofrecidos por diferentes empresas. Actualmente, estas empresas buscan contar con un mecanismo de interoperabilidad entre sus sistemas para llevar adelante su línea de negocios en forma eficiente. La integración de estos sistemas se ha visto dificultada por factores tecnológicos, como ser la heterogeneidad de las plataformas y protocolos de comunicación, la falta de estándares en definición de funcionalidades ofrecidas, y el contar con sistemas que no estaban preparados para este nuevo requerimiento del negocio. En este contexto se encuentra el ejemplo de la agencia de viajes, la cual quiere ofrecer un servicio a sus clientes haciendo uso de B2B con los proveedores turísticos. Una arquitectura orientada a servicios brinda el soporte necesario para satisfacer los requerimientos de B2B. Con este patrón se pueden generar procesos de negocio complicados a muy bajo costo, lo cual lo hace ideal para este contexto.

6.3.8 Consecuencias

La aplicación del patrón presenta los siguientes beneficios:

- *Mayor reuso*. Los servicios pueden ser reaprovechados en la definición de múltiples procesos de negocio. A su vez, es posible planificar el reuso al momento de la definición de servicios.

- *Soporte al cambio y la mantenibilidad.* La flexibilidad brindada por el motor de procesos y por el débil acoplamiento entre consumidores y productores son los pilares fundamentales para que el sistema sea mantenible y cambiable.

- *Experimentación*. Permite experimentar diferentes enfoques de negocio a muy bajo costo. La flexibilidad brindada por el motor de procesos permite evaluar y comparar diferentes estrategias de negocio a corto plazo y con poco esfuerzo.

La aplicación del patrón presenta las siguientes desventajas:

- *Pérdida de eficiencia*. La flexibilidad otorgada va en detrimento del rendimiento (performance) general del sistema, comparada con una arquitectura en donde los sistemas se comunican en forma directa.

- *Dificultad de establecer la granularidad correcta de los servicios*. En un sistema con una gran cantidad de servicios, sin mecanismos de agrupamiento de los mismos, es dificultoso obtener una comprensión global del sistema. Esta ausencia de percepción global, puede llevar a tomar malas decisiones, como por ejemplo crear un nuevo servicio e implementarlo en el contexto que ya existe uno que brinda una funcionalidad muy similar y que modificarlo tiene menor costo. La decisión sobre la granularidad de los servicios es difícil, pero resulta crucial para la calidad de la arquitectura.

- *Fuertes requerimientos de componentes off-the-shelf*. Se basa en componentes de infraestructura complejos de desarrollar, siendo preferible utilizar componentes off-the-shelf. Las empresas no cuentan con este tipo de componentes ya que son muy específicos a esta arquitectura, y por lo tanto es necesaria su adquisición. La desventaja reside en que, a diferencia de otros patrones de arquitectura, este patrón requiere de esa infraestructura adicional para su aplicación.

- *Mantenimiento de la configuración*. Estando el ecosistema en producción, los componentes de infraestructura requieren de esfuerzo de administración y mantenimiento por parte de especialistas en las herramientas.

6.4 Integración de Organizaciones y Procesos de Negocio

La adecuada calidad de la gestión que las organizaciones de hoy necesitan para desarrollar o apoyar la competitividad y el desarrollo de los servicios, es el reto más importante que hoy enfrentan los responsables de implementar tecnología de información en organizaciones públicas y privadas. Los modelos de gestión organizacional/empresarial tradicionales han perdido vigencia y están siendo reemplazados por la gestión orientada a los procesos y servicios, y no a los productos. En la búsqueda del máximo grado de eficiencia operativa y rentabilidad, las organizaciones tanto públicas como privadas, han comenzado a notar que su mejora no es un aspecto meramente interno y que no traspasa sus límites físicos. La integración de éstas conformando *redes virtuales de cooperación* (u *organizaciones virtuales)*, que persiguen un objetivo superior común, se está volviendo una necesidad más que una opción. A través de la integración de sus servicios las organizaciones pueden ofrecer productos más elaborados, o aumentar el alcance y la demanda de sus productos, entre otros. Sin embargo, integrar y coordinar organizaciones heterogéneas en pos del logro de objetivos inter-organizacionales, requiere interoperabilidad de la información y de los servicios que cada una de ellas ofrece, y capacidad de coordinación de los miembros de la red. Con acierto se argumenta que si bien en una cadena de valor o proceso organizacional/productivo cada unidad participante es importante, la fuerza resultante depende finalmente de la acción colectiva y no de las contribuciones individuales.
Por otra parte, cada organización autónoma y heterogénea que se integra a una organización virtual, pretende que el impacto interno derivado de la integración sea mínimo. Las organizaciones necesitan mantener su identidad y separar sus intereses internos, de los externos, para evitar que las demandas externas gobiernen el rediseño de sus procesos internos, perdiendo así su autonomía. Esta forma de operar requiere nuevos mecanismos de coordinación para aquellas actividades que se desempeñan de forma conjunta y que tienen repercusión interna a cada organización.

En este escenario la coordinación y la información compartida se vuelven dos elementos claves para el buen funcionamiento de una red. Si bien esa coordinación puede verse reflejada sólo a nivel estratégico o táctico, existe una tendencia creciente a realizar una integración de principio a fin, llegando aún a los niveles operativos de los procesos de negocio de las organizaciones. En este sentido, la gestión de la información que fluye

dentro de la red de organizaciones/empresas se convierte en un activo muy valioso y cuya generación, mantenimiento y adecuación, en muchos casos no es posible. Esto se debe a la escasa o nula visibilidad de la información que suelen permitir las organizaciones miembros de una red, para no sentirse espiadas.

Usualmente, para garantizar un nivel de coordinación aceptable y la circulación segura de la información compartida entre los miembros de la red, se requiere una entidad virtual que coordine adecuadamente los flujos de información y el ofrecimiento/consumo de servicios. Esta entidad virtual puede estar centralizada o distribuida y generalmente involucra una plataforma tecnológica que da soporte a la heterogeneidad de sistemas, redes y protocolos que cada organización/empresa utiliza, proveyendo adecuados mecanismos de interoperabilidad. Si bien los intentos de aplicar las Tecnologías de la Información y las Comunicaciones (TICs) a la mejora de esta situación no son recientes, se comienza a apreciar la creciente adopción de Internet como plataforma de apoyo a esos procesos. El tipo de TIC que se requiere para llevar a cabo esta integración es, en principio, económica y técnicamente alcanzable para la mayoría de las organizaciones, sin importar su tamaño u orientación. Sin embargo, el componente más importante para esta integración es la adopción de un enfoque arquitectónico que apoye la integración de servicios heterogéneos, sin convertirse en un obstáculo para los procesos de negocio.

6.4.1 Gestión de Procesos Distribuidos

La eficiencia de una organización/empresa no depende exclusivamente de cuán bien organizados estén sus procesos internos, ni sus sistemas de información, sino que en gran parte está condicionada por el grado en que se consigue alinear las metas, procesos y servicios, con el resto de los socios o miembros de una red. En la búsqueda de esta eficiencia global, el primer paso consiste en obtener una visión más amplia de esa cooperación conjunta y reconocer que los procesos de negocio tradicionalmente concebidos como internos, son en realidad un conjunto de procesos globales en los que cada organización interviene parcialmente en logro de un objetivo global. Conseguir alinear estos procesos de negocio distribuidos, requiere establecer mecanismos que permitan mejorar la fluidez de la información y coordinar los servicios que provee cada uno de los miembros de la red. Para conseguirlo es necesario contar con una arquitectura y un adecuado soporte tecnológico que permita:

- Definir los procesos de negocios globales y almacenarlos en repositorios para que sean accedidos posteriormente.
- Gestionar las distintas instancias de esos procesos, orquestando la secuencia de actividades entre los distintos recursos que intervienen en ellos.
- Proveer los mecanismos e interfaces que permitan automatizar la recolección y/o distribución de información entre los miembros de una red.
- Identificar y definir un conjunto de indicadores de desempeño global, para medir el nivel de logro de las metas globales.

Si bien estas características son importantes y no triviales, la naturaleza distribuida de los procesos de negocio dentro de una red, genera una necesidad de atender la variabilidad de

los sistemas de información y fuentes de datos que están presentes en ese entorno. Esto involucra un esfuerzo importante en lograr la interoperabilidad de la información. Si bien las organizaciones pretenden permanecer tan independientes como sea posible para gestionar sus propias políticas internas, la integración les demanda nuevas capacidades. Para ello, la orientación a servicios pareciera ser una alternativa interesante.

6.4.2 Arquitecturas Orientadas a Servicios de Apoyo a Procesos Inter-organizacionales

El modelo de orientación a servicios se basa en la existencia de un proveedor de servicios electrónicos, un consumidor de éstos y, opcionalmente, una tercera entidad que permita poner en contacto a unos con otros. Sobre este modelo se está construyendo la siguiente evolución de Internet, y ya existe un conjunto de tecnologías que permiten aplicarlo a la coordinación de actividades que componen un proceso de negocio distribuido.

En una arquitectura orientada a servicios, cada entidad participante pone a disposición de otros, un conjunto de servicios electrónicos que pueden ser consumidos bajo pedido. Bajo esta filosofía, un nuevo tipo de aplicaciones basadas en Internet, promueven la orquestación de invocaciones a estos servicios con el objeto de proveer una funcionalidad específica a un dominio de problema concreto. En este tipo de arquitecturas, el elemento básico es el *servicio Web*, entendido como un servicio que puede ser localizado y consumido por una aplicación de software, usando generalmente a Internet como medio de transporte. El concepto de servicio Web es utilizado por diferentes grupos para identificar conceptos muy variados. Desde un punto de vista tecnológico, un servicio Web puede definirse como: "*un sistema de software identificado unívocamente dentro de Internet, cuya interfaz pública e invocaciones son definidas utilizando XML. Su definición puede ser descubierta por otros sistemas de software. Éstos pueden interactuar con el servicio de una forma predefinida utilizando el envío de mensajes sobre los protocolos de Internet"* [Fran05].

Por otra parte, en un contexto más cercano al negocio, los servicios Web también son considerados como piezas de funcionalidad de negocio, que las organizaciones / empresas proveen a terceros utilizando las tecnologías vinculadas a Internet. Un servicio Web puede encapsular un servicio de negocio que representa una actividad, un subproceso o incluso un proceso completo. Bajo esta perspectiva, se puede establecer una correspondencia entre un modelo de un proceso de negocio global, una organización o red de organizaciones, y el proceso de orquestar una secuencia de invocaciones a servicios Web localizables en Internet.

6.5 Soporte a Procesos de Negocio Distribuidos Usando Servicios Web

Los procesos de negocio distribuidos pueden ser considerados como un conjunto de actividades que se asignan a diferentes miembros de una red de organizaciones. Las organizaciones participantes esperan que el proceso se lleve a cabo de manera eficaz y que contribuya al beneficio global. La utilización de Internet como soporte a este tipo de procesos, pareciera ser la primera opción [Gref99; Lazc01; Stri00].

Cuanto más distribuido está el proceso de negocio, más relevancia y dificultad adquiere la coordinación de las actividades/subprocesos. Esta coordinación puede centralizarse en un solo organismo, o bien puede distribuirse entre los participantes. En general, las estrategias de coordinación centralizada se parecen más a un sistema de flujo de trabajo tradicional (workflow), que coordina las actividades de los miembros de una red y les permite realizar acciones según los derechos que éstos tienen. Por otra parte, las estrategias de coordinación distribuida consideran a las entidades participantes como autónomas (denominadas ejecutores) [Fran05], y el proceso se coordinación de actividades se basa en protocolos de interacción estandarizados, a los que adhieren todos los miembros de una red.

Cuando se modela este tipo de procesos distribuidos (o de coordinación distribuida), no siempre es posible mantener el mismo nivel de abstracción para cada actividad o servicio Web ofrecido/consumido por organizaciones miembros de la red. En este sentido, el concepto de servicio Web es representado como la entidad que encapsula la definición de una actividad/subproceso/proceso de negocio (desde el punto de vista de la gestión de los flujos de información), al basarse en la definición y utilización de interfaces, en las cuales sólo se especifican las entradas y salidas (considerando el servicio Web como una caja negra). Haciendo esa correspondencia es posible obtener el primer borrador del proceso de negocio, y luego, éste puede refinarse tanto como sea necesario. Las primeras aproximaciones suelen ser vagas e incompletas. Sin embargo, en la medida que se avanza en el refinamiento del proceso se puede obtener el nivel de detalle y completitud deseados.

Dos elementos importantes que están presentes en los procesos de negocios distribuidos son las unidades de ejecución y los ejecutores. Una unidad de ejecución es *"un paquete de trabajo que puede estar compuesto por una actividad, un subproceso o un proceso, y que puede ser asignado a una entidad que se encargará de su ejecución, dado que posee el conocimiento y la capacidad para llevarlo a cabo como contribución a un proceso global" [Fran05]*. Una unidad de ejecución puede ser vista como una función computacional que transforma un conjunto de entradas en salidas, mediante la ejecución de un servicio Web. La entidad que se encarga de llevar a cabo esa ejecución se denomina ejecutor. Por su parte, un ejecutor *"es un proveedor de servicios (una organización o cualquiera de sus recursos) que es capaz de encargarse de la ejecución de un paquete de trabajo mediante la provisión y/o consumo de servicios web desde y hacia otros ejecutores" [Fran05b]*. Para realizarlo, éste cuenta con un modelo de datos de soporte, la lógica interna de los procesos de negocio y un conjunto de servicios capaces de gestionar esos datos (ver figura 5).

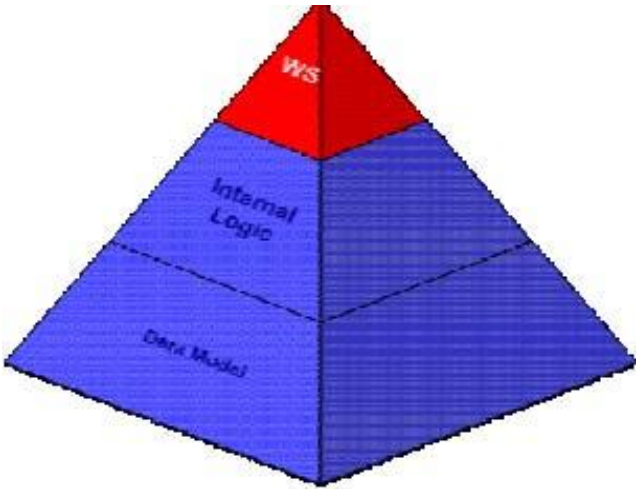


Figura 5: Arquitectura de un Ejecutor [13]

El *modelo de datos* es específico para el dominio dentro del cual el ejecutor desarrolla su actividad. Éste está dedicado a proveer la capa de interoperabilidad. La *lógica interna* recoge la lógica de negocio que el ejecutor debe implementar para tratar esos datos adecuadamente y que luego formará parte del paquete de servicios a ofrecer. Finalmente, la *capa de servicios Web* representa el conjunto de servicios que el ejecutor ofrece a otros ejecutores o aplicaciones para desarrollar su trabajo.

Este concepto de ejecutor es posible aplicarlo a diferentes tipos y tamaños de organizaciones (una repartición pública, una empresa, una oficina, una máquina, etc.) siempre que tengan la posibilidad de proveer o consumir servicios Web.

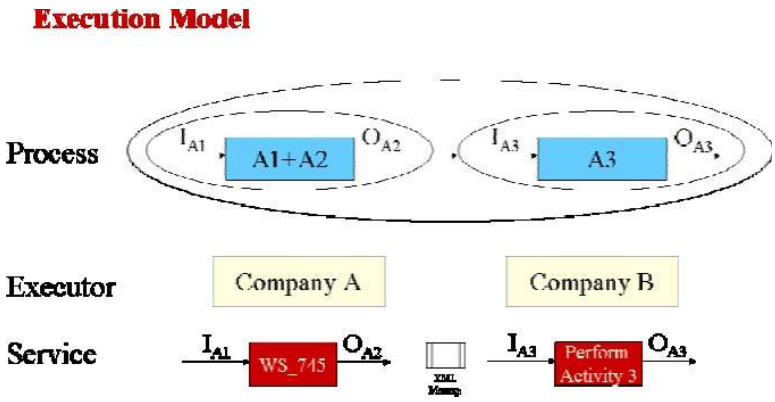


Figura 6. El modelo de ejecución de los ejecutores y los paquetes de trabajo [14]

[13] Extraído de: Franco, R.D., Ortiz Bas, A., Lario, F. *Cómo mejorar la coordinación en procesos productivos distribuidos utilizando un enfoque orientado a servicios e Internet*. IX Congreso de Ingeniería de Organización Gijón. España. Septiembre de 2005.

[14] Extraído de: Franco, R.D., Ortiz Bas, A., Anaya, V., Navarro R. "Improving Supply Chain Operations performance by using a collaborative platform based on a Service Oriented Architecture". 2nd Workshop on Computer Supported Activity Coordination. Miami, USA. 2005.

Una vez que se cuenta con los servicios Web, es necesario relacionar las unidades de ejecución (oficinas, reparticiones, etc.) con los servicios ofrecidos por cada ejecutor. Cada unidad de ejecución representa una unidad atómica de trabajo que es asignada a un ejecutor. La figura 6 muestra la orquestación de las invocaciones a servicios provistos por los distintos ejecutores del proceso [Fran05b]. Este modelo de proceso, al basarse en interfaces de servicios, exhibe un acoplamiento débil entre el ejecutor y el trabajo a realizar.

6.6 Una plataforma de Ejemplo

A partir de los conceptos de ejecutores y unidades de ejecución antes expuestos, se ha desarrollado una plataforma tecnológica que permite el modelado, ejecución y control de procesos de negocio distribuidos. Esta plataforma fue llamada IDIERE y tiene 6 componentes principales (ver Figura 7): (1) *generador de escenarios*, (2) *modelador de procesos de negocio*, (3) *motor de procesos*, (4) *repositorio de información compartida, (5) motor de aprendizaje organizativo e (6) inteligencia empresarial* [Fran05b].

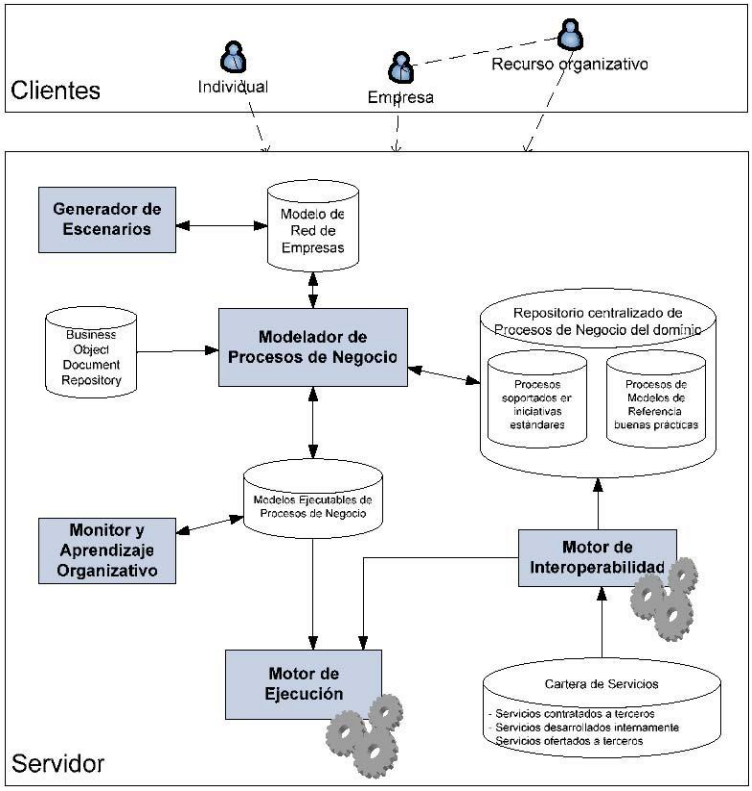


Figura 7. Arquitectura de la plataforma IDIERE [15]

[15] Extraído de Franco, R.D., Ortiz Bas, N., Navarro, R. "Enhancing Supply Chain Co-ordination by means of a collaborative platform based on Service Oriented Architecture". *Collaborative Networks and their Breeding Environments.* In proceedings of 6th IFIP Working Conference on Virtual Enterprises. PRO-VE '05. Ed. Springer Verlag. 2005.

El *generador de escenarios* permite la configuración de la red de organizaciones / empresas, definiendo el conjunto de ejecutores que la componen. El *modelador de procesos de negocio,* a partir del cual se construyen las instancias de los distintos procesos, define los paquetes de trabajo y los ejecutores a cargo de cada parte del proceso. El *motor de procesos* tiene una finalidad similar a la de cualquier herramienta de workflow; esto es, automatizar las distintas ejecuciones de actividades y contribuir a una mejor coordinación del trabajo conjunto. El *repositorio de información compartida* está destinado a almacenar las distintas definiciones de procesos, instancias en ejecución, y modelos de procesos existentes. Finalmente, el *motor de aprendizaje organizativo e inteligencia empresarial* es quien posee el conjunto de indicadores de nivel global y permite mostrarlos de una forma similar a un panel de mando.

6.7 Referencias

[Bass03] Bass L., Clements P., Kazman R. *Software Architecture in Practice 2nd Edition* Reading, MA: Addison-Wesley, 2003.

[Busch96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Peter Sommerlad, Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons; 1996.

[Clem02] Clements P., Buchmann F., Bass L., Garlan D., Ivers J., Little R., Nord R., Stafford J.. *Documenting Software Architectures. Views and Beyond*. SEI Series on Software Engineering. Addisson Wesley. 2002.

[Fran05] Franco, R.D., Ortiz Bas, A., Lario, F. *Cómo mejorar la coordinación en procesos productivos distribuidos utilizando un enfoque orientado a servicios e Internet*. IX Congreso de Ingeniería de Organización Gijón. España. Septiembre de 2005.

[Fran05a] Franco, R.D., Ortiz Bas, A., Anaya, V., Navarro R. "Improving Supply Chain Operations performance by using a collaborative platform based on a Service Oriented Architecture". 2nd Workshop on Computer Supported Activity Coordination. Miami, USA. 2005.

[Fran05b] Franco, R.D., Ortiz Bas, N., Navarro, R. "Enhancing Supply Chain Co-ordination by means of a collaborative platform based on Service Oriented Architecture". *Collaborative Networks and their Breeding Environments.* In proceedings of 6th IFIP Working Conference on Virtual Enterprises. PRO-VE '05. Ed. Springer Verlag. 2005.

[Gamm95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, 1995.

[Gref99] Grefen, P., Hoffner, Y. *CrossFlow – Cross Organizational Workflow support for Virtual Organizations*. Proceedings of 9th International Workshop on Research Issues on Data Engineering (RIDE-VE 99). Sidney, Australia. 1999.

[Lazc01] Lazcano, A., Schuldt, H., Alonso, G., Schek, H. *WISE: Process based E-Commerce*. IEEE Data Engineering Bulletin, Special Issue on Infrastructure for Advanced E-Services, Vol. 24, Nº 1, pp. 46-51. March 2001.

[Stri00] Stricker, C., Riboni, S., Kradolfer, M. and Taylor, J. *Market-Based Workflow Management for Supply Chain of Services*. Proceedings of the 33rd Hawaii International Conference on System Sciences, Hawaii, USA. 2000.

