Publicación en línea de contenidos con RDF Site Summary (RSS)

Facultad de Informática

Universidad Nacional de La Plata

Director: Luis Mariano Bibbó

Alumno: Emiliano José Calabrese

Legajo: 1498/8

Indice de contenidos

Introducción

En los primeros años de la Web, la mayoría de los sitios Web no se interesaban sobre el intercambio de datos entre ellos. Hoy la tendencia es que los sitios Web sean interdependientes y muchos integran contenidos que se originan en otros lados. Este contenido puede constar de noticias, eventos, proyectos, información corporativa, etc. La integración efectiva de contenidos usualmente requiere cierto esfuerzo por parte del proveedor de información así como del receptor del mismo, y esto multiplicado por cada fuente de datos. Compartir contenidos entre sitios es usualmente llamado con el término inglés *Syndication*. Término tomado de otros medios como el gráfico, para referirse a la publicación de contenidos licenciados en forma simultánea en diferentes periódicos. Las formas tradicionales de encarar este problema de incorporación de contenidos son: obtención y parseo de HTML, utilización de APIs propietarias, vuelcos de base de datos, etc. Algunos de los problemas que sufren estos métodos tradicionales son: fuerte dependencia de la presentación de los datos con las aplicaciones que procesan dichos datos, falta de flexibilidad para el trato de la información, cierta complejidad para llevar a cabo la tarea, falta de consenso en la elección de la estructura de datos por parte de los proveedores y receptores, caída de los tiempos de desarrollo de las aplicaciones. RDF Site Summary (RSS) soluciona estos problemas.

Presentación de RDF Site Summary (RSS)

RSS es un modelo para la publicación en línea de contenidos y metadatos. Ofrece un nuevo modelo de sindicación (*syndication*) en la Web, permitiendo atraer tráfico desde diferentes lugares de la Web. Es exitoso porque soluciona de una manera simple un problema mayor que la publicación de contenidos: el intercambio de datos/metadatos. RSS estandariza un formato para la distribución de contenidos. Esto facilita para un proveedor de contenidos la distribución amplia del contenido, y para los afiliados la recepción y proceso de contenidos de múltiples fuentes. En la mayoría de los casos el contenido no es realmente distribuido, sino que se publican sólo los títulos con una descripción asociada (metadatos). Esto permite que los usuarios finales tengan la opción de navegar hacia el sitio proveedor de los contenidos si les interesan los tópicos presentados en el documento RSS (a través de hipervínculos asociados a cada tópico). Una vez publicado un archivo (canal) RSS, se lo puede registrar en los sitios aggregators que se quieran. Los aggregators son el uso más común de los canales RSS. Un aggregator es un sitio Web o sistema que recolecta canales RSS (feeds) de múltiples fuentes y luego hace algo con los mismos. Generalmente esto involucra compaginar y exhibir los contenidos de cada feed y quizá la creación de nuevos feeds compuestos a partir de los recolectados.

Publicar un canal RSS es sólo el comienzo. Es natural su incorporación en servicios adicionales; los datos RSS pueden fluir dentro de otros productos y servicios como PDAs, teléfonos celulares, e-mails. Las redes asociadas de sitios que comparten un mismo área de interés (por ejemplo una colección de sitios sobre desarrollo con Java) pueden cosechar feeds RSS entre ellos mismos, y automáticamente visualizar los nuevos artículos de los otros sitios en la red, manteniéndose actualizados unos con otros de forma sistemática, y conduciendo más tráfico hacia todas partes. De esta manera, se emplea el modelo RSS para maximizar la lectura de contenidos de estos sitios Web asociados. Esto es, los usuarios finales mediante clicks en los tópicos navegan entre estos sitios que comparten una misma temática, los cuales se favorecen mutuamente con el tráfico.

Los estándares

RSS está basado en dos estándares: eXtensible Markup Language (XML) y Resource Description Framework (RDF).

- **eXtensible Markup Language (XML):** es un conjunto de reglas para definir tags sintácticos los cuales fraccionan un documento en partes y de esta manera las identifican. Es un lenguaje de meta-markup (meta conjunto de tags) que define una sintaxis utilizada para definir otros lenguajes de markup estructurados y específicos de un dominio, como por ejemplo RSS. Un lenguaje de markup (conjunto de tags) es un mecanismo para identificar estructuras dentro de un documento estructurado. La especificación de XML define una forma estándar para agregar markup a los documentos. XML está diseñado para que sea fácilmente procesado por computadoras, para almacenar e intercambiar datos.
- **Resource Description Framework (RDF):** provee la base para la interoperabilidad a través de diferentes comunidades de descripción de recursos. Uno de los mayores obstáculos para una comunidad de descripción de recursos es la multiplicidad de estándares incompatibles para la sintaxis y esquema de los lenguajes de definición de metadatos. Esto explica la falta de aplicaciones y servicios interdisciplinarios. RDF proporciona la solución a estos problemas. Está basado en tecnologías Web y por lo tanto es liviano y de alta distribución. Provee interoperabilidad entre aplicaciones que intercambian metadatos y está orientado a varias áreas de aplicación incluyendo: descripción de recursos, clasificación de contenidos, comercio electrónico, servicios colaborativos. RDF es el resultado del consenso entre miembros de estas comunidades para sus necesidades sintácticas y esfuerzos de distribución/instalación. Su objetivo central es soportar la interoperabilidad de los metadatos. RDF es, además, una aplicación de XML. El mayor beneficio que proporciona RDF es que permite a las comunidades de descripción de recursos focalizar sus esfuerzos en las cuestiones de la semántica en vez de la sintaxis y estructura de los metadatos.

Estructuración del trabajo

En este trabajo se estudia y analiza el modelo de publicación en línea que ofrece RDF Site Summary (RSS). Los usos y las perspectivas de crecimiento de RSS. Se trata su historia y evolución en el tiempo reflejada en sus diferentes versiones, se presenta en detalle su sintaxis, el mecanismo de extensibilidad vía la creación de módulos RSS. Además se explican con cierto nivel de detalle los estándares en los que está basado el formato RSS: XML y RDF. Se enumeran algunas aplicaciones basadas en RSS y algunos sitios que ofrecen contenidos en este formato. También se da un vistazo hacia otras tecnologías de publicación en línea basadas en XML, como NITF, XMLNews, ICE, etc. En dos anexos se tratan la importancia de la sindicación en la Web y se presentan todos los módulos RSS estándar y en estado propuesto a la fecha de creación de este documento.

Además del análisis del modelo de publicación de RSS, se desarrolló un framework que resume el comportamiento común de las aplicaciones de captura de contenidos RSS (recolectores de datos de documentos RSS), denominados aggregators. Y para ejemplificar el comportamiento de este framework para aggregators se implementaron también dos aggregators particulares derivados del framework. Estos dos aggregators particulares capturan datos de la misma fuente de información, es decir, de los mismos documentos (canales) RSS, pero poseen diferentes necesidades de información. También se desarrolló un generador de canales RSS para completar la demostración del funcionamiento de los aggregators. Esta aplicación genera dos canales RSS y los actualiza con una frecuencia especificada. Ambos aggregators particulares están suscriptos a estos dos canales. Conjuntamente, el framework para aggregators, los dos aggregators particulares derivados de dicho framework y el generador de canales RSS conforman el prototipo desarrollado.

Además de graficar el comportamiento de los aggregators RSS, este prototipo pretende visualizar las características principales del modelo de publicación que propone RSS como: liviano (se emplean documentos XML y no se necesitan desarrollos ni protocolos de comunicación especiales), la extensibilidad (los canales incorporan propiedades específicas de un área de interés particular), simultaneidad de publicación (los datos de un mismo canal son capturados por diferentes entidades al mismo tiempo) y la uniformidad para el tratamiento de la información (no existen diferencias de ningún tipo para la especificación de datos y metadatos).

Para la implementación del prototipo se empleó el lenguaje de programación Java (la implementación de Sun Microsystems, Java 2 SDK, Standard Edition, versión 1.4.1_01). Para el procesamiento de los documentos XML se utilizó la Simple API for XML versión 2 (SAX2). La portabilidad y extensibilidad de XML y de Java hacen que sean compañeros naturales para cubrir los requerimientos de flexibilidad e integración de las nuevas aplicaciones.

RDF Site Summary (RSS)

Introducción

En los primeros años de la web, la mayoría de los sitios no se interesaban sobre el intercambio de datos entre ellos. Hoy la tendencia es que los sitios sean interdependientes y muchos integran contenidos que se originan en otros lados. Este contenido puede constar de noticias, eventos, proyectos, información corporativa, etc.

La integración efectiva de contenidos usualmente requiere cierto esfuerzo por parte del proveedor de información así como del receptor del mismo, y esto multiplicado por cada fuente de datos.

Compartir contenidos entre sitios es usualmente llamado con el término inglés *Syndication* (Sindicación). Término tomado de otros medios como el gráfico, para referirse a la publicación de contenidos licenciados en forma simultánea en diferentes periódicos. Un ejemplo de esto son las columnas en los diarios. Para más detalles ver ***Anexo I - La Sindicación***.

RSS ofrece un nuevo modelo de sindicación en la Web. RSS provee un mecanismo para atraer tráfico desde diferentes lugares de la Web. RSS es exitoso porque soluciona de una manera simple un problema mayor que la publicación de contenidos simultáneos, que es el intercambio de datos. Es mas efectivo utilizar RSS que encarar el problema de forma tradicional como es obtener y parsear HTML, la utilización de APIs propietarias, vuelcos de base de datos o *cobranding*.

Tomar y parsear el HTML de un sitio Web proveedor de datos es el más común método para compartir datos. El problema con este método de "cortar y pegar" es que una aplicación debe ser desarrollada y mantenida para cada fuente de datos. Estas aplicaciones generalmente deben ser modificadas cada vez que el proveedor cambia la presentación HTML.

La utilización de APIs es un método mas efectivo pero puede traer problemas. Las APIs son generalmente dependientes del lenguaje, su funcionalidad no es extensible. Además los programadores deben mantener cierta pericia para con el trato de cada API.

Los sitios Web también intercambian datos mediante vuelcos de base de datos. Pero los datos deben ser convertidos a ambos extremos y no se elimina el problema de tratar con múltiples formatos de datos. Esta opción puede funcionar si todos los proveedores de contenido usan el mismo modelo de datos para la distribución de información, un escenario bastante improbable.

Cobranding es un método en el cual el proveedor de la información posee versiones personalizadas de la aplicación para cada cliente. Esto puede ser una solución para suscriptores que no tienen recursos de programación. El problema es que los datos son presentados en un formato genérico que no satisface la interfaz del cliente, o se requiere que el proveedor mantenga un plantilla diferente para cada cliente. La funcionalidad está limitada a lo que la aplicación Web puede brindar. Y también se necesita un monto considerable de planificación y desarrollo del lado del proveedor.

Bajo el modelo RSS, cada sitio publica un archivo describiendo el contenido de su canal. Otros sitios pueden suscribirse a ese canal y tomar su contenido. El archivo RSS puede ser transformado a HTML y visualizado directamente en el sitio suscriptor o puede ser editado para seleccionar ciertos ítems, de acuerdo a las necesidades de información de la audiencia del sitio. Una de las ventajas de este método es que una vez construido el sistema para la suscripción a un canal, puede ser utilizado para la suscripción a miles.

RSS y la sindicación

RSS estandariza un formato para la distribución de contenido. Esto facilita para un proveedor de contenidos la distribución amplia del contenido, y para los afiliados la recepción y proceso de contenidos de múltiples fuentes. En la mayoría de los casos el contenido no es realmente distribuido, sino que se publican sólo los títulos con una descripción asociada (metadatos). Esto significa que los usuarios finales naveguen hacia el sitio proveedor si les interesan los tópicos. Una vez publicado un archivo RSS, se lo puede registrar contra los aggregators RSS (recolectores de datos de documentos RSS) que se deseen. Su contenido puede fluir hacia sitios Web socios o afiliados, listas de e-mails, PDAs, teléfonos celulares, etc.

Los Weblogs

Un Weblog (también llamado *blog*) es un portal a la vida de un individuo o agrupación. Las ideas inmersas en un Weblog generalmente son de tipo de personal, político, técnico y/ o comentarios editoriales que son significativas para el autor. El sitio Web que popularizó el Weblog es probablemente Slashdot.org, un sitio de contenidos tecnológicos dirigido a fanáticos de las computadoras. Scripting.com es un ejemplo precoz de un Weblog, es un sitio en los que los lectores obtienen una comprensión personal sobre la mente de Dave Winer. Dave Winer usualmente combina sus opiniones sobre innovaciones técnicas con política, filosofía e historia.

RSS es una buena base para la creación de un Weblog. Un ejemplo es PerlXML.com, un sitio que contiene noticias y recursos de Perl/XML. Un módulo en Perl se emplea para agregar nuevos titulares de noticias. El script actualiza la página HTML frontal y los titulares RSS, que luego son recogidos por varios aggregators incluyendo a my.netscape.com y my.userland.com.

RSS y el tráfico

Publicar un canal RSS es sólo el comienzo. RSS, verdaderamente una mini base de datos conteniendo titulares y descripciones sobre las novedades de un sitio Web, es natural su incorporación en servicios adicionales. Además de exhibir las noticias sobre otros sitios y visualizadores de titulares, los datos RSS pueden fluir dentro de otros productos y servicios como PDAs, teléfonos celulares, e-mails. Los informativos por e-mail pueden ser fácilmente automatizados con RSS. Las redes asociadas de sitios que comparten un mismo área de interés (por ejemplo una colección de sitios sobre desarrollo con Java) pueden cosechar feeds RSS entre ellos mismos, y automáticamente visualizar los nuevos artículos de los otros sitios en la red, manteniéndose actualizados unos con otros de forma sistemática, y conduciendo más tráfico hacia todas partes. De esta manera, se emplea el modelo RSS para maximizar la lectura de contenidos de estos sitios Web asociados. Los usuarios finales mediante clicks en los tópicos navegan entre estos sitios que comparten una misma temática, los cuales se favorecen mutuamente con el tráfico.

RSS impulsa en contexto múltiples puntos de entrada hacia un único artículo primario, en vez de emplearse múltiples copias del mismo artículo (que introduce los problemas de mantenimiento). Como se sabe, los sitios con más links hacia ellos mismos ganan, y los que poseen el contenido más reciente también ganan. RSS crea una situación de triunfo para lo anteriormente descripto.

El modelo de publicación RSS

La información que un usuario de la Web accede día a día, por ejemplo: titulares de noticias, resultados de búsquedas, vacantes de trabajo, etc., una gran suma de este contenido puede ser pensado como una lista.

Mucha gente necesita seguir la pista de un cierto número de estas listas, pero resulta dificultoso debido a que existen muchas fuentes. Por eso, la gente tiene que ir a cada página, cargarla, recordar cómo está formateada, y encontrar qué dejaron en la lista para leer.

RSS permite a las computadoras de la gente traer y comprender la información, y así, todas las listas de cada persona pueden ser rastreadas por alguna actualización y personalizadas. Es un formato que tiene la intención de ser usado por las computadoras en nombre de las personas, en vez de ser directamente presentado a ellas, como HTML.

Para permitir ésto, un sitio Web construye un *feed* RSS (suministro de datos RSS ó canal), accesible al igual que cualquier otro archivo o recurso sobre el servidor. Una vez que el canal RSS está disponible, las computadoras pueden regularmente tomar el archivo para conseguir los ítems más recientes en la lista. Usualmente, las personas hacen esto con un *aggregator*, que es una aplicación que maneja un cierto número de listas y las presenta en una única interfase.

RSS puede ser utilizado para otra clase de información orientada a listas, cómo la sindicación de contenidos propios (generalmente los weblogs) junto con los links.

Un canal RSS contiene una lista de ítems, cada uno de ellos es identificado por un link. Cada ítem tiene una cantidad de metadatos asociados. El más básico conjunto de metadatos soportado por RSS incluye un título y una descripción para cada ítem. Adicionalmente, el canal mismo puede tener metadatos asociados, y así establecerle un título y una descripción, como otros campos como por ejemplo datos sobre el editor y los derechos de autor, etc.

Los aggregators son el uso más común de los canales RSS. Un aggregator es un sitio Web o sistema que recolecta canales RSS (*feeds*) de múltiples fuentes y luego hace algo con los mismos. Generalmente esto involucra compaginar y exhibir los contenidos de cada feed y quizá la creación de nuevos feeds compuestos a partir de los recolectados. Existen varios tipos aggregators. Los Web aggregators, a veces llamados portales, renderizan los canales RSS en HTML y los publican en una página Web. Los aggregators también pueden estar integrados dentro de clientes de e-mail, desktops de usuarios, o ser independientes (software dedicado). Los aggregators pueden ofrecer una variedad de características especiales, incluyendo la combinación de varios canales relacionados dentro de una única vista, ocultamiento de ítems que el observador ya vio, y clasificación en categorías de ítems y canales.

Otros usos de los canales RSS son seguimiento de sitios mediante motores de búsquedas y otros programas de software. Debido al hecho que el canal es legible por máquina, el software de búsqueda no tiene la necesidad de separar que partes son importantes de las que están solamente para la navegación y presentación. Estos canales pueden ser re-publicados en otros sitios Web, teniendo la libertad de presentar su contenido como requieran.

Con la propuesta de distribución de contenidos que ofrece RSS, los proveedores de contenidos pueden seguir reteniendo los derechos de autor, si lo desean. Proporcionando un canal RSS, se puede controlar que información es sindicada en el canal, si es un artículo completo o sólo un segmento. También el contenido puede ser protegido por los mecanismos actuales de control de acceso, si sólo se distribuyen los links y los metadatos asociados. También se puede proteger el canal RSS mediante la encriptación SSL y la autenticación HTTP de usuario y contraseña, si se necesita.

En muchas formas, RSS es similar a la suscripción de boletines de noticias que muchos sitios ofrecen para mantener a los lectores al día. La gran diferencia es que los lectores no tienen que proveer su dirección de e-mail, bajando la barrera de las cuestiones de privacidad, mientras se continúa brindando el canal directamente a los lectores.

Para la elección del contenido de un canal RSS, es buena candidata la información orientada a listas en el sitio que los lectores puedan estar interesados en seguirle la pista o re-utilizar. Esto puede abarcar titulares de noticias, lanzamientos de productos, listas de trabajo, calendarios de conferencias, rankings, etc. Por ejemplo:

- **Noticias y anuncios**: titulares, noticias y cualquier anuncio que puede ser añadido en el tiempo.
- **Lista de documentos**: listas de páginas añadidas o modificadas, de esta manera la gente no necesita chequear constantemente por diferentes contenidos.
- **Marcadores (bookmarks) y otros links externos**: es natural con RSS compartir listas de links externos entre sitios.
- **Calendarios**: listas de eventos pasados o próximos, fechas de vencimientos o de vacaciones.
- **Listas de mails**: para cumplimentar un registro basado en Web de listas públicas o privadas de e-mails.
- **Resultados de búsquedas**: para permitir que la gente rastree cambios o nuevos resultados en sus búsquedas.
- **Bases de datos**: listas de trabajo, puesta a la venta de productos de software, etc.

También se utiliza RSS para implementar vistas personalizadas de datos, esto es, las personas pueden elegir categorías de información que se mostrarán en su propia página home. Cada vista personalizada es un canal RSS. RSS es muy útil tanto en Internet como en una intranet. RSS puede ser una herramienta muy útil para compartir e integrar información dentro de una compañía.

El formato RSS 1.0

Definición

RDF Site Summary (RSS) es un formato para la descripción de metadatos y *sindicación* (proceso de compartición de contenidos entre sitios). Se atribuye las características de liviano, multipropósito y extensible. RSS es una aplicación de XML (Extensible Markup Language) que conforma la especificación de RDF (Resource Description Framework). RSS es extensible vía los espacios de nombres de XML y/ o la modularización basada en RDF. Un sumario RSS, como mínimo, es un documento describiendo un *canal* conformado por ítems recuperables vía URLs. Cada ítem consiste de un título, hipervínculo y una breve descripción. Los ítems tradicionalmente eran titulares de noticias, pero RSS ha sido reposicionado, abarcando nuevas áreas.

Historia

RSS 0.9 fue introducido en 1999 por Netscape como un framework para la descripción de canales y mecanismo de recolección de contenidos para el portal My Netscape Network (MNN) [*http://my.netscape.com*]. Proporcionando una simple "instantánea en un documento", los productores de sitios Web adquirían audiencia a través de la presencia de sus contenidos en My Netscape. MNN utilizaba RSS como un formato liviano de sindicación basado en XML. RSS rápidamente se convirtió en una alternativa viable para sistemas ad hoc de sindicación y en muchos escenarios donde estándares pesados como ICE (Information and Content Exchange) eran invencibles. RSS se expandió para transportar, además de sindicación de titulares de noticias, foros de discusión, anuncios de software y datos propietarios.

RSS 0.91, re-apodado *Rich Site Summary*, siguió en poco tiempo el camino de la versión 0.9. Dejó atrás sus raíces en RDF e incorporó nuevos elementos del formato *scriptingNews* de Userland, el más destacado es el nuevo elemento <description> al nivel de ítem. Estos cambios llevaron a RSS al campo de la sindicación liviana de contenidos.

Mientras Netscape suspendió su emprendimiento en RSS, Dave Winer de Userland dirigió a RSS para que fuera adoptado como un framework para la sindicación. La inclusión de RSS 0.91 como uno de los formatos de sindicación para su producto *Manila* y el servicio relacionado *EditThisPage.com* llevó a RSS a los mundos de la sindicación y del *weblog* (portal que expresa ideas de un individuo o agrupación).

Motivación

Ha medida que RSS continúa siendo reposicionado en diferentes campos, añadido y recategorizado, la necesidad de un enriquecimiento en metadatos crece. Los elementos `title` y `description` de los niveles de `channel` e `item` están siendo sobrecargados con metadatos y HTML. Algunos productores incluso están insertando elementos ad hoc no oficiales (por ejemplo `<category>`, `<date>`, `<author>`) como un intento de aumentar las escasas facilidades de metadatos de RSS.

Una solución propuesta es la inclusión de elementos simples al núcleo de RSS. Esta dirección, es la más simple en corto plazo, sacrifica la escalabilidad y requiere modificaciones iterativas al núcleo del formato, agregando funcionalidad demandada y removiendo funcionalidad sin utilidad.

Una segunda solución, y la adoptada por la especificación oficial de RSS 1.0, es la compartimentalización de funcionalidad específica dentro de módulos RSS enchufables. La modularización se logra gracias al uso de los espacios de nombres de XML para particionar vocabularios. Agregando y removiendo la funcionalidad RSS es cuestión de la inclusión de un conjunto particular de módulos apropiados para la tarea a acometer. De esta manera, no hay necesidad de que el núcleo de RSS sufra sucesivas modificaciones.

Aplicaciones avanzadas de RSS están demandando representaciones más ricas de relaciones entre elementos intra- e inter- channel. RDF provee un framework para el modelado de los metadatos requeridos por éstas aplicaciones. RSS 0.9 provee una básica (a pesar de ser limitada) plataforma de RDF sobre la cual se puede incorporar más estructura.

Objetivos de diseño

El objetivo de diseño de RSS 1.0 es ser un formato para la descripción de metadatos y sindicación, con las siguientes características: basado en XML, liviano, multipropósito y extensible.

- **Liviano**: mucho del éxito de RSS se desprende del hecho que es simplemente un documento XML en vez de un sistema completo de sindicación tal como XMLNews y ICE (Information and Content Exchange).
- **Multipropósito**: al poco tiempo de liberada la versión 0.91 se lo incorporó en nuevos usos. RSS fue instado a evolucionar con las necesidades de expansión de las aplicaciones: agregación, hilos de discusión, listados de trabajos, servicios de listados múltiples, resultados de encuentros deportivos, catalogación de documentos, etc. Mediante el mecanismo de modularización basado en los espacios de nombres XML y en RDF, RSS 1.0 construye un framework para la expansión estándar y ad hoc de los propósitos de uso.
- **Extensible**: el punto crucial de la diferencia entre RSS 1.0 y las versiones anteriores (o paralelas) recae en su extensibilidad mediante la compatibilidad con los espacios de nombres XML y RDF. Los módulos basados en espacios de nombres permiten la extensibilidad compartimentalizada. Esto permite a RSS ser extendido:

    - Sin la necesidad de re-escrituras iterativas de la especificación central.
    - Sin la necesidad de consenso en cada elemento.
    - Sin inflar RSS con elementos que la mayoría no utilizará en ninguna aplicación particular.
    - Sin colisiones de nombres.

- **Metadatos**: RDF permite la representación de relaciones ricas en metadatos más allá de lo que es posible con las estructuras planas de las versiones anteriores de RSS. La existencia de la base de RDF en RSS 0.9 fue la razón para la elección de construir sobre la versión más antigua de RSS.

- **Sindicación**: es definida, en este contexto, como disponer datos en línea para la recuperación y adicionalmente transmisión, acumulación o publicación en línea. RSS es una "foto en un documento" de lo que se considera más interesante/importante sobre un sitio en un determinado momento. Esta información se hace disponible para el mundo para tomar, agregar, publicar en línea, etc., con links apuntando directamente al sitio originador para cada ítem de información.

La forma de un documento RSS 1.0

La sintaxis principal de RSS 1.0 está construida sobre RSS 0.9. RSS 1.0 es extensible mediante la modularización basada espacios de nombres XML. Mientras que la extensibilidad ad hoc es impulsada, se espera que un conjunto principal de módulos pre-acordados emerja, cubriendo funcionalidades tales como: taxonomía, agregación, Dublin Core, etc.

La estructura de un documento RSS (llamado también canal o *feed*) básico es estructuralmente bastante simple:

Declaración XML

Contenedor

Descripción de Canal (elemento *channel*)

Descripción de Imagen (elemento opcional *image*)

Descripción de Item (elemento *item*)

Descripción de Item (elemento *item*)

...

Descripción de Item (elemento *item*)

Descripción de Entrada de Texto (elemento opcional *textinput*)

Cada recurso (channel, image, item(s), textinput) que describe un documento RSS debe tener una URI asociada para especificar canónicamente qué se está describiendo. Esto es llevado a cabo en RDF brindando un atributo `about`.

- Dado que el elemento `<channel>` expresa información sobre el canal RSS mismo, se empleará para el atributo `about` la URL donde se encuentra el documento RSS.
- El elemento `<image>` describe una imagen particular recuperable vía una URL, así que se empleará para el atributo `about` la `<url>` de la imagen.
- Cada `<item>` es una descripción de algo recuperable vía una URL, así que se empleará para el atributo `about` el valor del sub-elemento `<link>` del ítem.

- En el caso de `<textinput>`, se empleará para el atributo `about` el valor del sub-elemento `<link>`, que es la URL donde se envía el GET.

Sintaxis

La siguiente es una descripción completa de la sintaxis de un documento RSS 1.0.

Aclaración: en las descripciones de modelo de los elementos principales se utiliza una sintaxis similar a la de los DTDs, solamente por propósitos de presentación, pero la misma no implica orden de los sub-elementos.