CAPITULO VII: SEGURIDAD DE DOCUMENTOS ELECTRÓNICOS

Philippe Camacho, Agustín Villena

Departamento de Ciencias de la Computación, Universidad de Chile.
{pcamacho, avillena}@dcc.uchile.cl

7.1 Introducción

Los documentos electrónicos, al igual que sus pares del mundo físico, son usados como medio para acreditar acuerdos entre personas, tales como transacciones comerciales, contratos o para acreditar hechos significativos que las involucran como contrataciones, despidos, declaraciones juradas, etc.

Al circular la documentación entre diversos ambientes, está expuesta a vulnerabilidades tales como robo de información o la adulteración de su contenido. En el mundo físico es la firma manuscrita la que permite acreditar que el contenido de un documento basado en papel no ha sido adulterado, que es auténtico y que su contenido no puede ser repudiado por el firmante. Por su parte, en el mundo digital se han implementando mecanismos que implementan las mismas propiedades de seguridad utilizando tecnologías denominadas *criptográficas* y mecanismos legales que permiten a las personas adquirir *certificados de firma electrónica* con los que se podrán firmar documentos electrónicos.

En Chile la firma electrónica tiene validez legal desde el año 2002, existiendo a la fecha diversos trámites que la utilizan de forma cotidiana. En este capítulo se presentarán conceptos básicos sobre criptografía, cómo se ha implementado la firma electrónica en Chile y en qué consiste la firma electrónica aplicada al formato XML.

7.2 Conceptos básicos de Seguridad en Documentos Electrónicos

La seguridad aplicada a la documentación electrónica puede entenderse como el conjunto de medidas que buscan mitigar en lo posible el riesgo de intervenciones malintencionadas en la creación, uso y transporte de la información contenida en ella. En particular, lo que se busca es fortalecer los siguientes aspectos:

1. **Confidencialidad de las comunicaciones**: Supongamos que Patricio quiere mandarle un mensaje secreto a Silvia. Si no toma precauciones, un intruso o atacante, llamémoslo Tomás, podría leer el mensaje sin que los dos interlocutores pudiesen darse cuenta.
2. **Integridad de la información**: El participante malicioso Tomás no debería ser capaz modificar el mensaje mandado por Patricio sin que Silvia se dé cuenta.

3. **Autenticidad de la información:** Cuando Silvia recibe un mensaje, quisiera estar segura que fue el mismo mensaje que fue enviado por Patricio y no lo fue por otra persona.
4. **No repudiabilidad:** Que es una propiedad derivada de la autenticidad, e implica que Patricio no puede negar que el es el autor del mensaje recibido por Silvia.

Es importante recalcar el concepto de *gestión del riesgo* que subyace a este enfoque de seguridad, en donde no se apunta a eliminar los riesgos, sino que a mantenerlos controlados. Las herramientas técnicas que hacen posible implementar esquemas de seguridad en el mundo electrónico deben su existencia a la disciplina denominada *criptología,* que estudiaremos a continuación.

7.3 Elementos Básicos de Criptología

La criptología es la disciplina que tiene por objetivo resolver problemas vinculados con la comunicación en un ambiente hostil. Esta compuesta de dos partes: la criptografía, que es la ciencia que desarrolla herramientas para proteger información ante posibles ataques, tales como filtraciones o adulteraciones, y el criptoanálisis, que estudia esas herramientas y sus posibles fallas. Las tecnologías principales para implementar soluciones criptográficas son:

- **cifrado simétrico:** que permite resolver el problema de la confidencialidad y es usado para implementar ***criptografía de clave secreta***.
- **cifrado asimétrico:** que además de la confidencialidad, resuelve los problemas de la integridad, autenticidad y no repudio, usado para implementar ***criptografía de clave pública***.

7.4 Criptografía de clave secreta

Supongamos que Patricio quiere mandar un mensaje a Silvia de manera que sólo ella pueda entender el mensaje, es decir, que sea *confidencial*. Si el mensaje es enviado por un medio susceptible de ser intervenido por terceros, Patricio deberá modificar el mensaje original. Para esto usará un *algoritmo*, es decir, un conjunto determinado de pasos, mediante los cuales obtendrá a partir del mensaje original, que denominaremos *texto claro,* un mensaje ilegible, también llamado *texto cifrado*.[16] Silvia, deberá entonces seguir otro algoritmo que deshaga lo hecho por el usado por Patricio, y así obtener el texto claro del mensaje inicial. Como es posible observar en la figura 1, Patricio y Silvia deben haber acordado de antemano el par de algoritmos de cifrado/descifrado para poder comunicarse.

[16] Según definición de la Real Academia de la Lengua Española

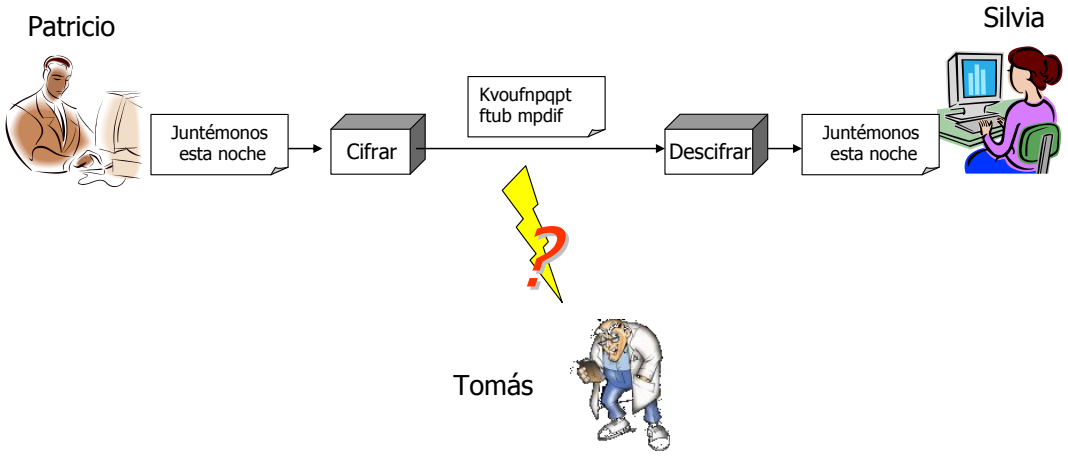


Figura 1. Ejemplo de Criptografía con Clave Secreta

De esta manera, si Tomás intercepta el mensaje cifrado, no podrá recuperar el mensaje original sin antes haber descubierto el par de algoritmos usados por Patricio y Silvia. Esto en la práctica, no es muy seguro, dado que desde la Segunda Guerra Mundial existe un vasto y público conocimiento compartido sobre algoritmos criptográficos, por lo cual la seguridad de un algoritmo no debe basarse en su confidencialidad, sino en un secreto compartido por las contrapartes, que llamaremos clave, el que se usará para alimentar los algoritmos de cifrado y descifrado. Por lo tanto, desde ahora supondremos que el atacante conoce el mecanismo usado para cifrar/descifrar los mensajes. En este contexto, la comunicación entre Patricio y Silvia sucederá de la siguiente forma:

a) Patricio y Silvia se ponen de acuerdo sobre una clave secreta en común
b) Patricio cifra el texto original con el algoritmo de cifrado y la clave secreta acordada, y envía el texto resultante a Silvia.
c) Silvia recibe el texto cifrado y recupera el mensaje inicial aplicando el algoritmo de descifrado y la clave secreta acordada.

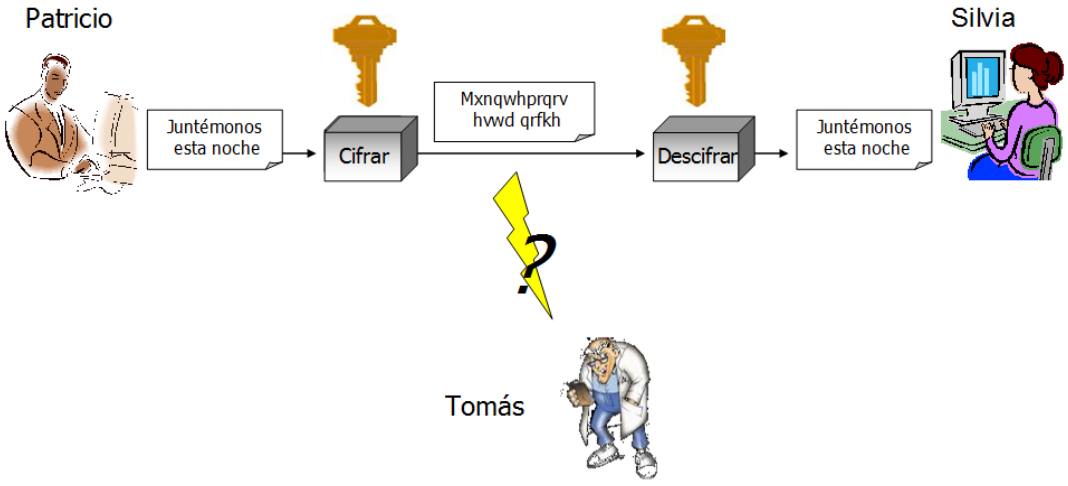


Figura 2. Ejemplo de Criptografía con Clave Secreta

Hay que hacer énfasis en que la cantidad de valores posibles para definir la clave tiene que ser muy grande, sino Tomás podría intentar *quebrar* el cifrado probando con todas la claves posibles.

Ejemplo de algoritmo simétrico

Un algoritmo básico que habría usado Julio César consiste en lo siguiente: para cifrar se toma cada letra del texto claro transformándola en la letra del alfabeto que viene *n* posiciones después, donde *n* es un entero entre 1 y 27 (numero de letras en el alfabeto). Si al hacer la traslación se sobrepasa la última letra (Z), entonces se vuelve a partir circularmente desde la letra A. Vamos a suponer que nuestro *n* es 3, entonces tenemos:

Texto claro:	A	B	C	D	E	F	...	W	X	Y	Z
Texto cifrado:	D	E	F	G	H	I	...	Z	A	B	C

El descifrado es muy simple, pues consiste en que para cada letra del cifrado se toma la letra del alfabeto que esta *n* posiciones *antes*, siempre operando de manera circular, es decir cuando se alcanza A, se sigue con Z, Y, X ...

Texto cifrado:	A	B	C	D	E	F	...	W	X	Y	Z
Texto claro:	X	Y	Z	A	B	C	...	T	U	V	W

Este par de algoritmos para cifrar y descifrar es tan simple como débil. Para quebrarlo, un atacante sólo tendría que probar todas las claves - que en este caso son 27 - y ver si el texto claro recuperado tiene sentido. También se puede armar un ataque usando la frecuencia de las letras en un idioma dado. Por ejemplo si llegan muchos textos cifrados y en ellos la letra la mas frecuente es la T, y además se sabe que el texto cifrado original estaba en español, entonces se puede deducir que la T del cifrado corresponde a la E del texto claro, pues E es la letra más frecuente del español.

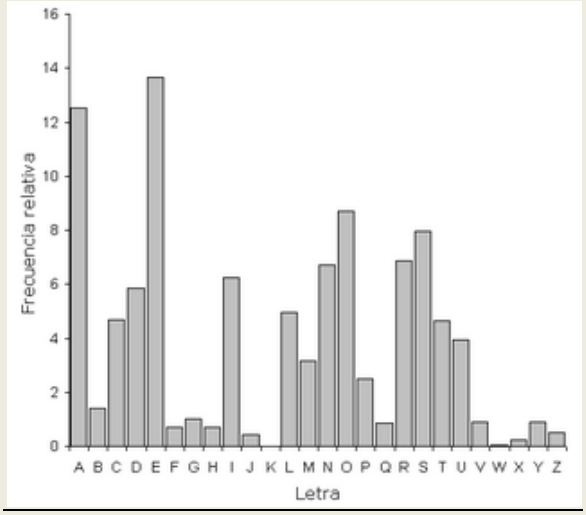


Figura 3: Letras con frecuencias relativa (%) en español (fuente: Wikipedia)

Por eso un algoritmo de cifrado robusto no debe permitir un ataque que use las frecuencias de las letras. Para lograr esto la idea es de no sólo cambiar la posición de las letras como en el cifrado descrito anteriormente, sino también sustituir letras del texto claro por otras, y obviamente usar claves que pueden tomar un gran número de valores posibles. Los algoritmos de cifrados modernos usan generalmente esas transformaciones (sustitución, intercambio de posición) y también otras muchas veces, operan sobre bits y no letras y manejan claves suficientemente grandes (en número de bits) para que no sea posible probarlas todas, ni siquiera con el más potente computador actual, según es posible ver en la Tabla 1.

Tabla 1: Tamaño de clave versus tiempo de descifrado con computadores actuales [17]

Tamaño de clave (bits)	Valores posibles	Tiempo necesario a 1 millón cifrado / s
32	$2^{32} = 4x10^{9}$	2,15 milisegundos
56	$2^{56} = 7,2 \times 10^{16}$	10,01 horas
128	$2^{128} = 3,4 \times 10^{38}$	$5,4 \times 10^{18}$ años

Como información adicional, el algoritmo de cifrado simétrico actual reconocido como el más seguro, el *Advanced Encryption Standard* (AES), permite usar claves de 128, 192 o 256 bits.

7.5 El problema de la distribución de claves

Como hemos visto, es posible establecer una comunicación confidencial usando un mecanismo basado en un algoritmo simétrico y claves secretas. Sin embargo, queda pendiente el problema de cómo dos interlocutores pueden ponerse de acuerdo en la misma clave. De hecho la única forma de estar realmente seguros de que la clave no será conocida por terceros es a través del intercambio manual. Como una manera de superar esta limitación nació la ***criptografía de clave pública***, que presentaremos a continuación.

7.6 Criptografía de clave pública

Aquí cada persona posee un par de claves: una *secreta* y otra *pública*, las que son utilizadas en un esquema denominado ***cifrado asimétrico***. La ***clave pública*** de una persona, tal como dice su nombre, está disponible para todos los que quieran comunicase con su dueño, y se usa para cifrar. La ***clave privada***, por su parte, sirve a su propietario para descifrar mensajes cifrados con su clave pública. Veamos un ejemplo:

[17] Los computadores actuales domésticos pueden ejecutar alrededor de 20 000 millones de operaciones básicas por segundo. En el caso de una clave de 128 bits, el ataque tipo fuerza bruta es realmente fuera de alcance dado que (5,4 x 1018)/20 000 = 2,7 x 10 14 ...

- Silvia pone su clave pública disponible para todos (por ejemplo, en su sitio web).
- Supongamos que Patricio quiere mandar un mensaje cifrado a Silvia. Para hacer ésto, sigue los siguientes pasos:
    - obtiene la clave pública de Silvia,
    - cifra el mensaje con un algoritmo de cifrado acordado y la clave pública de Silvia,
    - y le envía el texto cifrado.
- Al recibir el mensaje, Silvia lo descifrará usando el algoritmo acordado y su clave privada.

Obviamente Silvia tiene que guardar celosamente su clave privada, porque si Tomás u otra persona estuviera en su posesión, podría descifrar los mensajes destinados a ella. Es importante hace notar que la criptografía de clave pública se basa en tecnologías que hacen imposible en la práctica deducir la clave privada de Silvia a partir de la clave pública[18].

7.7 Otra aplicación de la criptografía de clave pública: Firma electrónica

Además de solucionar el problema de la distribución de claves, la criptografía de clave publica abre la posibilidad de firmar información. Antes de revisar esto en detalle, primero revisaremos cúal es el sentido que subyace al acto de firmar un documento:

1. que no sea posible que una persona firme en lugar de otra.
2. que no sea posible adulterar el contenido de lo firmado, sin que esto sea detectable.
3. la persona que firma un documento se hace de esta manera responsable de lo que allí está contenido, sin poder negarlo más adelante.

¿Cómo operaría el firmado de información usando criptografía de clave pública? Veamos un ejemplo:

1. Patricio, usando un algoritmo de firma y su propia clave privada, obtiene una una firma digital. Luego envía ésta y el documento original a Silvia.
2. Silvia obtiene la clave pública de Patricio, y con ella, la firma y el documento aplica un algoritmo para verificar la firma. Si el algoritmo acepta la firma, Silvia estará segura de que:
    - Dado que Silvia usó la clave publica de Patricio, el no podrá repudiar la autoría del documento, porque sólo con la clave privada de Patricio se pudo haber cifrado el documento original.
    - El documento original no fue adulterado antes de que ella lo recibiera, o sino el documento original no habría sido idéntico al documento descifrado.

[18] Es decir, que el tiempo requerido para calcular la clave privada a partir de la pública sea prohibitivo usando la tecnología actual, por ejemplo, 100.000 años.

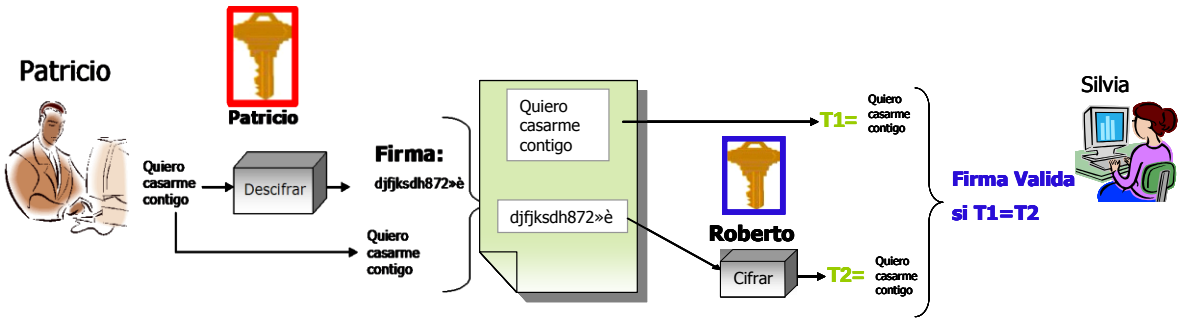


Figura 4: Ejemplo de firma con Clave Pública

7.8 Optimización a través el algoritmo de hash

El esquema anterior presenta dos dificultades:

- En la práctica los algoritmos criptográficos de clave pública son relativamente lentos, por lo cual firmar documentos grandes puede costar mucho tiempo.
- También el hecho que la firma del documento sea tan grande como el documento mismo, implica que transmitir y almacenar documentos firmados ocupa el doble de ancho de banda y espacio respectivamente.

En la actualidad existen tecnologías que permiten obtener una "huella" electrónica del documento de tamaño fijo y pequeño, la cual denominaremos *hash*. Estas tecnologías poseen las siguientes propiedades:

- Dado un documento y su huella, es imposible en la práctica obtener un documento distinto que tenga la misma huella[19]
- El resultado de una función de *hash* tiene un tamaño fijo.

Usando esta herramienta, un traspaso de documentos firmados sucede de la siguiente forma:

a) Ahora Patricio primero obtiene el hash del documento que quiere firmar.Luego, aplicando el algoritmo de firma con su clave privada al hash calculado obtiene la firma digital y envía ésta con el documento original a Silvia.

b) Silvia obtiene la clave pública de Patricio. Primero Silvia calcula el hash del documento original, luego aplica el algoritmo de verificación de firma con el hash firmado, el hash recién calculado y la clave pública. Si la verificación es exitosa Silvia estará segura de que Patricio firmó el documento, obteniéndose las mismas propiedades de integridad, autenticidad y no repudiabilidad del caso anterior.

[19] En la practica los algoritmos de hash más usados son MD-5 y SHA-1, sin embargo se han descubiertos en estos últimos años ataques sobre ellos y actualmente nuevos algoritmos candidatos estan al estudio.

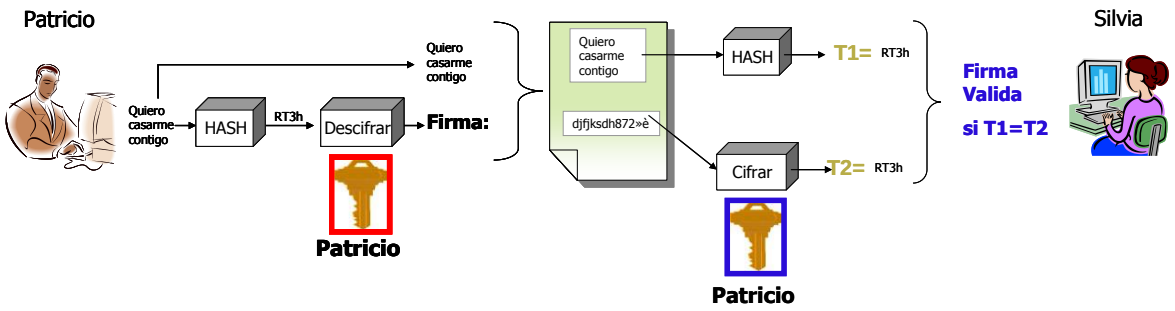


Figura 5: Ejemplo de firma con Clave Pública

En las figuras 4 y 5 se muestran esquemas de firmas (sin y con hash respectivamente) donde el algoritmo de firma es el mismo que el algoritmo para descrifar y el algoritmo de verificación de firma usa el algoritmo de encriptación. En este sentido la firma digital puede verse como la operación dual de la encriptación aunque no siempre es así.

Un algoritmo de clave pública famoso: RSA

En 1975 Whitfield Diffie y Martin Hellman idearon el concepto de criptografía de clave pública, pero las herramientas matemáticas requeridas no fueron hechas públicas hasta el año siguiente, por los matemáticos Rivest, Shamir y Adleman. Ellos descubrieron un algoritmo capaz de implementar un sistema criptográfico de clave pública, que fue bautizado *RSA* en honor a las iniciales de sus creadores. RSA es un algoritmo de cifrado de bloque (lo que significa que la información a cifrar está separada en varios pedazos del mismo tamaño) en el que la información a encriptar y el resultado de la encriptación es un numero entero entre 0 y n-1 para un n dado.

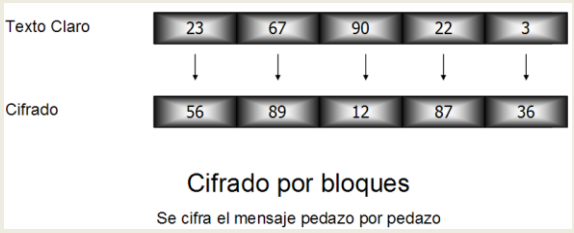


Figura 6: Ejemplo de Cifrado por Bloques

La operación de cifrado como de descifrado usa una operación llamada *exponenciación modular*. Dicha operación consiste a tomar un entero M, calcular M^e donde e, es también un entero y finalmente calcular M^e mod n (el resto de la división euclidiana de M^e por n). Veamos ahora como funciona RSA.

- Inicialización:
  - Escoger dos enteros primos *p* y *q* grandes, *p* diferente de *q*
  - Calcular $n = p \times q$
  - Calcular $\Phi(n) = (p-1) \times (q-1)$
  - Encontrar e tal que el *máximo común denominador* (mcd) entre $\Phi(n)$ y *e* sea *1*, con $1<e<\Phi(n)$ (lo cual se puede hacer en forma eficiente). Esta relación entre números se denomina "primos relativos entre sí".
  - Calcular el inverso modular de e, es decir d tal que $d \times e \bmod \Phi(n) = 1$ (lo que también se puede lograr en forma eficiente)
  - Aquí obtenemos la clave publica que es $CPub = \{e,n\}$ y la clave privada $CPriv = \{d,n\}$

- Calculo de cifrado:
  - Dado el texto claro M con $M<n$, el cifrado C será tal que: $C = M^e \bmod n$
- Calculo de descifrado:
  - Dado el texto cifrado C, se puede recuperar M haciendo:
    - $C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$. Y se puede demostrar que dado la elección de n,e y d tenemos para cualquier $M<n$ la igualdad: $M^{ed} \bmod n = M \bmod n = M$ (pues $M < n$).
    - En resumen tenemos: $C^d \bmod n = M$.