1. `<?xml version="1.0"?>`

   Al ser una aplicación de XML, un documento RSS no requiere comenzar con una declaración XML, pero se sugiere que aparezca como una buena práctica.

   Sintaxis: <?xml version="1.0"?>
   Requisito: Opcional (a menos que se especifique la codificación)

2. `<rdf:RDF>`

   El nivel más alto en cualquier documento RSS 1.0 es el elemento RDF. El tag de apertura RDF asocia el prefijo de espacio de nombres rdf: al esquema de sintaxis RDF y establece el esquema de RSS 1.0 como el espacio de nombres default para el documento. La utilización del prefijo rdf: es normativa.

   Sintaxis: <rdf:RDF xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns# xmlns="http://purl.org/rss/1.0/">
   Requisito: Requerida exactamente como se muestra, aparte de las declaraciones de espacios de nombres adicionales
   Modelo: (channel, image?, item+, textinput?)

`2.1 <channel>`

El elemento channel contiene metadatos describiendo el canal mismo, incluyendo un título, una breve descripción y un link URL que apunta al recurso descripto (por ejemplo, la página de inicio del proveedor del canal). La URL {recurso} del atributo rdf:about debe ser única con respecto a cualquier otro atributo rdf:about dentro del documento RSS y debe ser una URI que identifique el canal. Más comúnmente es, o la URL de la página de inicio que está siendo descripta o la URL donde se encuentra el archivo RSS.

Sintaxis: <channel rdf:about="{recurso}">
Requisito: Requerido
Atributo(s) Requerido(s): rdf:about
Modelo: (title, link, description, image?, items, textinput?)

2.1.1 <title>

Un título descriptivo para el canal.

Sintaxis: <title>{título del canal}</title>
Requisito: Requerido
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 40 (caracteres)

2.1.2 <link>

Haciendo una traducción HTML del canal, el título del mismo se vinculará a la URL que contiene este elemento link. Generalmente esta URL apunta a la página de inicio o de noticias del sitio.

Sintaxis: <link>{link_canal}</link>
Requisito: Requerido
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 500

2.1.3 <description>

Contiene una breve descripción del contenido, función, fuente, etc. del canal.

Sintaxis: <description>{descripcion_canal}</description>
Requisito: Requerido
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 500

2.1.4 <image>

Establece una asociación RDF entre el elemento opcional image y este canal RSS particular. El valor del atributo rdf:resource, el {imagen_uri}, debe ser idéntico al valor del atributo rdf:about del elemento image.

Sintaxis: <image rdf:resource = "{imagen_uri}" />
Requisito: Requerido sólo si el elemento image está presente
Modelo: Empty

2.1.5 <items>

Es una tabla RDF de contenidos, asociando los ítems del documento con este canal RSS particular. El rdf:resource {item_uri} de cada ítem en la secuencia debe ser el mismo que el rdf:about {item_uri} del elemento ítem asociado.

Una RDF Seq (secuencia) es utilizada para contener todos los ítems en vez de una RDF Bag, para denotar un orden de ítems para la traducción o reconstrucción.

Notar que los ítems que ocurren en el documento pero no aparecen como miembros de la secuencia de ítems del canal son propensos a ser descartados por los parsers RDF.

Sintaxis: <items><rdf:Seq><rdf:li resource="{item_uri}" /> ... </rdf:Seq></items>
Requisito: Requerido

2.1.6 <textinput>

Establece una asociación RDF entre el elemento opcional textinput y este canal RSS particular. El rdf:resource {textinput_uri} debe ser el mismo que el rdf:about {textinput_uri} del elemento textinput.

Sintaxis: <textinput rdf:resource="{textinput_uri}" />
Requisito: Requerido sólo si el elemento textinput está presente
Modelo: Empty

2.2 <image>

La imagen a ser asociada con una traducción HTML del canal. Esta imagen debe tener un formato soportado por la mayoría de los Web browsers.

Sintaxis: <image rdf:about="{imagen_uri}">
Requisito: Opcional; si esta presente debe también estar presente en el elemento channel.
Atributo(s) Requerido(s): rdf:about
Modelo: (title, url, link)

2.2.1 <title>

Texto alternativo (atributo "alt") asociado con el tag de imagen del canal, cuando se traduce a HTML.

Sintaxis: <title>{texto_alt_imagen}</title>
Requisito: Requerido si el elemento image está presente
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 40

2.2.2 <url>

La URL de la imagen a ser utilizada en el atributo "src" del tag de imagen del canal, cuando se traduce a HTML.

Sintaxis: <url>{image_url}</url>
Requisito: Requerido si el elemento image está presente
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 500

2.2.3 <link>

Es la URL que se vinculará a la imagen del canal en una traducción a HTML. La misma es, al igual que el link del título del canal, generalmente la página de inicio o de noticias del sitio.

Sintaxis: <link>{link_ imagen}</link>
Requisito: Requerido si el elemento image está presente
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 500

2.3 <item>

Generalmente es un titular de noticias, pero con la extensibilidad modular de RSS 1.0, puede ser cualquier cosa: cualquier objeto con una URI. Debe haber un mínimo de un ítem por documento RSS. RSS 1.0 no impone ningún límite superior, pero por compatibilidad con versiones anteriores, se recomienda un máximo de 15 ítems.

{item_uri} debe ser única con respecto a cualquier otro atributo rdf:about en el documento RSS y es una URI que identifica al ítem. {item_uri} debe ser idéntico al valor de <link>, sub-elemento de <item>, si es posible.

Sintaxis: <item rdf:about="{item_uri}">
Requisito: >= 1
Recomendación *(por compatibilidad hacia atrás con 0.9x):* 1-15
Atributo(s) Requerido(s): rdf:about
Modelo: (title, link, description?)

2.3.1 <title>

El título del ítem.

Sintaxis: <title>{titulo_item}</title>
Requisito: Requerido
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 100

2.3.2 <link>

La URL del ítem.

Sintaxis: <link>{link_ item}</link>
Requisito: Requerido
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 500

2.3.3 <description>

Una breve descripción/ resumen del ítem.

Sintaxis: <description>{descripcion_ item}</description>
Requisito: Opcional
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 500

2.4 <textinput>

El elemento textinput suministra un método para enviar un formulario de datos hacia una URL arbitraria. Se asume que el procesador del formulario en el extremo receptor sólo maneja el método HTTP GET.

El campo es típicamente utilizado como una caja de búsqueda o formulario de suscripción.

RSS 1.0 sugiere o bien su deprecación o su incremento con alguna forma de descubrimiento de recursos de este elemento en futuras versiones, mientras se mantiene por compatibilidad hacia atrás con RSS 0.9.

{textinput_uri} debe ser único con respecto a cualquier otro atributo rdf:about en el documento RSS y es una URI que identifica al textinput. {textinput_uri} debe ser idéntico al valor del sub-elemento <link> del elemento <textinput>, si es posible.

Sintaxis: <textinput rdf:about="{textinput_uri}">
Requisito: Opcional, si está presente, debe también estar presente en el elemento channel
Atributo(s) Requerido(s): rdf:about
Modelo: (title, description, name, link)

2.4.1 <title>

Un título descriptivo para el campo del textinput. Por ejemplo: "Búsqueda"

Sintaxis: <title>{textinput_title}</title>
Requisito: Requerido
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 40

2.4.2 <description>

Una breve descripción del propósito del campo del textinput.

Sintaxis: <description>{textinput_description}</description>
Requisito: Requerido
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 100

2.4.3 <name>

El nombre del campo del texto de entrada (la variable).

Sintaxis: <name>{textinput_varname}</name>
Requisito: Requerido
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 500

2.4.4 <link>

La URL a la cual el envío del formulario de datos será dirigido (utilizando el método GET).

Sintaxis: <link>{textinput_action_url}</link>
Requisito: Requerido
Modelo: (#PCDATA)
(Sugerido) Longitud Máxima: 500

Una restricción impuesta sobre los sub-elementos de los elementos del nivel más alto, es decir, los elementos `channel`, `image`, `item` y `textinput` es que los mismos no contengan sub-elementos repetidos (por ejemplo, `<item> <dc:subject/> <dc:subject/> </item>`). Esta restricción es sólo para los sub-elementos inmediatos. Cualquier profundidad mayor (con contenido más rico o elementos repetidos) está bien definida por el uso de la sintaxis RDF.

Ejemplos

El siguiente es un ejemplo de un documento RSS 1.0 básico, haciendo uso solamente del conjunto de elementos que conforman el núcleo (sin ningún módulo extra).

```
<?xml version="1.0"?>

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://purl.org/rss/1.0/"
>

  <channel rdf:about="http://www.xml.com/xml/news.rss">
    <title>XML.com</title>
    <link>http://xml.com/pub</link>
    <description>
      XML.com features a rich mix of information and services
      for the XML community.
    </description>

    <image rdf:resource="http://xml.com/universal/images/xml_tiny.gif"
/>
```

```
    <items>
      <rdf:Seq>
        <rdf:li
resource="http://xml.com/pub/2000/08/09/xslt/xslt.html" />
        <rdf:li
resource="http://xml.com/pub/2000/08/09/rdfdb/index.html" />
      </rdf:Seq>
    </items>

    <textinput rdf:resource="http://search.xml.com" />

  </channel>

  <image rdf:about="http://xml.com/universal/images/xml_tiny.gif">
    <title>XML.com</title>
    <link>http://www.xml.com</link>
    <url>http://xml.com/universal/images/xml_tiny.gif</url>
  </image>

  <item rdf:about="http://xml.com/pub/2000/08/09/xslt/xslt.html">
    <title>Processing Inclusions with XSLT</title>
    <link>http://xml.com/pub/2000/08/09/xslt/xslt.html</link>
    <description>
     Processing document inclusions with general XML tools can be
     problematic. This article proposes a way of preserving inclusion
     information through SAX-based processing.
    </description>
  </item>

  <item rdf:about="http://xml.com/pub/2000/08/09/rdfdb/index.html">
    <title>Putting RDF to Work</title>
    <link>http://xml.com/pub/2000/08/09/rdfdb/index.html</link>
    <description>
     Tool and API support for the Resource Description Framework
     is slowly coming of age. Edd Dumbill takes a look at RDFDB,
     one of the most exciting new RDF toolkits.
    </description>
  </item>

  <textinput rdf:about="http://search.xml.com">
    <title>Search XML.com</title>
    <description>Search XML.com's XML collection</description>
    <name>s</name>
    <link>http://search.xml.com</link>
  </textinput>

</rdf:RDF>
```

El de abajo es un documento RSS 1.0 que utiliza elementos de diferentes módulos con los prefijos de espacio de nombres: dc, sy, co y ti. Estos módulos fueron creados para este ejemplo.

```
<?xml version="1.0" encoding="utf-8"?>

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
  xmlns:co="http://purl.org/rss/1.0/modules/company/"
  xmlns:ti="http://purl.org/rss/1.0/modules/textinput/"
  xmlns="http://purl.org/rss/1.0/"
>

  <channel rdf:about="http://meerkat.oreillynet.com/?_fl=rss1.0">
    <title>Meerkat</title>
    <link>http://meerkat.oreillynet.com</link>
    <description>Meerkat: An Open Wire Service</description>
    <dc:publisher>The O'Reilly Network</dc:publisher>
    <dc:creator>Rael Dornfest (mailto:rael@oreilly.com)</dc:creator>
    <dc:rights>Copyright &#169; 2000 O'Reilly &amp; Associates,
Inc.</dc:rights>
    <dc:date>2000-01-01T12:00+00:00</dc:date>
    <sy:updatePeriod>hourly</sy:updatePeriod>
    <sy:updateFrequency>2</sy:updateFrequency>
    <sy:updateBase>2000-01-01T12:00+00:00</sy:updateBase>

    <image rdf:resource="http://meerkat.oreillynet.com/icons/meerkat-
powered.jpg" />

    <items>
      <rdf:Seq>
        <rdf:li resource="http://c.moreover.com/click/here.pl?r123" />
      </rdf:Seq>
    </items>

    <textinput rdf:resource="http://meerkat.oreillynet.com" />

  </channel>

  <image rdf:about="http://meerkat.oreillynet.com/icons/meerkat-
powered.jpg">
    <title>Meerkat Powered!</title>
    <url>http://meerkat.oreillynet.com/icons/meerkat-powered.jpg</url>
    <link>http://meerkat.oreillynet.com</link>
  </image>

  <item rdf:about="http://c.moreover.com/click/here.pl?r123">
    <title>XML: A Disruptive Technology</title>
    <link>http://c.moreover.com/click/here.pl?r123</link>
    <dc:description>
      XML is placing increasingly heavy loads on the existing
technical
      infrastructure of the Internet.
    </dc:description>
    <dc:publisher>The O'Reilly Network</dc:publisher>
    <dc:creator>Simon St.Laurent
(mailto:simonstl@simonstl.com)</dc:creator>
    <dc:rights>Copyright &#169; 2000 O'Reilly &amp; Associates,
Inc.</dc:rights>
    <dc:subject>XML</dc:subject>
    <co:name>XML.com</co:name>
    <co:market>NASDAQ</co:market>
    <co:symbol>XML</co:symbol>
  </item>

  <textinput rdf:about="http://meerkat.oreillynet.com">
    <title>Search Meerkat</title>
```

```
    <description>Search Meerkat's RSS Database...</description>
    <name>s</name>
    <link>http://meerkat.oreillynet.com/</link>
    <ti:function>search</ti:function>
    <ti:inputType>regex</ti:inputType>
  </textinput>

</rdf:RDF>
```

RSS y los estándares

La siguiente es una breve descripción de los estándares en los que se apoya el formato RSS.

XML (Extensible Markup Language)

Los archivos RSS son archivos XML. XML (Extensible Markup Language) es un conjunto de reglas para definir tags sintácticos los cuales fraccionan un documento en partes y de esta manera las identifican. Es un lenguaje de meta-markup (meta conjunto de tags) que define una sintaxis utilizada para definir otros lenguajes de markup estructurados y específicos de un dominio, como por ejemplo RSS. XML está diseñado para que sea fácilmente procesado por computadoras, para almacenar e intercambiar datos. XML proporciona estructura de datos y meta información. Esto realza el valor de los datos múltiples veces, permitiendo que el contenido sea empleado por una gran variedad de aplicaciones. Sin embargo, todas las partes que intercambian datos XML deben acordar en un vocabulario (sintaxis y semántica) común de los datos. La especificación de XML 1.0 provee un mecanismo para lo anterior con un DTD. Un DTD describe los tags y la jerarquía que los datos XML pueden incluir. A diferencia de HTML, que posee un conjunto fijo de tags (fija también la semántica asociada a cada tag), XML deja este conjunto libre para la creación por parte de los diseñadores.

RDF (Resource Description Framework)

Resource Description Framework (RDF) es un framework para la descripción e intercambio de metadatos. Este framework es extensible y permite el agregado de nuevos tipos de entidades. Brinda significado a los recursos para posibilitar el procesamiento automatizado de recursos de Web. RDF es un lenguaje declarativo y proporciona una manera estándar de uso de XML para representar metadatos en la forma de declaraciones sobre propiedades y relaciones de ítems sobre la Web. Estos ítems, conocidos como recursos, pueden ser casi cualquier cosa, mientras sean distinguibles vía una dirección de Web. Los recursos están identificados por una URI (Uniform Resource Identifier). La forma más común de URI es una URL, como por ejemplo http://www.javasoft.com/. Es posible asociar metadatos a una página Web, gráfico, archivo de audio, un GIF, etc.

A través de RDF, comunidades independientes pueden desarrollar vocabularios que satisfacen sus necesidades específicas. Además una comunidad puede compartir vocabularios con otras comunidades. Con el propósito de compartir vocabularios, el significado de los términos debe ser explicado en detalle. Las descripciones de estos conjuntos de vocabularios son denominados esquemas. Un esquema define el significado, características y relaciones de un conjunto de propiedades. El lenguaje RDF permite que cada documento conteniendo metadatos clarifique qué vocabulario está siendo utilizado mediante la asignación de una dirección de Web a cada vocabulario. El lenguaje de especificación de esquema es un lenguaje de representación declarativo influenciado por ideas de representación de conocimiento (por ejemplo: redes semánticas, frames, lógica de predicados), así como también lenguajes de especificación de esquemas de base de datos y modelos de datos de grafos. RDF emplea la idea de espacios de nombres XML para permitir efectivamente que declaraciones RDF referencien a un vocabulario particular o esquema.

Espacios de nombres

Un espacio de nombres XML es una colección de nombres, identificados por una referencia URI, que son utilizados en los documentos XML como tipos de elementos y nombres de atributos. Los espacios de nombres desambiguan elementos con el mismo nombre asignando URIs a elementos y atributos. Agrupan todos los elementos y atributos relacionados a una aplicación XML así el software puede reconocerlos fácilmente. Los espacios de nombres evitan colisiones de nombres que pueden confundir a las aplicaciones XML. La modularización basada en espacios de nombres posibilita la extensibilidad por compartimientos, y de esta manera RSS 1.0 es extendido.

La Web Semántica

La gramática RDF proporciona significado a los recursos, y eventualmente brindará lo que Tim Berners-Lee (inventor de la WWW, URIs, HTTP y HTML) denomina la Web Semántica. Esto es una Web utópica con metadatos y significado adosado al contenido y así las máquinas pueden procesarlo, y no solamente para fines de presentación, sino por diversas aplicaciones. Si esto tiene éxito, los agentes autónomos de software florecerán, los motores de búsqueda serán más relevantes y la Web como un todo se volverá mucho más aprovechable.

Módulos RSS 1.0

Uno de los principales objetivos de la especificación RSS 1.0 fue la extensibilidad. RSS 1.0 tiene una característica llamada módulos RSS que permite a cualquier persona extender RSS 1.0 para agregar nuevos campos para sus propios propósitos. Los módulos son una nueva URI o espacio de nombres que pueden ser añadidos a un documento RSS. Permiten a las personas adicionar nueva funcionalidad a RSS sin cambiar la especificación. Todo lo que se necesita es una URI. Luego cualquier persona puede utilizar esta URI para referir a su espacio de nombres. Los módulos RSS 1.0 emplean una característica de XML denominada espacios de nombres para crear espacios de nombres separados y no conflictivos para módulos, y de esta manera hacer posible que cualquier módulo RSS 1.0 sea incluido dentro de un documento RSS 1.0. El parser será capaz de reconocer a qué espacio de nombres pertenece un elemento dado y así poder ignorar elementos no conocidos.

Los módulos RSS 1.0 son mantenidos en documentos separados, disponibles en línea en *http://purl.org/rss/1.0/modules/*. Los módulos son clasificados como *Proposed* (propuestos) hasta que son aceptados como miembros estándar del grupo de trabajo de RSS-DEV.

El conjunto de módulos estándar RSS 1.0 hasta el momento es: Dublin Core, Syndication y Content.

Ver ***Anexo II - Módulos RSS 1.0*** para más información sobre módulos estándar y propuestos.

Guía para la creación de módulos

A continuación se presentan una serie de puntos a tener en cuenta en la creación de un módulo RSS:

- **Espacios de nombres XML**: cada módulo RSS 1.0 debe residir en su propio espacio de nombres XML y compartimentalizar algún agrupamiento de funcionalidad con un propósito especifico. Un módulo debe ser lo más estrechamente definido.
- **Modelos de contenido simple vs. rico**: la elección de modelo de contenido simple o rico que demanda la tarea es librada a los autores de los módulos. Adoptando un modelo simple, no imposibilita un incremento en la semántica posterior, mediante el módulo existente o mediante un módulo secundario.
- **RDF**: aunque los módulos no tienen requerido tomar el máximo provecho del framework RDF de RSS 1.0, los módulos construidos deben al menos ser compatibles con RDF y hacer lo mejor para proveer modelos de datos coherentes para RDF y XML plano. Teniendo en cuenta ésto, cualquier elemento conteniendo markup XML (por ejemplo: otros elementos) que no están escritos como RDF deben avisar a los parsers RDF que dicho markup no debe ser interpretado, más bien mantenidos como un valor, utilizando el atributo parseType="Literal".
- **Compatibilidad**: los módulos RSS no deben introducir conflictos por modificaciones ad hoc de los modelos de contenido de cualquier otro módulo o del conjunto principal de elementos. Las extensiones modulares no deben ser consideradas como sustitutos de elementos requeridos del núcleo (por ejemplo: dc:description al nivel de channel no debe obviar la necesidad de incluir el elemento requerido rss:description).

Generación de canales RSS

Métodos

Existen varias maneras de generar un canal a partir de un contenido:

- **Self-scraping**: es la manera más fácil de publicar un canal a partir de un contenido existente. Herramientas de Scraping traen la página Web y extraen las partes relevantes para el canal, de esta forma no se necesita cambiar el sistema de publicación previamente establecido. Se utilizan expresiones regulares o expresiones XPath, también podría requerirse de markup en la página Web para realizar indicaciones (generalmente usando tags o) que ayudan a decidir que debe colocarse en el canal.
- **Integración de canal**: si el sitio es generado dinámicamente (empleando por ejemplo Perl, Python, PHP o JSP/Servlets), puede tenerse disponible una librería RSS, y de esta manera integrar el canal dentro del proceso de publicación.
- **Comenzar a partir del canal**: alternativamente, se pueden manejar las partes orientadas a listas del contenido dentro del canal RSS mismo, y generar las páginas Web (así como otro contenido, como listas de e-mail) a partir del canal. Esto tiene la ventaja de tener siempre la correcta información en el canal, y las herramientas como XSLT hacen que esta opción sea simple.
- **Scraping por terceros**: si no se puede obtener el canal por ninguna de las anteriores opciones, terceras personas pueden rascar (*scrape*) el sitio propio y hacer que el canal esté accesible. Existen ciertos riesgos en la generación del canal debido a que el tercero puede no estar al tanto de todos los detalles del contenido, además se introducen las posibles fallas de comunicación (red, servidores, etc.).

Consejos

RSS es muy simple de trabajar, pero como en cualquier nuevo formato, pueden encontrarse problemas con su uso. Los siguientes son consejos para crear buenos feeds RSS:

- **Links descriptivos**: dar cada ítem en el canal una URL distinta en el tag *link*. De esta forma el software puede diferenciar entre ítems y reconocer ítems que ya se han visitado. Si dos ítems propiamente apuntan a la misma página, se pueden utilizar diferentes identificadores de fragmentos.
- **Metadatos significativos**: tratar que los metadatos sean propiamente útiles. Por ejemplo: evitar títulos demasiado cortos o incluir un artículo completo en el tag *description*. Generalmente, se debe incluir lo suficiente en el canal como para que la persona pueda decidir si seguir el link.

- **Codificación HTML**: evitar incluir markup HTML en el canal, debido a que no se sabe como será presentado. Si se necesita incluir un nombre de tag en el texto del canal, se deben utilizar las entidades apropiadas de HTML para escapar los ampersand y los corchetes angulares.
- **Codificación de caracteres**: para evitar problemas en la interpretación, la manera más segura es codificar el canal en UTF-8 y luego chequearlo realizando un parsing con un XML parser.
- **Comunicación con las máquinas**: utilizar los códigos de status HTTP apropiados si el canal fue realocado (usualmente `301 Moved Permanently`) o si no está mas accesible (`410 Gone` or `404 Not Found`).

Evolución del formato RSS

A continuación se presenta una enumeración de las diferentes versiones de RSS.

RSS 0.9

Netscape Communications liberó RSS 0.9 (RDF Site Summary) en marzo de 1999. Concebido para poblar el portal My Netscape con canales de noticias, 0.9 es un simple formato XML con 10 elementos utilizados para describir información sobre sitios Web, típicamente reportes de noticias o artículos. Estos elementos incluyen la noción de un "canal" que contiene hasta 15 ítems, cada uno de los cuales consiste en un título y un link. Los Webmasters pueden utilizar los archivos RSS 0.9 para sindicar su contenido, empleando links (sin descripciones) a sus artículos. El formato todavía es bastante popular.

RSS 0.91

En julio de 1999 Netscape introdujo RSS 0.91 (re-bautizado "Rich Site Summary"), incorporando la mayoría de las características del formato scriptingNews 2.0b1 de UserLand. La versión 0.91 agregó 14 elementos para la mejor descripción de canales, ítems y frecuencia de actualización, incluyendo un nuevo elemento description para los ítems. La belleza de estas primeras versiones de RSS es su simplicidad. RSS 0.9 y 0.91 fueron diseñados para la simplicidad. Esta es la razón de por qué son uno de los formatos XML más populares. Los humanos pueden leer y comprender fácilmente los archivos RSS y crearlos a mano, o con ayuda de programas como Weblog (de Jonathan Eisenzopf) o Manila (de Dave Winer de UserLand).

RSS 0.92

Userland Software liberó RSS 0.92 en diciembre del 2000. Basado en la versión 0.91 de Netscape, 0.92 agrega nuevas características opcionales que los desarrolladores solicitaban como síntesis de Weblog o blog (sub-elementos opcionales de ítem, las descripciones de ítem pueden contener HTML markup con las entidades codificadas), atribución de fuente (fuente de ítem), información de taxonomía (categoría de ítem) y el sub-elemento de canal, el cual notifica a los aggregators cuando el canal es actualizado para actualizaciones más puntuales y así economizar ancho de banda. RSS 0.92 es compatible ascendentemente con RSS 0.91, las nuevas características de 0.92 son opcionales. Un archivo 0.91 es también un archivo 0.92 válido.

RSS 1.0

RDF Site Summary (RSS) 1.0 fue liberado en diciembre de 2000 por el grupo de trabajo RSS-Dev. RSS 1.0 es una versión modularizada del formato RSS 0.9 original diseñado para equilibrar extensibilidad con simplicidad. Los módulos pueden extender la especificación RSS 1.0 para el uso en diferentes aplicaciones sin requerir nuevas versiones de RSS.

RSS 0.91+ vs. RSS 1.0

Desafortunadamente, RSS se dividió en dos caminos:

- RSS 0.91+ (0.91/0.92/0.93+) y
- RSS 1.0

Un problema de los diferentes caminos es la incompatibilidad. El núcleo de RSS 1.0 no es compatible hacia atrás con el núcleo 0.91 basado en DTD, pero lo es a través del módulo RSS091. Sin embargo, es compatible con la versión 0.9 porque está basado sobre RDF y espacios de nombres tal cual 0.9 fue basado. Por consiguiente, los desarrolladores pueden actualizar sus archivos RSS 0.91 a 1.0 dado que la mayoría de las aplicaciones comprenden 0.91 y 0.9. RSS 0.92+ es compatible hacia adelante con 0.91; todos los archivos 0.91 son archivos 0.92 válidos. 0.92 no es compatible hacia atrás con 0.9.

Resumen de las características de las versiones

Las siguientes son algunas de las características claves de las diferentes versiones de RSS:

- **0.9**: encabezados basados en RDF (espacios de nombres y RDF); ítems con link y título con un máximo de 15 instancias; links de 100 caracteres de límite; títulos de 500 caracteres de límite.
- **0.91**: basado en DTD; añade descripciones opcionales (500 caracteres de límite) y managingEditor, publisher, pubDate, webMaster, copyright.
- **0.92**: basado sobre 0.91; todos los sub-elementos de ítem son opcionales (source, enclosure, category); tag de lenguaje es opcional; la descripción de ítem permite HTML con entidades codificadas; sub-elemento channel; cloud permite notificación de actualización; todos los límites han sido eliminados.
- **1.0**: basado sobre 0.9; utilizando RDF y espacios de nombres; añade módulos (módulo Dublin Core, módulo que maneja la frecuencia de actualización, módulo que maneja la inclusión de contenido de ítem, etc.).

El futuro de RSS

El movimiento de RSS continúa, a pesar de su torpe comienzo en Netscape y luego su especialización en el manejo de noticias. Nuevas versiones de RSS están en desarrollo, como la versión 0.93 de UserLand. El grupo de trabajo de RSS-Dev está trabajando en un cierto número de módulos incluyendo un módulo de discusión, un módulo de contenido, un módulo de taxonomía, etc.

RSS 1.0 va más allá de la sindicación de noticias. Puede utilizarse para juntar datos de sistemas tales como manejadores de contenidos, administradores de conocimientos, portales de información corporativa, planeamiento de recursos empresariales, manejo de relaciones con clientes, etc.

Mediante la utilización de metadatos ricos para organizar noticias en nuevas y útiles maneras, los aggregators como Meerkat de O'Reilly y My UserLand añaden valor a los feeds RSS separados.

Aplicaciones RSS

La siguiente es una lista conteniendo aplicaciones basadas en RSS:

- **AmphetaDesk**: aplicación cliente de múltiple plataforma y *open source* que puede leer todas las versiones corrientes de RSS (todas las 0.9x y 1.0).
- **Meerkat**: un servicio de conexión abierta sobre RSS de Rael Dornfest de O'Reilly. Este portal de noticias de RSS incorpora principalmente noticias técnicas y combina *feeds* dentro de MOBs y categorías. Las noticias pueden ser filtradas por fecha, por categoría, por expresiones regulares o palabras clave.
- **PullRSS**: convertidor de RSS a HTML, *open source*, con redirecciones opcionales para el rastreo a través del click.
- **Radio Userland**: desarrollo de aplicación Web y entorno de ejecución del lado del cliente. Es esencialmente un servidor sobre el desktop.
- **Slashcode/Slashdot**: para Slashdot el nuevo motor de búsqueda toma solicitudes y responde vía RSS. De esta manera, buscando por Comments/Users/Stories se obtendrán datos mediante RSS.
- **Weblog 2.0**: utiliza RSS 1.0 y los módulos Taxonomy y Dublin Core para manejar canales en varios formatos más un directorio semejante a Yahoo. Por Jonathan Eisenzopf.

Aggregators RSS

Estos son algunos de los aggregators RSS más extensos:

- Syndic8 [*http://www.syndic8.com*]
- Userland [*http://aggregator.userland.com*]
- OnContent [*http://www.oncontent.com*]
- NewsIsFree [*http://www.newsisfree.com*]
- DayPop [*http://www.daypop.com*]

Algunos de los anteriores sitios proveen listas categorizadas de sus canales en formato OCS (Open Content Syndication). OCS es un formato XML para describir catálogos de canales, en vez de solamente un único canal. El formato usa RDF para expresar relaciones entre los ítems, y Dublin Core para expresar las propiedades que describen los atributos de los canales.

Contenidos RSS en vivo

La siguiente lista muestra algunos de los más conocidos canales de noticias RSS 1.0 disponibles en línea:

- ITN [*http://www.itn.co.uk/itn.rdf*]
- Meerkat (O'Reilly) [*http://www.oreillynet.com/meerkat/?_fl=rss10*]
- Monkeyfist [*http://monkeyfist.com/rss1.php3*]
- W3C [*http://www.w3.org/2000/08/w3c-synd/home.rss*]

Otras tecnologías de publicación XML

Otras tecnologías de publicación en línea (o sindicación) basadas en XML son: NITF, XMLNews, NewsML, PRISM, ICE. Todas estas aplicaciones XML han aparecido en respuesta a los problemas planteados por la distribución a múltiples destinos y por los nuevos modelos de negocio sobre contenidos que han surgido.

NITF

News Industry Text Format (NITF) es una aplicación XML para marcar ítems de noticias. Fue originalmente una aplicación SGML, antes del advenimiento de XML. Fue desarrollado en conjunto por el International Press Telecommunications Council (IPTC) y el Newspaper Association of America, las dos mayores organizaciones de estándar de la industria de las noticias. NITF está inspirado en parte por HTML. Fue diseñado para tener un DTD flexible, con el cual los usuarios pueden incluir marcas embebidas en el relato a gusto.

XMLNews

XmlNews es un conjunto de especificaciones para el intercambio de noticias e información utilizando estándares de la Web. Su objetivo es hacer que la producción, recepción y archivo de la información sean más fáciles. Y todo esto en forma independiente del hardware, software y lenguajes de programación utilizados. Este formato no sólo describe los elementos de la noticia y su locación, además acarrea el contenido. De esta manera las noticias pueden ser compartidas en forma más fácil a lo largo de la cadena de distribución, desde los reporteros en el campo y agencias de prensa internacionales a los usuarios finales como portales de noticias o intranets corporativas. XMLNews consiste en dos partes, XMLNews-Story para el contenido y XMLNews-Meta para los metadatos que describen el contenido. XMLNews-Story define el contenido textual de la noticia. Es un subconjunto de NITF.

XMLNews-Meta es un vocabulario extensible para la descripción de recursos referidos a noticias. Sólo se describe el contenido. Permite metadatos para cualquier clase información sobre noticias, incluyendo texto, fotos, audio y video. XMLNews-Meta es expresado en RDF, el cual es inherentemente extensible. Las organizaciones pueden extenderlo por su cuenta para lograr sus objetivos utilizando un vocabulario dentro de un espacio de nombres (namespace) customizado.

Una característica interesante de XMLNews es que un objeto puede ser marcado (embebido con tags) tanto como los usuarios lo requieran. La utilización de XML namespaces permite que cualquier usuario a lo largo de la cadena pueda agregar su propio marcado personalizado para definir el relato de la noticia de diferentes maneras sin crear conflictos con alguna adición previa o posterior al marcado.

NewsML

Con una especificación más nueva que NITF, desarrollado con el apoyo de IPTC. Es un formato de empaquetamiento para el contenido de noticias diseñado para solucionar el problema del transporte de ítems de noticias independientemente de su codificación. Todavía esta en desarrollo. Sus características principales son:

- Provee la estructura para representar los recursos de noticias sin importar el medio o formato. Puede representar fotos, audio, texto, video, gráficos.
- Provee representaciones alternativas para los ítems, por ejemplo, PDF, HTML, RTF, etc.
- Existe la posibilidad de adjuntar metadatos.

PRISM

La especificación de Publishing Requirements for Industry Standard Metadata (PRISM) define un estándar para la descripción de contenidos interoperables, intercambio y reuso en los contextos de la publicación tradicional y electrónica. PRISM recomienda el uso de ciertos estándares existentes, tales como XML, RDF, Dublin Core y varias especificaciones ISO para locaciones, lenguajes y formatos de fecha/ hora. Más allá de estas recomendaciones, define un pequeño número de espacios de nombres XML y vocabularios controlados de valores para lograr los objetivos anteriormente mencionados.

ICE

La especificación de ICE (Information and Content Exchange) es una de las aplicaciones XML más establecidas en este área. El estándar ICE hace más fácil para los sindicadores la tarea de distribución de la información en forma controlada a los suscriptores. No especifica el formato del contenido publicado, solamente lo empaqueta. Los caracteres de datos dentro del paquete definido por ICE pueden estar estructurados utilizando alguna de las representaciones de datos XML.

ICE define un mecanismo de publicación/ suscripción vía mensajes request y response (por ejemplo: mensajes XML sobre HTTP). Antes que la sindicación pueda ser usada, el sindicador debe configurar el servidor de sindicación, especificando qué tiene disponible y para qué suscriptores y a qué hora. El servidor mantiene un catálogo de todos los ofrecimientos disponibles incluyendo las políticas de distribución. Para satisfacer las suscripciones, el syndicator envía directivas en un paquete para actualizar a un suscriptor de un estado viejo a uno nuevo. ICE define cómo los documentos son distribuidos en una forma controlada, dejando libre el formato de esos documentos.

La mayoría de los miembros del grupo que define la especificación de ICE no vienen del tradicional mundo de las noticias. Esto se refleja en el hecho de que la sindicación bajo ICE es mas sofisticada que los modelos previamente utilizados por las grandes organizaciones.

La característica más notable de ICE es la habilidad para codificar un algunos aspectos del negocio de sindicación. Esto incluye los planes de distribución, activación de suscripciones, manejo de transacciones. Esto es práctico para los sindicadores que mantienen una gran base de clientes. También lo es para los aggregators de contenidos, ICE hace más fácil el manejo de las transacciones con múltiples proveedores de conocida confiabilidad.

ICE tiene múltiples implementaciones y es aplicado en donde un modelo confiable de *publish/ subscribe* sea requerido, por ejemplo un catálogo de partes.

Clasificación de las especificaciones

Las especificaciones basadas en XML para la industria de las noticias caen en cuatro grandes categorías:

- **Contenido textual**: objetos de noticias en texto para lectores humanos.
- **Metadatos**: información para la administración de noticias y clasificación. La mayoría de los formatos de metadatos están basados sobre RDF.
- **Empaquetado**: recopilan múltiples objetos dentro de un único paquete compuesto de noticias.
- **Sindicación**: manejan la distribución de noticias; la sindicación puede utilizar un modelo *push* (el proveedor inicia la transferencia, como un cable de noticias convencional) o un modelo *pull* (el receptor inicia la transferencia, como un Web browser)

La siguiente tabla muestra el alcance de cada especificación:

Contenido textual	NITF, XMLNews.
Metadatos	NewsML, NITF, PRISM, RSS, XMLNews.
Empaquetado	NewsML, PRISM.
Sindicación	ICE, RSS.

ICE (Information and Content Exchange)

ICE es un protocolo de sindicación de contenidos completo. Soporta ambos modelos de distribución *push* y *pull*.

NewsML

Es un formato de empaquetado y metadatos para objetos de noticias. NewsML provee un sofisticado soporte para objetos de noticias compuestos, conformados por diferentes medios y múltiples formatos. Aunque NewsML no fue pensado directamente como un formato de sindicación, soporta objetos de noticias en línea y fuera de línea, por lo tanto es compatible con ambos modelos de distribución push y pull.

NITF (News Industry Text Format)

Incluye soporte limitado de metadatos, pero permite una precisa colocación en línea de cierta información de metadatos.

PRISM (Publishing Requirements for Industry Standard Metadata)

Como RSS y XMLNews-Meta, PRISM proporciona un soporte completo para metadatos a través de RDF. El soporte para empaquetado es limitado, y está basado sobre Multipart-MIME en vez de un formato XML.

RSS

RSS 1.0 provee un soporte completo para metadatos a través de RDF, al igual que PRISM y XMLNews-Meta. RSS 0.91 tiene un soporte limitado para metadatos (solucionado en 1.0), y ambos posibilitan la sindicación por el modelo pull solamente.

XMLNews

Constituido por un par de especificaciones, *XMLNews-Meta* (metadatos) y *XMLNews-Story* (contenido textual). Como PRISM y RSS, XMLNews-Meta está basado en RDF.

Extensible Markup Language (XML)

Un documento RSS está basado en dos estándares: Extensible Markup Language (XML) y Resource Description Framework (RDF). A continuación se desarrolla una explicación sobre el primero de ellos.

Introducción a XML

Definición

XML (Extensible Markup Language) es un conjunto de reglas para definir tags sintácticos los cuales fraccionan un documento en partes y de esta manera las identifican. Es un lenguaje de meta-markup (meta conjunto de tags) que define una sintaxis utilizada para definir otros lenguajes de markup estructurados y específicos de un dominio, como por ejemplo RSS. Un lenguaje de markup (conjunto de tags) es un mecanismo para identificar estructuras dentro de un documento estructurado. La especificación de XML define una forma estándar para agregar markup a los documentos. XML está diseñado para que sea fácilmente procesado por computadoras, para almacenar e intercambiar datos.

XML proporciona estructura de datos y meta información. Esto realza el valor de los datos múltiples veces, permitiendo que el contenido sea empleado por una gran variedad de aplicaciones. Sin embargo, todas las partes que intercambian datos XML deben acordar en un vocabulario (sintaxis y semántica) común de los datos. La especificación de XML 1.0 provee un mecanismo para lo anterior con un DTD. Un DTD describe los tags y la jerarquía que los datos XML pueden incluir. A diferencia de HTML, que posee un conjunto fijo de tags (fija también la semántica asociada a cada tag), XML deja este conjunto libre para la creación por parte de los diseñadores.

Diferencias entre XML y HTML

Una de las principales diferencias entre estos dos lenguajes de marcas es que en HTML la semántica de los tags y el conjunto de tags están fijos. Por ejemplo, siempre un tag como <h1> significa un encabezado de primer nivel. La W3C, en conjunto con los fabricantes de los Web browsers y la comunidad WWW, trabaja constantemente para extender la definición de HTML, y así permitir que nuevos tags mantengan el ritmo con la tecnología cambiante y proporcionar variaciones en la presentación de la Web. Sin embargo, estos cambios están siempre sujetos a lo que los fabricantes de los Web browsers tienen implementado y por el hecho de que la compatibilidad hacia atrás es claramente un concepto superior. Y para las personas que desean diseminar información ampliamente, las características soportadas sólo por las últimas versiones de los Web browsers no son para nada útiles.

XML no especifica ni la semántica ni el conjunto de tags. De hecho, XML es realmente un meta-lenguaje para la descripción de lenguajes de marcas. En otras palabras, XML proporciona la facilidad para definir tags y las relaciones estructurales entre ellos. Partiendo que no existe un conjunto de tags predefinido, no existe entonces ninguna semántica preconcebida. Todas las semánticas de un documento XML estarán definidas por las aplicaciones que los procesan o por plantillas de estilo (*stylesheets*).

Breve historia

XML fue desarrollado por el *XML Working Group* (originalmente conocido como el *SGML Editorial Review Board*) formado bajo el auspicio de World Wide Web Consortium (W3C) en 1996. Se convirtió en una W3C Recommendation (estado final de un documento) recién en Febrero de 1998. Pero esta tecnología no es tan nueva. Antes de XML existe SGML (Standard Generalized Markup Language), desarrollado en el principio de los ‘80s, es un estándar ISO (ISO 8879) desde 1986 y utilizado ampliamente en grandes proyectos de documentación. El desarrollo de HTML comenzó en 1990. Los diseñadores de XML simplemente tomaron las mejores partes de SGML, guiados por la experiencia con HTML, y produjeron algo que no es menos potente que SGML y enormemente más regular y simple para usar. SGML es mayormente utilizado para documentación técnica.

Relación entre XML y SGML

XML está definido como un perfil de aplicación de SGML. SGML significa Standard Generalized Markup Language definido por la ISO 8879. SGML ha sido la manera estándar e independiente del proveedor de mantener repositorios de documentación estructurada por más de una década, pero no es apropiado para servir documentos sobre la Web.

Como XML es un perfil de aplicación de SGML, entonces cualquier sistema que conforme enteramente la especificación de SGML será capaz de leer documentos XML. Sin embargo, usando y comprendiendo documentos XML no significa que el sistema sea capaz de entender la generalidad completa de SGML. Es decir, que XML es una forma restringida de SGML. Para los técnicos puristas, es importante notar que pueden existir diferencias sutiles en la manera de entender documentos por sistemas XML y la manera de entender estos mismos documentos por sistemas SGML. En particular, el tratamiento del espacio en blanco adyacente a los tags puede diferir.

Razones para usar XML

Para comprender XML es importante entender por qué fue creado. XML fue creado para que los documentos estructurados enriquecidos puedan ser empleados en la Web. Las únicas alternativas viables, HTML y SGML, no son prácticas para este propósito. HTML viene ligado con un conjunto de semántica y no provee estructuras arbitrarias. SGML provee estructuras arbitrarias, pero es demasiado dificultoso para ser implementado por un Web browser. Los sistemas SGML resuelven grandes y complejos problemas que justifican su costo. Visualizar documentos estructurados enviados por la Web raramente vale dicha justificación. Esto no quiere decir que de XML se espera que reemplace completamente a SGML. Mientras XML ha sido diseñado para despachar contenido estructurado a través de la Web, algunas de las características que carece para hacerlo más práctico, hacen a SGML una solución más satisfactoria para la creación y almacenamiento a largo plazo de documentos complejos. En muchas organizaciones el filtrado de SGML a XML será el procedimiento estándar para la publicación en la Web.

XML es verboso por diseño, esto es, debido a que XML es un formato de texto y utiliza tags para delimitar los datos, los archivos XML son siempre mas extensos en comparación a los formatos binarios. Esto fue una decisión determinada por los diseñadores de XML. Las ventajas que ofrece un formato de texto son evidentes, y las desventajas son compensadas en diferentes niveles. El espacio en disco no es tan caro como solía ser. Además los programas compresores de archivos, como el Zip, funcionan eficientemente. También los protocolos de comunicación modernos como el HTTP/1.1 pueden comprimir datos en el acto, salvando el ancho de banda tan efectivamente como con los formatos binarios.

XML posee las siguientes ventajas:

- **Texto Plano**: como XML no está en formato binario, se pueden crear y editar archivos con cualquier editor de texto o con un entorno de desarrollo visual. Puede ser utilizado tanto para definir archivos de configuración así como para un repositorio de una compañía. El texto plano permite a la gente, si es necesario, mirar los datos sin el programa que lo produce y permiten a los desarrolladores depurar aplicaciones más fácilmente.

- **Identificación de datos**: XML expresa que clase de datos contiene, no cómo visualizarlos. Los tags identifican la información y dividen los datos en partes. Entonces diferentes programas pueden utilizar la información contenida en los tags de forma conveniente para sus propósitos. Es decir, los datos englobados en los tags pueden ser utilizados de diferentes maneras por diferentes programas.
- **Procesamiento simple**: una notación consistente y regular como la de los documentos XML hace sencillo la construcción de un programa que los procese, llamado XML parser. Además como XML es un estándar independiente del proveedor, se puede elegir la implementación del parser que se desee.
- **Jerárquico**: los documentos XML tienen una estructura jerárquica. Los documentos con estructura jerárquica son, en general, más rápidos para el acceso porque se pueden sondear hasta la parte que se necesita. Son además más fáciles para reorganizar porque cada pieza está delimitada.
- **Capacidad de transformación**: si se necesita visualizar un documento XML, puede usarse el estándar de estilo XSL. Este estándar especifica como mostrar los datos del documento XML mediante plantillas de estilo aplicadas a dicho documento XML. Desde que XML es totalmente libre de estilos, se le pueden aplicar diferentes plantillas para transformarlo en diferentes formatos de representación, como TEX, PDF, HTML, etc.
- **Reusabilidad en línea**: un documento XML puede ser compuesto por entidades separadas. Las entidades XML pueden ser incluidas en línea en el documento. Las secciones incluidas se ven como un documento normal, de una sola pieza. Es decir, la inclusión de una entidad en el documento es transparente al usuario/ aplicación final. Esto permite modularizar el documento con todas las ventajas que ofrece este mecanismo, como editar una sección y que el cambio sea reflejado en todos los lugares donde la sección sea utilizada.

Objetivos de desarrollo de XML

La especificación de XML establece los siguientes objetivos para XML:

1. **Debe ser directo el uso de XML sobre Internet.** Los usuarios deben poder visualizar documentos XML tan rápido y simple como los documentos HTML. En la práctica, esto es posible gracias a que los Web browsers más populares tienen conocimiento sobre la tecnología XML.
2. **XML debe soportar una gran variedad de aplicaciones.** XML debe ser provechoso para una gran variedad de aplicaciones: creación, presentación, análisis de contenido, etc. Aunque el foco inicial estaba puesto en servir documentos estructurados sobre la Web, no significó que XML fuera definido de una manera estrecha.

3. **XML debe ser compatible con SGML.** La mayoría de las personas involucradas en el emprendimiento XML vienen de organizaciones que poseen un gran suma de material en SGML. XML fue diseñado pragmáticamente para ser compatible con los estándares existentes mientras resuelve el relativamente nuevo problema del envío de documentos estructurados sobre la Web.
4. **Debe resultar simple escribir programas que procesen documentos XML.**
5. **El número de características opcionales en XML debe ser mantenido en un mínimo, idealmente cero.** Las características opcionales inevitablemente aumentan los problemas de compatibilidad cuando los usuarios quieren compartir documentos, terminando, a veces, en la confusión y frustración.
6. **Los documentos XML deben ser legibles por humanos y razonablemente claros.** Si el usuario no posee un browser XML y recibe una porción de XML desde alguna fuente, debe ser capaz de observarlo en un editor de texto y realmente comprender el significado del contenido.
7. **El diseño de XML debe ser preparado rápidamente.** Los esfuerzos estándar son notoriamente lentos. XML fue necesario inmediatamente y fue desarrollado tan velozmente como era posible.
8. **El diseño de XML debe ser formal y conciso.** Esto esencialmente quiere decir que XML debe poder ser expresado en EBNF y debe ser dócil para las técnicas y herramientas de compilación modernas. Existen muchas razones técnicas del por que la gramática de SGML no puede ser expresada en EBNF. La escritura de un correcto parser SGML requiere del manejo de una variedad de características lingüísticas raramente empleadas y muy dificultosas para parsear. XML, claramente, no cae en esa desventaja.
9. **Los documentos XML deben ser fáciles de crear.** Aunque existan editores sofisticados para el tratamiento de contenido XML, debe ser posible crear documentos XML de otras formas: directamente desde un editor de texto, con simples shell scripts o Perl scripts, etc.
10. **La brevedad en el markup de XML es de mínima importancia.** Varias características lingüísticas de SGML fueron diseñadas para minimizar el monto de tipeo requerido para el ingreso manual de documentos SGML. Estas características no son soportadas en XML. Desde un punto de vista abstracto, esos documentos son indistinguibles de sus formas más plenamente especificadas, pero el soporte de esas características agrega una considerable carga al parser SGML (o la persona que lo escribe, de todas formas). Además, la mayoría de los editores modernos ofrecen mejores facilidades para definir shortcuts para el ingreso de texto.

Cómo está definido XML

XML es una familia de tecnologías. XML 1.0 es la especificación que define la sintaxis de los documentos XML. Mas allá de XML 1.0, existe un conjunto de módulos que ofrecen útiles servicios para llevar a cabo diversas tareas.

XML está definido en varias especificaciones relacionadas:

- Extensible Markup Language (XML) 1.0 [http://www.w3.org/TR/WD-xml]

  Define la sintaxis de XML.

- Namespaces in XML [http://www.w3.org/TR/REC-xml-names]

  Provee un método simple para calificar elementos y atributos en documentos XML, permitiendo la asociación de espacios de nombres identificados por referencias URI.

- XML Schema [http://www.w3.org/TR/xmlschema-0/]

  El W3C XML Schema Definition Language es un lenguaje XML para describir y restringir el contenido de documentos XML.

- XML Pointer Language (XPointer) [http://www.w3.org/TR/1998/WD-xptr-19980303] y XML Linking Language (XLink) [http://www.w3.org/TR/1998/WD-xlink-19980303]

  Definen una forma estándar de representación de hipervínculos entre recursos. XPointer describe como direccionar un recurso, XLink describe como asociar dos o más recursos.

- Extensible Style Language (XSL) [http://www.w3.org/TR/WD-xsl]

  Define el lenguaje estándar de plantillas de estilo (stylesheet) para XML.

La sintaxis de XML

Forma de un documento XML

El siguiente es un ejemplo sobre un simple documento XML:

Ejemplo 1

```
<?xml version="1.0"?>

<oldjoke>

<burns>Say <quote>goodnight</quote>,
Gracie.</burns>
```

```
<allen><quote>Goodnight,
Gracie.</quote></allen>

<applause/>

</oldjoke>
```

Características a destacar:

- El documento comienza con una instrucción de procesamiento: `<?xml ...?>`. Esta es la declaración XML. No es requerida, pero su presencia identifica explícitamente el documento como un documento XML e indica la versión de XML que se empleó para su creación.
- No existe la declaración de tipo de documento. A diferencia de SGML, XML no requiere de la existencia de una declaración de tipo de documento. Sin embargo, una declaración de tipo de documento puede ser provista, y algunos documentos necesitarán de una para ser comprendidos sin ambigüedad.
- Los elementos vacíos (`<applause/>` en el ejemplo) tienen una sintaxis especial. Mientras que la mayoría de los elementos en un documento engloban cierto contenido, los elementos vacíos son simplemente marcadores indicando que algo ocurre (por ejemplo, el tag de regla horizontal `<hr>` de HTML). El final /> en esta sintaxis indica al programa que procesa dicho documento que el elemento es vacío y no posee su correspondiente tag de fin. Debido a que los documentos XML no requieren de una declaración de tipo de documento, sin la anterior indicación le sería imposible a un parser de XML determinar que tags fueron intencionalmente vacíos y cuales fueron dejados vacíos por error. XML ha suavizado la distinción entre elementos que fueron declarados como `EMPTY` (vacíos) y elementos que meramente no poseen contenido. En XML es legal el uso de la sintaxis de tag de elemento vacío en cualquiera de los dos casos. También es legal el uso del par de tags de comienzo y fin para elementos vacíos: `<applause></applause>`. Si se tiene en cuenta la interoperabilidad, es mejor reservar la sintaxis de tag de elemento vacío para los elementos que fueron declarados como `EMPTY` y utilizarla solamente para esos elementos.

Los documentos XML están compuestos por markup (conjunto de tags) y contenido. Existen seis clases de markup que pueden ocurrir en un documento XML: elementos, referencias de entidad, comentarios, instrucciones de procesamiento, secciones demarcadas y declaraciones de tipo de documento.

Elementos

Los elementos son la forma más común de markup. Delimitados por <>, la mayoría de los elementos identifican la naturaleza del contenido que engloban. Algunos elementos pueden ser vacíos, en este caso, no poseen contenido. Si un elemento no es vacío entonces comienza con un tag de principio: `<elemento>` y termina con un tag de fin: `</elemento>`.

Atributos

Los atributos son pares nombre-valor que aparecen dentro de los tags de comienzo, luego del nombre del elemento. Por ejemplo: significa un elemento `div` con el atributo `class` con valor `preface`. En XML, todos los valores de atributo deben estar encerrados entre comillas.

Referencias de entidad

Para introducir markup en un documento, algunos caracteres han sido reservados para identificar el comienzo del markup. El carácter “menor que” (<), por ejemplo, identifica el comienzo del tag de inicio o tag de fin de un elemento. Para poder insertar esos caracteres dentro de un documento como contenido, debe existir una manera alternativa para representarlos. En XML, las entidades son utilizadas para representar esos caracteres especiales. Las entidades son empleadas también para referirse a texto que usualmente se repite o varía. También se utilizan para incluir el contenido de archivos externos.