Este es una versión muy simplificada del algoritmo RSA. En la práctica hay que tomar muchas precauciones adicionales para evitar ciertos ataques. La documentación precisa del algoritmo se puede encontrar en http://www.rsa.com/rsalabs/node.asp?id=2125 (PKCS #1)

Ahora seguimos con un ejemplo simple:

- Escogemos dos números primos: $p=7$ y $q=13$
- Calculamos $n = p \times q = 7 \times 13 = 91$
- Calculanos $\Phi(n) = (p-1) \times (q-1) = 6*12 = 72$
- Encontramos e tal que e es primo relativo de $\Phi(n) = 72$ tal que $e < \Phi(n)$.
  Por ejemplo $e = 11$.
- Determinamos d tal modo que $d \times e \bmod 72 = 1$, $d < 72$
  $d=59$ sirve pues
  $59 \times 11 \bmod 72 = 649 \bmod 72 = (72 \times 9 + 1) \bmod 72 = 1$
  Ahora tenemos la clave publica CPub = {11,91} y la clave privada CPriv = {59,91}.
- Supongamos que queremos encriptar el mensaje $M=22$. Entonces:
- El cifrado sera: $C=M^{11} \bmod 91 = 22^{11} \bmod 91 = 29$
- Para recuperar el cifrado hay que calcular: $C^{59} \bmod 91 = 29^{59} \bmod 91 = 22 = M$

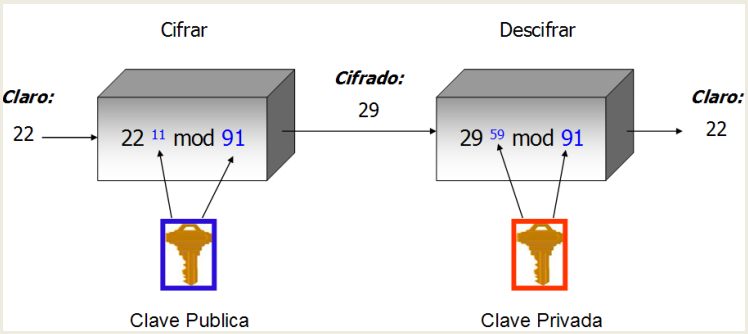


Figura 7: Ejemplo de cifrado/descifrado con RSA

Enfoquémonos ahora sobre el tema de la seguridad de RSA. Hay dos maneras naturales de intentar un ataque sobre RSA. La primera consiste en probar todos los valores posibles para la clave privada: es un ataque de tipo fuerza bruta. Como lo hemos visto, este ataque falla si la cantidad de valores que puede tomar la clave es suficientemente grande. La segunda forma, y sobre la que versan la mayor parte de los trabajos de criptoanálisis sobre RSA es la factorización de números enteros. Factorizar un numero entero es encontrar los números primos tales que el producto de éstos, sea igual al número que se quiere descomponer. En el caso de RSA, si se logra a factorizar n se puede descubrir la clave privada pues esta factorización genera los p y q primos tales que $n = p \times q$. Desde aquí se puede calcular $\Phi(n)=(p-1).(q-1)$. Después teniendo e, se puede encontrar d tal que $e \times d \bmod \Phi(n) = 1$. Y d junto a n es nuestra clave secreta. Si n es grande, su factorización es difícil, aunque cada vez se han logrado metas más exigentes con respecto a este problema. El último record fue obtenido en noviembre de 2005 por un equipo de investigadores alemanes que usaron varias máquinas en paralelo y lograron a factorizar un número de 193 dígitos, por lo cual recibieron una recompensa de 20.000 dólares.

7.9 Un problema final: Suplantación de identidad[20]

Con la criptografía de clave pública se soluciona esta vez el problema de la distribución de claves y permite además, firmar datos. Sin embargo existe una vulnerabilidad adicional, denominada "ataque del intermediario":

- Supongamos que Tomás, maliciosamente, envía su clave publica a Patricio, haciéndose pasar por Silvia. Del mismo modo, Tomás manda su clave publica a Silvia haciéndose pasar por Patricio.

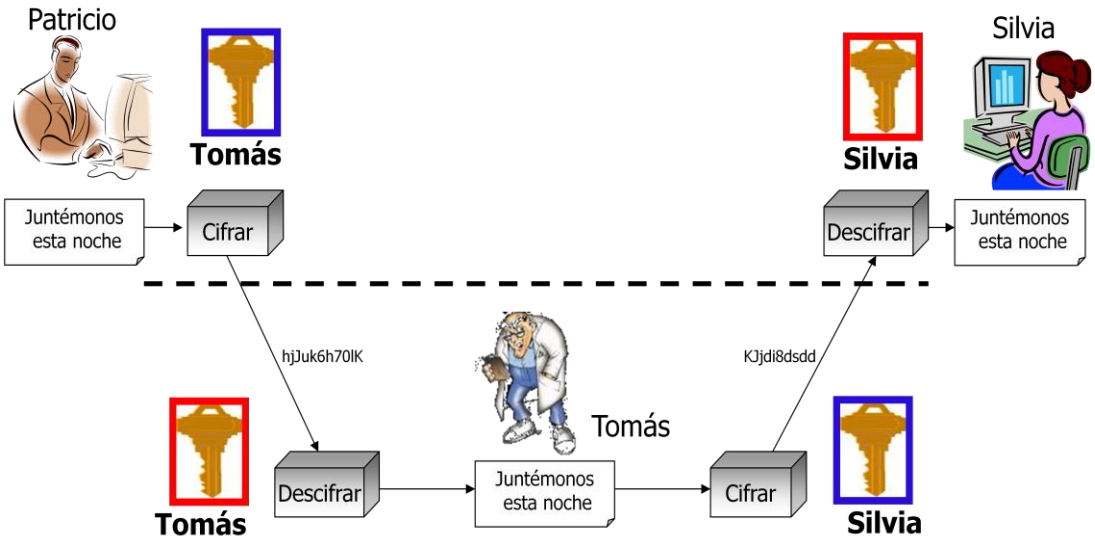


Figura 8: Ataque del Intermediario

- En estas condiciones, si Patricio envía un mensaje cifrado a Tomás, quien está suplantando a Silvia, este último podrá descifrarlo, leerlo o incluso adulterarlo, y usando la clave pública de Silvia, re-encriptar y enviarle el mensaje documento intervenido a ella. Así ni Patricio ni Silvia se dieron cuenta de que un intermediario pudo infiltrarse en medio de su comunicación, debido a que no poseían un mecanismo que les asegurase que las claves utilizadas correspondían a las personas que ellos creían.

7.10 La Solución: Infraestructura de Clave Pública

Una manera de asegurarse que una clave pública pertenece realmente a una persona dada, es constituir un organismo – denominado Autoridad Certificadora (AC) - que cumpla con los siguientes roles:

- Asegurar que una clave publica pertenece a una persona dada.

[20] En la literatura se denomina "Ataque del intermediario" (Middle Man Attack)

- Asegurar que la persona es confiable (es decir, que realmente existe, que no tiene antecedentes judiciales. etc.)
- La clave pública está vigente.

Para esto, la AC debe ser capaz de generar un ***certificado electrónico***, que es un documento firmado por la AC que contiene:

- Información válida sobre la identidad de la persona (nombres, apellidos, RUT en el caso chileno, etc...).
- La clave pública de la persona.
- La fecha de vencimiento del certificado.
- La firma de la Autoridad Certificadora de toda esa información.

Existen además *Autoridades de Registro* encargadas de los trámites administrativos para obtener y verificar los datos relevantes de la persona que quiere obtener un certificado electrónico. A continuación, las AC tienen a cargo la creación del certificado electrónico, con la consecuente generación de las claves privada y pública de la persona, y la firma de este certificado por parte de de la AC. También la AC posee un listado actualizado y firmado de los certificados que han sido revocados, tanto porque su dueño lo solicitó (porque fueron robado) o porque que el propietario no es considerado confiable.

Una *Infraestructura de Clave Publica* (PKI por sus siglas en inglés "*Public Key Infrastructure*"), es el conjunto de todas las organizaciones y medios informáticos que permiten crear, manejar y a revocar (si es que se requiere) dichos certificados.

7.11 Firma Electrónica en Chile

En el año 2001, la Comisión de Naciones Unidas para el Derecho Mercantil Internacional (UNCITRAL), definió un marco legal referencial para el uso de la Firma Electrónica (FE) en el comercio electrónico mundial, basado en los siguientes principios:

- **neutralidad tecnológica**, que indica que la FE no estará atada a ningún proveedor tecnológíco específico,
- **equivalencia funcional**, que indica que los documentos electrónicos con FE tendrán validez y aplicaciones similares a los documentos físicos firmados en forma manuscrita, y
- **autonomía de voluntad**, que establece que el usuario de FE contará con mecanimos que le permitirán contraer o renunciar a compromisos mediante ella, de acuerdo a su voluntad.

En Chile, y de manera coherente con la política gubernamental de incorporar las tecnologías de la información al quehacer de los organismos de la Administración Pública[21] y entre la ciudadanía en general, desde el año 2002 se han definido las herramientas legales

[21] Ver www.agendadigital.cl

para habilitar la adopción de mecanismos de seguridad aplicados a documentación electrónica. Entre ellas podemos destacar:

7.11.1 Ley Nº 19799 de Firma Electrónica

Publicada el 12 de abril de 2002, define un mecanismo a través del cual personas naturales pueden obtener un certificado electrónico, y con el cual se pueden firmar documentos electrónicos que ahora pueden ser reconocidos como legalmente firmados por dicha persona, de manera similar a como sucede con los documentos firmados basados en papel.

Además se definen dos modelos de Firma Electrónica, la *avanzada*, que asegura los atributos característicos de integridad, autenticidad y no repudio con certeza, y la *simple*, que otorga servicios similares pero sin la rigurosidad de la anterior. En general, un documento que constituye un instrumento público requiere necesariamente de Firma Electrónica Avanzada, pero existen casos en que es deseable que instrumentos privados posean un nivel de seguridad similar al anterior, como sucede en el caso del Documento Tributario Electrónico, que es de la familia de documentos XML que se utiliza para el modelo de Factura electrónica implementado por el Servicio de Impuestos Internos.

7.11.2 Decreto Supremo 81/2004 de Documento Electrónico

Publicado en el Diario Oficial el 23/Dic/2004, obliga a las reparticiones de la Administración Pública a adoptar gradualmente el estándar XML como codificación de su documentación, con fecha tope 31 de diciembre de 2009. Se ha permitido un proceso gradual de adopción, definido por los siguientes niveles de adopción:

1. ser capaces de recibir, visualizar, almacenar y reenviar documentos XML
2. ser capaces de generar documentos XML
3. ser capaces de procesar documentos XML, a través de un sistema informático de administración de flujos de trabajo

En este decreto se define que el estándar de firma de documentación electrónica para el gobierno de Chile será XML Signature.

7.11.3 Infraestructura de Clave Pública en Chile

En Chile, el Ministerio de Economía es el organismo encargado de acreditar las Autoridades Certificadoras (AC), es decir, los proveedores de certificados de firma electrónica, para lo cual ha conformado una Entidad Acreditadora (EA) que posee tal propósito[22].

Cada AC posee un "Certificado Raíz", que constituye una firma electrónica de la propia AC, y que es con la que se firman los certificados generados por dicho organismo. Dado que estas AC están respaldados por la EA, será posible verificar que un certificado es válido usando el certificado raíz de la AC que lo emitió. De hecho, es posible bajar los

[22] Ver www.entidadacreditadora.cl

certificados raíz de las AC desde el sitio web de la EA. Una aplicación práctica de este mecanismo es instalar dichos certificados en los navegadores de Internet, para que estos validen de manera automática la información que reciban firmada o encriptada con certificados emitidos por las AC.

7.11.4 Obtención y Administración de Certificados de Firma Electrónica

Para obtener un certificado de Firma Electrónica en Chile se deben seguir los siguientes pasos[23]:

7.11.4.1 Registro Presencial

En este paso, la persona natural debe personalmente ser registrada por una Autoridad de Registro (AR), que es un ente dependiente o reconocido por una AC y cuya función es acreditar que la persona es quién dice ser realmente. En la actualidad, el servicio de registro presencial es provisto por el Servicio de Registró Civil e Identificación, en alianza con algunas de las AC acreditadas en Chile.

7.11.4.2 Generación del Certificado de Firma Electrónica

Una vez que la AC ha recibido los datos del solicitante de certificado de firma electrónica, ésta se comunicará (usualmente mediante correo electrónico) para indicarle los pasos para generar su certificado. Este es usualmente un mecanismo automatizado vía web, en donde la clave privada es generada en el computador del usuario, sin que la AC se quede con una copia de dicha clave.

En el caso de necesitarse un certificado para firma electrónica avanzada, se requiere de un mecanismo que asegure que el titular del certificado lo mantendrá bajo su exclusivo control, de manera que se vincule únicamente al mismo y a los datos a los que se refiere. Para ésto se debe comprar a la AC un "dispositivo criptográfico", que es un mecanismo de hardware portátil dentro del cual podrán ser generadas y almacenados una o varias claves privadas, sin que sea posible obtener una copia externa de dichas claves.

7.11.4.3 Revocación

Un certificado de firma electrónica es revocado en los siguientes casos:

- La fecha de expiración ha vencido y no fueron renovados
- El titular ha solicitado su revocación, por ejemplo, porque ha perdido control sobre el certificado.
- La AC determina que el titular no es confiable.

[23] En este capítulo nos referiremos a certificados que identifican a personas naturales denominados "clase 3". Existen otros tipos de certificados emitidos por las AC, como los de firma de código o de sistema que esán fuera del alcance de este libro

Para informar del estado de un certificado, la AC a cada certificado emitido le coloca un número de serie, y diariamente actualiza una Lista de Certificados Revocados[24], la cual puede se consultada automáticamente para consultar el estado de un certificado[25]. Es importante hacer notar que, un documento firmado con certificado revocado no es confiable, y por lo tanto el receptor debería rechazarlo.

7.12 Firma electrónica según el estándar XML

El estándar XML-Signature[4] fue desarrollado por el grupo de trabajo XML DSig de la W3C[26] y el IETF[27] en respuesta en la ley estadounidense *"June 16 2000 e-sign act"* que reconoció un valor legal a la firma electrónica. Este estándar tiene muchas ventajas al lado de la firma electrónica implementada anteriormente en formatos tales como PDF[28] o Microsoft Word. Con XML Signature es posible:

1. Firmar una parte de un documento.
2. Aplicar múltiples firmas a un mismo documento. Esto es muy útil cuando se firma un contrato por dos o más partes. En las tecnologías de firma electrónica anteriores, en contraste, al firmar un documento ya firmado se invalida la firma inicial.
3. La firma electrónica XML Signature incluye información adicional (*metadatos*) sobre la firma misma, como por ejemplo el algoritmo que se usó para firmar. Eso permite aumentar el nivel de interoperabilidad.
4. XML Signature permite firmar cualquier tipo de datos puesto que se puede encapsular cualquier información en un XML.

7.13 Tipos de firma

Hay tres tipos de firma XML Signature:

- *Ensobrada*: la firma está incluida dentro del documento XML como un nodo de éste.
- *Ensobrante*: esta vez el documento XML firmado está incluido dentro del nodo "firma".
- *Separada*: la firma está incluida en un documento diferente del documento que tiene la información firmada.

[24] CRL por su sigla en inglés.
[25] Esta es una capacidad usual de los navegadores de internet actuales.
[26] El World Wide Web Consortium (www.w3c.org) es un organismo encargado de desarrollar tecnologias y estandares para promover la interoperabilidad en la web. Fue fundado en 1994.
[27] Internet Engeenering Task Force (IETF) es una comunidad internacional compuesta de desarrolladores, investigadores, operadores y vendedores que tiene por meta según la RFC 3935 de hacer funcionar internet mejor! ("Do internet work better").
[28] Portable Document Format

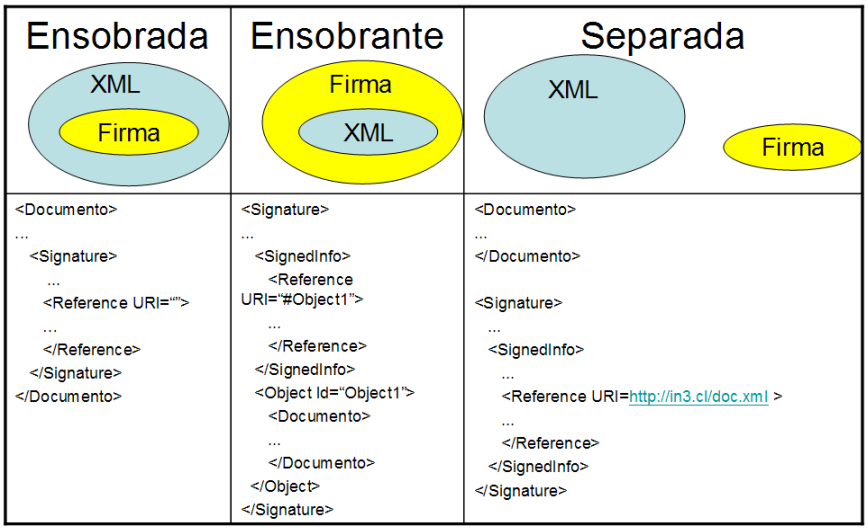


Figura 9: Tipos de Firma

7.14 Estructura de XML Signature

En esta sección se describirá la estructura de una firma XML Signature. Una firma XML Signature esta representada con el elemento XML llamado "Signature" y tiene la estructura siguiente:

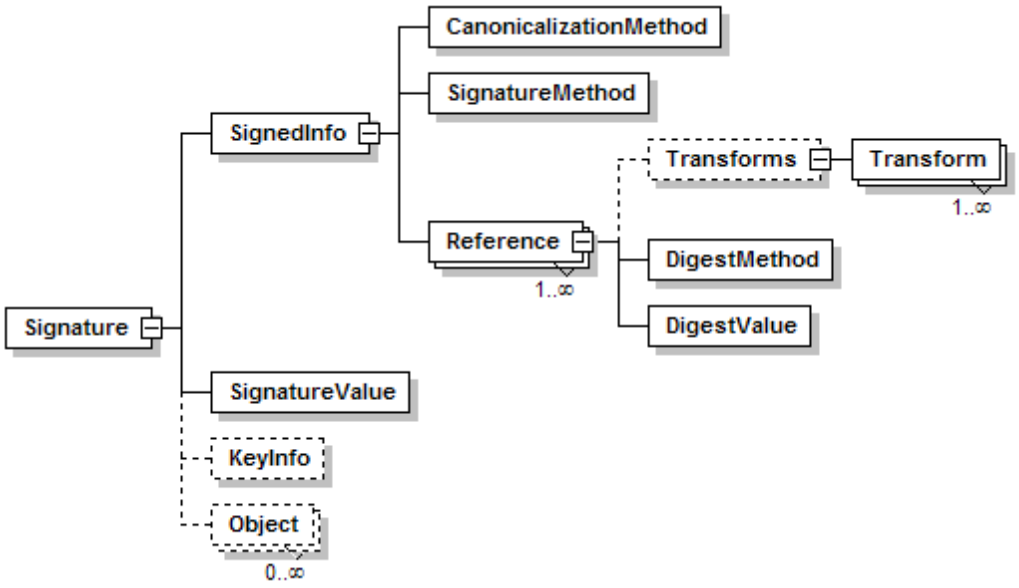


Figura 10: Estructura de una Firma

A continuación se describen cada uno de los elementos representados en la figura 10:

- **Signature** Contiene la firma. El elemento padre para la firma.

- **SignedInfo** Contiene la información sobre los datos firmados
- **CanonicalizationMethod** Contiene el nombre del algoritmo usado para generar una codificación estandarizada del XML. Este mecanismo es denominado *canonicalización*.
- **SignatureMethod** Contiene el algoritmo usado para generar la firma (operación de digest y de firma). Ejemplo: RSA-SHA1. Es importante notar que el nombre del algoritmo usado para firmar, es parte integrante de la información a firmar. Así se previene un ataque de substitución del algoritmo de firma por uno más débil.
- **Reference URI** Contiene el método para generar la huella del documento[29] y el valor resultante del digest. El atributo URI referencia la data a firmar .
- **DigestMethod** Contiene el algoritmo para generar la huella de los datos a firmar. Ejemplo SHA-1.
- **DigestValue** Contiene el valor de la huella de los datos a firmar.
- **SignatureValue** Contiene el valor de la firma.
- **KeyInfo** Contiene información sobre la clave que se necesita para verificar una firma. Puede ser un certificado X509 por ejemplo. Este elemento es opcional. Eso porque el autor del documento quizás no desea revelar información sobre la clave a todo los procesadores del documento. En este caso la clave podría estar directamente codificada en la aplicación que esta encargada de verificar la firma.
- **Object** Es un elemento opcional que se puede usar para dar información sobre la firma. Por ejemplo se puede agregar la fecha y hora (timestamp) del momento en que se generó la firma.

7.15 Pasos para generar y validar una firma XML Signature

En pocas palabras, para verificar la integridad de un documento se calcula su *hash* o huella, luego se compara ésta con la contenida en la firma (calculado con la clave pública). Si los dos son idénticos entonces la firma es válida. A continuación veremos en detalle este proceso.

7.16 Canonicalización del XML

Distintas aplicaciones podrían representar el mismo documento XML de manera diferente. Por ejemplo una pondría todo los atributos de los elementos en orden alfabético, con espacios en las definiciones de éstos, y otra quitaría los espacios irrelevantes y ordenaría alfabéticamente los atributos. Por ejemplo:

```
<documento  id   = ”mi documento” autor=”yo”>
Lindo documento
</documento>
```

y

[29] Ver punto 1.3.3.1

```
<documento autor=”yo” id=”mi documento” >
Lindo documento
</documento>
```

Generarán huellas diferentes. Por ende, antes de calcular la huella del documento a partir de la representación física del xml, tiene que aplicarse una estandarización, denominada *canocalización*. Así se cumplirá el objetivo de que dos XML que contienen la misma información significativa generen una huella igual.

Para generar o validar una firma hay dos pasos esenciales a seguir. Primero, crear o validar el elemento <Reference>, y después, generar o validar el resto (valor de la firma con la clave pública, etc.). Veamos en primer lugar como se genera una firma.

7.17 Generación de la firma

7.17.1 <Reference>

1) Recuperar la información a firmar.
2) Aplicar unas transformaciones opcionales (<Transforms>).
3) Calcular el hash o el digest de los datos después de la transformación con <DigestMethod> que tiene el algoritmo para calcularlo. Ejemplo “SHA-1”. Lo que se obtiene se pone en <DigestValue>.
4) Crear el elemento <Reference> que apunta a los datos a firmar.