Cada entidad debe poseer un único nombre. Para utilizar una entidad, simplemente se debe referenciar por su nombre. Las referencias de entidad comienzan con el ampersand (&) y finalizan con un punto y coma (;). Por ejemplo, la entidad `lt` inserta un literal < en un documento.

Una forma especial de referencia de entidad, llamada referencia de caracter, puede ser utilizada para insertar caracteres Unicode arbitrarios en un documento. Este es un mecanismo para la inserción de caracteres que no pueden ser tipeados directamente desde un teclado.

Las referencias de carácter toman una de dos formas: referencias decimales, `&#8478;` y referencias hexadecimales, `&#x211E;`. Ambas refieren al caracter de número U+211E de Unicode.

Comentarios

Los comentarios comienzan con `<!--` y finalizan con `-->`. Pueden contener cualquier dato excepto el literal string --. Se pueden ubicar comentarios entre markup en cualquier parte de un documento.

Los comentarios no son parte del contenido textual de un documento XML. Un procesador XML no está obligado a pasar los comentarios hacia la aplicación.

Instrucciones de procesamiento

Las instrucciones de procesamiento (PIs) son una puerta de escape para proveer información a una aplicación. Como los comentarios, no conforman la parte textual de un documento XML, pero el procesador XML está obligado a pasarlas hacia la aplicación.

Las instrucciones de procesamiento tienen la forma: <?nombre pidato?>. El nombre, llamado el objetivo PI, identifica la PI a la aplicación. Las aplicaciones deben procesar sólo los objetivos que reconocen e ignorar el resto de PIs. Cualquier dato que siga el objetivo PI es opcional, es para la aplicación que reconozca el objetivo. Los nombres de PI que comienzan con xml están reservados para la estandarización XML.

Secciones CDATA

En un documento, una sección `CDATA` le indica al parser que ignore los caracteres de markup. Las secciones `CDATA` son útiles, por ejemplo, para poder incluir listados de código fuente dentro de un documento XML sin que el parser interprete caracteres como < y & del listado como si fueran markup.

Las secciones `CDATA` comienzan con `<![CDATA[` y finalizan con `]]>`. Todos los caracteres de la sección son pasados directamente a la aplicación, sin interpretación alguna. Los elementos, referencias de entidad, comentarios e instrucciones de procesamiento son irreconocibles y los caracteres que comprenden son pasados a la aplicación. El único string que no puede aparecer en un sección `CDATA` es `]]>`.

Declaraciones de tipo de documento

Uno de los grandes poderes que posee XML es que permite la creación de nombres de tag específicos. Pero para una aplicación dada, probablemente carezca de sentido el hecho de que ocurran tags en un orden completamente arbitrario.

Si el documento debe tener significado, deben existir algunas restricciones en la secuencia y anidamiento de los tags. Las declaraciones son el lugar donde esas restricciones pueden ser expresadas.

Mas generalmente, las declaraciones permiten a un documento comunicar meta-información al parser sobre su contenido. La meta-información incluye: la secuencia y anidamiento permitidos, los valores de atributo y sus tipos y defaults, los nombres de los archivos externos que pueden ser referenciados y si contienen o no XML, los formatos de datos (no XML) externos que pueden ser referenciados, y las entidades que pueden encontrarse.

Existen cuatro clases de declaraciones en XML: declaraciones de tipo de elemento, declaraciones de lista de atributos, declaraciones de entidad y declaraciones de notación.

Declaraciones de tipo de elemento

Identifican los nombres de los elementos y la naturaleza de su contenido. Una típica declaración de tipo de elemento es:

```
<!ELEMENT oldjoke (burns+, allen, applause?)>
```

Esta declaración identifica el elemento nombrado `oldjoke`. Lo que sigue al nombre del elemento es el *modelo de contenido*. El modelo de contenido define cuales son los elementos que puede contener. En este caso, un `oldjoke` debe contener `burns` y `allen` y puede contener `applause`. Las comas entre los nombres de los elementos indican que deben aparecer en sucesión. El + después de `burns` indica que puede ser repetido más de una vez, pero debe ocurrir al menos una vez. El ? luego de `applause` indica que es opcional (puede estar ausente, o puede ocurrir exactamente una vez). Un nombre sin signos de puntuación, como `allen`, debe ocurrir exactamente una vez.

Las declaraciones de los elementos usados en cualquier modelo de contenido deben estar presentes para un procesador XML, para así poder chequear la validez de un documento. Además de nombre de elementos, el símbolo especial `#PCDATA` está reservado para indicar datos de caracter. PCDATA significa *parseable character data* (datos de caracter parseables).

Los elementos que contienen solamente otros elementos se dice que poseen *contenido de elemento*. Los elementos que contienen tanto otros elementos como `#PCDATA` se dice que poseen *contenido mixto*.

Por ejemplo, la definición de burns puede ser

```
<!ELEMENT burns (#PCDATA | quote)*>
```

La barra vertical indica una relación de disyunción (or), el asterisco indica que el contenido es opcional (puede ocurrir cero o más veces). De esta definición se desprende que burns puede contener cero o más caracteres y tags quote, mezclados en cualquier orden. Todo modelo de contenido mixto debe tener esta forma: `#PCDATA` debe aparecer primero, todos los elementos deben estar separados por barras verticales y el grupo entero debe ser opcional.

Otros dos modelos de contenido son posibles: `EMPTY` indica que el elemento no posee contenido (y consecuentemente no tiene tag de fin), y `ANY` indica que cualquier contenido es permitido. Se debe tratar de evitar a toda costa la utilización del modelo de contenido `ANY` en un ambiente de producción porque deshabilita todo chequeo de contenido del elemento.

Lo que sigue es un conjunto completo de declaraciones de elemento del **Ejemplo 1**:

```
<!ELEMENT oldjoke  (burns+, allen, applause?)>

<!ELEMENT burns    (#PCDATA | quote)*>

<!ELEMENT allen    (#PCDATA | quote)*>

<!ELEMENT quote    (#PCDATA)*>

<!ELEMENT applause EMPTY>
```

Declaraciones de lista de atributos

Las declaraciones de lista de atributos identifican qué elementos pueden tener atributos, qué atributos pueden tener, qué valores pueden mantener los atributos y qué valor es el default. Una típica declaración de lista de atributos puede ser la siguiente:

```
<!ATTLIST oldjoke
          name          ID                      #REQUIRED
          label         CDATA                   #IMPLIED
          status        (funny | notfunny)      'funny'
                >
```

En el ejemplo anterior, el elemento `oldjoke` tiene tres atributos: `name`, que es un ID y es requerido; `label`, que es un string (datos de caracter) y no es requerido; y `status`, que debe ser excluyentemente `funny` o `notfunny` y el valor default es `funny`, si no se especifica ningún valor.

Cada atributo en una declaración tiene tres partes: un nombre, un tipo y un valor default.

El autor de la declaración puede elegir cualquier nombre, siempre sujeto a ciertas restricciones leves. Además, se debe tener en cuenta que no se pueden repetir los nombres en el mismo elemento.

Existen seis tipos de atributo posibles:

- **CDATA**

  Los atributos CDATA son strings, cualquier texto es permitido.

- **ID**

  El valor de un atributo ID debe ser un nombre conforme con la especificación de XML. Todos los valores ID utilizados en un documento deben ser diferentes. Los IDs identifican unívocamente elementos individuales en un documento. Los elementos deben tener un único atributo de tipo ID.

- **IDREF ó IDREFS**

Un valor de atributo de tipo IDREF debe ser el valor de un solo atributo ID de algún elemento en el documento. El valor de un atributo IDREFS puede contener múltiples valores IDREF separados por espacio en blanco.

- **`ENTITY ó ENTITIES`**

  Un valor de atributo ENTITY debe ser el nombre de una sola entidad. El valor de un atributo ENTITIES puede contener múltiples nombres de entidad separados por espacio en blanco.

- **`NMTOKEN ó NMTOKENS`**

  Los atributos de nombre de token son una forma restringida de un atributo string. En general, un atributo NMTOKEN debe consistir en una sola palabra. El valor de un atributo NMTOKENS puede contener múltiples valores NMTOKEN separados por espacio en blanco.

- **Lista de nombres**

  Se puede especificar que el valor de un atributo debe ser tomado de una lista predeterminada de nombres. Esto es comúnmente llamado un tipo enumerativo, porque cada uno de los posibles valores está explícitamente enumerado en la declaración. Alternativamente, se puede especificar que esos nombres se deban corresponder a un nombre de notación.

Existen cuatro posibles valores de default:

- **`#REQUIRED`**

  El atributo debe poseer un valor explícitamente especificado en cada ocurrencia del elemento en el documento.

- **`#IMPLIED`**

  El valor del atributo no es requerido, y no se proporciona ningún valor como default. Si el valor no es especificado, el procesador de XML debe proceder sin ninguno.

- **"valor"**

  Un atributo puede tener cualquier valor legal como un default. El valor de atributo no es requerido en cada elemento en el documento, y si no está presente, se entregará el valor default especificado.

- **`#FIXED` "valor"**

Una declaración de atributo puede especificar que un atributo tiene un valor fijo. En este caso, el atributo no es requerido, pero si ocurre, debe tener el valor especificado. Si no está presente, se proveerá el valor especificado como default. Uno de los usos de los atributos `#FIXED` es asociar semántica a un elemento.

El procesador de XML realiza normalización de valor de atributo sobre valores de atributos: las referencias de caracter son reemplazadas por el carácter referenciado, las referencias de entidad son resueltas (recursivamente), y el espacio en blanco es normalizado.

Declaraciones de entidad

Las declaraciones de entidad permiten asociar un nombre con algún otro fragmento de contenido. Esta construcción puede ser un pedazo de texto regular, un pedazo de la declaración de tipo de documento o una referencia a un archivo externo conteniendo texto o datos binarios.

Algunas típicas declaraciones de entidad son:

Ejemplo 2

```
<!ENTITY ATI "ArborText, Inc.">

<!ENTITY boilerplate SYSTEM "/standard/legalnotice.xml">

<!ENTITY ATIlogo SYSTEM "/standard/logo.gif" NDATA GIF87A>
```

Existen tres clases de entidades:

- **Entidades Internas**

  Una entidad interna asocia un nombre a un string de texto literal.. La primera entidad en el **Ejemplo 2** es una entidad interna. Utilizando `&ATI;` en cualquier lugar en el documento insertará *ArborText, Inc.* en dicha ubicación. Las entidades internas permiten definir atajos (shortcuts) para texto frecuentemente tipeado o texto que se espera cambiar, como el estado de revisión de un documento.

  Las entidades internas pueden incluir referencias a otras entidades internas, pero es considerado un error si éstas tienen un tratamiento recursivo.

  La especificación XML predefine cinco entidades internas:

  - `<` produce <
  - `>` produce >

- `&` produce el ampersand, `&`
- `&apos;` produce el caracter de la comilla simple, `'`
- `&quot;` produce el caracter de la comilla doble, `"`

- **Entidades Externas**

  Las entidades externas asocian un nombre al contenido de otro archivo. Permite a un documento XML referenciar al contenido de otro archivo. Las entidades externas contienen texto o datos binarios. Si contienen texto, el contenido del archivo externo es insertado en el punto de referencia y parseado como parte del documento. Los datos binarios no son parseados y pueden ser únicamente referenciados en un atributo. Los datos binarios son usados para referenciar figuras y otro contenido no XML en el documento.

  La segunda y tercera entidad en el **Ejemplo 2** son entidades externas.

  Empleando `&boilerplate;` insertará el contenido del archivo `/standard/legalnotice.xml` en la ubicación de la referencia de entidad. El procesador XML parseará el contenido de ese archivo como si ocurriera literalmente en esa locación.

  La entidad `ATIlogo` es también una entidad externar, pero su contenido es binario. La entidad `ATIlogo` puede ser únicamente utilizada como el valor de un atributo de tipo ENTITY (ó ENTITIES). El procesador XML pasará esta información a la aplicación, pero no intentará procesar el contenido de `/standard/logo.gif`.

- **Entidades de Parámetro**

  Las entidades de parámetro pueden sólo aparecer en la declaración de tipo de documento. Una declaración de entidad de parámetro es identificada colocando `%` (signo de porcentaje-espacio) delante de su nombre en la declaración. El signo de porcentaje es también empleado en referencias a entidades de parámetro, en lugar del ampersand. Las referencias de entidades de parámetro son inmediatamente expandidas en la declaración de tipo de documento y el texto reemplazado es parte de la declaración, cuando por el contrario, las referencias de entidad normales no son expandidas. Las entidades de parámetro no son reconocidas en el cuerpo de un documento.

  Observando las declaraciones de elemento anteriores se notará que dos de ellas tienen el mismo modelo de contenido:

```
<!ELEMENT burns     (#PCDATA | quote)*>

<!ELEMENT allen     (#PCDATA | quote)*>
```

Estos dos elementos son lo mismo sólo porque tienen la misma definición literal. Con el objeto de hacer más explícito el hecho de que estos dos elementos son semánticamente lo mismo, se emplea una entidad de parámetro para definir sus modelos de contenido. Son dos las ventajas de usar una entidad de parámetro: permite dar un nombre descriptivo al contenido y permite el cambio del modelo de contenido en un solo lugar, si se quieren actualizar las declaraciones de elementos, asegurándose que siempre permanecerán iguales:

```
<!ENTITY % personcontent "#PCDATA | quote">

<!ELEMENT burns (%personcontent;)*>

<!ELEMENT allen (%personcontent;)*>
```

Declaraciones de notación

Las declaraciones de notación identifican tipos específicos de datos binarios externos. Esta información es pasada a la aplicación procesante, la cual puede hacer uso según desee. Una típica declaración de notación es la siguiente:

```
<!NOTATION GIF87A SYSTEM "GIF">
```

Discusión sobre la necesidad de una Declaración de Tipo de Documento

El contenido XML puede ser procesado sin una declaración de tipo de documento. Sin embargo, existen algunas instancias donde la declaración es requerida:

- **Ambientes de autoría**

  La mayoría de los ambientes de autoría necesitan leer y procesar declaraciones de tipo de documento con el objeto de entender y promover los modelos de contenido del documento.

- **Valores default de atributo**

  Si un documento XML cuenta con valores default de atributos, al menos parte de la declaración debe ser procesada para obtener los valores default correctos.

- **Manejo de espacio en blanco**

> La semántica asociada con el espacio en blanco en contenido de elemento difiere de la semántica asociada con el espacio en blanco en contenido mixto. Sin un DTD, el procesador de XML no tiene manera de distinguir entre estos dos casos, y todos los elementos se tratarán efectivamente como contenido mixto.

En las aplicaciones donde una persona compone o edita los datos (en contraposición de datos que pueden ser generados directamente desde una base de datos, por ejemplo), es probable que sea requerido un DTD si se desea tener garantizada cierta estructura.

Incluyendo una Declaración de Tipo de Documento

Si está presente, la declaración de tipo de documento debe ser la primera construcción en el documento luego de opcionales instrucciones de procesamiento y comentarios.

La declaración de tipo de documento identifica el elemento raíz del documento y puede contener declaraciones adicionales.

Todos los documentos XML deben tener un único elemento raíz que engloba todo el contenido del documento. Las declaraciones adicionales pueden venir de un DTD externo, llamadas subconjunto externo, o estar incluidas directamente en el documento, llamadas subconjunto interno, o ambas:

```
<?XML version="1.0" standalone="no"?>

<!DOCTYPE chapter SYSTEM "dbook.dtd" [

<!ENTITY %ulink.module "IGNORE">

<!ELEMENT ulink (#PCDATA)*>

<!ATTLIST ulink

    xml:link       CDATA  #FIXED "SIMPLE"

    xml-attributes CDATA  #FIXED "HREF URL"

    URL            CDATA  #REQUIRED>

]>

<chapter>...</chapter>
```

Este ejemplo referencia un DTD externo, `dbook.dtd`, e incluye declaraciones de elemento y atributo para el elemento `ulink` en el subconjunto interno.

Vale destacar que las declaraciones en el subconjunto interno sustituyen las declaraciones en el subconjunto externo.

El procesador XML lee el subconjunto interno antes que el subconjunto externo y la primera declaración toma precedencia.

Con el motivo de determinar si un documento es válido, el procesador XML debe leer la declaración de tipo de documento completa (ambos subconjuntos interno y externo). Pero para algunas aplicaciones, la validez puede no ser requerida, y puede ser suficiente para el procesador XML leer solamente el subconjunto interno.

Se puede comunicar esta información en una *declaración de documento standalone*. La declaración de documento standalone, `standalone="yes"` o `standalone="no"` ocurre en la declaración XML. El valor `yes` indica que sólo las declaraciones internas necesitan ser procesadas. Un valor `no` indica que *ambas* declaraciones internas y externas deben ser procesadas.

Otras cuestiones importantes

Además del markup, existen algunas cuestiones a considerar: manejo de espacio en blanco, normalización de valores de atributos, el lenguaje en el que el documento es escrito.

Manejo de espacio en blanco

Sólo se puede determinar si el espacio en blanco es significativo si se conoce el modelo de contenido de los elementos en cuestión. En pocas palabras, el espacio en blanco es significativo en contenido mixto y *no* es significativo en contenido de elemento.

La regla para un procesador XML es que se deben pasar todos los caracteres que no son markup hacia la aplicación. Si el procesador es un procesador validador, debe informar también a la aplicación cuales caracteres de espacio en blanco son significativos.

El atributo especial `xml:space` puede ser usado para indicar explícitamente que el espacio en blanco es significativo. En cualquier elemento que incluya la especificación de atributo `xml:space='preserve'`, todo el espacio en blanco dentro de ese elemento (y dentro de los sub-elementos que no explícitamente limpien el `xml:space`) es significativo.

Los únicos valores legales para `xml:space` son `preserve` y `default`. El valor `default` indica que el procesamiento default es el deseado.

Normalización de valores de atributos

El procesador XML realiza normalización de valores de atributos: las referencias de caracter son reemplazadas por el caracter referenciado, las referencias de entidad son resueltas (recursivamente), y el espacio en blanco es normalizado.

Identificación del lenguaje

Muchas aplicaciones que procesan documentos pueden beneficiarse con la información sobre el lenguaje natural en el que el documento es escrito. XML define el atributo `xml:lang` para identificar el lenguaje. El propósito de este atributo es estandarizar la información a lo largo de las aplicaciones, por eso, la especificación XML también describe cómo los lenguajes deben ser identificados.

La validez de un documento XML

Existen dos categorías de documentos XML: bien-formados y válidos.

Documentos bien-formados

Un documento puede únicamente estar bien-formado si obedece la sintaxis de XML. Un documento que incluye secuencias de caracteres de markup que no pueden ser parseadas o son inválidas no puede estar bien-formado.

Además, el documento debe satisfacer todas las siguientes condiciones:

- La instancia de documento debe conformar la gramática de documentos XML. En particular, algunas construcciones de markup (referencias de entidad de parámetro, por ejemplo) son sólo permitidas en lugares específicos. Si lo anterior no se respeta, entonces el documento no puede estar bien-formado.
- El texto de reemplazo para todas las entidades de parámetro referenciadas dentro de una declaración de markup consiste en cero o más declaraciones de markup completas.
- No puede aparecer un atributo más de una vez en el mismo tag de comienzo.
- Los valores de atributo de tipo string no pueden contener referencias a entidades externas.
- Los tags no vacíos deben estar apropiadamente anidados.
- Las entidades de parámetro deben estar declaradas antes de ser utilizadas.
- Todas las entidades excepto las siguientes: `amp`, `lt`, `gt`, `apos`, y `quot` deben estar declaradas.

- Ninguna entidad binaria puede estar referenciada en el flujo de contenido, solamente puede ser usada en un atributo declarado como `ENTITY` o `ENTITIES`.
- Ninguna entidad de texto ni de parámetro está permitida ser recursiva, directa o indirectamente.

Por definición, si un documento no está bien-formado, *no* es un documento XML.

Documentos válidos

Un documento bien-formado es válido sí y solo sí contiene una correcta declaración de tipo de documento y si el documento obedece las restricciones de esa declaración (la secuencia y anidamiento de elementos es válida, los atributos requeridos son proporcionados, los valores de los atributos son del tipo correcto, etc.).

Espacios de nombres en XML

La especificación de los espacios de nombres en XML fue adoptada con el objetivo de brindar a los creadores de software el fácil soporte de vocabularios muy ricos en markup concebidos con XML.

Definición del problema

El punto más importante de XML es permitir a los usuarios la facilidad de crear únicos tags que identifican su información de una manera más significante.

Mientras ésto ofrece a los usuarios una gran flexibilidad, se exponen problemas para el intercambio y la integración de software.

El problema que se enfrenta es el siguiente: qué sucede cuando un documento hace uso del mismo nombre de tag en diferentes contextos. Dentro de un documento, el término “título” puede referirse al documento mismo, al nombre del libro, y al apelativo formal asociado a su autor. El problema no es sólo para los nombres de elementos, se extiende a los atributos también.

La potencial colisión de los diferentes usos de los mismos nombres expone problemas para cualquier software basado en la escritura de XML.

Solución

La especificación de espacios de nombres en XML enfrenta esta cuestión permitiendo a los tags y atributos inscribirse en un contexto. Este contexto es el espacio de nombres del tag o de atributos, el cual es una simple dirección de Web.

En forma concisa: los espacios de nombres son una manera simple y directa de distinguir nombres (de tags y atributos) utilizados en documentos XML, sin importar de donde vienen.

Por ejemplo:

```
<html xmlns="http://www.w3.org/HTML/1998/html4"
      xmlns:xdc="http://www.xml.com/books">
 <head><title>Book Review</title></head>
 <body>
  <xdc:bookreview>
   <xdc:title>XML: A Primer</xdc:title>
   <table>
    <tr align="center">
     <td>Author</td><td>Price</td>
     <td>Pages</td><td>Date</td></tr>
    <tr align="left">
     <td><xdc:author>Simon St. Laurent</xdc:author></td>
     <td><xdc:price>31.98</xdc:price></td>
     <td><xdc:pages>352</xdc:pages></td>
     <td><xdc:date>1998/01</xdc:date></td>
    </tr>
   </table>
  </xdc:bookreview>
 </body>
</html>
```

En el ejemplo anterior, los elementos con prefijo `xdc` están asociados al espacio de nombres `http://www.xml.com/books`, y cualquier otra cosa sin prefijo se asume que está en el espacio de nombres default `http://www.w3.org/HTML/1998/html4`, que es el nombre de espacio de nombres para HTML.

Los prefijos están vinculados a los nombres completos utilizando los atributos sobre el elemento más alto cuyos nombres comienzan con `xmlns`. Los prefijos no significan nada, son un atajo sintáctico para los nombres completos. Estos nombres completos, son URIs, es decir, direcciones de Web. A la especificación de los espacios de nombres no le interesa adonde apuntan (si es que apuntan a algún sitio) dichas URIs.

Una cosa que el espacio de nombres no define es qué significan los tags o los atributos. El grupo de trabajo de esquemas XML está trabajando en el hecho de permitir a los diseñadores de DTDs o de esquemas XML referenciar a conjuntos de tags públicos.

XML Schemas

El W3C XML Schema Definition Language es un lenguaje XML para describir y restringir el contenido de documentos XML. El propósito de un esquema es definir una clase de documentos XML.

Los XML Schemas pueden eventualmente suplantar a los DTDs como el mecanismo primario para restringir datos XML. Un XML Schema, que está en un formato de un documento XML, brinda la misma función que un DTD mientras corrige algunas de sus limitaciones. Mientras que un DTD restringe el tipo del tag que va dentro de un documento XML, no tiene forma de restringir rangos de un atributo dado (por ejemplo: edad entre 0 y 150 años). Los XML Schemas proveen a los desarrolladores de un tipado de datos más potente para los contenidos de elementos y valores de atributos. XML Schema suplementa el mecanismo DTD básico incluido en XML Versión 1.0 con un framework más riguroso para declarar estructura y contenido de documentos XML. Además, proveen un tipado de datos más potente para los valores de atributos.

Limitaciones de los DTDs

XML heredó los Document Type Definitions (DTDs) de SGML. Los DTDs son el mecanismo de esquema para SGML. Los DTDs pueden ser usados para definir modelos de contenido, es decir, el orden y anidamiento válido de los elementos. También los DTDs se utilizan para definir, de manera limitada, los tipos de datos de los atributos.

Los DTDs sufren de numerosas limitaciones:

- Son escritos en una sintaxis diferente (no XML) .
- No tienen soporte para espacios de nombres.
- Ofrecen un sistema de tipado de datos extremadamente limitado. Los DTDs pueden sólo expresar el tipo de datos de los atributos en términos de enumeraciones explícitas y en pocos formatos ordinarios de strings. No tienen facilidades para describir números, fechas, valores de moneda, etc.
- Tienen un complejo y frágil mecanismo de extensión basado en un poco más que la sustitución de strings. La peor cuestión sobre el mecanismo de extensión del DTD (entidades de parámetro) es que no hace explícitas las relaciones. Dos elementos definidos para tener el mismo modelo de contenido no son la misma cosa en ninguna forma explícita. Asimismo, un grupo de atributos definidos como una entidad de parámetro y reusados no son lógicamente un grupo, son sólo “coincidentalmente” un grupo.

XML Schema supera estas limitaciones y es mucho más expresivo que el DTD. La expresividad adicional permitirá a las aplicaciones Web intercambiar datos XML de una forma mucho más robusta sin tener que apoyarse en herramientas de validación ad hoc.

Aunque XML Schema está pensado para reemplazar a los DTDs, en el corto tiempo los DTDs seguirán beneficiados de una serie de ventajas:

- Soporte muy difundido de herramientas. Todas las herramientas SGML y muchas herramientas XML pueden procesar DTDs.
- Extenso deployment. Un gran número de tipos de documento están definidos utilizando los DTDs: HTML, XHTML, DocBook, TEI, J2008, CALS, etc.
- Extensa especialización y muchos años de aplicación práctica.

Además, los DTDs son bien comprendidos por una gran comunidad de programadores y consultores de SGML y XML.

Características principales de XML Schema

XML Schema ofrece las siguientes características:

- **Tipos de datos más ricos**. Define booleanos, fechas, URIs, enteros, números decimales, números reales, intervalos de tiempo, etc. Además de estos tipos simples predefinidos, existen facilidades para crear otros tipos.
- **Tipos definidos por el usuario**. Se permite definir un tipo de datos nombrado. Esto es más potente que simplemente definir que dos elementos tienen la misma estructura porque la información sobre el tipo de dato está disponible para el procesador.
- **Agrupamiento de atributos**. Esta característica permite al autor del esquema hacer explícitas las relaciones de los atributos. En los DTDs, el agrupamiento puede llevarse a cabo con una entidad de parámetro, simplificando el proceso de creación de un DTD, pero la información no es pasada al procesador.
- **Soporte de espacio de nombres**. Con la introducción de espacios de nombres en XML, la validación se ha vuelto más dificultosa. La especificación de XML Schema describe un mecanismo para la composición de esquemas (permitiendo a los esquemas soportar múltiples espacios de nombres para ser combinados de una manera racional así la validación puede ser efectuada).