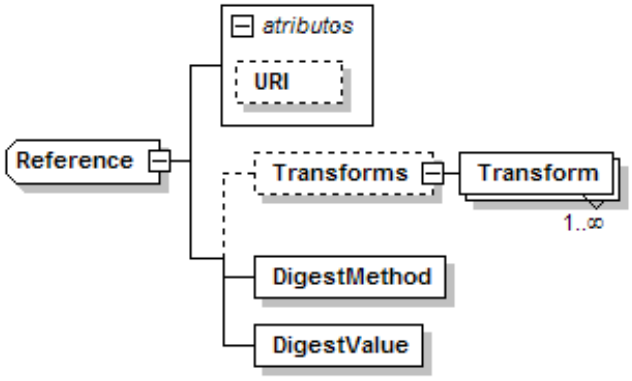


Figura 11: Estructura de una Referencia

7.17.2 <Signature>

1) Crear el elemento <SignedInfo> con <SignatureMethod>, <CanocalizacionMethod> y <Reference>(s).

2) Canocalizar el elemento <SignedInfo> y después calcular la firma que se pone en <SignatureValue> del resultado de esta canocalizacion, dado los algoritmos especificados en <CanocalizationMethod> y <SignatureMethod>.
3) Construir el elemento <Signature> con <SignedInfo>, <SignatureValue>, y si necesario <KeyInfo> con la información sobre la clave pública que se puede usar para verificar la firma, y <Object> para agregar información como la fecha de creación de la firma.

```
 <Signature  Id="MyFirstSignature"  xsi:schemaLocation="http://www.w3.org/2000/09/xmldsig#
xmldsig-core-schema.xsd">
   <SignedInfo>
     <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20000710" />
     <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa" />
     <Reference URI="http://www.w3.org/TR/xml-stylesheet/">
       <Transforms>
         <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
         <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#null"/>
       </Transforms>
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
       <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
     </Reference>
     <Reference URI="http://www.w3.org/TR/REC-xml-names/">
       <Transforms>
         <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
       </Transforms>
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
       <DigestValue>UrXLDLBIta6skoV5/A8Q38GEw44=</DigestValue>
     </Reference>
   </SignedInfo>
   <SignatureValue>MC0CFFrVLtRlkMc3Daon4BqqnkhCOlEaAhUAk8pH1iRNK+q1I+sisDTz2TFEALE=
   </SignatureValue>
   <KeyInfo>
     <KeyValue>
       <DSAKeyValue>...</DSAKeyValue>
     </KeyValue>
   </KeyInfo>
   <Object>
     <SignatureProperties>
       <SignatureProperty Target="#MyFirstSignature">
        <ts:timestamp>
            this is a test of the mixed content model
         </ts:timestamp>
       </SignatureProperty>
     </SignatureProperties>
   </Object>
 </Signature>
```

Figura 12: Ejemplo de firma XML

7.18 Validación de la firma

7.18.1 <Reference>

Para cada elemento <Reference>
1) Recuperar los datos a firmar.
2) Aplicar las transformaciones opcionales.
3) Calcular el hash o digest con <DigestMethod> de los datos después de la(s) transformación(es) y verificar contra <DigestValue>.

7.18.2 <Signature>

1) Recuperar la información sobre la clave publica desde <KeyInfo> o desde una fuente externa.
2) Calcular la forma canónica de <SignedInfo> usando el algoritmo definido en el elemento <CanocalizationMethod>.
3) Con la información sobre la clave pública, <SignatureMethod>, aplicar el algoritmo de validación de firma sobre <SignatureValue> y comparar con <SignedInfo> canocalizado.

7.19 Consideraciones sobre Seguridad en la firma XML

El estándar de firma electrónica XML Signature tiene la virtud de ser muy flexible, lo cual trae el peligro de que sea posible generar agujeros de seguridad, si la implementación de este estándar no es correcta. A continuación veremos algunas recomendaciones.

7.20 El problema derivado de transformaciones y canonicalizaciones

Como se vio con anterioridad, el procedimiento de generación de firma, incluye al principio la aplicación de unas transformaciones sobre los datos iniciales (Object Data). Al resultado de esas transformaciones, después se canonicaliza y luego se aplica la firma. Esto significa que lo que se firma realmente es lo que se obtiene después de las transformaciones y de la canonicalización. Por lo tanto, es muy importante que la aplicaciones usadas para firmar muestren, al momento de firmar, no el dato inicial, sino aquel lo más cercano posible de lo obtenido después de la canonicalización.

Por ejemplo, si el documento XML tiene una hoja de estilo XSL asociada, y esa transformación es parte del elemento <Transforms> de la firma, entonces lo que se debería mostrar al usuario que quiere firmar, es el xml después de la transformación XSL. De hecho en este caso es muy recomendable agregar una referencia a la hoja XSL (elemento <Reference>), porque sino el código XSL podría cambiar la información sin adulterar la firma.

7.21 El Entorno de Seguridad

Este estándar, permite usar claves públicas y toda una gama de algoritmos criptográficos asociados. No hay que olvidar, sin embargo, que la seguridad de un documento firmado según XML Signature no es independiente del entorno en la cual aquél es manipulado: la infraestructura de clave pública, el tamaño de clave usado, y los procedimientos administrativos que sigue el personal involucrado en el manejo de dichos documentos, serán determinantes en que la información involucrada sea o no susceptible de ser vulnerada.

7.22 Un ejemplo aplicado

A continuación presentaremos el uso de herramientas concretas que se usan para manejar certificados y para firmar / validar documentos electrónicos, contextualizados en el siguiente caso: la generación de una partida de nacimiento electrónico en un hospital. Este documento lo recibe un médico que lo va a tener que firmar antes que sea entregado al administrativo del Hospital.

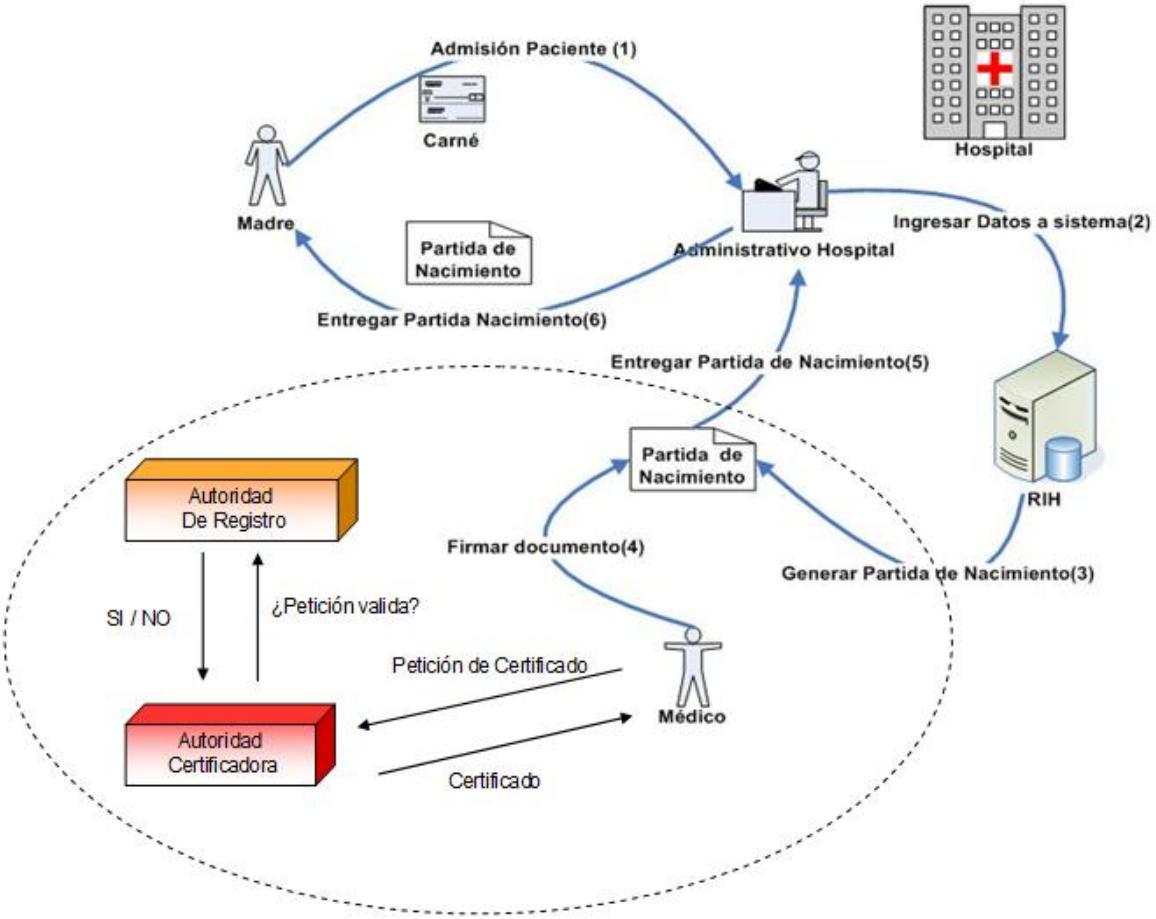


Figura 13: Ejemplo del Uso de Firma Electrónica

Este caso estará compuesto de 5 etapas:

1. Vamos a crear una Autoridad Certificadora (AC) ficticia. Ella va a generar un certificado auto-firmado que después se usará para verificar certificados de usuarios "afiliados" a esa Autoridad.
2. El médico va a generar un par de claves (pública/privada) y una solicitud de certificado a la AC.
3. Vamos a suponer que la verificación de la información contenida dentro de la solicitud se hizo exitosamente (en el caso real seria contra el Registro Civil) y que esa AC firme dicha solicitud, creando así un certificado que se va a entregar al médico.
4. El médico teniendo ya su certificado, va a firmar el documento XML recibido según el estándar XML Signature.
5. El Administrativo va a recibir el documento XML firmado por el médico y verificará su validez usando el certificado del médico que fue firmado por la AC.

7.23 Entorno de desarrollo

Usaremos el intérprete de comandos de Windows (cmd.exe) o un terminal de Unix. Además, usaremos las siguientes herramientas:

7.23.1 OpenSSL

Es una biblioteca criptográfica abierta (http://www.openssl.org) hecha en lenguaje C y que se puede usar bajo la forma de una herramienta utilizable desde una línea de comandos. Con esa biblioteca vamos a ver cómo se pueden manejar claves y certificados.

7.23.2 XMLSec

Esta biblioteca también es de código abierto y hecha en lenguaje C. Esta biblioteca sigue la implementación oficial del estándar XML Signature. Está construida sobre la biblioteca libxml2, disponible en www.xmlsoft.org. Veremos cómo se puede firmar y verificar la firma de un XML gracias a esta herramienta.

Cómo obtener las bibliotecas usadas en este ejemplo:

<table>
<tr><th colspan="2">OpenSSL</th></tr>
<tr><th>Windows</th><th>Linux</th></tr>
<tr><td>Para Windows se pueden bajar los binarios precompilados desde la web. Al momento de esta edición, una dirección para bajar los archivos es http://www.slproweb.com/products/Win32OpenSSL.html</td><td>Linux viene generalmente con una versión de openssl preinstalada. Para verificarlo abrir un Terminal y escribir:

<pre>$openssl version
OpenSSL 0.9.8 05 Jul 2005</pre></td></tr>
</table>

xmlsec	
Windows	**Linux**
En Linux En Windows se pueden instalar los binarios en `http://www.zlatkovic.com/libxml.en.html`. No olvidarse de bajar las bibliotecas requeridas libxml2, libxslt, iconv y zlib.	Se puede bajar y compilar el código fuente en el sitio de la biblioteca: `http://www.aleksey.com/xmlsec/`

7.24 Creación de la Autoridad Certificadora

Una autoridad certificadora puede ser acreditada, reconocida por otra autoridad o ser autoridad raíz es decir que no es reconocida por otra sino por ella misma. Los certificados de esas autoridades vienen generalmente ya instalados en el navegador.

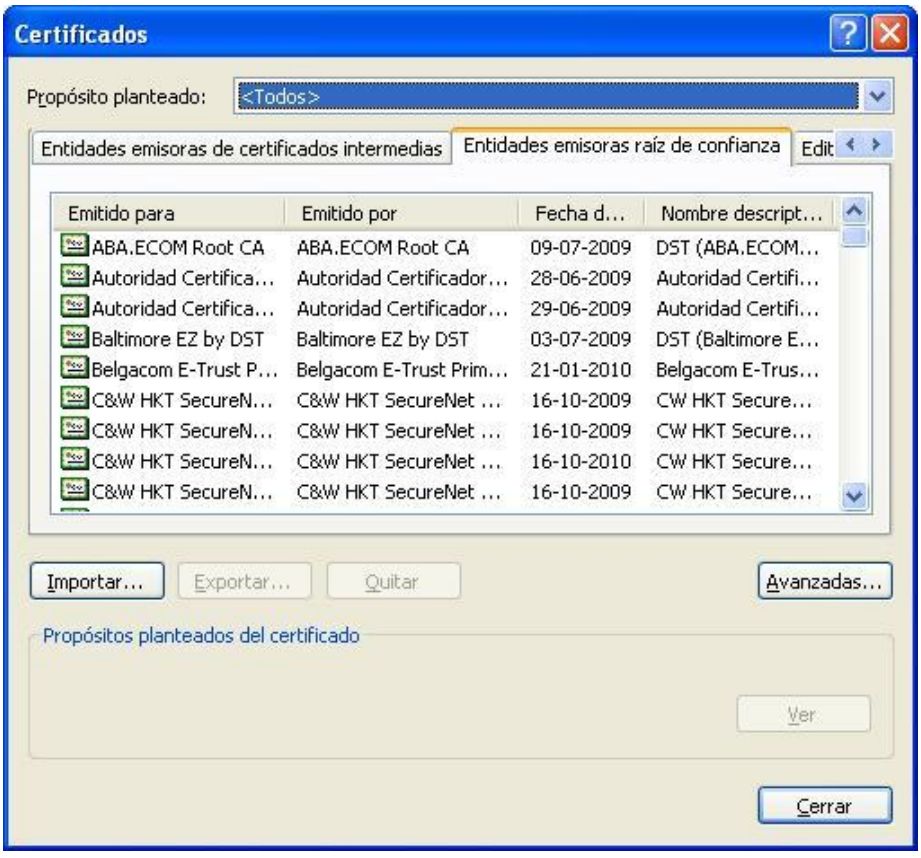


Figura 14: Ejemplo de certificados raíces instalados en el navegador Internet Explorer

Vamos a suponer entonces que nuestra Autoridad Certificadora es raíz. Dado eso, hay que generar un certificado autofirmado por la AC. El comando siguiente va a permitir crear un

par de claves (pública /privada), junto al respectivo certificado autofirmado X509 para nuestra AC. Las claves se generan por defecto usando el algoritmo RSA con tamaño de 1024 bits, y el certificado va a tener una vigencia de 365 días. (Se mostrará en negrita los datos ingresados por el usuario).

```
$ openssl req -new -x509 -days 365 -keyout ca-privkey.pem -out ca-cert.pem
Generating a 1024 bit RSA private key
........++++++
............++++++
writing new private key to 'ca-privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CL
State or Province Name (full name) [Some-State]:REGION METROPOLITANA
Locality Name (eg, city) []:SANTIAGO
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IN3
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:Autoridad Certificadora Imaginaria
Email Address []:aci@in3.cl
$ ls
ca-cert.pem  ca-privkey.pem
```

Se generaron el certificado (ca-cert.pem) y la clave privada protegida por una contraseña (*pem phrase*) ca-privkey.pem. Estos archivos, tal como lo indica la extensión, están en formato *Privacy Enhanced Email* (PEM).

Sobre la seguridad de una contraseña:
Es muy importante que el certificado con la clave privada sólo sea accesible para quién la generó y quién la va a usar. En el caso que sea robado, se podría ejecutar un ataque para encontrar la contraseña que protege la clave privada denominado *de diccionario*, que consiste en probar claves posibles construyendo una lista con combinaciones de nombres de personas, palabras del diccionario, números y algunos otros caracteres. Ejemplos: gato22, #VivaChile#, Silvia2006 etc. .. Esta lista tiene la propiedad de que es mucho más pequeña que todas las combinaciones posibles y permite encontrar las claves con alta probabilidad. Algunos softwares, al solicitar al usuario la creación o cambio de contraseña detectan si la clave ingresada es demasiado débil, es decir, que podría ser descubierta con un ataque de diccionario. Para escoger una contraseña se recomienda recordar una frase y construir la contraseña con sólo las primeras letras de cada palabra. Ejemplo: "*Me gusta mucho este libro sobre el documento electrónico*" produce la contraseña "*Mgmelsede*".

7.25 Generación de solicitud de certificado del médico

Siguiendo con el ejemplo, ahora el médico va a generar su par de claves en su computador, y luego creará una solicitud de certificado que contendrá solamente la clave pública, pues la clave privada debe permanecer en su posesión exclusiva. Esta solicitud se mandará al la AC para que se genere el certificado.

Nota: Este caso es ligeramente distinto al que sucede con el modelo que siguen las AC Chilenas, dado que con estas últimas el solicitante debe acreditar su identidad frente a una Autoridad de Registro – que muchas veces es provista por la misma AC -, la que envía la solicitud a la AC, para posteriormente generarse las claves mediante una interacción entre el solicitante y el sitio web de la AC.

Primero hay que generar un par de claves (pública / privada). Se van a usar claves para el algoritmo RSA de un tamaño de 1024 bits.

```
$ openssl genrsa -out key.pem 1024
Generating RSA private key, 1024 bit long modulus
....++++++
..............................++++++
e is 65537 (0x10001)
```

Se puede ver el archivo pem que se generó. La clave esta codificada en base 64 conformemente al estándar Privacy Enhanced Email (PEM).

```
En Linux
$ cat key.pem

En Windows
> type key.pem

-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQDAEbMfXb+qEf8s5ZkgXINJHyEGBn6zxjn9GzZl0IZ5s/PhyQH3
<varias líneas omitidas>
ARID2C2WKvT7LzjG2SEklmgNCgkTxhR3npGNjxXnoA==
-----END RSA PRIVATE KEY-----
```

También se puede usar openssl para ver los diferentes componentes del par de claves:

```
$ openssl rsa -noout -text -in key.pem
Private-Key: (1024 bit)
modulus:
    00:c0:11:b3:1f:5d:bf:aa:11:ff:2c:e5:99:20:5c:
    <varias líneas omitidas>
    e6:4c:e7:c3:28:f9:39:7b:35
```

```
publicExponent: 65537 (0x10001)
privateExponent:
    0b:7b:83:67:e6:a4:e4:b7:fa:b7:66:6a:87:22:c0:
    <varias líneas omitidas>
    38:e5:d8:cc:c5:26:6a:41
prime1:
    00:e4:e4:e5:88:f1:86:67:00:d4:07:4c:e8:36:7e:
    <varias líneas omitidas>
    d9:0d:59:e3:6d
prime2:
    00:d6:d0:6c:c9:1f:58:c2:75:ce:3c:89:27:f7:92:
    <varias líneas omitidas>
    a6:0a:c4:51:e9
exponent1:
    2c:ac:7c:1c:31:3f:91:24:f9:3c:ff:86:a6:f4:1b:
    <varias líneas omitidas>
    fd:0d:19:35
exponent2:
    3a:28:3d:7b:8a:08:0e:c5:b4:2b:41:7a:d7:95:07:
    <varias líneas omitidas>
    6e:0f:ce:99
coefficient:
    7b:fe:05:f2:f0:06:00:18:25:2b:df:a0:8c:14:e5:
     <varias líneas omitidas>
    8f:15:e7:a0
```

Aparecen el módulo, los exponentes (privado y público), los números primos y también *exponent1*, *exponent2* y *coefficient* que son valores que permiten optimizar la operación de descifrado.

Con el par de claves ya se puede crear una solicitud de certificado. Se pide entrar la información que será verificada por la Autoridad Certificadora (País, Nombre etc.)

```
$ openssl req -new -key key.pem –out medico.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CL
State or Province Name (full name) [Some-State]:REGION METROPOLITANA
Locality Name (eg, city) []:SANTIAGO
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Hospital
Organizational Unit Name (eg, section) []:Neonatologia
Common Name (eg, YOUR name) []:Patricio Perez
Email Address []:patricio.perez@hospital.cl

Please enter the following 'extra' attributes
```

```
to be sent with your certificate request
A challenge password []:no.la.sabras
An optional company name []:
```

El "*challenge password*" estará incluída en la solicitud que se va a generar. La idea es usar esa clave para verificar la identidad del autor de la solicitud al entregarle el certificado.

7.26 Creación del certificado del médico firmado por la CA

Este archivo (*medico.csr*) se mandó ahora a la CA para que esa pueda crear el certificado.

```
$ openssl x509 -req -days 365 -in medico.csr  -CA ca-cert.pem -CAkey ca-privkey.pem -
CAcreateserial -out medico.cer
Loading 'screen' into random state - done
Signature ok
subject=/C=CL/ST=REGION METROPOLITANA/L=SANTIAGO/O=Hospital/OU=Neonatologia/CN=P
atricio Perez/emailAddress=patricio.perez@hospital.cl
Getting CA Private Key
Enter pass phrase for ca-privkey.pem:
```

Se creo el archivo *medico.cer* que es el certificado que se manda al médico usando el formato x509, con una validez de 365 días. Se puede notar que para firmar se pidió la contraseña de la clave secreta de la AC para poder acceder a la clave privada guardada en *ca-privkey.pem*.

7.27 Firma del documento XML

Como lo vimos antes, el estándar XML Signature ofrece varios tipos de firma: ensobrada, ensobrante y separada. En nuestro ejemplo vamos a crear una firma ensobrada. Para eso se necesita crear una documento XML plantilla donde vamos a tener la información a firmar y los elementos cuyo valor serán completados al momento de firmar (`<DigestValue>`, `<SignatureValue>` entre otros). Aquí esta el xml que vamos a procesar con XMLSec:

```
<PartidaNacimiento>
 <MedicoResponsable>
    <RUT>1111111-1</RUT>
    <email>patricio.perez@hospital.cl</email>
    <organizacion>Hospital</organizacion>
 </MedicoResponsable>
 <DatosRecienNacido>
       <RUTMadre>22222222-2</RUTMadre>
       <RUTPadre>33333333-3</RUTPadre>
       <FechaNacimiento>2006-03-01</FechaNacimiento>
```

```
      <Sexo>Femenino</Sexo>
 </DatosRecienNacido>
 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod   Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"
/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue></DigestValue>
     </Reference>
    </SignedInfo>
    <SignatureValue/>
    <KeyInfo>
    </KeyInfo>
  </Signature>
</PartidaNacimiento>
```

Este archivo, que llamaremos *partida-nacimiento-sin-firmar.xml*, se puede firmar con XMLSec de la manera siguiente:

```
      $ xmlsec --sign --privkey-pem key.pem -out put partida-nacimiento-firmada.xml
partida-nacimiento-sin-firmar.xml
```

Donde *key.pem* contiene la clave privada y *partida-nacimiento-firmada.xml* es el documento firmado. Con esta herramienta la firma se puede configurar al arbitrio del usuario (algoritmo de firma, algoritmo de hash (*digest*), de canonicalización, agregar transformaciones) tan sólo cambiando la plantilla de firma. Obviamente cambiar el algoritmo de firma implica usar claves diferentes hechas a la medida de éste.

7.28 Verificación del documento XML

La partida de nacimiento ha sido enviada al Registro Civil, los que deben certificar su autenticidad antes de dar validez a su información. Para verificar el documento firmado, se necesita certificado del autor. Esa clave pública puede venir incrustada en el XML, o puede usarse como archivo. En nuestro caso se trata del certificado *medico.cer* entregado por la AC. Para verificar la firma con XMLSec:

```
$ xmlsec --verify --pubkey-cert-pem medico.cer partida-nacimiento-firmada.xml
OK
SignedInfo References (ok/all): 1/1
Manifests References (ok/all): 0/0
```

7.29 Resumen

La seguridad en documentación electrónica replica las características requeridas con la documentación física, es decir, confidencialidad, integridad, autenticidad y no repudiabilidad, y la gestión de seguridad apunta principalmente a mitigar los riesgos de vulnerabilidades en la manipulación de la información contenida. La criptología, es la disciplina que estudia los problemas derivados de la transmisión de información en un ambiente hostil, siendo la criptografía la rama orientada a desarrollar herramientas para aumentar la seguridad de la información, y el criptoanálisis, la rama orientada a estudiar las posibles vulnerabilidades de las herramientas criptográficas.

La criptografía simétrica, se puede implementar con algoritmos secretos, pero dado que dichos algoritmos tarde o temprano son conocidos, se prefiere el uso de una clave secreta compartida entre las contrapartes de la comunicación. El uso de una clave secreta plantea el problema de cómo distribuirla entre las contrapartes. Para resolver este problema se inventó la criptografía de clave pública en donde cada contraparte posee una par de claves, pública y privada, que tienen la propiedad de que el mensaje cifrado con una puede ser descifrado con la otra, pero que no es posible en la práctica conocer una clave a partir de la otra. Una aplicación adicional de esta herramienta es la firma electrónica, en donde una emisor adjunta al texto claro, una versión cifrada de dicho texto con la clave privada del emisor. La contrapartes podrán luego, usando la clave pública del emisor, comparar ambos mensajes, y en caso de se iguales comprobarán que efectivamente el mensaje fue enviado por el emisor. Como una manera de hacer más eficiente este mecanismo, se agrega el uso de un resumen o huella digital, la cual es la realmente firmada por el emisor.

La criptografía de clave pública puede ser afectada por suplantaciones de identidad, para solucionar ésto se ha inventado la Infraestructura de Clave Pública, que define Autoridades Certificadoras que son reconocidas por una autoridad superior (por ejemplo el Estado) para emitir cerificados que acreditan que el dueño de una clave pública es realmente quien dice ser.

En Chile, la Firma Electrónica es una realidad legal desde el año 2002, existiendo a la fecha varias Autoridades Certificadoras y de Registro en operación. El estándar de codificación de documentos para el gobierno de Chile es el XML. Esta codificación tienen asociada un estándar para firma electrónica denominado XML Signature, que permite agregar varias firmas a un sólo documento, usar diversos algoritmos de firma, todos esto mediante mecanismo estandarizado y de código abierto.

CAPITULO VIII: SERVICIOS WEB PARA EL GOBIERNO ELECTRÓNICO

José Selman[1], *Rodrigo Frez*[1], *Andrés Neyem*[2]

[1]Departamento de Ciencias de la Computación, Universidad de Chile.
{jselman, rfrez}@dcc.uchile.cl

[2]Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile.
aneyem@ing.puc.cl

8.1 Introducción

Los Servicios Web son aplicaciones computacionales creadas con el propósito de intercambiar información entre distintas aplicaciones en la Web. Poseen una serie de componentes que se han estandarizado para describir su funcionamiento mediante capas de especificaciones. Esta tecnología pretende lograr una alta interoperabilidad y permitir una eficiente integración entre distintos sistemas distribuidos heterogéneos en la Web.

Las aplicaciones de hoy se alejan cada vez más de los sistemas con alto nivel de centralización, dirigiéndose a soluciones levemente acopladas, basadas en componentes distribuidos. Pero las primeras soluciones no lograron satisfacer completamente la necesidad de integración entre distintas plataformas. Los Servicios Web han sido diseñados para cumplir con los requerimientos de las aplicaciones actuales, entregando una especificación basada en un conjunto de estándares, que siendo independientes de la plataforma, logran un alto nivel de aceptación y utilización.

El más básico nivel de interacción entre aplicaciones, en el marco de los Servicios Web, es el intercambio de mensajes Extensible Markup Language (XML). La capacidad de XML de ser transportado a través de la mayoría de los protocolos presentes en la Web, asegura la interoperabilidad entre aplicaciones.

Los Servicios Web son aplicaciones web autocontenidas. No solamente son capaces de realizar operaciones por sí mismos, sino que también pueden incorporar otros Servicios Web. Esto permite completar transacciones complejas y de alto nivel. La naturaleza neutral de los Servicios Web, permite la composición de entidades elementales o complejas, independiente del proveedor.

8.2 Arquitectura de Servicios Web

Los Servicios Web están definidos, esencialmente por tres tecnologías, que proveen un marco sistemático y extensible para las interacciones entre aplicaciones. Estas tecnologías están construidas sobre los protocolos existentes en la Web, basados en estándares abiertos como XML. Principalmente son:

- Web Service Descrption Language (WSDL v1.1), que describe formalmente, las interfaces de un Servicio Web. Su principal característica es la de adoptar la especificación XML Schema, como su sistema de tipos, para describir tipos de datos y formato de mensajes. Fue diseñado para ser extendido, por lo que está preparado para servir a nuevos requerimientos.

- Simple Object Access Protocol (SOAP v1.1), que permite la comunicación entre Servicios Web y aplicaciones clientes. Soporta los más comunes protocolos de transmisión presentes en Internet como HTTP, SMTP y FTP. Posee implementación en prácticamente cualquier lenguaje de programación, debido a que los datos son representados utilizando XML.

- Universal Description, Discovery and Integration (UDDI) , que permite registrar y publicar ServiciosWeb, en especial sus características

La arquitectura de Servicios Web posibilita la Composición, esto es la combinación de distintos servicios localizados en nodos remotos, que le entregan a la arquitectura, la posibilidad de crear operaciones complejas.

La composición de Servicios Web es la integración coordinada de Servicios Web simples o complejos, para conseguir una tarea determinada. Este problema ha sido abordado por dos paradigmas complementarios. Uno es el modelamiento de procesos de negocio complejos y el segundo, la web semántica. Bajo las dos perspectivas es posible realizar composiciones automatizadas, pero la industria ha respaldado con más fuerza los esfuerzos para modelar procesos.

Figura 1: Arquitectura de Servicios Web

8.3 Reseña sobre la evolución de la interoperabilidad de sistemas

Los principales fundamentos de los Servicios Web, provienen de las especificaciones de la computación distribuida. La distribución de operaciones computacionales a través de múltiples estaciones de trabajo, involucra estudiar conceptos de comunicación síncrona y asíncrona, además de contar con herramientas de bajo nivel que permitan la integración entre aplicaciones.

Pero la comunicación de mensajes, requiere muy a menudo de la lógica presente en los lenguajes procedurales, como por ejemplo la invocación de funciones. A fines de la década de los 80, la Open Software Foundation (OSF), consorcio de fabricantes de computadores, desarrolló el estándar Distributed Computing Environment (DCE), que creó iniciativas para unificar y estandarizar las tecnologías existentes para la invocación de procedimientos remotos, lo que más tarde se formalizó con el nombre de "Remote Procedure Call" (RPC1).

Mas tarde, el consorcio "Object Management Group" (OMG) se propuso estandarizar la tecnología de procedimientos remotos. Este consorcio concentró sus esfuerzos en construir una especificación, que permitiera a distintas aplicaciones multiplataforma, efectuar requerimientos en objetos remotos. Así nació la arquitectura "Common Object Request Broker Architecture" (CORBA). Su objetivo fue plantear una arquitectura única, orientada al objeto, para la integración y desarrollo de aplicaciones distribuidas. Su principal característica fue hacer transparente el estado y ciclo de vida de los objetos. Al mismo tiempo Microsoft lanzaba su arquitectura "Component Object Model "(COM) y su protocolo de acceso a objetos remotos "Distributed COM" (DCOM), que en conjunto definen una especificación binaria de componentes. En fechas cercanas a la irrupción de los Servicios Web y debido a la popularidad del lenguaje de programación Java, nació la especificación "Remote Method Invocation" (RMI), una implementación especial de RPC.

El gran problema de CORBA y COM/DCOM, fue su pobre escalabilidad. Además no se adecuaron al surgimiento de la Web, que se convirtió en la arquitectura distribuida más popular de la historia, principalmente por su bajo acoplamiento y la masificación de Internet.

A comienzos del nuevo siglo, la computación distribuida necesitaba soluciones a sus más inmediatas necesidades, como la capacidad de intercambiar modelos que sirvan tanto a desarrolladores como integradores de sistemas. Además sería necesario el intercambio de información, tanto dentro como fuera de una organización, lo que plantea el problema de operar en plataformas heterogéneas.

8.3.1 Web Service Description Language (WSDL)

Esta especificación describe qué mensajes deben ser intercambiados para que la interacción con un Servicio Web sea exitosa. WSDL en su versión 1.1, es un documento XML para describir Servicios Web como una colección de puntos de comunicación, que pueden intercambiar ciertos mensajes. Para agentes y usuarios, el formato permite realizar una descripción formal de una interacción cliente-servicio. Un Servicio en esta descripción, es un punto abstracto en Internet, definido con un protocolo de red y un formato de mensaje,

donde las operaciones y mensajes descritos son también completamente abstractos. WSDL es extensible, lo que permite describir puntos y sus mensajes sin importar el formato o el protocolo de red usados para comunicarse. Sin embargo, la especificación 1.1, sólo describe como usar WSDL en conjunto con SOAP en su versión 1.1, HTTP GET/POST y MIME. Una completa descripción WSDL de un Servicio Web, provee dos tipos de información:

- Descripción del servicio a nivel de aplicación, o interfaces abstractas.
- Los detalles específicos del protocolo del cual se depende.

Luego, para acceder de forma exitosa a un Servicio Web, se debe seguir el protocolo definido. El documento WSDL introduce una gramática para describir puntos de emisión de mensajes, conocidos popularmente como "endpoints". Los elementos que permiten describir "endpoints" son:

- Messages: Contienen referencias a XML Schema, que definen las diferente partes de un mensaje.
- Operations: Lista de Mensajes involucrados en un flujo de mensaje. Por ejemplo una operación de requerimiento y respuesta contiene dos mensajes.
- PortType: Es el set de flujo de mensajes (Operations), que se encuentran en un "endpoint", sin detallar el transporte ni la codificación.
- Binding: Medio de transporte y codificación particular de un PortType.
- Port: Dirección en la red de un "endpoint" asociado al Binding que lo caracteriza.
- Service: Colección de "endpoints" relacionados.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions                        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://www.in3.cl/webservices/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://www.in3.cl/webservices/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
  </wsdl:types>
  <wsdl:message name="VerificarFirmaSoapIn">
  </wsdl:message>
  <wsdl:message name="VerificarFirmaSoapOut">
  </wsdl:message>
  <wsdl:message name="VerificarHuellaSoapIn">
  </wsdl:message>
  <wsdl:message name="VerificarHuellaSoapOut">
  </wsdl:message>
  <wsdl:portType name="ServicioVerificadorSoap">
    <wsdl:operation name="VerificarFirma">
      <wsdl:input message="tns:VerificarFirmaSoapIn" />
      <wsdl:output message="tns:VerificarFirmaSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="VerificarHuella">
```

```
      <wsdl:input message="tns:VerificarHuellaSoapIn" />
      <wsdl:output message="tns:VerificarHuellaSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ServicioVerificadorSoap" type="tns:ServicioVerificadorSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <wsdl:operation name="VerificarFirma">
    </wsdl:operation>
    <wsdl:operation name="VerificarHuella">
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="ServicioVerificador">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
    <wsdl:port name="ServicioVerificadorSoap" binding="tns:ServicioVerificadorSoap">
      <soap:address location="http://www.in3.cl/VerificadorFirma/Verificador.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

8.3.2 Simple Object Access Protocol (SOAP)

Este protocolo en su versión 1.1, se adapta a la naturaleza distribuida y heterogénea de la Web, en donde los mecanismos de comunicación deben ser independientes de la plataforma, y lo más liviano posible. XML es un lenguaje que permite la codificación de datos e información independiente de la plataforma, es la elección adecuada para SOAP, que es, en efecto, un protocolo liviano para el intercambio de información en ambientes descentralizados y distribuidos.
SOAP consiste de tres partes:

- Un envoltorio que define un marco de trabajo que describe qué existe en el mensaje y cómo procesarlo.
- Un set de reglas de codificación para expresar instancias de tipos definidos en una aplicación.
- Una convención para representar llamadas y respuestas a procedimientos remotos.

Documento SOAP de Requerimiento:

```
POST /VerificadorFirma/Verificador.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.in3.cl/webservices/VerificarFirma"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <VerificarFirma xmlns="http://www.in3.cl/webservices/">
```

```
      <FirmaDigitalizada>base64Binary</FirmaDigitalizada>
      <Rut>int</Rut>
      <DV>int</DV>
    </VerificarFirma>
  </soap:Body>
</soap:Envelope>
```

Documento SOAP de Respuesta:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <VerificarFirmaResponse xmlns="http://www.in3.cl/webservices/">
      <VerificarFirmaResult>boolean</VerificarFirmaResult>
    </VerificarFirmaResponse>
  </soap:Body>
</soap:Envelope>
```

8.3.3 Universal Descrption, Discovery and Integration (UDDI)

La especificación UDDI ofrece a los usuarios de Servicios Web un modo sistemático para encontrar proveedores de Servicios Web mediante un registro centralizado. Es equivalente a un gran directorio automatizado y en línea de servicios. Existen directorios accesibles mediante Navegadores Web convencionales, disponibles para todo público, como también directorios privados de organizaciones que necesitan organizar y publicar sus Servicios Web internos. La información en un directorio UDDI se describe mediante tres categorías principales:

- Páginas Blancas: Se refiere al nombre y dirección física del negocio, información de contacto, nombre del sitio web y alguna ota información de identificación.
- Páginas Amarillas: Contiene el tipo de negocio, como localizarlo, los productos y servicios que ofrece.
- Páginas Verdes: Se refiere a la información técnica acerca de los servicios del negocio, principalmente a los medios y formas de interactuar con él y las definiciones de los procesos de negocio.

UDDI, en su versión 2.0, provee de dos especificaciones básicas, que definen la estructura y operación de un servicio de registro:

- Una definición de la información a publicar de cada servicio y como codificarlo (esto es la UDDI Data Structure).
- Una Application Program Interface (API), que permita publicar y consultar (esto es la UDDI Programmer's API).

El acceso al registro puede ser logrado mediante programación, usando el protocolo SOAP, tanto para la publicación como consulta. UDDI codifica tres tipos de información (mediante documentos XML) acerca de un Servicio Web, llamados entidades:

- businessEntity, información que incluye, por ejemplo, el nombre y otros datos del proveedor del Servicio.
- businessService, información de categorización basada en negocio o tipos de servicio.
- tModel, información de los detalles técnicos del Servicio Web, que incluye un URI como referencia al documento WSDL que describe al Servicio Web.

La entidad businessEntity puede contener uno o más elementos XML businessService. Ambos pueden especificar un elemento XML categoryBag que categoriza el negocio o servicio. Cada entidad definida es identificada por una llave única garantizando su unicidad por ser un identificador único universal (UUID). La entidad de Servicio (businessService), contiene una lista de elementos XML llamados bindingTemplate, que codifican el acceso técnico a la información del servicio. Cada uno de estos elementos representa un punto de acceso de un Servicio Web. A su vez éstos contienen elementos llamados tModelInstanceDetails. Proveen la información técnica del servicio, en la forma de referencia a una entidad (llamada tModel), que describe las especificaciones técnicas que cumple el Servicio Web.

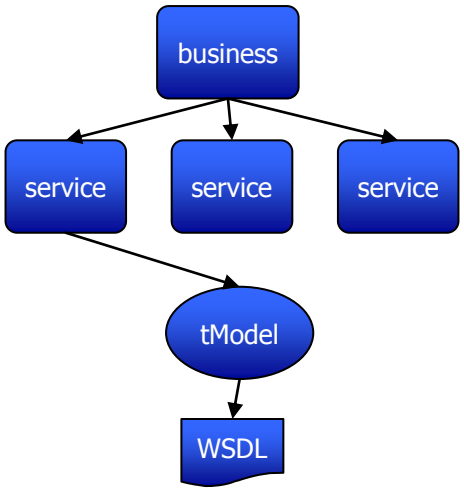


Figura 2: Arquitectura de un directorio UDDI

Al poseer un alto número de campos de información opcional y extensible, UDDI se presenta como una arquitectura que es fácil de utilizar, la cual puede contener casi cualquier tipo de información. Pero con toda esta flexibilidad y extensibilidad se hace

difícil concebir un patrón real de registro y búsqueda que permita soportar descubrimiento y búsqueda de valor. Por ejemplo, el único campo de información requerido para registrar un negocio es el nombre de éste. La búsqueda por nombres no entrega generalmente resultados correctos, debido, entre otros, a la ortografía y una serie de atributos que poseen los nombre, por ejemplo S.A. (Sociedad Anónima). Por ésto, lo mas probable es que un usuario reciba una lista de nombres, para luego examinar uno a uno hasta encontrar el correcto. Además no existe un formato obligatorio para la información de contacto, la información de identificación o la información de clasificación.

UDDI es un concepto que promete mucho para los Servicios Web. El problema que UDDI trata de resolver es enorme, pero con un gran beneficio potencial.

8.4 Extensiones y Actualizaciones

8.4.1 WSDL 2.0

El cambio desde la versión 1.1 de WSDL es significativo del punto de vista de la sintaxis y el modelamiento. Los principales cambios son:

- Inclusión de nueva semántica al idioma descriptivo. Uno de los efectos inmediatos es la obligatoriedad del atributo “targetNamespace", que define el espacio de nombres de cada documento WSDL.
- Se elimina el elemento XML “Message", en favor del sistema de tipos de XML Schema.
- Se elimina el soporte para sobrecarga de operaciones.
- Se renombra el elemento XML “PortType"por “Interface".
- Se agrega soporte para herencia de interfaces, a través del atributo “extends"del elemento XML “interface".
- Se renombra el elemento XML “Port" por “Endpoint".

La descripción de un Servicio Web en la especificación WSDL 2.0 se modela en dos partes. La primera de ellas es abstracta. WSDL 2.0 describe un Servicio Web en términos de mensajes que envía y recibe, a través del sistema de tipos de XML Schema (elemento XML “Type"). Los mensajes intercambian patrones que definen la secuencia y cardinalidad de éstos. Una operación (elemento XML “Operation") asocia el intercambio de patrones de mensajes con uno o más mensajes. Finalmente, una interfaz (elemento “Interface") agrupa estas operaciones independientemente del protocolo de transporte.

La segunda parte de la descripción, es la concreta. Los enlaces (elemento XML “Binding") especifican el protocolo de transporte de las interfaces. Un punto de emisión (elemento XML “Endpoint") de un servicio (elemento XML “Service"), asocia una dirección en la red con un enlace. Por último, un servicio agrupa los puntos de emisión que implementan una interfaz común.

8.4.2 SOAP with Attachments

Aunque SOAP y XML son tecnologías que responden acertadamente a la necesidad de describir datos, en muchas aplicaciones existen datos que necesitan de una descripción distinta. Por ejemplo los datos binarios contenidos en una fotografía digitalizada, o un audio de una conversación. En estos casos XML no es muy adecuado. SOAP with Attachments responde a esta necesidad, combinando el protocolo SOAP con el formato MIME, para permitir que cualquier dato pueda ser incluido como parte del mensaje SOAP. Es el mismo modelo utilizado para incluir mensajes atachados en el correo electrónico.

El formato MIME permite que múltiples bloques de datos sean encadenados en un mensaje. El encabezado MIME delimita donde comienza y termina cada bloque. En el caso de SOAP with Attachments, el mensaje consiste de múltiples partes MIME, o bloques de datos. La primera parte es el SOAP Envelope, y las restantes, representan los mensajes atachados.

8.4.3 SOAP v1.2

En esta nueva versión se introducen cambios estructurales y en la sintaxis. El mayor cambio es que SOAP ahora se basa en un set abstracto de información XML, que incluye reglas de serialización de objetos. Esto ayuda a que otras tecnologías de la arquitectura de Servicios Web consuman, entiendan y extiendan de mejor forma los mensajes SOAP. En resumen, las principales diferencias con la versión 1.1 son:

- Se basa en XML Infoset, SOAP v1.1 se basa en XML 1.0.
- Los espacios de nombre para el “Envelope” y el “Encoding” han cambiado, para poder distinguir fácilmente entre las versiones 1.1 y 1.2.
- Agrega nuevos códigos para el manejo de excepciones.
- El modelo de procesamiento de mensajes es más sencillo a través del cumplimiento de una serie de reglas.
- Agrega semánticas para manejar mensajes desarrollados en distintas versiones de SOAP. Si un nodo SOAP v1.2 recibe un mensaje SOAP v1.1, responderá con un mensaje SOAP v1.1 con una excepción de versión errónea (“VersionMismatch")
- SOAP v1.1 define un sólo enlace hacia http, SOAP 1.2 define un marco de enlace abstracto, que permite definir requerimientos y conceptos comunes a cualquier tipo de especificación de enlace.

8.4.4 UDDI V3.0

La versión 2.0 de UDDI facilitó el crecimiento de los Servicios Web, principalmente debido a la introducción de relaciones de negocios, validaciones de jerarquías externas, internacionalización y un método de búsqueda o consulta. Lamentablemente muchos problemas persisten con UDDI, lo que ha confinado su adopción principalmente a registros privados en las intranets de las empresas. Tanto los usuarios de UDDI como los proveedores de soluciones han identificado un número de áreas que requieren mayor evolución, lo que permitió focalizar las áreas de esfuerzo para el desarrollo de UDDI versión 3.0:

- Soporte para ambientes con muchos servidores de registros UDDI.
- Separación de las políticas de negocio de la implementación.
- Introducir características de seguridad.
- Mejorar los mecanismos de consulta.

Algunas de las características que agrega la versión 3.0 de UDDI pueden resumirse en:

- Mejoramiento de la interfaz para aplicaciones, en lo que se refiere a custodia o almacenamiento de registros.
- Validaciones externas simplificadas, en lo que se refiera a las jerarquías arbitrarias usadas en las categorizaciones.
- Mejoras en las características de replicación.
- Mejoras en el uso y el contenido del registro en si.
- Soporte para WSDL mejorado.
- Una mejor definición de esquema de datos.
- Categorizaciones complejas, permitiendo que las categorizaciones puedan ser agrupadas, por ejemplo la longitud y la latitud.
- Soporte para extensiones en las funcionalidades del registro.

8.5 Seguridad para Servicios Web

WS-Security es una especificación propuesta inicialmente por Microsoft, IBM y VeriSign en el año 2002. Fue enviada a la organización de estándares OASIS, quién la estableció en Abril de 2004 como un estándar abierto.

WS-Security se enfoca en proveer seguridad para SOAP, proveyendo seguridad para los mensajes involucrados en transacciones de Servicios Web. WS-Security no define prácticamente ninguna nueva tecnología de seguridad, sino que se enfoca en la correcta y efectiva aplicación de las tecnologías existentes como aserciones SAML[30], firmas digitales XML Signature, encriptación usando XML Encryption a los mensajes SOAP.

WS-Security se enfoca en asegurar los mensajes de Servicios Web, sin importar dónde ni cómo este mensaje SOAP viaje, en contraste de otras tecnologías en la capa de transporte donde se establece un canal de comunicación seguro a través del cual viajan los mensajes quedando completamente expuestos al llegar a su destino.

WS-Security saca provecho a la flexibilidad de XML Signature y XML Encryption permitiendo asegurar selectivamente partes de los mensajes. Un problema común que ha molestado a los proveedores de firewalls es que los paquetes SSL/TLS son completamente opacos para los mismos. Esta situación no ocurre con WS-Security. Con Servicios Web como un firewall, puede leer las partes del mensaje que necesita. Esto ha resultado

[30] Ver http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

imposible con tecnologías anteriores que solamente entregan un canal de comunicación seguro.

Cada mensaje puede tener su propia estrategia de seguridad. En el caso de una simple llamada a un Servicio Web, tendremos dos mensajes: un requerimiento y una respuesta. Es completamente posible cifrar el requerimiento pero transferir la respuesta sin ser cifrada, no sería posible utilizando seguridad de transporte. Las principales diferencias de las implementaciones de seguridad en las distintas capas se pueden apreciar en la tabla 1.

Tabla 1: Comparación Seguridad Según Capa

Seguridad de Transporte	Seguridad de Mensajería
Punto a Punto	Destino a Destino
Madura, su implementación es relativamente directa	Nueva, relativamente compleja con muchas opciones de seguridad
No granular, enfoque del todo o nada	Muy granular, puede aplicar selectivamente a trozos de mensajes y solamente a los requerimientos o respuestas
Dependiente del Transporte	La misma estrategia puede aplicarse a distintas tecnologías de transporte

Esta flexibilidad trae un problema consigo: complejidad. Esto se debe a la complejidad de las tecnologías subyacentes como XML Signature y XML Encryption. Sin embargo, con el paso del tiempo, esta complejidad ira siendo absorbida por los ambientes y herramientas de desarrollo, que permitirán definir reglas para la manera que deben asegurarse los Servicios Web.

Al seleccionar seguridad en la capa de mensajería debemos estar concientes que estamos optando por una solución extremadamente poderosa y flexible. Se dice que la más grande amenaza de seguridad es la complejidad, por ello, a pesar que una de las mayores fortalezas de WS-Security es su flexibilidad, puede ser también su mayor debilidad.

8.5.1 Extendiendo SOAP con seguridad

WS-Security define un encabezado de seguridad para SOAP que provee un lugar estándar para poner artefactos de seguridad. El propósito de WS-Security no es inventar un nuevo tipo de seguridad, sino que de proveer un contenedor común para incrustar información de seguridad en los mensajes SOAP. El encabezado WS-Security esta compuesto de tres componentes principales:

- **Tokens de seguridad**: Cero, uno ó más tokens de seguridad. Usualmente no más de uno.
- **Elementos de contenido cifrado con XML Encryption**: Cero, uno ó más elementos XML Encryption. Estos pueden ser los elementos <ReferenceList> o <EncryptedKey> de XML Encryption.

- **Elementos de contenido firmados digitalmente con XML Signature**: Cero, uno ó más firmas XML Signature. Usualmente, si se incluye una firma, ésta firma como mínimo, alguna parte del cuerpo del mensaje.

En el siguiente listado se aprecia la estructura general de un encabezado WS-Security.