Resource Description Framework (RDF)

El otro estándar en el que se basa RSS es Resource Description Framework (RDF).

Introducción a RDF

La World Wide Web fue originalmente construida para consumo humano, y aunque todo es legible por una computadora, estos datos no son comprendidos o interpretados por una computadora. Es muy difícil automatizar algo en la Web, y debido al gigantesco volumen de información que contiene la Web, es imposible manejarla en forma manual. La solución que RDF propone es utilizar metadatos para describir los datos contenidos en la Web. Metadatos son datos sobre datos, por ejemplo, el catálogo de una biblioteca esta conformado por metadatos porque el mismo describe las publicaciones de la biblioteca. Los metadatos en el contexto de la Web son los datos que describen recursos en la Web. La distinción entre datos y metadatos no es absoluta, es creada por una aplicación particular, y muchas veces el mismo recurso será interpretado de ambas maneras simultáneamente.

RDF es una infraestructura para el procesamiento de metadatos. Provee interoperatibilidad entre aplicaciones que intercambian información que es comprendida por computadora, en la Web. RDF proporciona facilidades para facultar el procesamiento automatizado sobre recursos de Web. RDF puede ser utilizado en una amplia gama de aplicaciones. Por ejemplo, en el área de descubrimiento de recursos, para proveer mejores capacidades a un motor de búsqueda; en el área de catalogación, para describir el contenido y las relaciones entre el contenido disponibles en un determinado sitio Web, página o biblioteca digital; en agentes de software inteligente, para facilitar el intercambio común de conocimiento; en clasificación de contenidos, para describir colecciones de páginas que conforman una sola unidad lógica o documento; en la descripción de derechos de propiedad intelectual de páginas Web; también puede ser usado para expresar preferencias de privacidad de un usuario, y de la misma forma, las políticas de privacidad de un sitio Web.

RDF con las firmas digitales serán la clave para la construcción del "Web of Trust" (Web de confianza) para el comercio electrónico, trabajo colaborativo y otras aplicaciones.

La sintaxis utilizada para la codificación y transporte de metadatos RDF hace uso de Extensible Markup Language (XML): uno de los objetivos de RDF es hacer posible la especificación de semánticas para datos basados en XML de una manera estandarizada e interoperable. RDF y XML son complementarios: RDF es un modelo de metadatos y solo direcciona por referencia muchas de las cuestiones de codificación que el transporte y el almacenamiento de archivos requiere, como ser, la internacionalización, los conjuntos de caracteres, etc. Por estas razones, RDF delega en el soporte de XML. Es importante aclarar que esa sintaxis de XML es sólo una de las posibles sintaxis para RDF y otras maneras alternativas de representar el mismo modelo de datos RDF pueden surgir.

El objetivo principal de RDF es definir un mecanismo para la descripción de recursos que no hace suposiciones sobre un particular dominio de aplicación, ni define (a priori) las semánticas de ningún dominio de aplicación. La definición del mecanismo debe ser neutral con respecto al dominio, este mecanismo debe ser conveniente para la descripción de información acerca de cualquier dominio.

Para facilitar la definición de metadatos, RDF tendrá un sistema de clases, al igual que muchos de los sistemas de programación y modelado orientados a objetos. Una colección de clases, típicamente creadas para un propósito especifico o un dominio, es llamada un *esquema* (schema). Las clases son organizadas en una jerarquía, y ofrecen extensibilidad a través del refinamiento por subclasificación. De esta forma, a la hora de crear un esquema levemente diferente a uno existente, no será necesario reinventar la rueda. Uno puede proveer modificaciones incrementales a un esquema base. Mediante la capacidad de compartir esquemas comunes, RDF soportará la reusabilidad de las definiciones de metadatos. Debido a la extensibilidad incremental proporcionada por RDF, los agentes procesadores de metadatos serán capaces de rastrear los orígenes de los esquemas que no están familiarizados y realizar acciones significantes sobre los metadatos, para los cuales, estos agentes no fueron diseñados para procesar.

La habilidad de compartir y extender que posee RDF también permite a los autores de metadatos utilizar herencia múltiple para mezclar definiciones, para proveer múltiples vistas de sus datos, aprovechando el trabajo hecho por otros autores. Además, es posible crear instancias de datos RDF basadas en múltiples esquemas de múltiples fuentes (intercalación de diferentes tipos de metadatos). Los esquemas pueden ser escritos en RDF.

Como resultado del acuerdo de muchas comunidades sobre los principios básicos de la representación y transporte de los metadatos, RDF ha sido influenciado por varias fuentes diferentes. La mayor influencia proviene de la *Web standardization community*, en la forma de metadatos HTML y PICS, la *library community,* el *structured document community* en la forma de SGML y más importante XML, y también la *knowledge representation (KR) community.* Tambíen existen otras áreas de la tecnología que contribuyeron al diseño de RDF; estas incluyen los lenguajes de programación y modelado orientados a objetos, así como las bases de datos. Debido al hecho que RDF fue influenciado también por la comunidad KR, los lectores familiarizados en este campo fueron advertidos que RDF no especifica un mecanismo de razonamiento. RDF puede ser caracterizado como un sistema de *simple frame*. Un mecanismo de razonamiento puede ser construido sobre este sistema de *frame*.

Historia

La historia de los metadatos en el W3C comienza en 1995 con PICS (Platform for Internet Content Selection). PICS es un mecanismo para la comunicación de valuaciones (*ratings*) de páginas Web desde un servidor hacia los clientes. Estos ratings contienen información sobre el contenido de páginas Web: por ejemplo, si una página contiene violencia, nudismo, sexo, lenguaje obsceno, etc. En vez de proporcionar un conjunto fijo de criterios, PICS introdujo un mecanismo general para la creación de ratings. Diferentes organizaciones podrían clasificar contenido basándose en sus propios objetivos y valores, y los usuarios podrían configurar sus browsers para filtrar cualquier página Web que no iguale dicho criterio.

Luego de varias reuniones, se identificaron limitaciones en la especificación de PICS, y se delinearon los requerimientos funcionales para atacar el problema mas general de asociar información descriptiva a los recursos de Internet. Mas tarde, se dieron cuenta que la infraestructura diseñada para PICS podía aplicarse a diferentes campos, entonces se consolidó el grupo de trabajo W3C Resource Description Framework.

RDF es el resultado de varias comunidades de metadatos reuniendo sus necesidades para proveer una robusta y flexible arquitectura para el soporte de metadatos en la Web. Ningún individuo u organización ha inventado RDF. RDF es un esfuerzo de diseño colaborativo. Varias compañías que poseen membresía en la W3C han contribuido con sus recursos intelectuales. RDF está bosquejado sobre el diseño de XML. Otros emprendimientos de metadatos, como el Dublin Core y el Warwick Framework han influenciado el diseño de RDF.

El modelo RDF básico

La base de RDF es un modelo para la representación de propiedades con nombre y valores de propiedad. El modelo RDF está basado en principios bien establecidos de varias comunidades de representación de datos. Las propiedades RDF también representan relaciones entre recursos y, por consiguiente, un modelo RDF puede parecerse a un diagrama de entidad-relación. Mas precisamente, RDF Schemas, los cuales son instancias de modelos de datos RDF, son diagramas ER. En la terminología de diseño orientado a objetos, los recursos corresponden a objetos y las propiedades corresponden a variables de instancia.

El modelo de datos de RDF es una forma, independiente de la sintaxis, de representar expresiones RDF. La representación del modelo de datos es utilizada para evaluar la equivalencia en significado. Dos expresiones RDF son equivalentes sí y solo sí sus representaciones de modelo de datos son las mismas. Estas definiciones de equivalencia permiten algunas variaciones sintácticas en las expresiones sin alterar el significado de las mismas.

El modelo de datos básico consiste en tres tipos de objetos:

1. **Recursos**: son todas las cosas descriptas por las expresiones RDF. Un recurso puede ser una página Web completa (por ejemplo: un documento HTML). Puede ser una parte de una página Web; por ejemplo: un elemento HTML o XML específico dentro de un documento fuente. Puede ser también una colección de páginas; por ejemplo: un sitio Web entero. Un recurso puede ser también un objeto que no es directamente accesible vía la Web; por ejemplo: un libro impreso. Los recursos son siempre referenciados por URIs mas opcionales *anchor ids*. Cualquier cosa puede tener una URI; la extensibilidad de las URIs permite la inserción de identificadores para cualquier entidad imaginable.
2. **Propiedades**: una propiedad es un aspecto específico, una característica, un atributo, o una relación, utilizada para describir un recurso. Cada propiedad tiene un significado específico, define sus valores permitidos, los tipos de recursos que puede describir, y sus relaciones con otras propiedades.
3. **Aserciones**: un determinado recurso junto con una propiedad nombrada mas el valor de esa propiedad para ese recurso es una aserción (*statement*) RDF. Estas tres partes individuales de una aserción son llamadas, respectivamente, el sujeto, el predicado y el objeto. El objeto de una aserción (el valor de la propiedad) puede ser otro recurso o puede ser un literal; por ejemplo: un recurso (especificado por una URI) o un simple string u otro tipo de datos primitivo definido en XML. En términos RDF, un literal puede tener contenido que sea markup (conjunto de tags) XML pero no es evaluado por el procesador de RDF.

Ejemplos

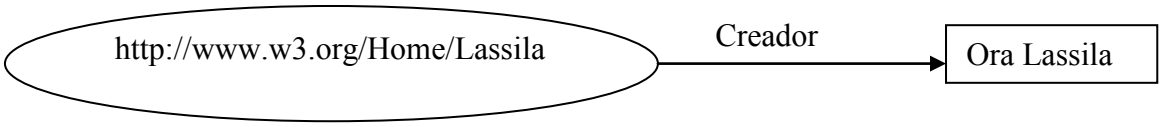
Consideremos la siguiente sentencia:

Ora Lassila es el creador del recurso http://www.w3.org/Home/Lassila.

Esta sentencia tiene las siguientes partes:

1. Sujeto (*recurso*): http://www.w3.org/Home/Lassila
2. Predicado (*propiedad*): Creador
3. Objeto (*literal*): “Ora Lassila”

La siguiente es una representación gráfica de la aserción RDF anterior. Para esto, se utilizará un grafo dirigido con etiquetas. En este diagrama, los nodos (los óvalos) representan recursos y los arcos representan propiedades con nombre. Los nodos que representan literales de string son dibujados como rectángulos.



La sintaxis RDF básica

El modelo de datos de RDF provee un framework abstracto y conceptual para la definición y uso de metadatos. Una sintaxis concreta es necesaria para los propósitos de creación e intercambio de estos metadatos. La especificación de RDF utiliza la codificación XML como sintaxis para el intercambio. RDF también requiere la facilidad de espacios de nombres de XML para asociar precisamente cada propiedad con el esquema que define dicha propiedad.

Una simple aserción RDF raramente aparece aislada; mas comúnmente, varias propiedades de un recurso son presentadas en forma conjunta. La sintaxis RDF XML ha sido diseñada para facilitar el agrupamiento de múltiples aserciones para un mismo recurso dentro del elemento Description. Este elemento nombra, en su atributo about, el recurso al cual todas las aserciones se aplican.

El elemento RDF es un simple empaquetador que marca los límites dentro de un documento XML cuyo contenido es pensado explícitamente para ser mapeado dentro de una instancia de modelo de datos RDF. Description contiene los elementos restantes que provocan la creación de aserciones dentro de la instancia del modelo. Típicamente existirán mas de una aserción hecha sobre un recurso; el elemento Description provee la forma de brindar el nombre de recurso una sola vez para varias aserciones. El valor del atributo about es interpretado como una referencia URI.

Un simple elemento `Description` puede contener más de un elemento *propiedad* (`<nombreProp>`contenido`</nombreProp>`). Cada elemento *propiedad* agrega un arco al grafo. La interpretación de este grafo es definida por el diseñador del esquema. Dentro del elemento *propiedad*, puede aparecer un atributo `resource` especificando que otro recurso es el valor de esta propiedad, es decir, el objeto de la aserción es otro recurso identificado por una URI en vez de un literal. Los nombres de propiedad deben ser asociados a un esquema. Esto puede lograrse calificando los nombres de elemento con el prefijo del espacio de nombres para conectar, evitando toda ambigüedad, la definición de la propiedad con el esquema RDF correspondiente o declarando un espacio de nombres default.

El ejemplo de la anterior sección:

Ora Lassila es el creador del recurso http://www.w3.org/Home/Lassila.

Es representado en RDF/XML de la siguiente manera:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creador>Ora Lassila</s:Creador>
  </rdf:Description>
</rdf:RDF>
```

Aquí el prefijo de espacio de nombres *'s'* refiere al espacio de nombres elegido por el autor de esta expresión RDF y definida en la declaración XML del espacio de nombres como: `xmlns:s="http://description.org/schema/"`.

El nombre (URI) del espacio de nombres en la declaración, es un identificador único global para un esquema particular que el autor de metadatos está utilizando para definir el uso de la propiedad Creador. Otros esquemas pueden también definir una propiedad llamada Creador y ambas propiedades serán distinguidas por sus identificadores de esquema. Un esquema generalmente define varias propiedades. Entonces un sola declaración de espacio de nombres será suficiente para disponer de un extenso vocabulario de propiedades.

El documento XML completo, contenedor de la descripción anteriormente introducida será:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creador>Ora Lassila</s:Creador>
  </rdf:Description>
</rdf:RDF>
```

Utilizando el espacio de nombres default para el espacio de nombres RDF, el anterior documento puede también ser escrito de la siguiente manera:

```
<?xml version="1.0"?>
<RDF
  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <Description about="http://www.w3.org/Home/Lassila">
    <s:Creador>Ora Lassila</s:Creador>
  </Description>
</RDF>
```

El atributo parseType

El atributo `parseType` es un atributo especial que puede aparecer en cualquier elemento. Tiene el efecto de cambiar la interpretación del contenido del elemento donde aparece. El atributo `parseType` debe tener uno de los valores `Literal` o `Resource`. El valor `Literal` especifica que el contenido del elemento sea tratado como un literal RDF/XML, es decir, el contenido no debe ser interpretado por un procesador de RDF. El valor `Resource` especifica que el contenido del elemento debe ser tratado como si fuera el contenido del elemento `Description`. Otros valores de `parseType` están reservados para futuras especificaciones de RDF. En todos los casos, el contenido de un elemento con el atributo `parseType` debe cumplir con la restricción XML de *bien formado*. El contenido de un elemento con el atributo `parseType="Resource"` debe además cumplir con el modelo de contenido del elemento `Description`.

Esquemas y espacio de nombres

Cuando escribimos una sentencia en un lenguaje natural, utilizamos palabras que comunican cierto significado. Este significado es crucial para entender la afirmación y, en el caso de aplicaciones de RDF, es crucial para determinar que el correcto procesamiento ocurre como fue pensado. Es crítico que ambos roles, el escritor y el lector de la afirmación entiendan el mismo significado para los términos utilizados, como *Creador*. De lo contrario resultará todo en una confusión. En un medio de escala global como la World Wide Web no es suficiente confiar en conceptos culturales comunes, como el concepto de creador. Debe ser todo lo más preciso posible.

El significado en RDF es expresado a través de la referencia a un esquema (*schema*). Un esquema es una especie de diccionario. Un esquema define los términos que serán usados en las aserciones RDF y brinda un significado específico a cada uno de ellos. Una variedad de formas de esquema pueden ser utilizadas con RDF. Un esquema es un lugar donde las definiciones y restricciones de uso para las propiedades están documentadas. Para evitar confusiones entre definiciones independientes (potenciales de caer en conflictos) sobre el mismo término, RDF hace uso de la facilidad de espacio de nombres provista por XML. Los espacios de nombres son una manera sencilla de atar un uso específico de una palabra en contexto al diccionario (esquema) donde la definición propuesta será encontrada. En RDF, cada predicado usado en una aserción debe ser identificado con exactamente un solo espacio de nombres, o esquema. Sin embargo, un elemento `Description` puede contener aserciones con predicados de muchos esquemas.

Contenedores

Frecuentemente es necesario referirse a una colección de recursos. Por ejemplo, decir que un trabajo fue hecho por más de una persona, o listar todos los módulos de software de un paquete. Los contenedores RDF son usados para mantener estas listas de recursos o literales.

RDF define tres tipos de objetos contenedores:

1. **Bag**: es una lista sin orden de recursos o literales. Son utilizados para declarar que una propiedad tiene múltiples valores y que no es significativo el orden en el cual los valores son dados. Son permitidos valores duplicados.
2. **Sequence**: es una lista ordenada de recursos o literales. Son utilizados para declarar que una propiedad tiene múltiples valores y que el orden de los valores es significativo. Pueden usarse para preservar el orden alfabético de valores. Son permitidos valores duplicados.
3. **Alternative**: es una lista de recursos o literales que representa alternativas para un solo valor de una propiedad. Puede ser usado para proveer una lista de sitios espejo en Internet en el cual un recurso puede ser hallado. Una aplicación que hace uso de una propiedad cuyo valor es una colección *Alternative* es consciente que puede elegir uno de los items en la lista como sea apropiado.

El modelo contenedor

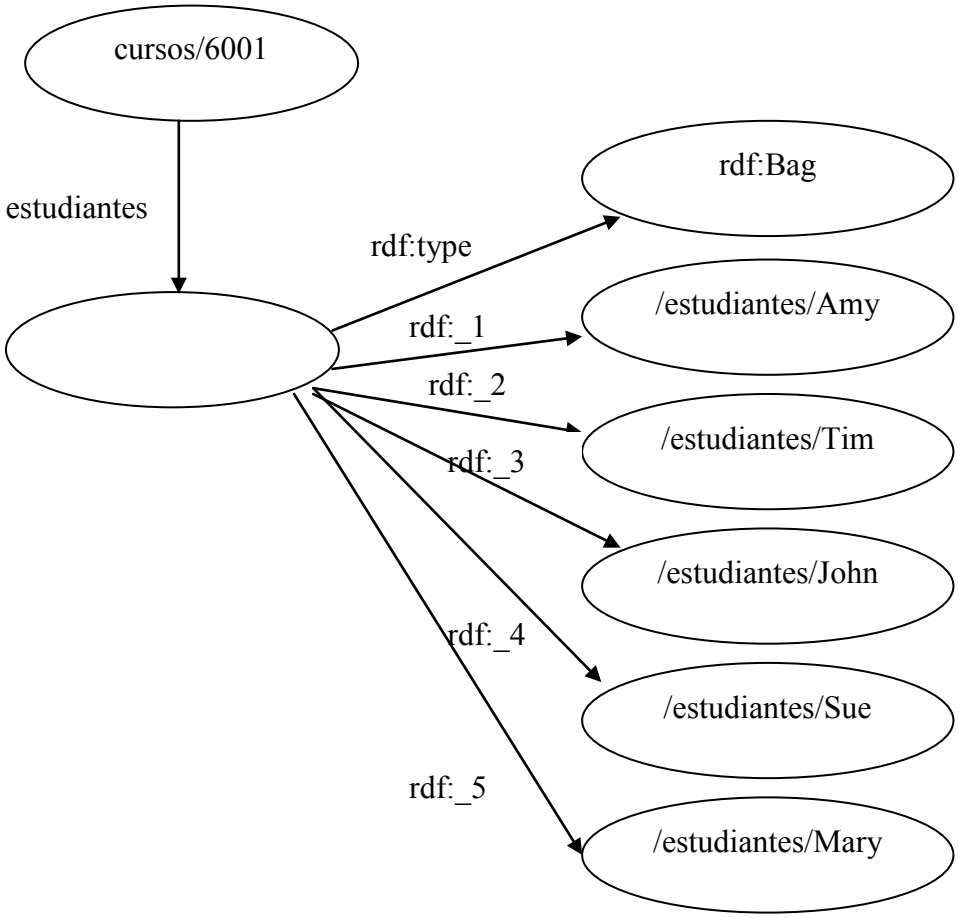
Para representar una colección de recursos, RDF utiliza un recurso adicional que identifica la colección específica (una instancia de una colección, en la terminología de diseño orientado a objetos). Este recurso debe ser declarado como una instancia de uno de los tipos de objetos contenedores anteriormente definidos. La propiedad *type*, definida abajo es usada para construir esta declaración. La relación de pertenencia entre este recurso contenedor y los recursos que pertenecen a la colección está definida por un conjunto de propiedades definidas expresamente para este propósito. Esas propiedades de pertenencia son simplemente nombradas "_1", "_2", "_3", etc. Los recursos contenedores pueden tener otras propiedades en adición a las propiedades de pertenencia y la propiedad *type*. Cualquiera de las aserciones adicionales describen al contenedor mismo.

El uso común de los contenedores es como valor de una propiedad. Cuando son usados de esta forma, la aserción continúa teniendo un único objeto sin importar el número de miembros dentro del contenedor. El recurso contenedor mismo es el objeto de la aserción.

Por ejemplo, para representar la siguiente aserción:

Los estudiantes del curso 6001 son Amy, Tim, John, Mary, y Sue.

El modelo RDF es:



La sintaxis de los contenedores

La sintaxis de los contenedores RDF tiene la siguiente forma:

1. **Sequence**: es definida por el tipo de elemento `rdf:Seq`
2. **Bag**: es definida por el tipo de elemento `rdf:Bag`
3. **Alternative**: es definida por el tipo de elemento `rdf:Alt`

El contenido de los elementos contenedor toma la forma de uno o varios de los elementos `rdf:li` donde cada uno de ellos, a su vez, contienen un string literal, una descripción o un contenedor, excluyentemente.

Los contenedores pueden ser utilizados en cualquier lugar donde un elemento `Description` sea permitido.

RDF/XML usa `li` como un elemento conveniente para evitar tener que explícitamente enumerar cada miembro. El elemento `li` asigna propiedades _1, _2, y siguiendo, como sea necesario. El nombre de elemento `li` fue elegido como mnemónico del término *list item* de HTML.

El contenedor `Alt` requiere tener al menos un elemento. Este miembro será identificado como la propiedad _1 y es el valor default o preferido.

Ejemplos

El modelo de la sentencia:

Los estudiantes del curso 6001 son Amy, Tim, John, Mary, y Sue.

Es escrita en RDF/XML como sigue:

```
<rdf:RDF>
  <rdf:Description about="http://miuniversidad.edu/cursos/6001">
    <s:estudiantes>
      <rdf:Bag>
        <rdf:li resource="http://miuniversidad.edu/estudiantes/Amy"/>
        <rdf:li resource="http://miuniversidad.edu/estudiantes/Tim"/>
        <rdf:li resource="http://miuniversidad.edu/estudiantes/John"/>
        <rdf:li resource="http://miuniversidad.edu/estudiantes/Mary"/>
        <rdf:li resource="http://miuniversidad.edu/estudiantes/Sue"/>
      </rdf:Bag>
    </s:estudiantes>
  </rdf:Description>
</rdf:RDF>
```

Aserciones sobre aserciones

RDF además de soportar la construcción de aserciones sobre recursos de la Web, también puede ser utilizado para crear aserciones sobre otras aserciones RDF. Estas aserciones son llamadas aserciones de *alto orden*. Para poder construir aserciones sobre otras aserciones, se deberá conformar un modelo de la aserción original. Este modelo es un recurso nuevo al cual se adjuntarán propiedades adicionales.

El modelado de aserciones

Las aserciones son construidas sobre recursos. El modelo de una aserción es el recurso que se necesita para permitir la construcción de nuevas aserciones (aserciones de alto orden) sobre la aserción modelada.

Por ejemplo:

Ora Lassila es el creador del recurso http://www.w3.org/Home/Lassila.

RDF considerará la anterior sentencia como un hecho. Si, en vez, se construye la siguiente sentencia:

Ralph Swick dice que Ora Lassila es el creador del recurso http://www.w3.org/Home/Lassila.

No se afirma nada sobre el recurso http://www.w3.org/Home/Lassila; por el contrario, sólo se expresa un hecho sobre la aserción que Ralph hizo. Para poder expresar este hecho en RDF, se tendrá que modelizar la aserción original como un recurso con cuatro propiedades. Este proceso es formalmente llamado *reificación* en la comunidad de Knowledge Representation. Un modelo de una aserción es llamado una aserción reificada *(reified statement)*.

Para modelizar aserciones RDF define las siguientes propiedades:

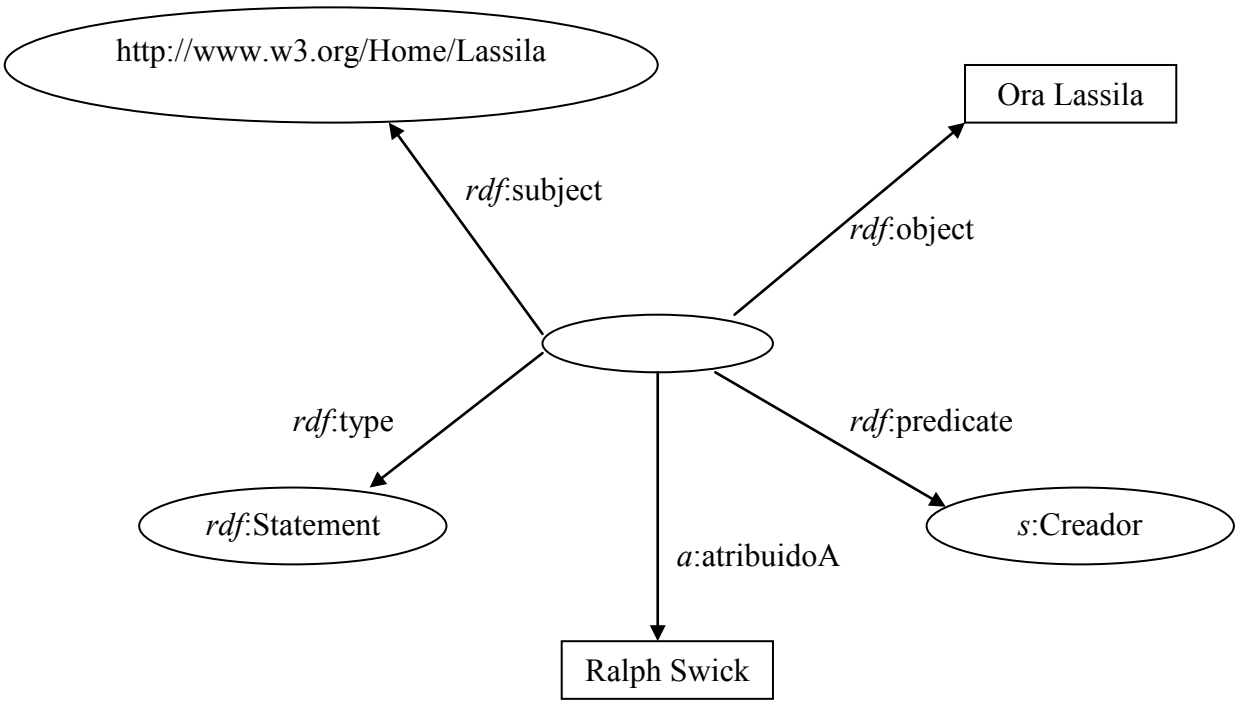
- **`subject`**: esta propiedad identifica el recurso que está siendo descripto por la aserción modelada. Esto es, el valor de la propiedad `subject` es el recurso sobre el cual la aserción original fue hecha (en el ejemplo, http://www.w3.org/Home/Lassila).
- **`predicate`**: esta propiedad identifica la propiedad original en la aserción modelada. El valor del la propiedad `predicate` es un recurso representando la propiedad específica en la aserción original (en el ejemplo, creador).
- **`object`**: esta propiedad identifica el valor de propiedad en la aserción modelada. El valor de la propiedad `object` es el objeto en la aserción original (en el ejemplo, "Ora Lassila").
- **`type`**: el valor de la propiedad `type` describe el tipo del nuevo recurso. Todas las aserciones reificadas son instancias de `RDF:Statement`, es decir, tienen una propiedad `type` cuyo objeto es `RDF:Statement`. La propiedad `type` es también usada generalmente para declarar el tipo de cualquier recurso.