```
<S:Envelope>
      <S:Header>
            <wsse:Security>
                  <wsse:UsernameToken>
                        …
                  </wsse:UsernameToken>
                  <ds:Signature>
                        …
                  </ds:Signature>
                  …
                  <xenc:ReferenceList>
                        …
                        <xenc:DataReference URI=”#body”/>
                        …
                  </xenc:ReferenceList>
            </wsse:Security>
      </S:Header>
      <S:Body>
            <xenc:EncryptedData Id=”body” Type=”content”>
                  …
            </xenc:EncryptedData>
      </S:Body>
</S:Envelope>
```

8.5.2 Espacios de nombre

En general, los espacios de nombre utilizados en los encabezados de seguridad WS-Security son los encontrados en la tabla 2.

Tabla 2: Espacios de Nombre en WS-Security

Prefijo	Significado	Espacio de Nombre
Ds	Digital Signature	http://www.w3.org/2000/09/xmldsig
Wsse	WS-Security Extensión	http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
Wsu	Web Services Utility	http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
Xenc	XML Encryption	http://www.w3.org/2001/04/xmlenc

8.5.3 Tokens de Seguridad en WS-Security

Los tokens de seguridad son artefactos de seguridad incluidos en los mensajes que son típicamente usados para autenticación o autorización. Una gran variedad de tokens han sido definidos para ser usados en WS-Security. Además, WS-Security es lo suficientemente extensible para habilitar el uso de nuevos tokens de seguridad a medida que vayan emergiendo.

8.5.4 Token de Seguridad de Nombre Usuario y Contraseña

El *UsernameToken* provee la clásica opción nombre de usuario y contraseña. Recordemos algunos de los problemas típicos de este enfoque:

- Si las contraseñas son demasiado simples, entonces son fáciles de adivinar.
- Si las contraseñas son muy complejas, entonces generalmente son escritas para recordarlas, lo que hace que cualquiera pueda tener acceso a ellas.

Una situación peculiar respecto a poner un nombre de usuario y contraseña en los mensajes SOAP es que uno de los principales objetivos de usar seguridad en esta capa, es que el mensaje sea auto-protegido, por lo que tener la contraseña como texto legible incrustado en el mensaje se opone a este propósito. Un ejemplo de esto se puede apreciar en siguiente listado (ejemplo, Username Token):

```
<S:Envelope>
      <S:Header>
              …
              <wsse:Security>
                      <wsse:UsernameToken>
                              <wsse:Username>juanito</wsse:Username>
                              <wsse:Password>1234</wsse:Password>
                      </wsse:UsernameToken>
                      …
              </wsse:Security>
              …
      </S:Header>
      …
</S:Envelope>
```

WS-Security provee una alternativa al enfoque de la contraseña como texto legible. Este enfoque es llamado *PasswordDigest* y es una alternativa viable siempre y cuando tengamos la contraseña legible en ambos lados del intercambio. El proceso involucra el uso del mismo algoritmo de hashing en ambos lados del intercambio. En general, se utilizan tres elementos básicos como entrada para los algoritmos de hashing, por ejemplo para SHA1:

- **Estampilla de tiempo:** Se usa una estampilla de tiempo para agregar entropía al resultado. También ayuda para asegurar que el servidor acepte mensajes recientes,

descartando mensajes enviados N minutos antes de su recepción, para evitar ataques de repetición.

- **Un valor aleatorio (Nonce):** Un conjunto aleatorio de bytes que son utilizados para evitar ataques de repetición al agregar entropía. El lado del cliente tiene la responsabilidad de generar el denominado *Nonce*, posiblemente de manera aleatoria o guardando alguna especie de contador para que no sean repetidos dentro de algún período de validez. WS-Security recomienda al servidor, mantener un caché de *Nonces* durante su vida útil, rechazando los mensajes que sean recibidos de manera duplicada.

- **Constraseña:** Este elemento representa la contraseña o un secreto compartido.

En el siguiente listado se aprecia el uso de la versión *PasswordDigest* del *UsernameToken*.

```
<wsse:Security>
     <wsse:UsernameToken>
            <wsse:Username>juanito</wsse:Username>
            <wsse:PasswordType=”wsse:PasswordDigest”>
                   D2A12DFE8D90FC6…
            </wsse:PasswordType>
            <wsse:Nonce>EFD89F06CCB28C89</wsse:Nonce>
            <wsu:Created>2005-05-08T20:21:23Z</wsu:Created>
     </wsse:UsernameToken>
</wsse:Security>
```

8.5.5 Tokens de Seguridad Binarios

Un *BinarySecurityToken* puede ser contenedor de uno de los pocos tipos de credenciales binarias disponibles en WS-Security. Estos son certificados X.509 Versión 3 y tickets Kerberos. Dado que son binarios, se debe especificar el tipo de codificación para poder ser representados en XML.

8.5.5.1 Certificados X.509

Un certificado X.509 Versión 3 es un contenedor para la llave pública, para un par público/privado (asimétrico) de llaves. En el siguiente listado se ve un certificado X.509 incrustado en un token de seguridad binario.

```
<wsse:BinarySecurityToken ValueType=”wsse:X509v3” EncodingType=”wsse:Base64Binary”>
NIFEPzCCA9CrAwIBAgIQEmtJZc0…
</wsse:BinarySecurityToken>
```

WS-Security específica el cómo se debe incrustar un certificado en el encabezado de seguridad.

8.5.5.2 Kerberos

Kerberos es un protocolo de autenticación basado en tecnologías de llaves secretas que involucra un punto centralizado de distribución de llaves. WS-Security habilita el paso de un ticket Kerberos incrustado en un token de seguridad binario.

8.5.6 Tokens de Seguridad XML

Solamente los tokens de nombre de usuario y binarios fueron incluidos en la especificación original de WS-Security. No obstante, poco tiempo después fueron agregados perfiles para una serie de tokens de seguridad XML.

8.5.6.1 Tokens SAML

Las aserciones SAML tienen un problema similar a los certificados X.509. El receptor debe ser capaz de determinar que el remitente, o la entidad (siendo representada por el emisor en el caso que este tercero atestigüe a favor de la identidad del remitente), realmente posea pruebas de su identidad. Por esto, el receptor debe confirmar el sujeto de la aserción usando alguno de los siguientes métodos:

- **Sostenedor de Llave *(holder-of-key)***: Con este método, el remitente (solicitante) incluye una firma digital XML Signature que contiene un bloque *KeyInfo* con una referencia apuntando a la aserción SAML.

- **Remitente Atestigua *(sender-vouches)***: Con este método, el remitente del mensaje atestigua a favor de la aserción firmándola digitalmente.

La mayor diferencia entre ambos recae en que con el primero, la firma es generalmente creada por el sujeto de la aserción, mientras que en el segundo, la firma es creada por una tercera entidad que atestigua a favor de la aserción.

El concepto de usar tokens SAML con mensajes SOAP es poderoso, debido a que muchos de los escenarios descritos en la especificación SAML encajan perfectamente con los escenarios de Servicios Web. El uso de aserciones SAML en los encabezados de seguridad de los mensajes, permite al cuerpo del mensaje enfocarse en la información propia de la transacción y al encabezado a contener información de seguridad necesaria para su ejecución. WS-Security especifica el cómo incrustar aserciones SAML en el encabezado de seguridad en el perfil para este token. La estructura general de una aserción SAML en un encabezado de seguridad se puede ver en el siguiente listado:

```
<S:Envelope xmlns:S="...">
    <S:Header>
        <wsse:Security xmlns:wsse="...">
            <saml:Assertion
                      MajorVersion="1"
                      MinorVersion="0"
                      AssertionID="SecurityToken-ef912422"
                      Issuer="juanito"
                      IssueInstant="2005-05-14T16:47:05.6228146-07:00"
                     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
                ...
            </saml:Assertion>
            ...
        </wsse:Security>
    </S:Header>
    <S:Body>
        ...
    </S:Body>
</S:Envelope>
```

8.5.6.2 Tokens XrML

XrML *(eXtensible Rights Markup Language)* es un lenguaje XML que define una sintaxis para administración de derechos digitales (DRM). En la especificación WS-Security, se define como usar un elemento <*license*> en el encabezado de seguridad. XrML tiene un problema similar al encontrado en SAML y certificados digitales con respecto a la confirmación de las demandas de un sujeto representadas en la "licencia". Estas demandas son confirmadas al incrustar una firma digital XML Security que referencia a la información de la licencia que provee la información necesaria para validar la firma y en consecuencia probar que la demanda fue legítimamente atada al mensaje. En el siguiente listado, vemos que el elemento <*KeyInfo*> contiene una referencia en su elemento <*SecurityTokenReference*> al token de licencia identificado por el atributo *licenseId* mediante su equivalencia con la URI.

```
<S:Envelope>
  <S:Header>
    <wsse:Security>
      <r:license licenseId="urn:foo:SecurityToken:ab12345">
        <r:grant>
          <r:keyHolder>
            <r:info>
              <ds:KeyValue>…</ds:KeyValue>
            </r:info>
          </r:keyHolder>
          <r:possessProperty/>
          <sx:commonName>José Miguel Selman</sx:commonName>
        </r:grant>
        <r:issuer>
          <ds:Signature>…</ds:Signature>
        </r:issuer>
      </r:license>
      <ds:Signature>
        <ds:SignedInfo>
          …
          <ds:Reference URI="#msgBody">
            …
          </ds:Reference>
          …
        </ds:SignedInfo>
        …
```

```
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="urn:foo:SecurityToken:ab12345" ValueType="r:license"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </S:Header>
  <S:Body wsu:Id="msgBody">
    <PictureRequest xmlns="http://www.jselman.com/pics">
      <Picture format="image/gif">
        AxE1TrsRGGH…
      </Picture>
    </PictureRequest>
  </S:Body>
</S:Envelope>
```

8.5.6.3 Tokens XCBF

Los documentos XCBF *(XML Common Biometric Format)* también pueden ser usados como tokens de seguridad en el encabezado de seguridad, según lo descrito en la especificación de WS-Security, en la cual se presenta el ejemplo que se muestra a continuación. Los documentos XCBF son documentos XML, que codifican los formatos de patrones definidos en el formato *Common Biometric Exchange File Format* (CBEFF), contienen información biométrica (ADN, huellas digitales, escáner de retina, geometría de manos, etc.) de alguna identidad, cuyo propósito es ser utilizada para autenticación y autorización.

```
<S:Envelope xmlns:S="...">
  <S:Header>
    <wsse:Security xmlns:wsse="...">

      <wsse:XCBFSecurityToken

         xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
         Id="XCBF-biometric-object"
         ValueType="wsse:XCBFv1"
         EncodingType="wsee:XER">

         <BiometricSyntaxSets>
            <BiometricSyntax>
               <biometricObjects>
                  <BiometricObject>
                     <biometricHeader>
                        <version> 0 </version>
                        <recordType> <id> 4 </id> </recordType>
                        <dataType> <processed/> </dataType>
                        <purpose> <audit/> </purpose>
                        <quality> -1 </quality>
                        <validityPeriod>
                           <notBefore> 1980.10.4 </notBefore>
                           <notAfter>2003.10.3.23.59.59</notAfter>
                        </validityPeriod>
                        <format>
                           <formatOwner>
                              <oid> 2.23.42.9.10.4.2 </oid>
                           </formatOwner>
                        </format>
                     </biometricHeader>
                     <biometricData>
                        0A0B0C0D0E0F1A1B1C1D1E1F2A2B2C2D2E2F
```

```
                    </biometricData>
                  </BiometricObject>
               </biometricObjects>
            </BiometricSyntax>
         </BiometricSyntaxSets>

      </wsse:XCBFSecurityToken>

    </wsse:Security>
  </S:Header>
  <S:Body>
     ...
  </S:Body>
</S:Envelope>
```

8.5.7 Referencias a Tokens de Seguridad

Un problema relacionado con los tokens de seguridad es como referenciarlos desde otros lugares. Diferentes tipos de tokens de seguridad pueden, y en general tienen, diferentes estrategias para identificadores únicos. Por ejemplo, en el caso de XrML el elemento *licenseId* resulta ser el identificador único más apropiado.

Como solución, WS-Security introduce un elemento <*SecurityTokenReference*>, como una manera estándar para referenciar tokens de seguridad sin importar su formato. Dentro de un elemento <*SecurityTokenReference*> podemos referenciar un token de tres diferntes maneras: usando elementos directos, identificadores llave, nombres llave ó incrustado.

El uso de elementos directos sencillamente apunta a un URI, que opcionalmente provee una pista en el atributo *ValueType* que indica a qué tipo de token de seguridad se encuentra apuntando, como se aprecia en el siguiente código:

```
<wsse:SecurityTokenReference xmlns:wsse="…">
  <wsse:Reference URI="http://www.dcc.uchile.cl.cl/#X509token" ValueType="wsse:X509v3"/>
</wsse:SecurityTokenReference>
```

La estrategia de identificadores llave, incluye un identificador apropiado de la llave en el elemento. Este enfoque permite el uso de un identificador completamente único basado en el tipo específico de token, como se puede apreciar en el siguiente ejemplo:

```
<wsse:SecurityTokenReference xmlns:wsse="…">
  <wsse:KeyIdentifier ValueType="wsse:X509v3">
     uTHyQBrcgFu4xmo14mD/iYgyyIg=
  </wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
```

El enfoque de nombres llave establece el uso de un nombre arbitrario de la llave de un elemento <*ds:KeyName*> para referirse a un token de seguridad. La especificación, sin embargo, no recomienda el uso de esta estrategia debido a que este elemento no es necesariamente único y podría erróneamente referirse a múltiples tokens de seguridad.

La última estrategia contempla incrustar el token de seguridad en el elemento de referencia como se muestra a continuación:

```
<wsse:SecurityTokenReference xmlns:wsse="…">
  <wsse:Embedded>
    <wsse:BinarySecurityToken    ValueType="wsse:X509v3"    EncodingType="wsse:Base64Binary"
wsu:Id="X509Token">
      MIIEZzCCA9CgAwgIQEmtJZc0rqrJh5i…
    </wsse:BinarySecurityToken
  </wsse:Embedded>
</wsse:SecurityTokenReference>
```

Con el transcurso del tiempo emergerán nuevos tipos de tokens de seguridad que serán fácilmente incluidos en la especificación WS-Security, dada su gran flexibilidad. Los tokens existentes a la fecha proveen una poderosa tecnología de autenticación y autorización no solamente para el sector empresarial, sino que además para cualquier aplicación Internet basada en Servicios Web.

8.5.8 XML Signature en WS-Security

Existen algunas consideraciones especiales para usar XML Signature en un mensaje SOAP, pero técnicamente no es más que una firma XML Signature incrustada dentro del encabezado de seguridad de WS-Security. Este mecanismo es utilizado en WS-Security por dos grandes razones:

- Necesidad de verificación de las credenciales incrustadas en un token de seguridad.
- Proveer integridad persistente.

Dada la mutabilidad de los encabezados SOAP, los intermediarios pueden agregar o hacer cambios a los mismos, el tipo de firma que tiene más sentido para ser usado dentro de SOAP son las firmas *Detached* o separadas, en las cuales siempre existe un elemento explícito con referencias a lo que es firmado.

La posibilidad de cambios legítimos que pueden realizar intermediarios sobre el mensaje, podría ocasionar problemas con las expresiones XPath utilizadas para indicar la referencia. Hay que tener extremo cuidado al construir estas expresiones para que permanezcan válidas en el punto de validación de la firma.

El encabezado puede contener más de una firma XML Signature, pudiendo las firmas incluso tener traslapos entre sí. En varias circunstancias es necesario firmar los tokens de seguridad, de manera de verificar su integridad y veracidad.

```
<S:Envelope>
  <S:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken ValueType="wsse:X509v3" EncodingType="wsse:Base64Binary"
wsu:Id="X509Token">
        ...
      </wsse:BinarySecurityToken>
    </wsse:Security>
    <ds:Signature>
```

```
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="…"/>
        <ds:SignatureMethod Algorithm="…"/>
        <ds:Reference URI="#body">
          <ds:Transforms>
            <ds:Transform Algorithm="…"/>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="…"/>
          <ds:DigestValue>…</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>
        …
      </ds:SignatureValue>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:Reference URI="#X509Token"/>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
    </ds:Signature>
  </S:Header>
  <S:Body wsu:Id="body">
    …
  </S:Body>
</S:Envelope>
```

En el listado anterior, vemos que se firma el cuerpo del mensaje SOAP, debido al uso del URI *#body*. En muchas circunstancias es necesario firmar los tokens de seguridad. Como ya hemos descrito, existe una manera estándar para referenciar a los tokens de seguridad usando un elemento <*SecurityTokenReference*>. Debido a que este mecanismo es diferente al usado por URIs, la especificación WS-Security introduce una nueva transformación cuyo objetivo es dejar fuera del contenido firmado al elemento <*SecurityTokenReference*> pero no a la referencia. Un ejemplo del uso de esta nueva transformación introducida por WS-Security se puede ver en el ejemplo siguiente.

```
<S:Envelope>
  <S:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken Id="miTokenX509">
        …
      </wsse:BinarySecurityToken>
      <wsse:SecurityTokenReference wsu:Id="miTokenDeSeguridad">
        <wsse:Reference URI="#miTokenX509"/>
      </wsse:SecurityTokenReference>
      …
      <ds:Signature>
        <ds:SignedInfo>
          <ds:Reference URI="#miTokenDeSeguridad">
            <ds:Transforms>
              <ds:Transform Algorithm="…#STR-Transform">
                <wsse:TransformationParameters>
                  <ds:CanonicalizationMethod Algorithm="…"/>
                </wsse:TransformationParameters>
              </ds:Transform>
            </ds:Transform>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>…</ds:SignatureValue>
      </ds:Signature>
      …
    </wsse:Security>
  </S:Header>
  <S:Body>
    …
```

```
  </S:Body>
</S.Envelope>
```

WS-Security provee un contenedor estándar en su encabezado de seguridad SOAP para situar elementos de firmas digitales XML Signature. El uso principal de firmas digitales en sobres SOAP es para proveer mecanismos de verificación de integridad y de credenciales. Dada la gran flexibilidad de XML Signature se pueden firmar selectivamente trozos de los sobres SOAP.

8.5.9 XML Encryption en WS-Security

XML Encryption es utilizado en WS-Security para selectivamente, esconder información sensible dentro de los mensajes SOAP. Como vimos anteriormente, en XML Encryption se puede utilizar una llave compartida de "sesión" para obtener los beneficios de eficiencia de criptografía simétrica junto con los beneficios de administración y confianza de las tecnologías criptográficas asimétricas. Al igual que en el caso de XML Signature, el uso de esta tecnología se limita a la incrustación de un elemento XML Encryption en el encabezado de seguridad WS-Security.

```
<S:Envelope>
  <S:Header>
    <wsse:Security>
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod Algorithm="…"/>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier EncodingType="wsse:Base64Binary" ValueType="wsse:X509v3">
              F2jFla0GxSq…
            </wsse.KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>…</xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>
          <xenc:DataReference URI="#body"/>
        </xenc:ReferenceList>
      </xenc:EncryptedKey>
    </wsse:Security>
  </S:Header>
  <S:Body>
    <xenc:EncryptedData Id="body">
      <xenc:CipherValue>…</xenc:CipherValue>
    </xenc:EncryptedData>
  </S:Body>
</S:Envelope>
```

En el código anterior se puede ver que el elemento <*EncryptedKey*> usa una llave pública para cifrar una llave compartida, la cual a su vez es utilizada para cifrar el cuerpo del mensaje. Esta técnica, de encapsular la llave simétrica cifrada con algoritmos asimétricos dentro de los mensajes, es conocida como "envolvimiento de llave" ó "contenedor digital".

El uso de una llave de sesión no es obligatorio en WS-Security. Al igual que en XML Enryption, se puede usar una estrategia basada únicamente en el uso de una llave

compartida ó simétrica, un enfoque completamente asimétrico, o bien, una estrategia híbrida. Esta última alternativa es la más ampliamente utilizada.

WS-Security no provee un nuevo mecanismo para cifrar documentos XML, sino provee un contenedor para situar dicha información en el encabezado de seguridad SOAP. Los elementos principales SOAP, como <*Envelope*>, <*Header*> y <*Body*> nunca son cifrados debido a que son necesarios por la infraestructura SOAP para su manejo efectivo. La gran flexibilidad de XML Encryption habilita la encriptación selectiva de trozos del mensaje SOAP para que distintos intermediarios puedan tener acceso a distintos trozos del mensaje. El contenido puede cifrarse usando tecnologías simétricas, asimétricas o ambas, utilizando una llave de sesión.

8.5.10 Estampillas de Tiempo

En general, la especificación WS-Security no define ningún nuevo mecanismo de seguridad, solamente provee una manera estándar para incorporar los mecanismos existentes a los mensajes SOAP. La pequeña excepción a esto son las estampillas de tiempo, dado que WS-Security define una estructura específica para incluirlas dentro del encabezado de seguridad. Dicha estructura se puede apreciar en el siguiente ejemplo.

```
<S:Envelope>
       <S:Header>
              <wsse:Security>
                     …
                     <wsu:Timestamp>
                            <wsu:Created>2005-05-14T19:31:22Z</wsu:Created>
                            <wsu:Expires>2005-05-14 T19:46:22Z</wsu:Expires>
                     </wsu:Timestamp>
                     …
              </wsse:Security>
       </S:Header>
</S:Envelope>
```

El encabezado de seguridad SOAP provisto por WS-Security provee un contenedor estándar para la información relacionada a los diversos mecanismos de seguridad que pueden ser utilizados. WS-Security no introduce nuevas tecnologías ni conceptos, solamente estandariza la manera en la cuál tecnología existente debe ser aplicada a SOAP.

Los tokens de seguridad son artefactos que proveen mecanismos de autenticación y autorización de una manera muy flexible. Los tokens de seguridad, en general, heredan tecnologías similares fuera de WS-Security, como el uso de nombre de usuario y contraseña, certificados X509, aserciones SAML, tickets Kerberos, etc.

Las firmas digitales y encriptación son logradas mediante el uso de XML Signature y XML Encryption respectivamente, tecnologías que se adhieren perfectamente a la especificación, proveyendo mecanismos de gran flexibilidad que satisfacen todos los requerimientos de seguridad de transacciones basadas en Servicios Web.

El uso de tokens de seguridad compone uno de los aspectos más significativos de WS-Security, debido a la gran flexibilidad y extensibilidad que se adquiere al poder agregar nuevos documentos a la especificación sin tener que modificarla. Así, pueden agregarse nuevos tokens de seguridad a medida que vayan emergiendo.

Un problema que podría considerarse con WS-Security es que no especifica como asegurar un mensaje SOAP. Por ejemplo, no indica que partes de un mensaje deben ser cifradas o firmadas, sino que solamente provee una ubicación para incluir este tipo de información. Debe considerarse a WS-Security como un conjunto de herramientas que pueden ser aplicadas a escenarios de Servicios Web. La determinación de las políticas de seguridad queda en las manos de desarrolladores, arquitectos o administradores, pudiendo ser ésta, una tarea extremadamente compleja, debido a la amplia gama de posibilidades ofrecidas por WS-Security.

Debe tenerse presente que la gran flexibilidad, principal fortaleza y ventaja de WS-Security a sus predecesores compone a la vez su más grande debilidad debido a la complejidad que introduce.

8.5.11 Normas de Interoperabilidad

La WS-I, es una organización abierta cuyo objetivo es promover la interoperabilidad de Servicios Web entre plataformas, sistemas operativos y lenguajes de programación. Esta organización crea, promueve y soporta protocolos genéricos para el intercambio interoperable de mensajes.

La WS-I se ubica entre los organismos de especificaciones Internet ubicuos como los son la W3C, OASIS e IETF, y desarrolladores y usuarios. De esta manera, se definen perfiles de interoperabilidad tanto para Servicios Web por sí solo como para WS-Security, lo que define la manera para asegurar aplicaciones basadas en Servicios Web, pues recordemos que WS-Security solamente provee herramientas para asegurar Servicios Web y no define como asegurar un mensaje SOAP.

A la fecha, la WS-I ha tenido una gran aceptación y activa participación por parte de proveedores líderes de tecnología como lo son BEA, IBM, Microsoft, Oracle, SAP, etc. por lo que resulta crucial, en el tema de seguridad, desarrollar servicios conformes a lo indicado en los documentos del grupo de trabajo del perfil básico de seguridad, pues en caso contrario, nuestros servicios serán seguros pero no interoperables, llevándonos nuevamente al punto de inicio, quebrantando la razón de ser de la tecnología Servicios Web.

8.6 Servicios Web de Segunda Generación

Las especificaciones que componen la primera generación de Servicios Web (WSDL, SOAP y UDDI) proveen protocolos para describir, proveer y consumir servicios. Cada vez más, los Servicios Web involucran a una gran cantidad de participantes formando grandes unidades computacionales distribuidas, conocidas como actividades. La estructura de las actividades es generalmente compleja, al igual que las relaciones entre sus participantes. La

ejecución de éstas puede tomar un largo tiempo, debido a latencias de negocio e interacciones usuarias.

La segunda generación de especificaciones para Servicios Web ha emergido y evolucionado lentamente para posicionar a la plataforma de Servicios Web como un sucesor viable a anteriores plataformas distribuidas, entregándole las herramientas necesarias para satisfacer los complejos requerimientos de las aplicaciones e integraciones de hoy. Algunos de los requerimientos de infraestructura no satisfechos por la primera generación de especificaciones son:

- Contexto y Administración de Transacciones: Sin un contexto activo y duradero los Servicios Web actúan de manera independiente y no pueden participar en transacciones distribuidas.
- Definición y Ejecución de Procesos de Negocio: Para poder componer Servicios Web en flujos de trabajo estructurados, es necesario contar con un vocabulario estándar que permita definir procesos que puedan ser ejecutados.
- Confiabilidad y Mensajería: La arquitectura de Servicios Web debe ser eficiente, flexible y tolerante a fallas.
- Políticas y Meta Datos: Mecanismos para abstraer reglas de negocio de alto nivel, reglas de seguridad, y propiedades descriptivas que puedan ser aplicadas a grupos de servicios como políticas.

Existen varias especificaciones independientes que proveen soluciones a los requerimientos enunciados. El hecho que estas existan como módulos independientes compone un aspecto arquitectural clave de la plataforma de Servicios Web. Esta característica se ha tornado cada vez más significante a medida que nuevas especificaciones han ido emergiendo, estructurándose alrededor del concepto de aplicaciones componibles. El rango de funcionalidades de estas especificaciones continuará creciendo al igual que las implementaciones de las mismas. En general, todas las especificaciones proveen contenedores que se deben incrustar en la sección de encabezado de los sobres SOAP, por lo que también se les conoce como Extensiones SOAP (Ver Fig. 3).

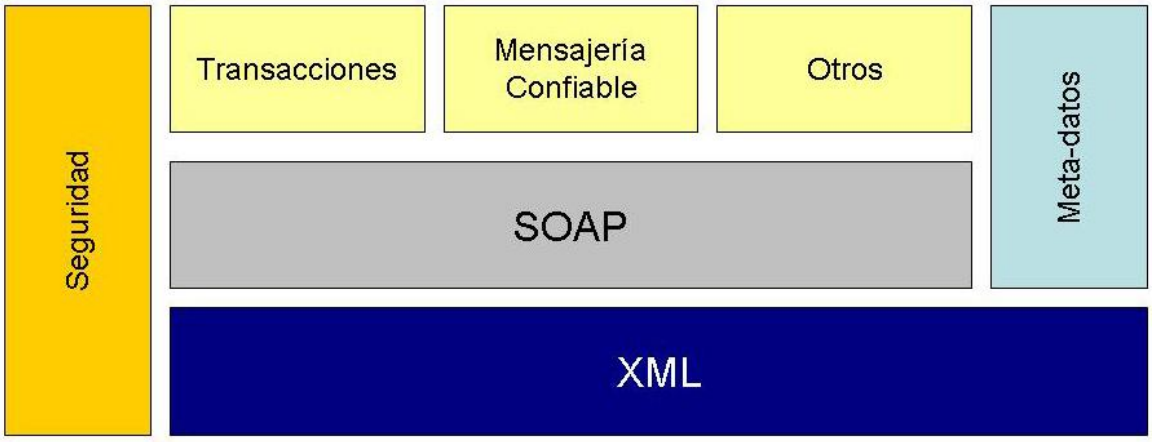


Figura 3: Extensiones SOAP

La mayoría de estas extensiones son mantenidas por cuerpos de estándares como W3C, IETF, OASIS, etc.

8.6.1 Transacciones

La naturaleza de bajo acople de Servicios Web hace que estos requieran un enfoque no tradicional para mantener un contexto conversacional persistente, siendo necesario contar con un servicio de administración de contextos para preservar la integridad de las actividades. Cuando un grupo de Servicios Web interactúan para ejecutar una unidad lógica, requieren compartir un contexto común, de manera que la actividad recibe una identificación que es propagada a cada servicio participante.

Las especificaciones relacionadas con transacciones para Servicios Web definen mecanismos para interoperabilidad transaccional entre dominios de Servicios Web, proveyendo un medio para componer las cualidades transaccionales del Servicio en aplicaciones de Servicios Web. Las especificaciones proveen un framework extensible de coordinación, WS-Coordination, y tipos específicos de coordinación para:

- Procesos de Corta Duración, o Transacciones Atómicas, WS-AtomicTransaction.
- Procesos de Larga Duración, o Actividades de Negocio, WS-BusinessActivity.

8.6.1.1 WS-Coordination

WS-Coordination describe un framework extensible para proveer protocolos que coordinan acciones de transacciones distribuidas. Tales protocolos de coordinación son usados para entregar soporte a un gran número de aplicaciones, incluyendo a aquellas que alcanzan acuerdos consistentes de acuerdo al resultado de actividades distribuidas.

Describe una definición de la estructura del contexto y de los requerimientos para la propagación de contexto entre servicios que cooperan entre si.

8.6.1.2 WS-AtomicTransaction

La especificación WS-AtomicTransaction provee la definición del tipo de coordinación necesario para transacciones atómicas, o de corta duración, que debe ser usado con el framework extensible descrito en la especificación de WS-Coordination. Esta especificación define tres protocolos específicos de coordinación:

- Terminación *(Completion).*
- 2-fases volátil *(Volatile Two-Phase Commit).*
- 2-fases durable *(Durable Two-Phase Commit).*

Los desarrolladores pueden usar cualquiera de estos protocolos al construir aplicaciones que requieren la propiedad transaccional de todo o nada, sin embargo la recomendación es

aprovechar la característica de composición de especificaciones manteniendo el nivel de acople lo más bajo posible.

8.6.1.3 WS-BusinessActivity

WS-BusinessActivity provee la definición del tipo de coordinación necesario para transacciones tipo Actividades de Negocio, o de larga duración, que debe ser usado con el framework extensible descrito en la especificación de WS-Coordination. Esta especificación define dos protocolos específicos de coordinación:

- Acuerdo de Negocio con Terminación de Participantes *(BusinessAgreementWithParticipantCompletion)*.

- Acuerdo de Negocio con Terminación de Coordinador *(BusinessAgreementWithCoordinatorAgreement)*.

Al igual que en el caso de WS-AtomicTransaction la recomendación es aprovechar la característica de composición de especificaciones manteniendo el nivel de acople lo más bajo posible.

8.6.2 Mensajería Confiable

8.6.2.1 WS-ReliableMessaging

La especificación WS-ReliableMessaging describe un protocolo que permite a mensajes ser entregados de manera confiable entre aplicaciones distribuidas en presencia de fallas de componentes de Software, Sistemas o Red. El protocolo es descrito de una manera independiente, permitiendo que sea implementado, usando diferentes tecnologías de transporte de red. Para promover la interoperabilidad de Servicios Web es definida una atadura SOAP dentro de esta especificación.

Muchos errores pueden interrumpir una conversación. Los mensajes pueden perderse, ser duplicados o re-ordenados, luego los sistemas pueden experimentar fallas traduciéndose en una pérdida del estado volátil conversacional.

WS-ReliableMessaging provee un protocolo ínter operable que puede ser usado por un origen (cliente del servicio) y destinatario para garantizar que un mensaje enviado será entregado. Esta garantía es especificada en un aseguramiento, quedando bajo la responsabilidad del origen y el destino el proveerlos, debiendo levantar los errores correspondientes en caso contrario. Las consideraciones de persistencia relacionadas con la habilidad de un destino de satisfacer los aseguramientos, son responsabilidad de la implementación y no afectan al protocolo de transporte. Existen cuatro tipos de aseguramiento básicos que pueden ser provistos por los destinos:

- A lo más uno (*AtMostOne*): los mensajes serán entregados a lo más una vez sin duplicidad o un error será levantado. Es posible que algunos mensajes no sean entregados.
- A lo menos uno (*AtLeastOnce*): Todo mensaje enviado será entregado o un error será levantado. Algunos mensajes pueden duplicarse, siendo entregados más de una vez.
- Exactamente uno (*ExactlyOne*): Cada mensaje enviado será entregado sin duplicidad o un error será levantado.
- En orden (*InOrder*): Los mensajes serán entregados en el orden que fueron enviados, en caso contrario un error será levantado. Puede ser combinado con los aseguramientos anteriores pues no hace referencia a mensajes duplicados u omisiones.

La manera en la cual el emisor de un mensaje determina que el mensaje fue recibido por el destinatario es mediante la recepción de un mensaje de reconocimiento, como se puede ver en la Fig. 4.

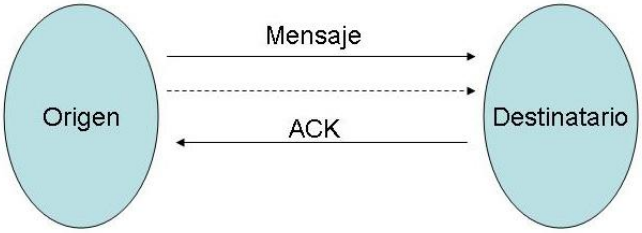


Figura 4: Mensaje de Reconocimiento

Los mensajes de reconocimiento pueden ser enviados al emisor en un mensaje separado, o bien como parte del mensaje de respuesta SOAP. En el siguiente código se aprecia un ejemplo en el cual el mensaje número 4 fue perdido.

```
<wsrm:SequenceAcknowledgment>
       <wsu:Identifier>http://www.dcc.uchile.cl/ServicioDummy/</wsu:Identifier>
       <wsrm:AcknowledgmentRange Upper=”3” Lower=”1”/>
       <wsrm:AcknowledgmentRange Upper=”6” Lower=”5”/>
</wsrm:SequenceAcknowledgment>
```

8.6.3 Otros

8.6.3.1 WS-Addressing

WS-Addressing provee un mecanismo independiente del protocolo de transporte para asignar direcciones a Servicios Web y mensajes. WS-Addressing logra ésto de una manera interoperable, definiendo dos unidades que contienen información típicamente provista por protocolos de transporte y sistemas de mensajería. Estas unidades normalizan esta información subyacente en un formato uniforme que puede ser procesado independientemente del transporte o aplicación. Estas unidades son:

- Referencias a “endpoints”.
- Encabezados de información del mensaje.

Los “endpoints” son definidos como una entidad, procesador o recurso (que puede ser referenciado) donde los mensajes de Servicios Web pueden ser enviados. Los encabezados de información del mensaje transportan características destino a destino que incluyen origen, destino e identidad de los mensajes. Ambas unidades han sido diseñadas para ser extensibles y re-usables de manera que puedan ser referenciadas y apalancadas por otras especificaciones.

A modo de ejemplo consideremos la siguiente porción de código donde se aprecia un mensaje enviado por http://www.dcc.uchile.cl/ServicioDummy hacia http://www.in3.cl/webservices/VerificarFirma.

```
<S:Envelope xmlns:S="http://www.w3.org/2002/12/soap-envelope"
         xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
<S:Header>
<wsa:ReplyTo>
<wsa:Address> http://www.dcc.uchile.cl/ServicioDummy </wsa:Address>
</wsa:ReplyTo>
<wsa:To> http://www.in3.cl/webservices/VerificarFirma </wsa:To>
<wsa:Action> http://www.in3.cl/webservices/VerificarFirma </wsa:Action>
</S:Header>
<S:Body>
       …
</S:Body>
</S:Envelope>
```

8.6.3.2 WS-Attachments

WS-Attachments es una especificación para especificar como transportar carga de trabajo fuera del cuerpo de un sobre SOAP (Ver Fig. 5). Existen dos situaciones en las cuales resulta conveniente enviar la información a ser entregada adjunta:

- Un archivo binario, por ejemplo una imagen. En este caso, el procesamiento que involucra codificar y decodificar los datos es no deseable debido a lo intenso que resulta en términos de ciclos de CPU.

- Un documento XML, o parte de un documento XML que no es fácilmente representado dentro del cuerpo del mensaje, por ejemplo documentos XML que usan un estándar de codificación diferente que el del mensaje en cuestión.

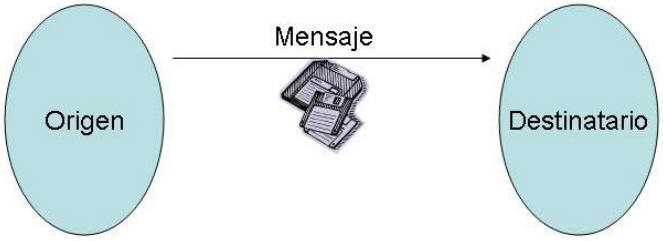


Figura 5: Mensaje con Datos Adjuntos

Cualquiera de estos escenarios podría causar problemas de procesamiento. La especificación WS-Attachments introduce una estructura SOAP compuesta, consistente de una parte principal, el mensaje SOAP base, y una parte secundaria que representa la información adjunta. Provee un mecanismo para que las partes principales reverencien partes secundarias, y para que las partes secundarias se reverencien entre ellas. WS-Attachments especifica a DIME[31] como el formato de codificación de la información adjunta, alternativa a MIME[32] que pese a ser menos popular resulta más apropiado para su uso con SOAP debido a su encapsulación liviana. SOAP With Attachments (SwA) es una especificación alternativa que usa MIME con estándar de codificación.

Es recomendable estar familiarizado con ambas especificaciones: WS-Attachments y SwA, manteniendo el nivel de acople con alguna de ellas lo más bajo posible, idealmente especificable de manera declarativa.

8.6.4 Síntesis

Hemos realizado una pequeña introducción a una pequeña muestra de las especificaciones WS-*. Nuevas especificaciones de esta familia aparecen con mucha frecuencia, al igual que sus implementaciones. En general ofrecen los servicios de infraestructura requeridos para funciones de negocio, lo que ha posicionado al framework de Servicios Web como una alternativa viable a tecnologías distribuidas tradicionales. La segunda generación de especificaciones de Servicios Web se encuentra construida alrededor del concepto de aplicaciones componibles, lo que representa una característica arquitectural clave de las mismas. El rango de funcionalidades ofrecidas por estas especificaciones crece a diario, así como especificaciones que traslapan o compiten entre sí, por lo que resulta indispensable hacer uso de ellas de manera declarativa, de manera que los servicios e implementaciones de las mismas sean provistos por el framework de desarrollo de Servicios Web.

Se recomienda, previa adopción de cualquiera de estas especificaciones, hacer un pequeño recorrido analizando el estado de las mismas o aparición de nuevas especificaciones y revisar la publicación de normas de interoperabilidad por parte de WS-I.

[31] DIME: Direct Internet Message Encapsulation
[32] MIME: Multipurpose Internet Mail Extensions.

8.7 Buenas Prácticas para Servicios Web

En general, debido a que la tecnología de Servicios Web reside sobre XML, las buenas prácticas para XML son válidas también. En lo que se refiere a la construcción y diseño de Servicios Web se puede enunciar el siguiente conjunto de buenas prácticas posibles de aplicar en un equipo de desarrollo que involucre el uso de Servicios Web:

8.7.1 Prácticas para planear proyectos con Servicios Web

8.7.1.1. Saber cuándo usar Servicios Web

Un pequeño proyecto de bajo riesgo es una buena oportunidad para dar el primer paso dentro del mundo de los Servicios Web. En este ámbito será posible integrar esta tecnología de manera limitada y controlada, ya que el ideal es no ir muy lejos esforzándose en integraciones que en poco tiempo quedarán obsoletas, ya sea por cambios en los estándares o por mejores alternativas de integración. Un conjunto de razones para considerar Servicios Web serían las siguientes:

1. La industria está adoptando y promoviendo el uso de Servicios Web. Mientras antes se incorpore al equipo de desarrollo, mejor será el entendimiento acerca de las nuevas arquitecturas de aplicaciones.
2. No es necesario crear una nueva arquitectura para usar Servicios Web. El diseño desacoplado de ellos permite agregar un número importante de servicios sencillos sin traer mayor impacto en el resto del sistema.
3. Si ya se está utilizando diseños orientados a servicios o modelos de negocio, los Servicios Web son el siguiente paso lógico. La incorporación de los paradigmas de orientación a servicios puede motivar técnicamente migrar a Servicios Web.
4. Muchas herramientas de desarrollo incluyen soporte para creación y uso de Servicios Web, ocultando al desarrollador el detalle técnico de bajo nivel. Esto permite tener una pronunciada y favorable curva de aprendizaje.

8.7.1.2. Saber cómo usar Servicios Web.

Siempre es mejor limitar el ámbito de implementación de Servicios Web, de manera que esté a la par con los límites del propio conocimiento. Aunque el concepto fundamental tras los Servicios Web tiene mucho en común con el diseño basado en componentes, posee diferencias significativas. Agregar Servicios Web de manera incorrecta a una aplicación, lleva generalmente a desarrollar desde cero mucho antes de lo esperado.

Si las funcionalidades a desarrollar son críticas para la organización, lo mejor es esperar hasta tener la absoluta certeza de la manera en que los Servicios Web deben ser integrados. Idealmente limitar el uso de servicios a prototipos de bajo riesgo que lleven a entender de mejor manera donde los Servicios Web son un mayor aporte dentro la propia tecnología.

8.7.1.3. Saber cuándo evitar Servicios Web

Incorporar Servicios Web sólo tiene sentido cuando su inclusión provee valor. Si no son o serán parte importante de la organización en un futuro cercano es mejor evitarlos, principalmente teniendo en cuenta:

1. La evolución y diferencias en la implementación de las especificaciones de la arquitectura de Servicios Web, en particular las especificaciones de segunda generación. Mejor es esperar hasta que cada especificación posea una amplio respaldo de la industria.
2. El peligro de utilizar ciegamente las herramientas que asisten al desarrollador en la creación de Servicios Web. Aunque ahorran trabajo en el lenguaje de marcado (WSDL, SOAP), éstos no asisten en optimizar o mejorar el diseño de la aplicación.
3. Las extensiones no estandarizadas que agregan soluciones de desarrollo propietarias, lo que crea dependencia en el proveedor. Seguir el camino de estas soluciones puede comprometer las oportunidades futuras de interoperabilidad.
4. Las aplicaciones autónomas no poseen como requisito el uso de Servicios Web. El uso de éstos se toma en consideración principalmente cuando la interoperabilidad es el tema central.

8.7.1.4. Actualizarse de forma incremental

Para realizar la transición a una arquitectura que considere el uso de Servicios Web, es mejor comenzar con una solución que utilice conceptos de orientación a servicios, pero que no dependa de la tecnología de Servicios Web para su implementación. Con esta aproximación es posible revertir el diseño al modelo basado en componentes, sin que exista gran impacto en el diseño general de la aplicación. De esta manera también será posible migrar de forma definitiva y natural a una arquitectura que descanse tecnológicamente en los Servicios Web en el futuro.

8.7.1.5. Considerar el verdadero peso de los sistemas legados

Siempre es mejor considerar la reutilización de la lógica de los sistemas legados, antes de reemplazarla por completo. Los Servicios Web pueden entregar una ventaja comparativa tanto en costo como en tiempo, al entregar las herramientas para incorporar la lógica incorporada en los sistemas legados, a una organización que necesita un alto grado de interoperabilidad.

Adicionalmente es importante tener en cuenta las limitaciones propias de incorporar un sistema legado que no fue diseñado para integraciones externas. En los casos en que estas limitaciones estén claras, adaptar un Servicio Web para que entregue cierta lógica de un sistema legado tiene mucho sentido.

8.7.1.6. El costo y retorno de inversión

Un buen Servicio Web requiere una gran cantidad de trabajo para asegurar que verdaderamente funciona y es confiable. Cada servicio desarrollado puede convertirse en

una pieza clave dentro de toda una infraestructura tecnológica. Aparte de exponer interoperablemente aplicaciones legadas y variados tipos de funcionalidades reutilizables dentro de un negocio, los Servicios Web pueden representar e incluso constituir procesos de negocios completos.

Si se construyen Servicios Web, el costo del desarrollo puede ser significativamente menor, si se utilizan las actuales herramientas de desarrollo que la organización posee, siempre ésta soporte la creación de Servicios Web.

La calidad de la interfaz que representa a un Servicio Web es muy importante tomando en cuenta las nuevas posibilidades de integración. Aunque los Servicios Web son catalogados como una tecnología muy poco acoplada, una vez que una serie de servicios se encuentran altamente integrados a una plataforma organizacional mayor, son la posibilidad de encontrar dependencias sobre las interfaces. Cambiar una interfaz una vez que ya ha sido construida, puede ser una tarea costosa y sucia, especialmente en ambientes en que un servicio depende funcionalmente de otro.

Sin embargo en la mayoría de los casos, ensamblar sistemas nuevos y sistemas legados a una arquitectura de Servicios Web requerirá una fracción del costo y el esfuerzo de un proyecto de integración tradicional. Debido a ésto, existirán retornos concretos y medibles a la inversión que se deberán tomar en consideración. Lo ideal es hacerlo bien al primer intento, pero lograrlo tendrá sus costos.

Aunque muchas organizaciones ya han invertido en proyectos orientados a arquitecturas XML, dar el paso a integrar tecnologías basadas en Servicios Web necesita de justificaciones adicionales, lo que es especialmente cierto cuando inversiones significativas ya se han dedicado a proyectos que dentro de una organización cumplen de buena forma su objetivo.

La receta general para mantener visible el retorno a la inversión es evaluarla cada vez que nueva información se encuentre disponible. Es muy probable que los beneficios de una arquitectura que utilice Servicios Web sean mucho mayores que los originalmente previstos, ésto debido que los beneficios de la migración son esencialmente tecnológicos y de implementación, y no dirigidos a los beneficios organizacionales de alto nivel.

El retorno de la inversión en la implementación de Servicios Web puede en algunos casos ser mas fácil de justificar que otras tecnologías basadas en XML. Los beneficios tienden a ser más tangibles debido a que la interoperabilidad resultante presenta ahorros que son fácil e inmediatamente identificables.

8.7.1.7. Diseñar mirando el futuro

Se puede tomar ventaja del potencial de interoperabilidad de los Servicios Web, aún si una integración no es demandada inmediatamente. Según ésto, es recomendable diseñar aplicaciones pensando en las futuras posibilidades de integración.

Introducir conceptos integradores requerirá cambios en los estándares de diseño, la arquitectura de desarrollo y la manera de pensar del equipo desarrollador. De esta manera se podrá facilitar, por ejemplo, que clientes locales y futuros clientes remotos posean interfaces de Servicios Web estandarizadas y con convenciones de nombre.

8.7.2 Prácticas relacionadas con la estandarización

8.7.2.1. Incorporar el estándar

Como en cualquier proyecto de desarrollo en una organización, donde existen distintos grupos desarrolladores construyendo distintas partes de un sistema, estandarizar el diseño de cada una de ellas es importante por todos los motivos tradicionales (robustez, mantenimiento, etc.). En el mundo de los Servicios Web, el beneficio real viene en el establecimiento de una interfaz estandarizada. En una organización ésto puede traducirse en sistemas estandarizados para navegar a través de la lógica de una aplicación, la arquitectura de la integración, los repositorios de datos corporativos y las partes de la infraestructura de la organización.

Poseer ambientes no estandarizados siempre retrasará el progreso e introducirá nuevos riesgos. Cuando se desarrollan Servicios Web dentro de una organización, se establece una infraestructura en la cual desarrolladores, integradores, y tal vez usuarios remotos, podrán utilizarla de forma común en años venideros.

La sola inclusión de una plataforma común para el intercambio de datos no es suficiente para asegurar la calidad de un ambiente orientado a los Servicios Web.

8.7.2.2. Tomar en cuenta las convenciones de nombre

Cuando se ensamblan las partes de una arquitectura integrada pueden aparecer múltiples componentes interdependientes, siendo cada uno de ellos necesario para la solución. Los ambientes de trabajo consisten en la mayoría de los casos en una mezcla de componentes de aplicación legados y contemporáneos, lo que por si sólo puede presentar algunas inconsistencias.

En un proyecto es muy fácil etiquetar los componentes de una aplicación de forma arbitraria. Un nombre es algo que se agrega rápidamente, de manera de poder concentrarse en tareas funcionales más importantes. Los beneficios de la estandarización de nombres no son evidentes hasta bien avanzados los ciclos de un proyecto, cuando es necesario comenzar a ensamblar los componentes.

Los nombres utilizados para identificar componentes públicos e inveraces de Servicios Web, actúan como puntos de referencia para otros componentes y Servicios Web. Cuando se modifica uno de estos nombres, se necesita cambiar todas las referencias a él. Como resultado, renombrar todos los componentes y Servicios Web se convierte en un proyecto en si mismo, durante el cual todo otro desarrollo es puesto en espera.

Utilizar una convención de nombres no sólo mejorará la eficiencia de la administración de una solución, hará de la migración y la implementación de los nuevos proyectos de integración, una tarea mucho más sencilla. Las convenciones de nombre reducen el riesgo de error humano y la posibilidad de que un pequeño cambio lleve a la solución a misteriosas caídas.

8.7.2.3. Diseñar Servicios Web contra una interfaz, no al revés

Para asegurar la consistencia en el diseño de "endpoints" de Servicio Web, el proceso común de desarrollo debe ser invertido. En vez de construir la lógica de la aplicación y luego expresar su funcionalidad a través de una interfaz de Servicio Web, se necesita tomar el diseño de tal interfaz como la primera tarea.

Con el conocimiento de los Servicios Web que deben ser encapsulados, es posible crear una interfaz consistente y genérica con características que cumplen con un modelo estandarizado. La mejor forma de asegurar la consistencia a través de todas las interfaces dentro de un marco de Servicios Web es hacer responsable a un rol del diseño de interfaces.

8.7.2.4. Separar el diseño de la implementación

Diseñar un Servicio Web es una tarea aparte de la implementación. Un diseñador de interfaces de Servicios Web es responsable de asegurar que las interfaces externas de todos los Servicios Web son consistentes y representan claramente la función de negocio del Servicio Web. Este diseñador típicamente será el dueño del documento WSDL, al cual los desarrolladores proveerán de códigos para su implementación. También puede estar a cargo de los documentos SOAP para asegurar también un formato consistente de mensajes.

Responsabilidades típicas de un diseñador de interfaces de Servicios Web:

- Documento WSDL
- Documento de mensajería SOAP
- Claridad de la interfaz
- Extensión de la interfaz
- Estandarización y convenciones de nombre de la interfaz

Los típico requisitos para este rol son poseer conocimientos en diseño de componentes, habilidades y conocimientos de SOAP y WSDL, capacidad de análisis del negocio y un buen entendimiento del ámbito de negocio de la organización.

8.7.2.5. Utilizar algún mecanismo de clasificación

Cada Servicio Web es único, pero muchos de ellos terminan realizando funciones similares y exhibiendo características comunes, lo que permite pensar en la clasificación de ellos. A continuación se expresan algunos beneficios del uso de clasificaciones:

- Es posible aplicar estándares de diseño específico a las distintas clasificaciones de Servicios Web.
- El tipo de clasificación comunica instantáneamente el rol del Servicio Web en la arquitectura.
- Las clasificaciones pueden alinearse con las políticas y estándares de seguridad de la organización.

8.7.3 Prácticas en la implementación

8.7.3.1. Utilizar un registro privado de Servicios Web

Una vez que los Servicios Web se establezcan como una entidad común dentro de la organización, naturalmente éstos comenzarán a evolucionar, requiriendo actualizaciones a las interfaces o la irrupción de nuevas generaciones de Servicios Web. Muy pronto se hará difícil mantener registro de todas las interfaces de cada servicio, especialmente tomando en cuenta que muchos de ellos siempre se encontrarán en estado de transición (debido a las migraciones desde los sistemas legados).

Un registro privado puede almacenar las descripciones de todos los Servicios Web que posea la organización. Actúa como un repositorio central para las actuales interfaces de servicios, a la cual puede acceder, previa autentificación, cualquier interesado en conocer acerca del marco de Servicios Web de la organización. Algunos de los beneficios inmediatos son:

- Acceso eficiente a las interfaces de cada servicio.
- Se preserva la integridad de todas las interfaces de servicio publicadas en el registro.
- Se motiva y promueve el descubrimiento de Servicios Web genéricos y reutilizables.

El registro de servicio debiera ser un proceso obligatorio dentro del marco de Servicios Web en una organización. Los usuarios no deben perder confianza en el registro, por lo que deben tener las garantías de que las interfaces publicadas corresponden a las últimas versiones disponibles.

Otro punto clave para hacer de un registro una parte importante y fundamental del marco de Servicios Web, es asignar un rol que sea responsable de mantener el registro de Servicios Web. En resumen es responsable de la administración del repositorio del registro de Servicios Web.

8.7.3.2. Considerar la administración

Una tarea subestimada en los proyectos de sistemas orientados al uso de Servicios Web es la administración necesaria para mantener la solución funcionando en el tiempo. El aumentar la interoperabilidad tiene como riesgo la más alta dependencia entre ambientes de aplicación. Con un mayor nivel de integración, mayor es la responsabilidad de mantener los Servicios Web funcionando de forma correcta, independiente de los parámetros que reciba

o devuelva. Los altos volúmenes de uso, las condiciones de error y otras variables de ambiente son situaciones que deben ser consideradas y anticipadas.

En este caso también es recomendable crear un rol de Administrador de Servicios, el cual es responsable de la mantención y monitoreo de las variables de ambiente. En ambientes donde un gran número de Servicios Web son utilizados debe existir una administración de sistema, para garantizar la ejecución confiable dentro del ambiente que acoge a los Servicios Web. Este rol es especialmente relevante en organizaciones que ofrecen Servicios Web que pueden ser accesados externamente. Para manejar efectivamente volúmenes de uso impredecibles, este rol debe ser capaz de responder rápidamente cuando el rendimiento comienza a caer.

El rol de Administrador de Servicios debiera ser responsable entre otros de:

- Solucionar la necesidad de contar con herramientas y máquinas que permitan realizar la administración, proponiendo y evaluando productos.
- Monitorear el uso de Servicios Web en forma individual.
- Analizar estadísticas de uso, para identificar patrones.
- Prever el impacto del uso de un Servicio Web sobre aplicaciones o componentes legados.
- Identificar y mantener registro de las dependencias entre los Servicios Web implementados.
- Manejar el control de versiones sobre los Servicios Web.
- Estar involucrado con el control de versiones de aplicaciones legadas representadas por Servicios Web.

8.8 Problemas de las especificaciones tradicionales de Servicios Web

8.8.1 Problemas en la descripción

La actual arquitectura de Servicios Web utiliza jerarquías semánticas y vocabularios estandarizados que entregan muy poca flexibilidad y expresividad. Esto es motivado por la orientación a personas en vez de máquinas. Para automatizar el proceso de descripción como el de descubrimiento, es necesario poseer lenguajes de descripción de Servicios Web que sean capaces de soportar entre otros:

1. Datos semi-estructurados
2. Tipos
3. Restricciones
4. Herencia

Sin contar con este tipo de funcionalidades y grado de flexibilidad, no podrán existir agentes automatizados que sean capaces de decidir cual Servicio Web disponible en un dominio se acomoda de mejor forma a sus requerimientos.

8.8.2 Problemas en el descubrimiento

Los estándares que componen la arquitectura de Servicios Web están diseñados para proveer descripciones del mecanismo de transporte (SOAP), y las interfaces utilizadas (WSDL). Sin embargo estas características no permiten lograr una automática localización de Servicios Web en base a las capacidades que poseen y las funciones que cumplen.

UDDI intenta describir Servicios Web, mediante un directorio jerárquico de negocios, que a través de una serie de atributos, permite navegar por una jerarquía clasificada de negocios que poseen servicios. Pero tampoco es capaz de representar las capacidades de un Servicio Web, por lo tanto, no entrega herramientas adecuadas para buscar servicios en base a lo que proveen. Aún más, la búsqueda sólo permite como entrada, palabras clave o *keywords*, lo que lleva a recibir resultados amplios y con una muy baja o nula relación.

Un Servicio Web puede existir en distintos repositorios UDDI, con distintos identificadores. No existen protocolos de comunicación estandarizados entre directorios, que permitan manejar una sincronización de servicios publicados. Además, un mismo servicio puede ser categorizado con atributos completamente distintos en dos directorios UDDI.

Cabe recordar que la experiencia práctica en el uso de directorios UDDI públicos (un directorio privado es siempre más riguroso), muestra que existe un gran número de descripciones de Servicios Web incompletas, erróneas o mal clasificadas. Un caso típico es encontrar Servicios Web publicados en un directorio, que ni siquiera cuentan con atributo que indique donde encontrar la descripción WSDL del servicio.

8.9 Servicios Web Semánticos

8.9.1 Introducción Web Semántica

Según Tim Berners-Lee, creador de la Web y director del W3C, la idea de la web semántica es hacer de la web un medio más colaborativo y entendible. Es colocar datos en la web de forma tal, que las máquinas puedan entenderlos de una manera natural. Emerge por la necesidad de solucionar los problemas de sobrecarga de información presente en la web, en la que una gran cantidad de ella se encuentra ubicada en sistemas cerrados, además de tener una administración de contenido deficiente. Los conceptos claves a considerar en el marco de la web semántica son:

Metadatos: Los metadatos son datos sobre los mismos datos. Se han utilizado en el área de ciencias de la información, donde se mantienen índices para la catalogación de libros. Entre los esquemas clásicos se encuentra el uso de fichas, que indican título, autor, país, ubicación física, y algunas taxonomías. Para utilizar metadatos no se requiere mantener un orden de la información, y típicamente se maneja una gran cantidad de metadatos. El hecho que un sitio Web contenga sus propios metadatos, permite mantener la información de catalogación y descripción de manera descentralizada.

Taxonomías: Una taxonomía es una jerarquía semántica en la cual entidades de información son relacionadas, ya sea por subclasificaciones o subclases. Las taxonomías permiten orientar la navegación por la información.

Ontologías En el área de la Inteligencia Artificial, una ontología se puede ver como un entendimiento compartido y común sobre un dominio, que puede ser usado entre personas y aplicaciones. Hablar de ontologías, es hablar de vocabularios con significado interpretable por máquinas. Por ejemplo, para describir un documento en la Web, existe la ontología Dublin Core, la cual constituye una base para el marcado en la Web. Una ontología se puede expresar utilizando los siguientes componentes:

Clases objetos en general
Instancias objetos en particular
Relaciones entre objetos
Propiedades de los objetos
Funciones procesos inmersos en los objetos
Restricciones reglas que circundan los objetos

La integración semántica en la Web, añade metadatos a las fuentes de información y utiliza ontologías para definir los esquemas conceptuales y sus relaciones. Esquemáticamente tiene tres niveles:

- **Objetos,** representados por documentos.
- **Conocimiento,** acerca de los objetos representado por metadatos y taxonomías.
- **Dominio,** representado por ontologías

8.9.2 Introducción a RDF

Resource Description Framework (RDF) es la solución recomendada por la W3C para la representación de metadatos. El modelo consiste en un grafo etiquetado y dirigido, donde el orden no es relevante. Para la serialización, RDF describe por referencia muchos de los aspectos de codificación que requiere el almacenamiento y transferencia de archivos. Por ejemplo, internacionalización y conjuntos de caracteres. Para estos aspectos, RDF cuenta con el soporte de XML.

El objetivo general de RDF es definir un mecanismo para describir recursos en la Web, o documentos electrónicos disponibles en la Web, sobre un dominio de aplicación particular, sin definir (a priori) la semántica del dominio. La definición del mecanismo debe ser neutral con respecto al dominio, sin embargo, debe ser adecuado para describir información sobre cualquier dominio.

El lenguaje RDF define afirmaciones, las que se componen por triples, que es una secuencia de tres elementos. Cada uno de los elementos de una afirmación, son identificados mediante una URI, que se utiliza como esquema de identificación universal de recursos. El objeto dentro de una afirmación es un literal, por lo que no es un URI. Es

clásico representar una afirmación en RDF, a través de un grafo dirigido y etiquetado, como lo muestra la Figura

8.9.3 Introducción a RDFS

RDF Schema (RDFS), es un esquema que establece un sistema de tipos simples, con el objetivo de definir vocabularios RDF. RDFS define una notación de clases, permitiendo la definición y descripción de nuevas clases, que se convierten en nuevos recursos. Permite la definición de jerarquías, a través de subclases y propiedades. Las propiedades pueden ser restringidas a una clase, ya sea en dominio o en rango.

8.9.4 Introducción a OWL

OWL permite definir ontologías para recursos presentes en la Web. Es por ello que posee más facilidades para expresar significados y semánticas que XML,RDF y RDFS.

8.9.5 Servicios Web Semánticos

Los Servicios Web son uno de los más importantes recursos presentes en la Web. La Web semántica tiene dentro de sus objetivos, el proveer mecanismos en los que agentes, entendidos como una máquina o aplicación, puedan localizar, seleccionar, utilizar, componer y monitorear Servicios Web automáticamente. Esto permitirá encontrar Servicios Web por sus metadatos, más que por palabras clave.

Las ontologías entregan gran expresividad y flexibilidad, junto con la habilidad de expresar datos semi-estructurados, restricciones, tipos y herencia. Los Servicios Web proveen modularización, manejo y escalabilidad. Los beneficios de ambas tecnologías pueden fusionarse, para entregar una arquitectura que describa de mejor forma los Servicios Web.

8.9.6 Ontología de Servicios Web

La comunidad de la Web semántica, ha desarrollado una serie de lenguajes de marcado como Ontology Web Language (OWL), y su predecesor, DARPA Agent Markup Language + Ontology Inference Layer (DAML+OIL). Tales lenguajes permiten la creación de ontologías para cualquier dominio, con lo que es posible instanciar estas ontologías para la descripción de cualquier sitio Web.

Para hacer uso de un Servicio Web, un agente necesita de una descripción del servicio procesable por una máquina, además de los medios y forma para accesarlo. Los sitios Web debieran utilizar un set de clases y propiedades básicas, para declarar y describir servicios. El mecanismo de estructuración de ontologías del lenguaje OWL provee un marco apropiado para realizarlo.

Los Servicios Web pueden ser simples, en el sentido que sólo involucran el acceso a una operación de un Servicio Web, como por ejemplo un servicio que retorne el código verificador del Rut. La otra posibilidad es que sean complejos, es decir el Servicio Web

está compuesto de múltiples servicios simples, que posiblemente requieran de una interacción o conversación entre el servicio y los agentes. La ontología OWL para Servicios (OWL-S) soporta ambos tipos de Servicios Web, aunque la principal motivación del lenguaje, son los Servicios Web compuestos.

En particular OWL-S presenta la clase "Service", que representa un Servicio Web. Es presentada por la clase "ServiceProfile", descrita por la clase "ServiceModel" y soporta una clase "ServiceGrounding".

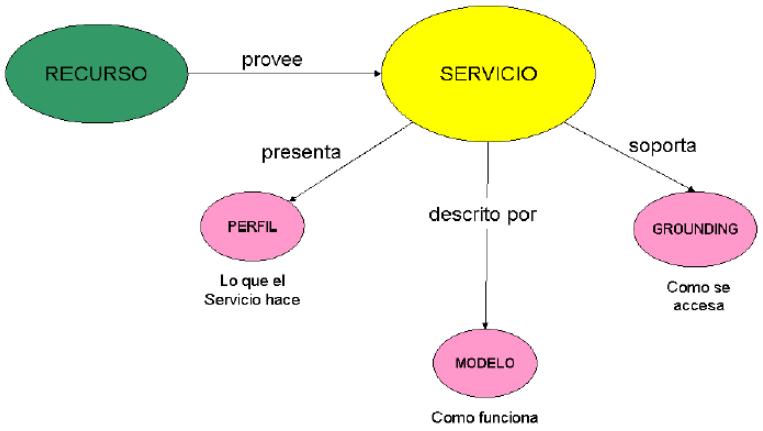


Figura 6: Ontología de Servicios Web

8.9.7 Arquitectura de Servicios Web Semánticos

Ejemplificando a través del escenario en que un agente necesita encontrar e invocar un Servicio Web, para poder cumplir los requerimientos de usuario, se tiene la siguiente cronología que cruza la arquitectura de Servicios Web Semánticos:

1. Los agentes encuentran un Servicio Web determinado, beneficiándose de los motores de registro y de búsqueda presentes en la Web, los cuáles usan la tecnología de directorios UDDI, o actúan como un sistema de búsqueda sobre los metadatos que define OWL-S, encapsulando la descripción WSDL y el medio de transporte.

2. Se obtiene la especificación de como coordinar la ejecución de un Servicio Web compuesto, esto es la ejecución coordinada y controlada de un número determinado de operaciones de distintos Servicios Web, ya sea de manera secuencial o paralela.

3. Se realiza el intercambio de mensajes entre un servicio y un agente a través del protocolo descrito. El agente puede monitorear el estado de ejecución del servicio, dado que ya conoce la especificación, condiciones previas y efectos de la tarea.

8.10 Referencias

[ERL] Thomas Erl. “Service-Oriented Architecture – A Field Guide to Integrating XML and Web Services”. Prentice Hall PTR. New Jersey. USA. ISBN: 0-13-142898-5. 2004

[ERN] Eric Newcomer. “Understanding Web Services – XML, WSDL, SOAP, and UDDI”. Addison-Wesley Pearson Education. Indianapolis. USA. ISBN: 0201750813. Agosto 2004.

[UDDIV3] UDDI.org, “UDDI Version 3 Features”. OASIS Group. http://uddi.org/pubs/uddi_v3_features.htm. Last visit: May 2009.