Un nuevo recurso con las anteriores cuatro propiedades representa la aserción original. Esta puede ser usada como el objeto de otras aserciones, asi como también, pueden hacerse aserciones adicionales sobre la misma. El recurso con las cuatro propiedades no es un reemplazo para la aserción original, es el modelo de la aserción. Una aserción y su correspondiente aserción reificada existen independientemente en un grafo RDF y ambas pueden ser presentadas sin la otra.

Para modelar el previo ejemplo, se adjuntará otra propiedad a la aserción reificada (`atribuidoA`) con el valor apropiado (en este caso, `Ralph Swick`). Utilizando la sintaxis RDF/XML, esto tiene la siguiente forma:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:a="http://description.org/schema/">
  <rdf:Description>
    <rdf:subject resource="http://www.w3.org/Home/Lassila" />
    <rdf:predicate resource="http://description.org/schema/Creador" />
    <rdf:object>Ora Lassila</rdf:object>
    <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Statement" />
    <a:atribuidoA>Ralph Swick</a:atribuidoA>
  </rdf:Description>
</rdf:RDF>
```

La anterior descripción RDF se representa gráficamente de la siguiente manera:



RDF Schema

Las descripciones utilizadas en las aplicaciones RDF pueden ser modeladas como relaciones entre recursos Web. El modelo de datos de RDF define un modelo simple para la descripción de interrelaciones entre recursos en los términos de propiedades con nombre y valores. Sin embargo el modelo de datos de RDF, no provee mecanismos para declarar estas propiedades, tampoco provee mecanismos para definir las relaciones entre estas propiedades y otros recursos. Ese es el rol de RDF Schema.

Las comunidades de descripción de recursos requieren la habilidad de expresar ciertas cosas sobre ciertas clases de recursos. Por ejemplo, para describir recursos bibliográficos se necesita definir atributos como "autor", "título", "tema". Para la certificación digital son requeridos atributos como "checksum" y "autorización". La declaración de estas propiedades (atributos) y sus correspondientes semánticas son definidas en el contexto de RDF como un esquema RDF (*RDF schema)*. Un esquema define no solo las propiedades de un recurso (como título, autor, tema, color, etc.) sino también puede definir las clases de recursos a describir (libros, páginas Web, personas, compañías, etc.).

RDF define un lenguaje de especificación de esquemas. Es decir, RDF no especifica un vocabulario de elementos descriptivos como "autor", en cambio, especifica los mecanismos necesarios para definir estos elementos, para definir clases de recursos que pueden ser utilizados con estos elementos, para restringir las posibles combinaciones de clases y relaciones, y para detectar violaciones a estas restricciones. Más brevemente, el mecanismo del RDF Schema provee un sistema de tipado básico para usar en los modelos RDF. Define recursos y propiedades como `rdfs:Class` y `rdfs:subClassOf` que son utilizados para la especificación de esquemas específicos a una determinada aplicación. El sistema de tipado es especificado en términos del modelo de datos básico de RDF, como recursos y propiedades. De esta forma, los recursos constituyentes de este sistema de tipado se convierten en parte del modelo RDF de cualquier descripción que lo utilice.

El lenguaje de especificación de esquemas RDF es un lenguaje de representación declarativo influenciado por ideas de representación de conocimiento (como redes semánticas, frames, lógica de predicados), así también como de lenguajes de especificación de esquemas de base de datos (NIAM) y grafos para expresar modelos de datos. El lenguaje de especificación de esquemas RDF es menos expresivo, pero más simple de implementar, que los lenguajes de cálculo de predicados como el CycL y KIF.

RDF y el lenguaje RDF Schema también están basados en las investigaciones de metadatos realizadas en la comunidad Digital Library. En particular, RDF adopta un acercamiento modular hacia los metadatos que puede ser considerado como una implementación del Warwick Framework. RDF representa la evolución del modelo de Warwick Framework en el aspecto que el Warwick Framework permite a cada vocabulario de metadatos ser representado en diferentes sintaxis. En RDF, todos los vocabularios están expresados dentro de un único modelo bien definido. Esto permite una mezcla más refinada de vocabularios procesables por máquina. Y satisface la necesidad de crear metadatos en los cuales las aserciones pueden ser construidas sobre múltiples vocabularios que son manejados de forma descentralizada por comunidades especializadas independientes.

Los esquemas RDF pueden ser contrastados con los XML Document Type Definitions (DTDs) y los XML Schemas. A diferencia de un XML DTD o XML Schema, que brindan restricciones específicas sobre la estructura de un documento XML, un esquema RDF provee información sobre la interpretación de las aserciones dadas en un modelo de datos RDF. Mientras que un XML Schema puede ser utilizado para validar la sintaxis de una expresión RDF/XML, un esquema sintáctico solo no es suficiente para los propósitos de RDF. Los esquemas RDF pueden también especificar restricciones que deben ser respetadas por estos modelos de datos. Futuros trabajos sobre RDF Schema y XML Schema podrían permitir la combinación de reglas de sintaxis y semántica.

La especificación de RDF Schema fue influenciada directamente por la consideración de los siguientes problemas:

1. **Platform for Internet Content Selection (PICS)**

   El modelo y la sintaxis de RDF es adecuado para representar etiquetas PICS, sin embargo no provee un mapeo de propósito general de sistemas de clasificadores PICS hacia una representación RDF.

2. **Metadatos Web**

   Una de las aplicaciones obvias de RDF es en la descripción de páginas Web. Este es uno de los principales objetivos de la Dublin Core Metadata Initiative. El Dublin Core Element Set es un conjunto de 15 elementos proyectados para la aplicación global en la descripción de recursos Web y así permitir sus descubrimientos. El Dublin Core ha sido la mayor influencia para el desarrollo de RDF. Una consideración importante en el desarrollo del Dublin Core fue no sólo la simple habilidad de descripción, sino también proveer la capacidad de calificar descripciones para soportar una elaboración de dominio específico y una descripción precisa.

   La especificación de RDF Schema provee un sistema comprensible por máquina para la definición de esquemas para vocabularios descriptivos como el Dublin Core. Permite a los diseñadores especificar clases de recursos y propiedades para comunicar descripciones de esas clases, relaciones entre propiedades y clases, y limitaciones sobre las combinaciones de clases, propiedades y valores.

3. **Mapas de sitios (Sitemaps) y Navegación Conceptual**

   Un *sitemap* es una descripción jerárquica de un sitio Web. Una taxonomía de temas es un sistema de clasificación que puede ser utilizado por creadores de contenidos o terceras partes de confianza para organizar o clasificar recursos Web. La especificación de RDF Schema provee un mecanismo para la definición de vocabularios necesarios por dichas aplicaciones. También RDF Schema provee recursos suficientes para la creación de modelos RDF que representen la estructura lógica de diccionarios y otros sistemas de clasificación de biblioteca.

4. **P3P**

   La W3C Platform for Privacy Preferences Project (P3P) ha especificado una gramática para la construcción de aserciones sobre las prácticas de recolección de datos y preferencias personales acerca de un sitio Web, así como una sintaxis para el intercambio de datos estructurados.

   Aunque las prácticas de recolección de datos personales han sido descriptas en P3P empleando un conjunto de tags XML específicos de aplicación, existen beneficios para la utilización de un modelo de metadatos general para estos datos. La estructura de las políticas P3P puede ser interpretada como un modelo RDF. Haciendo uso de un esquema de metadatos para describir las semánticas de descripciones de prácticas de privacidad permitirá que los datos de las prácticas de privacidad sean utilizados entre otros metadatos dentro de una consulta para el descubrimiento de recursos. Y permitirá que un agente genérico de software actúe sobre metadatos de privacidad empleando las mismas técnicas usadas para otros metadatos descriptivos.

Resumen

La World Wide Web permite el acceso sin precedentes a información distribuida a lo largo y ancho del planeta. Los metadatos perfeccionan el acceso a esta información y RDF es un estándar de la W3C para definir la arquitectura necesaria para el soporte de metadatos. RDF es una aplicación de XML que impone las restricciones estructurales necesarias para proveer métodos sin ambigüedad para la expresión de semánticas. Estás semánticas permiten la codificación consistente, intercambio, y procesamiento sistematizado de los metadatos. RDF adicionalmente provee el mecanismo para la publicación de vocabularios, legibles por humanos y procesable por computadoras, diseñados para alentar el intercambio, uso y extensión de semánticas de metadatos entre diferentes comunidades de información.

El uso efectivo de metadatos entre aplicaciones requiere de convenciones comunes sobre semánticas, sintaxis y estructura. El diseño de infraestructuras, como RDF, capaces de soportar estas construcciones proporciona las bases necesarias para el manejo de información en la Web y provee la habilidad de transformar la Web en un más útil y potente recurso de información.

El framework para aggregators RSS

Introducción

Se desarrolló un framework que resume el comportamiento común de las aplicaciones de captura de contenidos RSS (recolectores de datos de documentos RSS), denominados *aggregators*. Y para ejemplificar el comportamiento de este framework se implementaron también dos aggregators particulares derivados del framework. Estos dos aggregators particulares capturan datos de la misma fuente de información, es decir, de los mismos documentos (canales o feeds) RSS, pero sirven a entidades que tienen diferentes necesidades de información, a pesar de que pertenecen al mismo área de interés.

También se desarrolló un generador de canales RSS para completar la demostración del funcionamiento de los aggregators y del framework. Esta aplicación genera dos canales RSS y los actualiza con una frecuencia de segundos. Ambos aggregators particulares están suscriptos a estos dos canales. Conjuntamente, el framework para aggregators, los dos aggregators particulares derivados de dicho framework y el generador de canales RSS conforman el prototipo desarrollado.

Tecnología empleada

Para la implementación de todo el prototipo se utilizó el lenguaje de programación Java, más específicamente la API Java™ 2 Platform, Standard Edition, versión 1.4.1 (compilando con la implementación de Sun Microsystems, Java 2 SDK, Standard Edition, versión 1.4.1_01). La portabilidad y extensibilidad de XML y de Java hacen que sean compañeros naturales para cubrir los requerimientos de flexibilidad e integración de las nuevas aplicaciones.

Para el procesamiento de los documentos XML se empleó directamente la API para el parsing de datos XML, Simple API for XML versión 2 (SAX2). Su elección se basó en la gran agilidad que posee y los pocos recursos que consume, debido fundamentalmente a que procesa el documento XML mientras lo va leyendo y a que no genera ninguna estructura interna para almacenar ningún dato tomado del documento XML. La implementación del parser XML utilizada para el desarrollo fue Apache Xerces2 Java Parser, versión 2.3.0.

Comportamiento del aggregator RSS

Lo siguiente es un resumen del comportamiento del aggregator RSS modelizado en el framework.

Cada canal RSS es actualizado por su autor con una frecuencia no determinada. El aggregator tiene registradas en una tabla las URLs de los canales a los que se va a conectar periódicamente para extraer los datos de su propio interés. Con una frecuencia especificada, el aggregator se conecta a las URLs de los canales que tiene registrados y verifica si hubo algún cambio (usualmente lee el header HTTP que contiene la última fecha de actualización del recurso, en este caso del documento RSS) con respecto a la última lectura realizada del canal que tiene documentada. Si se determina que el canal ha sido actualizado desde su última lectura por parte del aggregator, entonces se toman solamente los datos de interés, utilizando algoritmos de parsing y mapping XML. Luego se registra esta lectura como la última lectura completada para el canal. Una vez finalizada la recorrida por los canales registrados buscando cambios, se entregan los datos extraídos de los canales mapeados a objetos a un proceso, registrado previamente como cliente del aggregator. Este proceso realiza la tarea para con los datos extraídos de los canales, encomendada por el sistema que emplea el aggregator particular. Esta tarea puede ser: almacenar los datos en una base de datos, renderizar los datos directamente en HTML, etc.

Cada instancia particular de aggregator RSS debe especificar qué canales debe consultar y para cada canal, qué datos XML y cómo mapearlos a objetos Java.

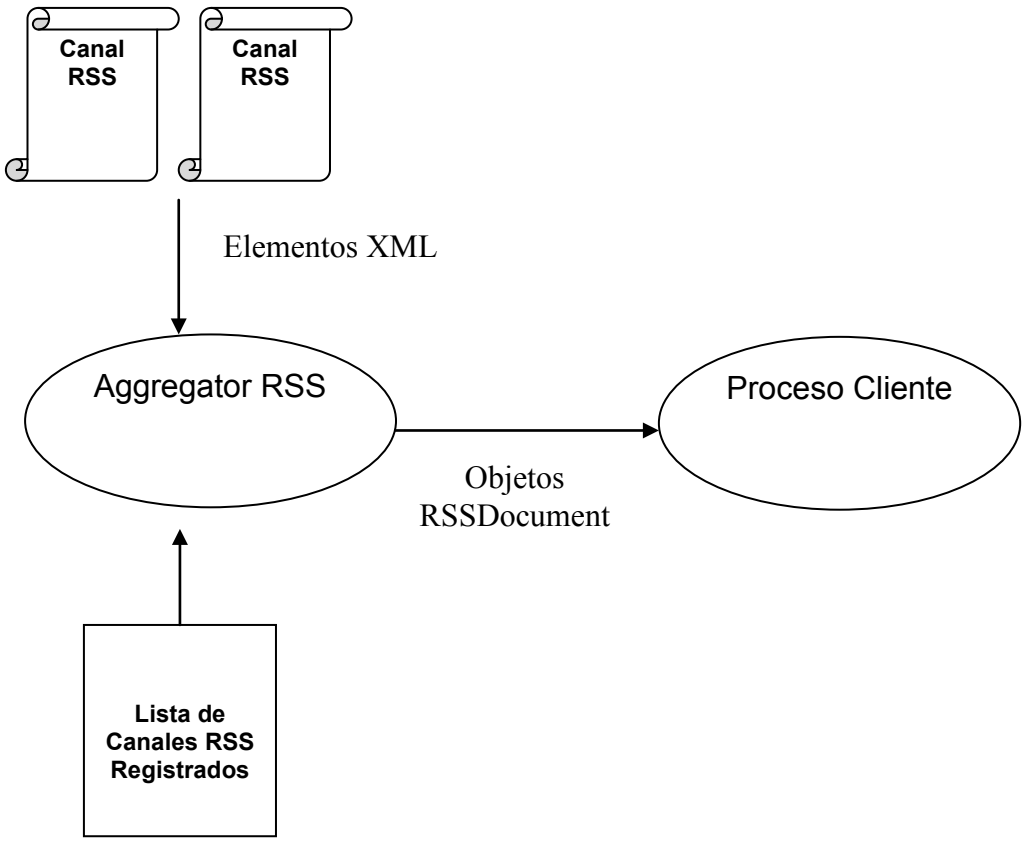
El proceso cliente del aggregator, encargado de recibir la información tomada de los canales en cada ejecución del aggregator y hacer algo con la misma, está sujeto a los requerimientos de un dominio de problema específico y por lo tanto, está fuera del alcance de este framework.

La estructura de los datos XML depende de los módulos RSS utilizados por el canal. Además cada aggregator particular es libre de tomar los datos que le interese de cada módulo en el canal, según sus necesidades de información.

En resumen, el desarrollador de un aggregator RSS particular, derivado del framework para aggregators RSS, debe definir:

1. La lista canales RSS a consultar periódicamente.
2. Las clases que van a contener los datos XML a tomar de cada canal.
3. La manera de mapear estos datos XML dentro de las clases mencionadas en el punto anterior.
4. El proceso cliente del aggregator, encargado de hacer algo con los datos tomados de los canales.

El gráfico de abajo ilustra los conceptos anteriormente explicados, clarificando el alcance del aggregator RSS modelizado en el framework y las relaciones con las otras entidades.



Pasos para derivar un aggregator particular

Un aggregator RSS particular, derivado del framework para aggregators RSS debe seguir los cinco pasos siguientes para aprovechar su funcionalidad:

1. **Implementación de los conjuntos de propiedades.**

   Implementar una clase que defina la estructura de objetos del conjunto de propiedades, es decir, los elementos y atributos XML de interés a tomar de un módulo RSS determinado. Cada módulo RSS extiende el formato básico RSS añadiendo propiedades para un área de interés específico. Es decir, esta clase será la responsable de alojar al subconjunto de propiedades de un módulo RSS particular. Dichas propiedades son las requeridas para este aggregator particular. La clase debe implementar la interface *RSSPropertySet* perteneciente al paquete *rssAggregator*. El único método que define esta clase es `getId()`, que retorna un String identificando al conjunto de propiedades de interés, generalmente se retorna el propio nombre de la clase. Para cada módulo RSS al cuál se deseen tomar ciertas propiedades (o su totalidad) se deberá definir una clase que implemente la interface *RSSPropertySet*.

2. **Implementación de los mapeadores customizados.**

   Un mapeador customizado es una clase que implementa la interface *RSSCustomMapper* en el paquete *rssAggregator*. Esta clase define cómo las propiedades (elementos y atributos XML) de interés para el aggregator y pertenecientes a un módulo RSS específico se mapean a objetos Java. Más específicamente define cómo estas propiedades se extraen del canal y se almacenan en la instancia colaboradora apropiada, que representa al conjunto de propiedades (*RSSPropertySet*), implementada en el punto anterior. La interface *RSSCustomMapper* define un par de métodos para cada elemento básico RSS (channel, item, image y textinput). Estos métodos son `composeXXX` y `connectXXXTagTracker`, donde `XXX` es un elemento básico RSS. El primer método del par crea la instancia adecuada que representa al conjunto de propiedades (*RSSPropertySet*) y la registra contra la clase que representa al elemento básico RSS (*RSSItem, RSSChannel, RSSImage* ó *RSSTextInput*). El segundo método detalla cómo se mapean las propiedades del canal RSS a objetos, creando una red de `TagTracker`s (detallados más adelante en el punto *Clases para el mapeo de XML* de la sección *Clases que conforman el framework*). Por ejemplo, si se quieren mapear propiedades de un módulo RSS asociadas al elemento básico `channel`, deben implementarse `composeChannel` y `connectChannelTagTracker`. Para un canal RSS dado pueden existir múltiples mapeadores customizados (cada uno de ellos con su apropiada clase colaboradora *RSSPropertySet*) correspondientes a los diferentes módulos RSS que el propio canal acarrea.

3. **Composición del documento XML de canales RSS registrados.**

   Comúnmente nombrado *RegRSSChannels.xml*. En este documento figuran todos los canales que el aggregator particular quiere consultar periódicamente para extraer información. Para cada canal RSS a consultar (elemento `channel`) se especifican: el nombre del canal, la URL, una breve descripción y la lista de mapeadores (elemento `customMappers`). Cada ítem de esta lista (subelemento `mapper`) declara el nombre de la clase responsable del mapeo de las propiedades de interés (instancias que implementan la interface *RSSCustomMapper*) de acuerdo a los requerimientos de un aggregator determinado.

4. **El proceso cliente del aggregator.**

   En cada trabajo del aggregator, luego de recorrer la lista de canales registrados buscando alguna actualización en los mismos, se obtiene un conjunto de *RSSDocument* (que representa la información tomada de un canal que ha sufrido alguna actualización). Estas instancias de *RSSDocument* son entregadas finalmente a un proceso cliente. Este último es dueño de aplicar cualquier tratamiento a los *RSSDocument*, como guardarlos en un medio persistente, clasificarlos, categorizarlos, cruzarlos con otra información o con otros *RSSDocument*, renderizarlos en HTML, etc. Este proceso debe ser creado según los requerimientos del sistema que utiliza al aggregator y debe registrarse como cliente del aggregator. Para esto, sólo tiene que implementar la interface *RSSAggregatorListener* del paquete *rssAggregator*. Esta interface define un único método (`mappedDocs`) que tiene la responsabilidad de recibir al conjunto de *RSSDocument* y aplicarles el tratamiento requerido.

5. **Comienzo del aggregator.**

   Una vez que están codificadas las clases de conjuntos de propiedades (*RSSPropertySet*), las clases de los mapeadores customizados (*RSSCustomMapper*), compuesto el documento *RegRSSChannels.xml* y codificado el proceso cliente, finalmente, se procede a invocar a `RSSAggregator.start`. La clase *RSSAggregator* en el paquete *rssAggregator* es la fachada de todo el framework, es decir, un aggregator particular interactúa directamente sólo con esta clase para realizar las tareas del aggregator. El método estático `start` toma como argumentos la URL del documento *RegRSSChannels.xml*, la frecuencia en segundos de activación del aggregator (define la periodicidad con que se consultan los canales registrados) y la instancia que implementa *RSSAggregatorListener* (el proceso cliente del aggregator particular).

El prototipo desarrollado

Para ilustrar el funcionamiento del framework para aggregators RSS, se desarrollaron dos aggregators RSS particulares derivados de dicho framework. Estos dos aggregators particulares capturan datos de la misma fuente de información, es decir, de los mismos canales RSS. Pero estos aggregators son utilizados por entidades que tienen diferentes necesidades de información, a pesar de que pertenecen al mismo área de interés. La fuente de información de los aggregators serán dos canales RSS que presentan información específica del área de interés, que para este desarrollo se eligió el mundo de la música Rock.

También se desarrolló un generador de canales RSS para completar la demostración del funcionamiento de los aggregators y del framework. Esta aplicación genera dos canales RSS y los actualiza con una frecuencia de segundos. Ambos aggregators particulares están suscriptos a estos dos canales. Conjuntamente, el framework para aggregators, los dos aggregators particulares derivados de dicho framework y el generador de canales RSS conforman el prototipo desarrollado.

Además de graficar el comportamiento de los aggregators RSS, este prototipo pretende visualizar las características principales del modelo de publicación que propone RSS como: liviano (se emplean documentos XML y no se necesitan desarrollos ni protocolos de comunicación especiales), la extensibilidad (los canales incorporan propiedades específicas de un área de interés particular, mediante los módulos RSS definidos), simultaneidad de publicación (los datos de un mismo canal son capturados por diferentes entidades al mismo tiempo) y la uniformidad para el tratamiento de la información (no existen diferencias de ningún tipo para la especificación de datos y metadatos). También se podrá observar la gran flexibilidad y libertad que poseen los aggregators RSS a la hora de obtener los datos de un canal, dado que cada uno captura la información que necesita.

Se definen para este área de interés elegido, dos módulos RSS:

- **Módulo RockAlbum**: provee información sobre un álbum de Rock editado por un artista/banda.
- **Módulo RockNews:** provee información sobre una noticia relacionada a un artista/banda del Rock.

Más adelante se detallan las propiedades de cada uno de estos módulos.

Se definen dos canales RSS:

- **albums.rss**: canal que presenta los álbumes de Rock que están actualmente de oferta. Utiliza el módulo RockAlbum. Cada ítem en el canal refiere a un álbum de Rock.
- **news.rss**: canal conteniendo las últimas noticias de Rock. Utiliza el módulo RockNews. Cada ítem en el canal provee información sobre una noticia referida a un artista del Rock.

Los hipervínculos que aparecen en los canales son ficticios. Estos datos son incluidos con el único fin de visualizar el funcionamiento del prototipo.

Los dos aggregators particulares están suscriptos a los dos canales RSS definidos, *news.rss* y *albums.rss*, y conocen (saben como interpretar) los módulos RockNews y RockAlbum. A pesar de que ambos leen los mismos canales, poseen diferentes necesidades de información, diferentes requerimientos. Por lo tanto, cada uno toma los datos que le interesan de cada canal. En la implementación, esto se refleja en que cada aggregator particular tiene sus propios mapeadores customizados con sus correspondientes conjuntos de propiedades para cada canal, ajustados a sus requerimientos.

Descripción de las necesidades de información de cada aggregator:

- **Aggregator1**: los datos capturados son utilizados en un sitio Web que pone a disposición de los usuarios una base de datos de la música Rock. Captura toda la información que proveen ambos canales. Estos canales son utilizados, por este aggregator, básicamente como empaquetadores de datos. No aprovecha los hipervínculos al sitio Web proveedor, suministrados en cada ítem de cada canal.
- **Aggregator2**: los datos capturados son utilizados en un sitio Web que presenta noticias del Rock como paneles informativos de lectura rápida, sin detalles, donde lo más importante son las noticias más actuales. Se captura la información más relevante de cada ítem de cada canal. Estos canales son utilizados, por este aggregator, como resumen (metadatos) del contenido del sitio Web proveedor de los canales. Utilizando los hipervínculos que exponen los items de cada canal, un usuario interesado puede navegar hacia información más detallada, brindada ya por el sitio Web proveedor del canal.

Las clases implementadas para cada uno de los aggregators RSS particulares del prototipo son:

- **Aggregator1**: todas las clases propias de Aggregator1 se encuentran en el paquete `customAgg1` dentro del *customAgg1.jar*. La clase principal es `Agg1RSSAggregator`. Esta clase realiza la invocación `RSSAggregator.start` para comenzar la ejecución del aggregator particular. Para el canal *albums.rss* implementa el mapeador customizado `AlbumAgg1Mapper` y el conjunto de propiedades `AlbumAgg1PropSet`. Para el canal *news.rss* implementa el mapeador customizado `NewsAgg1Mapper` y el conjunto de propiedades `NewsAgg1PropSet`. Como proceso cliente registra una instancia de la clase `HTMLWriterAgg1` que simplemente renderiza en HTML los datos capturados de cada canal. Esto es para que sea más visible la diferencia entre los distintos prototipos, sin adicionar complejidad en algo que está fuera del alcance del framework para aggregators RSS.

- **Aggregator2**: todas las clases propias de Aggregator2 se encuentran en el paquete `customAgg2` dentro del *customAgg2.jar*. La clase principal es `Agg2RSSAggregator`. Esta clase realiza la invocación `RSSAggregator.start` para comenzar la ejecución del aggregator particular. Para el canal *albums.rss* implementa el mapeador customizado `AlbumAgg2Mapper` y el conjunto de propiedades `AlbumAgg2PropSet`. Para el canal *news.rss* implementa el mapeador customizado `NewsAgg2Mapper` y el conjunto de propiedades `NewsAgg2PropSet`. Como proceso cliente registra una instancia de la clase `HTMLWriterAgg2` que simplemente renderiza en HTML los datos capturados de cada canal.

Ambos aggregators particulares tienen un archivo *app.properties* propio, donde se definen los atributos (o parámetros) de la aplicación. Abajo se describen las propiedades que aparecen en cada uno de estos archivos:

- **`regChannelsURL`**: URL del RegRSSChannels.xml.
- **`secsFrequency`**: intervalo de espera entre cada recolección de datos por parte del aggregator. Especificado en segundos.
- **`albumTemplate`**: URL del template para renderizar los datos tomados del canal *albums.rss*, utilizado por el proceso cliente.
- **`newsTemplate`**: URL del template para renderizar los datos tomados del canal *news.rss*, utilizado por el proceso cliente.
- **`albumOut`**: URL de la página HTML resultante de aplicar el template a`lbumTemplate` a los datos tomados del canal *albums.rss*.
- **`newsOut`**: URL de la página HTML resultante de aplicar el template `newsTemplate` a los datos tomados del canal *news.rss*.
- **`org.xml.sax.driver`**: nombre de la clase del driver para la SAX, responsable del parseo XML.

Además del archivo de propiedades *app.properties*, cada aggregator particular tiene su propio archivo XML *RegRSSChannels.xml*, donde especifica los canales a los que está suscripto y qué mapeadores (estándar o customizados) emplear para cada canal.

Archivos y directorios

Todo el prototipo desarrollado está contenido en el archivo *Rss.zip*. Al descompactar el archivo *Rss.zip* se crea la siguiente estructura de directorios (los mismos aparecen como sub-directorios del directorio donde se descompactó el .zip):

- **lib**: contiene el framework para aggregators RSS dentro del archivo *rssAggregator.jar*. También contiene *xercesImpl.jar*, con la implementación del parser XML Apache Xerces2 Java Parser, versión 2.3.0.
- **source**: incluye el archivo *sourceRssAggregator.zip* que comprende todos los fuentes del prototipo.

- **generator**: contiene el generador de canales RSS residente en el archivo *RSSGenerator.jar*. Esta aplicación se lanza tipeando en la línea de comandos `java -jar RSSGenerator.jar frec`, donde `frec` es la frecuencia en segundos de actualización de los canales, con un default de 3 segundos.
- **channels**: incluye dos sub-directorios, rockAlbums conteniendo el canal *albums.rss* y el otro sub-directorio rockNews conteniendo el canal *news.rss*. Ambos canales se actualizan mediante la aplicación *RSSGenerator.jar* previamente descripta.
- **agg1**: incluye al aggregator particular denominado *Aggregator1* cuyas clases están empaquetadas en el archivo *customAgg1.jar*. También contiene un archivo de propiedades de configuración para este aggregator particular, *app.properties* y la tabla de canales registrados para dicho aggregator en el archivo XML *RegRSSChannels.xml*. Para el funcionamiento de este prototipo los elementos `<url>` de cada canal declarado, deben tener la URL correspondiente del canal que se actualiza automáticamente mediante el generador, incluidos en el sub-directorio channels. agg1 contiene el sub-directorio albums con el documento HTML *albumesAgg1.htm*, resultado de renderizar en HTML el canal albums.rss junto con las plantillas *albumTemplateAgg1.htm* y *albumAgg1.css* necesarias para la renderización. De la misma manera, agg1 incluye un sub-directorio news conteniendo el documento HTML *newsAgg1.htm*, resultado de renderizar en HTML el canal *news.rss* junto con las plantillas *newsTemplateAgg1.htm* y *newsAgg1.css*. La aplicación se lanza tipeando en la línea de comandos: `java -jar customAgg1.jar`.
- **agg2**: incluye al otro aggregator particular *Aggregator2*. Tiene una conformación idéntica al directorio agg1 anteriormente comentado, con la salvedad que los nombres con referencia al número *1*, están sustituidos por el número *2*. La aplicación se lanza tipeando en la línea de comandos: `java -jar customAgg2.jar`.

Módulos RSS definidos

Lo que sigue son las dos especificaciones de los módulos RSS *RockAlbum* y *RockNews*, definidos para el prototipo. Las URIs de los espacios de nombres son ficticias.

Módulo RockAlbum

Provee información sobre un álbum de Rock editado por un artista/banda.

Declaración de espacio de nombres:

xmlns:ra="http://musicCommunity.org/modules/rockAlbum/"

Todos son sub-elementos de <item>:

- <ra:title> (#PCDATA)

  Título del álbum.

- <ra:artist> (#PCDATA)

  Nombre del artista/banda.

- <ra:URL> (#PCDATA) [URI]

  URL donde se encuentra toda la información referente al álbum.

- <ra:releaseDate> (#PCDATA) [W3CDTF]

  Fecha de lanzamiento.

- <ra:company> (#PCDATA)

  Nombre de la compañía discográfica.

- <ra:genre> (#PCDATA)

  Género o subgénero musical, por ej.: 'rockAndRoll', 'pop', 'R&B', etc. Existe una lista de géneros reconocidos pero está sujeta a actualizaciones, por lo que no se especifica una lista de valores fijos.

- <ra:themes> [lista rdf:Bag]

  - <ra:theme>

    - <ra:name> (#PCDATA)
    - <ra:order> (positive integer)
    - <ra:length> (mm:ss)

  Lista de temas. Para cada tema se especifica: nombre, orden de aparición y duración (en minutos y segundos).

Módulo RockNews

Provee información sobre una noticia relacionada a un artista/banda del Rock.

Declaración de espacio de nombres:

xmlns:rn="http://musicCommunity.org/modules/rockNews/"

Todos son subelementos de <item>:

- <rn:artist> (#PCDATA)

  Artista/Banda referenciado en la noticia.

- <rn:agency> (#PCDATA)

  Nombre de la agencia de noticias.

- <rn:dateTime> (#PCDATA) [W3CDTF]

  Fecha y hora de la noticia.

- <rn:URL> (#PCDATA) [URI]

  URL donde se encuentra toda la información referente a la noticia.

- <rn:country> (#PCDATA)

  País donde ocurrió la noticia.

- <rn:headline> (#PCDATA)

  Título de la noticia.

- <rn:body> (#PCDATA)

  Texto con el cuerpo completo de la noticia.

Leyendas

- [lista rdf:Bag]

  Indica que el contenido del elemento es una lista (sin orden) rdf:Bag de acuerdo con la especificación de contenedores soportados por la especificación de RDF. Es decir que el contenido del elemento tiene la siguiente sintaxis:

```
<elemento>
        <rdf:Bag>
                <rdf:li>
                        <!-- contenido de elemento -->
                </rdf:li>

                <rdf:li>
                        <!-- contenido de elemento -->
                </rdf:li>

                ...

        </rdf:Bag>
</elemento>
```

- [URI]

  El contenido del elemento debe poder ser interpretado como una URI.

- [W3CDTF]

  El contenido del elemento debe poder ser interpretado como una fecha y hora de acuerdo a lo especificado en Date and Time Formats de W3C [http://www.w3.org/TR/1998/NOTE-datetime-19980827]

Clases que conforman el framework

A continuación se describen las principales clases que conforman el framework para aggregators RSS.

Clases que representan a un documento RSS

- **RSSDocument**: representa a un documento RSS. Está conformado por un RSSChannel, uno o varios RSSItem y opcionalmente, un RSSImage y RSSTextInput.
- **RSSChannel**: representa el elemento *channel* de un documento RSS. Posee un identificador y una tabla donde registrar las propiedades que interesan mapear de los diferentes módulos RSS asociados a este elemento.
- **RSSItem**: representa el elemento *item* de un documento RSS. Posee un identificador y una tabla donde registrar las propiedades que interesan mapear de los diferentes módulos RSS asociados a este elemento.

- **RSSImage**: representa el elemento *image* de un documento RSS. Posee un identificador y una tabla donde registrar las propiedades que interesan mapear de los diferentes módulos RSS asociados a este elemento.
- **RSSTextInput**: representa el elemento *textinput* de un documento RSS. Posee un identificador y una tabla donde registrar las propiedades que interesan mapear de los diferentes módulos RSS asociados a este elemento.

Las clases RSSChannel, RSSItem, RSSImage y RSSTextInput son subclases de la clase abstracta **AbstractRSSCoreElement**. Esta última concentra todo el comportamiento común de las clases mencionadas.

La tabla que tiene registradas las propiedades a tomar de cada módulo asociado al elemento RSS determinado (*channel*, *item*, *image*, *textinput*) es una instancia de **RSSPropertySetContainer**. Esta clase es un *wrapper* de una *hashtable* donde cada elemento es una clase que implementa la interface ***RSSPropertySet***.

La interface ***RSSPropertySet*** fue creada para tipificar a un conjunto de propiedades (o elementos y atributos XML) de interés asociadas a un elemento básico de RSS (*channel*, *item*, *image*, *textinput*) definidas en un módulo RSS determinado. Estas propiedades son solamente las necesarias para satisfacer a una determinada entidad. Cada conjunto de propiedades debe ser registrado en la instancia de la clase apropiada (RSSChannel, RSSItem, RSSImage, RSSTextInput) para poder luego ser aprovechadas.

Clases para propiedades estándar RSS

Las siguientes clases representan las propiedades estándar de los elementos básicos de un documento RSS, es decir, los sub-elementos de *channel*, *item*, *image* y *textinput* definidos en la especificación de RSS 1.0. Estos conjuntos de propiedades estándar también deben ser registrados con la instancia de la clase apropiada si se desean usar.

- **RSSChannelStdPropSet**: representa las propiedades estándar del elemento RSS *channel*.
- **RSSItemStdPropSet**: representa las propiedades estándar del elemento RSS *item*.
- **RSSImageStdPropSet**: representa las propiedades estándar del elemento RSS *image*.
- **RSSTextInputStdPropSet**: representa las propiedades estándar del elemento RSS *textinput*.

Clases para el mapeo de XML

Las siguientes clases conforman el motor de mapeo de XML a Java:

- **SaxMapper**: se encarga de mapear a objetos los datos de un documento XML.

Utiliza para el parsing la API SAX2. Es una clase abstracta. En cada subclase de la misma se debe definir un método que construye la red de TagTrackers y otro que se encarga de retornar el objeto mapeado al final del proceso. La red de TagTrackers provee la información de cuáles son los elementos y atributos XML del documento que se van a mapear a objetos y de cómo se va a hacer este proceso.

- **TagTracker**: representa una acción a tomar cuando se encuentra un determinado elemento XML en el documento. Esta acción a tomar generalmente es capturar el valor de un atributo o tomar el contenido del elemento XML y mapearlo a un objeto determinado. Los objetos TagTracker trabajan en grupo dentro de una red de TagTrackers. Se comienza creando un root y luego se crea un TagTracker para cada elemento XML que ocurra en el elemento root del documento y que se desee mapear. Estos objetos se vinculan al root, y así siguiendo para cada elemento XML. SaxMapper y TagTracker conforman el patrón de diseño *Builder*.

Clases para el mapeo customizable

Las siguientes clases conforman el modelo de mapeo customizable de un documento RSS.

- **RSSDocMapper**: se encarga del mapeo de un documento RSS a un objeto RSSDocument. Es una subclase de SaxMapper. Posee un mecanismo de mapeo customizable, es decir, se pueden registrar objetos que se dediquen a mapear las propiedades (elementos y atributos XML) que interesan de módulos RSS específicos que están presentes dentro del documento. Estos objetos deben implementar la interface *RSSCustomMapper*. De esta forma, un aggregator particular es configurado con los custom mappers requeridos según las necesidades de información de una entidad particular.
- ***RSSCustomMapper* (interface)**: debe ser implementada por una clase que quiera especificar cómo deben ser mapeados a objetos las propiedades que interesan tomar de los módulos RSS. Estas propiedades están asociadas con los distintos elementos RSS básicos (*channel*, *item*, *image* y *textinput*). La interface define el comportamiento mediante los métodos `composeXXX` y `connectXXXTagTracker` donde `XXX` es el nombre del elemento RSS básico al cual las propiedades a absorber del documento RSS están asociadas. En la implementación, el `composeXXX` debe registrar el conjunto de propiedades necesario para el mapeo. Es decir, debe registrar una instancia que implementa la interface *RSSPropertySet*. El método `connectXXXTagTracker` debe decir cómo serán mapeadas las propiedades presentes en un documento RSS pertenecientes a un determinado módulo hacia el conjunto de propiedades registrado para tal elemento básico RSS. Por lo tanto debe existir una coherencia entre un mapeador (instancia que implementa *RSSCustomMapper*) definido por un usuario y su correspondiente conjunto de propiedades (instancia que implementa *RSSPropertySet*). La primera define cómo serán mapeadas las propiedades que se necesitan y la segunda define la estructura de objetos donde serán alojadas.

Clases para mapeadores de propiedades estándar RSS

Las siguientes clases son mapeadores provistos por el framework. Todas implementan mapeadores de propiedades estándar correspondientes a cada uno de los elementos básicos RSS. Como se detalló previamente, estas clases deben conformar a la interface *RSSCustomMapper*.

- **RSSChannelStdMapper**: especifica el mapeo de las propiedades estándar asociadas al elemento básico RSS *channel*, modelizadas en RSSChannelStdPropSet.
- **RSSItemStdMapper**: especifica el mapeo de las propiedades estándar asociadas al elemento básico RSS *item,* modelizadas en RSSItemStdPropSet.
- **RSSImageStdMapper**: especifica el mapeo de las propiedades estándar asociadas al elemento básico RSS *image,* modelizadas en RSSImageStdPropSet.
- **RSSTextInputStdMapper**: especifica el mapeo de las propiedades estándar asociadas al elemento básico RSS *textinput,* modelizadas en RSSTextInputStdPropSet.

Clases para el aggregator RSS

A continuación se enumeran las clases responsables del comportamiento de un aggregator RSS y clases colaboradoras.

- **RegRSSChannel**: representa a un canal RSS registrado en el aggregator particular. Posee la URL del canal e información de estado del mismo, como por ejemplo, la última fecha de actualización del canal que ha sido consultada por el aggregator. Además tiene registrada la lista de nombres de clases correspondientes a los custom mappers que el aggregator luego le solicitará instanciar para mapear las propiedades elegidas del canal.
- **RegRSSChannelsMapper**: encargada de mapear los canales registrados desde un documento XML (denominado generalmente *RegRSSChannels.xml*). Retorna una lista de instancias de RegRSSChannel con la información obtenida de dicho documento XML.
- ***RSSAggregatorListener*** (interface): debe ser implementada para poder ser cliente de un aggregator particular. Una aplicación implementa esta interface y se registra contra un aggregator particular, entonces luego de cada ejecución del aggregator, este último le entrega a la aplicación registrada los objetos RSSDocument obtenidos.
- **RSSAggregatorWorker**: modeliza el comportamiento de una ejecución de un aggregator RSS. Es decir, realiza el trabajo de recorrer la lista de canales registrados, verificar si los canales fueron actualizados desde la anterior recorrida y mapear las propiedades requeridas, obteniendo instancias de RSSDocument. Estas últimas son entregadas a la aplicación registrada como cliente del aggregator. Mantiene una lista de canales registrados (instancias de RegRSSChannel) y la aplicación cliente (instancia que implementa la interface *RSSAggregatorListener*). Delega en RSSDocMapper la responsabilidad de mapear los canales RSS que se consultan.

- **RSSAggregator**: responsable de sincronizar y planificar las ejecuciones del aggregator, la tarea en cada ejecución es delegada en RSSAggregatorWorker**.** Es la fachada de todo el framework, es decir, un aggregator particular interactúa directamente sólo con esta clase para realizar las tareas del aggregator.

Conclusiones

El prototipo desarrollado, más específicamente, el framework para aggregators RSS y los aggregators particulares derivados del mismo, ilustraron el funcionamiento de la aplicación basada en RSS más importante, el aggregator RSS (recolector de documentos RSS). Además, este prototipo permitió visualizar las principales características del modelo de publicación propuesto por RSS:

- **Liviano**: RSS es un formato de documento XML y propone un modelo de publicación muy simple. No necesitando de software específico en el cliente ni en el servidor, ni de protocolos de comunicaciones especiales para el intercambio de los datos (generalmente se utiliza HTTP).
- **Flexibilidad**: no existen reglas para el intercambio de documentos RSS. Tampoco existen reglas para el tratamiento de la información capturada de un documento RSS.
- **Extensibilidad**: mediante el mecanismo de modularización de RSS (basado en RDF y XML Namespaces) los canales RSS permiten incluir propiedades específicas de un área de interés particular. De esta manera se enriquece el formato RSS sin necesidad de reestructurar el conjunto básico de elementos (el núcleo). Esto es, se permite a las personas adicionar nueva funcionalidad a RSS sin cambiar la especificación.
- **Uniformidad para el tratamiento de la información**: no existen diferencias de ningún tipo (sintácticas, de empaquetamiento, etc.) para la especificación de los datos y de los metadatos.
- **Metadatos**: RSS provee un soporte completo para metadatos a través de RDF. RDF permite la representación de relaciones ricas en metadatos, más allá de las simples estructuras planas.
- **Simultaneidad de publicación**: los datos de un mismo canal RSS son tomados por diferentes entidades que poseen diferentes necesidades de información. Es decir, un canal RSS publicado puede ser capturado por diferentes aggregators RSS. Quedando a criterio de estos últimos, qué información existente en el canal les interesa capturar. Así, la información (total o parcial) expuesta en un determinado canal RSS puede aparecer en diferentes sitios Web (y otros medios, como e-mails, teléfonos celulares, etc.) simultáneamente.
- **Multipropósito**: RSS va más allá de la sindicación de noticias. Puede utilizarse para juntar datos de sistemas tales como manejadores de contenidos, administradores de conocimientos, portales de información corporativa, planeamiento de recursos empresariales, manejo de relaciones con clientes, etc.

RSS estandariza un formato para la distribución de contenidos. Esto facilita para un proveedor de contenidos la distribución amplia del contenido, y para los afiliados la recepción y proceso de contenidos de múltiples fuentes.

RSS es exitoso porque soluciona de una manera simple un problema mayor que la publicación de contenidos: el intercambio de datos/metadatos. Las siguientes son las soluciones que RSS aporta al problema general del intercambio de información:

- **Administración inteligente de contenidos**: RSS impulsa en contexto múltiples puntos de entrada hacia un único artículo primario, en vez de emplearse múltiples copias del mismo artículo (que introduce los problemas de mantenimiento). Como se sabe, los sitios Web con más links hacia ellos mismos ganan, y los que poseen el contenido más reciente también ganan. RSS crea una situación de triunfo para lo anteriormente descripto.
- **Maximización de la lectura de contenidos**: RSS permite atraer nuevos visitantes a un sitio Web, distribuyendo ampliamente el canal RSS generado a partir de los contenidos del sitio Web en cuestión. El canal RSS puede ser capturado por diferentes aggregators RSS (recolectores de canales RSS) e incluido en otros sitios Web. También el canal RSS puede viajar por otros medios digitales como e-mail.
- **Actualización automática de contenidos**: dos o más sitios Web asociados, que comparten un mismo área de interés (por ejemplo el lenguaje de programación Java) pueden generar sus propios canales RSS (a partir del contenido que cada uno publica) e intercambiárselos entre ellos para mantenerse automáticamente actualizados con respecto a las novedades del resto.
- **Derechos**: con la propuesta de distribución de contenidos que ofrece RSS, los proveedores de contenidos pueden seguir reteniendo los derechos de autor, si lo desean. Proporcionando un canal RSS, se puede controlar qué información es sindicada en el canal, si es un artículo completo o sólo un segmento. También el contenido puede ser protegido por los mecanismos corrientes de control de acceso, si sólo se distribuyen los links y los metadatos asociados.

Por todas las anteriores características se concluye que el modelo de publicación de contenidos de RSS es más potente que el de sus pares.

La utilización del framework para aggregators RSS como herramienta para crear aggregators RSS particulares (específicos de un dominio de problema), ofrece la principal ventaja de resumir el trabajo de los desarrolladores, permitiendo derivar un aggregator particular fácilmente y en poco tiempo, focalizando el esfuerzo en tareas propias del dominio particular, sin invertir tiempo en abstraer el comportamiento común de los aggregators para obtener una solución genérica. De esto último se ocupa el framework. El desarrollador de un aggregator RSS particular, derivado del framework, debe preocuparse solamente por definir qué canales debe consultar, y para cada canal, qué datos XML interesan y cómo mapearlos a objetos Java. El tratamiento de la información obtenida de los canales RSS mediante el aggregator es específico de cada problema, quedando fuera del alcance del framework. Para esto último, el desarrollador registra el proceso que trata la información tomada de cada canal contra el aggregator particular. Para mas detalles ver *Pasos para derivar un aggregator particular* del capítulo anterior, *El framework para aggregators RSS*.

Anexo I - La Sindicación

La forma de las relaciones del contenido y el negocio sobre la Web está atada al viejo concepto. La sindicación (*syndication*), extraída del mundo cerrado del medio de comunicación tradicional, puede ser el modelo que permita a la Web permanecer abierta. Los convenios de sindicación constituyen el alma de las industrias de la radiodifusión, cable y periódicos. En tales arreglos, las entidades que crean contenidos los licencian a los distribuidores, quienes los integran junto con sus otros ofrecimientos. La sindicación en línea excepcionalmente toma el camino más corto a través de los idiomas de contenido, comercio y computación. Si bien comúnmente es visto como un artefacto de medios de comunicación tradicionales y pasivos, la sindicación encaja perfectamente con la fluidez e interactividad de la Web.

La sindicación y la Web

Muy pronto, la sindicación será un tema central para el desarrollo de la mayoría de los negocios en Internet. Al mismo tiempo, es el futuro modelo de millones de sitios Web independientes y personales que le otorgan vitalidad a Internet. La sindicación posibilita a los sitios extender su presencia hacia sus clientes, y brinda a estos clientes herramientas para añadir la información y las funciones que ellos quieren ver.

La sindicación funciona muy bien en línea porque todo toma la forma de información. En el mundo físico, la sindicación implica muchísima impresión, ensamblado y conducción de carreteles de video. Sobre la Web, como la transferencia de contenido es más simple, las relaciones pueden ser más complejas. Sumar a esto la habilidad de armar información dinámicamente o incluso ejecutar aplicaciones con derechos y privilegios asignados entre varias partes, la cuestión cobra mucho interés. No existe una demarcación estricta entre contenido en línea y salidas de aplicaciones basadas en Web. Estas últimas pueden tener la habilidad de tomar como entrada los perfiles personales de los clientes y generar de acuerdo a estas configuraciones diferentes presentaciones (formatos) del mismo contenido.

Los modelos de sindicación fuera de línea están limitados a los radiodifusores y periódicos, que consecuentemente, pueden extenderse para comerciar transacciones e incluso procesos de negocio centrales sobre Internet.

El valor de la sindicación

La sindicación provee varios beneficios. Fundamentalmente, facilita la especialización y diversificación funcional. Sin la sindicación, cualquier originador de contenidos debe también encontrar la manera de distribuir dicho contenido a los usuarios finales, o debe establecer una relación exclusiva con un distribuidor de contenidos. Sobre el otro extremo del espectro, los sitios destino y los aggregators deberían invertir tiempo y dinero para generar contenido original de alta calidad.

Invocando esta des-intermediación no significa que necesariamente todo funcione para las compañías o sus clientes. Algunos intermediarios agregan fricción, pero otros hacen que el mercado sea más fluido y eficiente. En un mundo sindicado, las compañías e individuos pueden elegir donde concentrar sus esfuerzos. El resultado es un rico ecosistema con varios nichos potencialmente provechosos.

Desde la perspectiva de negocio, la sindicación permite a los originadores y sindicadores aglomerar una gran cantidad de transacciones de poco valor dentro de un negocio beneficioso. Los originadores de contenidos pueden generar un ingreso incremental (por retribuciones de suscripción o desde la publicidad) con un pequeño o ningún costo adicional mediante la sindicación de materiales a través de un gran número de sitios Web. La sindicación posibilita a los originadores expandir su alcance y acelerar el tiempo de distribución.

Escalada, escasez y sindicación

La evolución de la sindicación en línea, al igual que cualquier otro desarrollo mayor sobre Internet, está ligado a dos factores: la escalada y la escasez.

Los servicios basados en Web deben ser capaces de soportar un crecimiento de escala masivo y veloz, porque inclusive el pedazo de información más aislado está potencialmente disponible para una audiencia de decena de millones.

La restricción sobre la escalada es la escasez. Un sistema puede sólo crecer tan rápido como su link más débil. Internet escala tan rápidamente porque elimina elementos de escasez del mundo físico, tales como el tiempo y el costo de distribución de información a los clientes.

Los modelos de negocio para la sindicación fuera de línea asumen escasez en varios puntos. Los originadores de contenidos impresos y para difusión de radio o TV, típicamente firman tratos exclusivos con sindicadores, y estos sindicadores tienen tan sólo un número limitado de puestos de distribución (canales de TV, estaciones de radio, diarios, etc.) para divulgar los contenidos. Internet elimina esos puntos de escasez. No únicamente la Web carece de centro, sino que cualquier sitio potencialmente puede convertirse en una mini-Web de recursos estrechamente integrados de múltiples fuentes.

Orden emergente del caos

Las redes de sindicación son una forma de orden emergente. Existe un volumen abrumador de información en la Web, por consiguiente los consumidores no pueden encontrar por sus propios medios todo el contenido que los creadores y originadores puedan interesarles. Se torna muy difícil para las personas encontrar las cosas que buscan, y para aquellos que ofrecen cosas encontrar la gente que las quiera.

Una solución para el caos de la Web es crear nuevos centros en torno de los cuales el resto gire. Los portales toman esta oportunidad. La sindicación puede proveer una alternativa de auto-organización. Al igual que las redes neuronales del cerebro, las Webs de sindicación abarcan actores interactivos pero independientes que no necesitan del otro para soportar transacciones finales.

Medios e intermediarios

La tecnología realmente ha posibilitado que la distancia entre creadores y consumidores crezca a lo largo del tiempo. Antes de la revolución industrial los artesanos vendían sus artículos directamente. Luego los mayoristas y minoristas vinieron.

Los canales de distribución de múltiples niveles en comercio y los contratos de sindicación en broadcasting, ambos imponen todavía más capas, pero funcionan porque han creado valor para cada uno involucrado. Esta tendencia continúa en línea. Incluso los fabricantes emplean Internet para tocar a los consumidores directamente, los portales se interponen entre los minoristas y sus consumidores.

Con la sindicación, cualquier información puede estar en cualquier parte, porque el vínculo entre la creación y la distribución está roto. Existirán muchos caminos posibles entre las compañías y sus audiencias. Muchos de estos caminos existirán simultáneamente. La gran oportunidad para la tecnología y para los proveedores de servicios es tomar ventaja de la mejor cadena de distribución para un cliente dado en un determinado momento.

Anexo II - Módulos RSS 1.0

Módulos RSS estándar

A continuación se presentan los únicos tres módulos RSS estándar, al momento de ser redactado este documento.

Módulo: Dublin Core

La declaración del espacio de nombres es:

xmlns:dc="http://purl.org/dc/elements/1.1/"

El Dublin Core Metadata Element Set provee algunos elementos estándar de metadatos, como por ejemplo, los siguientes:

- <dc:title> (#PCDATA)
- <dc:creator> (#PCDATA)
- <dc:subject> (#PCDATA)
- <dc:description> (#PCDATA)
- <dc:publisher> (#PCDATA)
- <dc:contributor> (#PCDATA)
- <dc:date> (#PCDATA)
- <dc:type> (#PCDATA)
- <dc:format> (#PCDATA)
- <dc:identifier> (#PCDATA)
- <dc:source> (#PCDATA)
- <dc:language> (#PCDATA)
- <dc:relation> (#PCDATA)
- <dc:coverage> (#PCDATA)
- <dc:rights> (#PCDATA)

Todos pueden ocurrir como sub-elementos de <channel>, <item>, <image> y <textinput>.

Módulo: Syndication

La declaración del espacio de nombres es:

xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"

Provee indicaciones de sindicación a los aggregators y a otros recolectores de datos RSS, considerando con qué frecuencia son actualizados.

Define tres elementos, todos sub-elementos de channel:

- **<sy:updatePeriod>** ('hourly' | 'daily' | 'weekly' | 'monthly' | 'yearly')

  Describe el período sobre el cual el formato del canal es actualizado. Si es omitido se asume el valor 'daily' (diario).

- **<sy:updateFrequency>** (entero positivo)

  Utilizado para describir la frecuencia de las actualizaciones en relación con *updatePeriod*. Un entero positivo indica cuantas veces en ese período es actualizado el canal. Si es omitido se asume el valor de 1.

- **<sy:updateBase>** (#PCDATA)

  Define la fecha base a ser empleada en combinación con *updatePeriod* y *updateFrequency* para calcular el cronograma de publicación. El formato de la fecha toma la forma: yyyy-mm-ddThh:mm.

Módulo: Content

La declaración del espacio de nombres es:

xmlns:content="http://purl.org/rss/1.0/modules/content/"

Es un módulo para el contenido existente en los sitios Web, con sus múltiples formatos.

Módulos RSS propuestos

La siguiente es una lista de los módulos clasificados como *Proposed*, al momento de ser redactado este documento.

Módulo: Administrative

La declaración del espacio de nombres es:

xmlns:admin="http://webns.net/mvcb/"

Este módulo provee propiedades administrativas que pueden ser utilizadas para mejorar la robustez y fiabilidad del amplio consumo de RSS entre proveedores, aggregators, clientes y otros usuarios.

Todos sub-elementos de channel:

- **<admin:errorReportsTo rdf:resource="URI"/>**

  Es una URI (típicamente una URL 'mailto:') para contactar la persona o fuente de la instancia particular de información RSS/RDF. Generalmente se utilizaría para reportar errores técnicos en el uso de RSS, RDF ó XML.

- **<admin:generatorAgent rdf:resource="URI"/>**

  Es la URI del software que fue utilizado para generar el RSS/RDF. Es similar al campo *User-Agent* en HTTP o el campo *X-Mailer* en email. Puede ser usado para reportar errores en el software generador y para información de estadísticas. La URI debe incluir la información de versión, y si es una URL resoluble debe retornar una página HTML legible para el usuario.

Módulo: Aggregation

La declaración del espacio de nombres es:

xmlns:ag="http://purl.org/rss/modules/aggregation/"

Proporciona información adicional para los aggregators sobre la fuente original del ítem RSS.

Todos sub-elementos de item:

- **<ag:source>** (#PCDATA)
- **<ag:sourceURL>** (#PCDATA)
- **<ag:timestamp>** (#PCDATA) [ISO 8601]

Módulo: Annotation

La declaración del espacio de nombres es:

xmlns:annotate="http://purl.org/rss/1.0/modules/annotate/"

Provee soporte para recursos que comentan, reiteran o referencian a otros recursos. Los ejemplos incluyen a sistemas de acotación como *crit*, *Usenet* y la mayoría de los grupos de debate basados en Web. El módulo puede ser utilizado en combinación con otros módulos (como Dublin Core) para proporcionar información adicional sobre los recursos.

Sub-elemento de item:

<annotate:reference> (rdf:resource)

Módulo: mod_audio

La declaración del espacio de nombres es:

xmlns:audio="http://media.tangent.org/rss/1.0/"

Para módulos tales como el *mod_mp3* para el servidor Apache y para sindicar datos de audio en forma apropiada, se necesitan campos adicionales. mod_audio agrega esos campos que se necesitan. Este módulo RSS extiende la sintaxis (y semántica) con elementos, a nivel de ítem, sobre metadatos específicos de los datos de audio.

Todos sub-elementos de item:

- **<audio:songname>** (#PCDATA)
- **<audio:artist>** (#PCDATA)
- **<audio:album>** (#PCDATA)
- **<audio:year>** (#PCDATA)
- **<audio:comment>** (#PCDATA)
- **<audio:genre>** (#PCDATA)
- **<audio:recording_time>** (#PCDATA)
- **<audio:bitrate>** (#PCDATA)
- **<audio:track>** (entero positivo)
- **<audio:genre_id>** (entero positivo)
- **<audio:price>** (#PCDATA)

Módulo: mod_cc

La declaración del espacio de nombres es:

xmlns:cc="http://web.resource.org/cc/"

Este módulo apunta a brindar metadatos relativos a la licencia de los derechos de autor (copyright) bajo la cual el documento RSS, y los objetos a los que apunta, están sujetos. Como el nombre sugiere, se origina del proyecto *Creative Commons*, pero no está restringido a las licencias que produce. Por el contrario, está diseñado para permitir la inclusión de cualquier licencia existente y futura.

Este módulo se destaca por ser el primer módulo RSS 1.0 que utiliza completamente la sintaxis RDF. Requiere la inclusión de otro elemento al mismo nivel que `<channel>` e `<item>` dentro del documento.

- **<cc:license rdf:resource="URI OF LICENSE" />**

  Puede ser un sub-elemento de `<item>`, `<channel>` ó `<image>`. La licencia a la que se refiere es aplicada al elemento padre. Como en cualquier elemento RDF, la ocurrencia de `<cc:license>` dentro de cualquier elemento implica el copyright del documento contenido en el atributo `rdf:about` de ese elemento padre, y no el documento apuntado por el sub-elemento `<link>`. El elemento contiene un solo atributo rdf:resource que lleva la URI de la licencia aplicada.

- **<cc:License rdf:about="URI OF LICENSE">**

  Es un elemento que aparece sobre el primer nivel del documento RSS (el mismo nivel de los elementos `<channel>` e `<item>`). Se requiere que este elemento aparezca una única vez por cada licencia referida por el elemento `<cc:licence rdf:resource="">`. Necesita una combinación de dos elementos `<cc:permits>` y `<cc:requires>`.

- **<cc:permits rdf:resource="URI OF BEHAVIOUR" />**

  Es un sub-elemento del elemento `<cc:License rdf:about="">`. Lleva un atributo `rdf:resource` que debe contener la URI de un concepto de comportamiento que la licencia permita. Actualmente, *Creative Commons* sugiere tres posibles valores:

  ```
  <cc:permits
  rdf:resource="http://web.resource.org/cc/Reproduction"/>
  ```

  El trabajo puede reproducirse.

  ```
  <cc:permits
  rdf:resource="http://web.resource.org/cc/Distribution"/>
  ```

El trabajo (y, si es autorizado, los trabajos derivados) pueden ser distribuídos, públicamente exhibidos y públicamente desempeñados.

```
<cc:permits
rdf:resource="http://web.resource.org/cc/DerivativeWorks"/>
```

Trabajos derivados pueden ser creados y reproducidos.

Se debe notar que este módulo no se limita a los estándares de *Creative Commons*. El elemento `<cc:permits>` puede referir a conceptos de cualquier autoridad relevante.

- **<cc:requires rdf:resource="URI OF BEHAVIOUR" />**

Es un sub-elemento de elemento `<cc:License rdf:about="">`. Lleva un atributo `rdf:resource` que contiene una URI de un concepto de comportamiento que la licencia requiere. Actualmente, *Creative Commons*, sugiere cuatro posibles valores:

```
<cc:requires
rdf:resource="http://web.resource.org/cc/Notice"/>
```

La información de copyright y licencia debe quedar intacta.

```
<cc:requires
rdf:resource="http://web.resource.org/cc/Attribution"/>
```

Se debe otorgar un crédito al titular del copyright y/o autor.

```
<cc:requires
rdf:resource="http://web.resource.org/cc/Copyleft"/> -
```

Los trabajos derivados deben ser licenciados bajo los mismos términos que el trabajo original.

```
<cc:requires
rdf:resource="http://web.resource.org/cc/Noncommercial"/>
```

Los derechos pueden no ser practicados para propósitos comerciales.

Otra vez notar que este módulo no se limita a los estándares de *Creative Commons*. El elemento `<cc:requires>` puede referir a conceptos de cualquier autoridad relevante.

Módulo: changedPage

La declaración del espacio de nombres es:

xmlns:cp="http://my.theinfo.org/changed/1.0/rss/"

Incluye información necesaria para suscribirse y así ser notificado cuando este canal RSS sufre alguna actualización.

Las aplicaciones deben seguir la especificación de *changedPage* [http://my.theinfo.org/changed/1.0/] utilizando el elemento `cp:server` como el notificador.

Sub-elemento de channel:

<cp:server> (rdf:resource)

Módulo: mod_company

La declaración del espacio de nombres es:

xmlns:company="http://purl.org/rss/1.0/modules/company"

Se necesitan nuevos campos para la sindicación de noticias y datos de mercados, acompañados por símbolos de ticker (código bursátil de la empresa). Este módulo agrega esos campos necesarios para identificar el símbolo de ticker y el mercado asociado, así también como metadatos sobre la compañía.

Sub-elementos de item:

- **<company:name>** (#PCDATA)
- **<company:symbol>** (#PCDATA)
- **<company:market>** (#PCDATA)
- **<company:category>** (taxo:topic)

Módulo: Qualified Dublin Core

La declaración del espacio de nombres es:

xmlns:dcterms="http://purl.org/dc/terms/"

Este módulo está diseñado para permitir que los metadatos de Qualified Dublin Core puedan ser utilizados en adición a los metadatos de Dublin Core definidos en su módulo correspondiente.

Al momento de crear este documento, este módulo se encuentra en estado Borrador (*Draft*).

Módulo: Email

La declaración del espacio de nombres es:

xmlns:email="http://purl.org/rss/1.0/modules/email/"

Este módulo está diseñado para representar los encabezados de e-mail.

Sub-elementos de item:

- **<email:from>** (#PCDATA)
- **<email:to>** (#PCDATA)
- **<email:subject>** (#PCDATA)
- **<email:date>** (#PCDATA)
- **<email:message-id>** (#PCDATA)
- **<email:sender>** (#PCDATA)
- **<email:reply-to>** (#PCDATA)
- **<email:in-reply-to>** (#PCDATA)
- **<email:references>** (#PCDATA)
- **<email:content-type>** (#PCDATA)
- **<email:content-disposition>** (#PCDATA)
- **<email:mime-version>** (#PCDATA)
- **<email:user-agent>** (#PCDATA)
- **<email:content-type>** (#PCDATA)

Módulo: Event

La declaración del espacio de nombres es:

xmlns:ev="http://purl.org/rss/1.0/modules/event/"

Con RSS se tiene la habilidad de tomar noticias y resúmenes de otros sitios Web y presentarlos en portales. Sin embargo, una propiedad clave de las noticias es que las mismas describen algo que ya pasó. Por lo tanto, la noticia más interesante es la noticia "más nueva", la última.

No es así con eventos tales como conferencias, lanzamientos de productos, cursos de entrenamiento, etc. Los eventos se ubican en una cierta fecha en el futuro, y pueden ser anunciados con años o pocos días de anterioridad. De esta manera, cuando se anuncian, se necesita ordenarlos por la fecha del evento, debido a que los eventos más interesantes son aquellos que ocurrirán en el futuro más cercano.

Este módulo brinda la posibilidad al usuario final de tener una especie de calendario en su sitio Web. Este calendario tiene la capacidad de automáticamente absorber anuncios de eventos (conferencias, IRC chats, lanzamientos de libros, etc.) de otros sitios Web de interés.

Sub-elementos de item:

- **<ev:startdate>** (#PCDATA) [W3CDTF]

  Elemento requerido. Representa la fecha/hora de comienzo del evento. Si este elemento no especifica la zona horaria, entonces ésta última es inferida del elemento `ev:location`. No se permiten intervalos de tiempo.

- **<ev:enddate>** (#PCDATA) [W3CDTF]

  La fecha/hora de finalización del evento. Si este elemento es idéntico al elemento `ev:startdate` o no es especificado, entonces el evento no tiene duración. Este elemento puede tener una zona horaria diferente a la del elemento `ev:startdate`. No se permiten intervalos de tiempo.

- **<ev:location>** (#PCDATA)

  El lugar del evento. El contenido es una descripción breve. Utilizar semántica aumentada si se desea proveer datos adicionales, como por ejemplo la URL del lugar.

- **<ev:organizer>** (#PCDATA)

  El nombre de la organización o persona que organiza el evento. Utilizar semántica aumentada si se desea proveer datos adicionales, como el número de teléfono del organizador.

- **<ev:type>** (#PCDATA)

  El tipo del evento, por ejemplo: conferencia, lanzamiento, reunión de proyecto. El propósito es promover o filtrar ciertos tipos de eventos que el usuario tiene o no interés.

Módulo: Link

La declaración del espacio de nombres es:

xmlns:l="http://purl.org/rss/1.0/modules/link/"

Este módulo soporta la sindicación de información de link de sitios junto con el suministro de datos de RSS 1.0. El mecanismo de link del W3C HTML proporcionó la inspiración original para este módulo.

La información de link puede ser utilizada para:

- Versiones de links alternativas de un canal o ítems RSS, basados en tipo de medio, lenguaje o parentesco.

- Provee enlace a la fuente de contenidos, tópico y descripciones de servicios Web (WSDL).
- Acoplamiento a información RDF adicional para un canal.
- Soporte de links específicos de la aplicación y proveedor. Esto puede ser usado para construir vínculos basados en el código específico de cliente sin quebrar ninguna aplicación basada en RSS 1.0 existente.
- Además, los links pueden ser empleados para proveer funcionalidades que no están normalmente disponibles si RDF/XML fuese incluido en línea con el RSS original.

Los links son implementados como aristas direccionales desde el canal RSS actual hacia un recurso externo. Las relaciones de link son provistas de forma que las aplicaciones puedan determinar cómo un determinado link debe ser tratado.

Este módulo define las siguientes relaciones de link:

- **print** - http://purl.org/rss/1.0/modules/proposed/link/#print

  Versión imprimible del ítem RSS actual. Para documentos HTML, ésto puede ser una URL sin ítems de navegación ni otras opciones que pueden empañar el documento cuando el mismo es enviado a la impresora.

- **permalink** - http://purl.org/rss/1.0/modules/proposed/link/#permalink

  Versión permalink del ítem RSS actual. Un permalink está definido como una URL para un recurso que está siempre disponible (nunca se invalidan a pesar del paso del tiempo).

- **service** - http://purl.org/rss/1.0/modules/proposed/link/#service

  Link a un servicio o archivo de descripción de servicio (WSDL). Puede ser utilizado para el descubrimiento en tiempo de ejecución de Web Services dentro de un archivo RSS.

- **source** - http://purl.org/rss/1.0/modules/proposed/link/#source

  Incluye un vínculo a la fuente del ítem RSS. Esto puede ser usado para la correcta atribución de la versión original del ítem RSS.

- **topic** - http://purl.org/rss/1.0/modules/proposed/link/#topic

  Proporciona un link al actual canal o ítem RSS por tópico. Este es generalmente un archivo HTML o RSS que contiene ítems RSS con el mismo tópico que el ítem actual.

- **alternate** - http://purl.org/rss/1.0/modules/proposed/link/#alternate

  Provee versiones alternativas por lenguaje, tópico, etc. de un ítem o canal.

Sub-elemento de channel o item:

- **<l:link>**

  Provee la URL del target de este link.

- **<l:type>**

  Brinda una indicación del tipo de contenido del contenido disponible en la dirección target del link.

- **<l:title>**

  Un título legible para humanos de este link.

- **<l:rel>**

  Especifica la relación del documento.

- **<l:lang>**

  El código de lenguaje para el medio sobre un target específico.

- **<l:charset>**

  Especifica la codificación de caracteres del recurso designado por el link.

Módulo: RSS091

La declaración del espacio de nombres es:

xmns:rss091="http://purl.org/rss/1.0/modules/rss091#"

Este módulo provee compatibilidad lateral con RSS 0.91 proporcionando una versión modularizada de los elementos incorporados a la versión 0.91.

Sub-elementos de channel:

- **<rss091:language>** (#PCDATA) [LANG]
- **<rss091:rating>** (#PCDATA) [PICS]
- **<rss091:managingEditor>** (#PCDATA)
- **<rss091:webmaster>** (#PCDATA)
- **<rss091:pubDate>** (#PCDATA) [RFC 822]
- **<rss091:lastBuildDate>** (#PCDATA) [RFC 822]
- **<rss091:copyright>** (#PCDATA)
- **<rss091:skipHours rdf:parseType="Literal">** (hour+)
- **<rss091:hour>** ('0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' | '19' | '20' | '21' | '22' | '23')

- **<rss091:skipDays rdf:parseType="Literal">** (day+)
- **<rss091:day>** ('Monday' | 'Tuesday' | 'Wednesday' | 'Thursday' | 'Friday' | 'Saturday' | 'Sunday')

Sub-elementos de image:

- **<rss091:width>** (#PCDATA)
- **<rss091:height>** (#PCDATA)

Sub-elementos de item:

- **<rss091:description>** (#PCDATA)

Módulo: Search

La declaración del espacio de nombres es:

xmlns:search="http://purl.org/rss/1.0/modules/search/"

Proporciona información adicional sobre los ítems resultantes de la consulta a un motor de búsqueda.

Sub-elementos de item:

- **<search:relevance>** (#PCDATA)
- **<search:scope>** (#PCDATA)

Módulo: Service Status

La declaración del espacio de nombres es:

xmlns:ss="http://purl.org/rss/1.0/modules/servicestatus/"

Este módulo extiende a RSS para incluir elementos que permiten la descripción del estado y actual disponibilidad de servicios y servidores.

Algunos datos, tal como si el servidor está levantado o no, normalmente serán generados automáticamente, en tanto que otros datos, tal como el texto explicativo para humanos, pueden ser el resultado de procesamiento adicional o de una entrada de datos directa por humanos. Un servicio, visto desde el punto de vista de un usuario puede no tener una correspondencia uno a uno con un servicio visto desde el punto de vista del sistema. Por ejemplo, un servicio de usuario dependiente de más de un servicio de sistema está disponible, si y solo sí, todos los servicios de sistema están disponibles. Esto puede ser fácilmente calculado con la entrada y salida conforme a esta especificación.

Sub-elementos de channel:

- **<ss:aboutStats>** (rdf:resource)

  Debe brindar acceso a una descripción de cómo `ss:availability` y `ss:averageResponseTime` fueron calculados, y de esta forma dar su significado.

Sub-elementos de item:

- **<ss:lastChecked>** (#PCDATA) [W3CDTF]

  Provee el tiempo más reciente que el servicio fue probado.

- **<ss:lastSeen>** (#PCDATA) [W3CDTF]

  Provee el tiempo más reciente que el servicio fue respondido.

- **<ss:responding>** (true | false) [boolean]

  Es verdadero o falso, dependiendo si el servicio respondió al último chequeo. En el caso de que el servicio está respondiendo los dos tiempos (`ss:lastChecked` y `ss:lastSeen`) deben ser iguales. Si el servicio no está respondiendo, entonces es útil tener alguna aproximación de cuánto tiempo estuvo caído, mediante la comparación de estos dos tiempos.

- **<ss:availability>** (integer) [percentage]

  Es una estadística que puede ser calculada como se desee.

- **<ss:averageResponseTime>** (float) [seconds]

  Idem a `ss:availability`.

- **<ss:statusMessage>** (#PCDATA)

  Contiene un mensaje sobre el servicio y su estado. Usualmente no será necesario. Sin embargo mensajes como “existe un problema intermitente que estamos monitoreando” o “el servicio se re-arrancará durante el almuerzo” pueden ser apropiados en ciertos casos. Un mensaje puede ser incluido independientemente si el servicio está actualmente respondiendo o no.

Módulo: Slash

La declaración del espacio de nombres es:

xmlns:slash="http://purl.org/rss/1.0/modules/slash/"

Slash es el código fuente y base de datos que fue originalmente usado para crear Slashdot, y ahora ha sido liberado con la Licencia Pública General GNU. Es un proyecto *Open Source/Free Software*.

Este módulo aumenta el núcleo de RSS y el módulo de metadatos de Dublin Core con elementos a nivel de canal e ítem específicos de sitios basados en Slash.

Sub-elementos de item:

- <**slash:section**> (#PCDATA)
- **<slash:department>** (#PCDATA)
- **<slash:comments>** (entero positivo)
- **<slash:hit_parade>** (enteros separados por comas)

Módulo: Streaming

La declaración del espacio de nombres es:

xmlns:str="http://hacks.benhammersley.com/rss/streaming/"

Este módulo satisface las necesidades adicionales de los proveedores de streaming-media. Es visto como un suplemento a módulos estándar existentes y propuestos, especialmente Dublin Core.

Las principales características involucran a las aplicaciones asociadas al medio stream, el codec del stream es codificado con tags adicionales para la segmentación de broadcasts vivo/continuo.

Es información predominantemente técnica/práctica: conceptualiza información tal como estilo de música, contenido de video, contribuyentes y manejo de derechos para ser tratado por Dublin Core, etc.

Sub-elementos de item y channel:

- **<str: type>** ('audio' | 'video' | 'both')

  Puede ser usado para describir cada ítem o el canal entero. 'video' implica un video con una pista de sonido. 'both' implica una mezcla de ítems de video y audio, y por lo tanto es usado sólo dentro de una descripción de canal.

- **<str: associatedApplication>** (#PCDATA)
- **<str: associatedApplication.version>** (#PCDATA)
- **<str: associatedApplication.downloadUri>** (#PCDATA)
- **<str: codec>** (#PCDATA)
- **<str: codec.name>** (#PCDATA)
- **<str: codec.version>** (#PCDATA)
- **<str: codec.url>** (#PCDATA)

Una página que describe el codec

- **<str: codec.downloadUri>** (#PCDATA)

  La URI para obtener el codec.

- **<str: codec.sampleRate>** (#PCDATA)

  Un valor en kHz de velocidad de muestreo de audio.

- **<str: codec.stereo>** ('stereo' | 'mono')
- **<str: codec.ResolutionX>** (#PCDATA)

  Un valor en pixels de ancho en el eje X.

- **<str: codec.ResolutionY>** (#PCDATA)

  Un valor en pixels de ancho en el eje Y.

- **<str: duration>** (#PCDATA) HH:MM:SS
- **<str: live>** ('live' | 'recorded')
- **<str: live.scheduledStartTime>** (#PCDATA)

  W3CDTF para broadcasts en vivo, hh:mm:ss para lugares en un código de tiempo para grabaciones.

- **<str: live.scheduledEndTime>** (#PCDATA)

  W3CDTF / hh:mm:ss como arriba.

- **<str: live.location>** (#PCDATA)

  Un string literal, como las líneas directivas de Dublin Core para locaciones, o puede extenderse con RDF para usar espacios de nombres adicionales específicos de locaciones.

- **<str: live.contactUri>** (#PCDATA)

  Por ejemplo: mailto:, http:, aim:, irc:.

Módulo: Subscription

La declaración del espacio de nombres es:

xmlns:sub="http://purl.org/rss/1.0/modules/subscription/"

Este módulo soporta la sindicación de información de suscripción a sitios junto con datos RSS 1.0. Esta información puede ser usada para facilitar un número de características:

- Permitir a agentes de usuario presentar una lista en frente del lector y así el mismo poder navegar hacia otros canales RSS y sitios Web.
- Capacitar a aggregators RSS para explícitamente descubrir comunidades Web entre grupos de canales RSS.
- Simplificar a robots RSS el descubrimiento de datos RSS buscando aggregators.
- Facultar a los algoritmos de búsqueda extensibles en P2P, los cuales pueden usar los bordes direccionales expuestos por este módulo para navegar a través de canales RSS.

Existen dos roles principales en este módulo. Cuando el mismo es utilizado en un Weblog personal, esto es esencialmente suscripciones de usuarios. Y cuando se sindica desde un sitio corporativo (profesional), esto es esencialmente recomendaciones de formularios de sitios externos.

Módulo: Taxonomy

La declaración del espacio de nombres es:

xmlns:taxo="http://purl.org/rss/1.0/modules/taxonomy/"

Proporciona la habilidad de clasificar canales e ítems bajo uno o más esquemas taxonómicos.

- **<taxo:topic>** (taxo:link, taxo:topics), atributo: rdf:about

  Elemento de primer nivel. Opcionalmente utilizado para definir un tópico.

- **<taxo:topics>** (rdf:Bag)

  Elemento de segundo nivel (puede ser incluido en channel, item o taxo:topic). Proporciona (usando una estructura rdf:Bag/rdf:li) una lista de tópicos. Estos tópicos pueden o no ser definidos dentro del alcance del canal corriente. La ubicación de los documentos que describen los tópicos puede opcionalmente ser proporcionada utilizando un módulo genérico de inclusión a ser definido.

Módulo: Threading

La declaración del espacio de nombres es:

xmlns:thr="http://purl.org/rss/1.0/modules/threading/"

Proporciona la habilidad para los ítems de especificar relaciones padre/hijo.

Sub-elemento de item:

<thr:children> (rdf:Seq)

Bibliografía

- Extensible Markup Language (XML) 1.0, W3C Recommendation, 6 October 2000. http://www.w3.org/TR/REC-xml
- Namespaces in XML, W3C Recommendation, 14 January 1999. http://www.w3.org/TR/REC-xml-names
- XML Schema Part 0: Primer. W3C Recommendation, 2 May 2001. http://www.w3.org/TR/xmlschema-0/
- Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, 22 February 1999. http://www.w3.org/TR/REC-rdf-syntax
- RDF Site Summary (RSS) 1.0 Specification. 2001-05-30. http://purl.org/rss/1.0/spec
- RDF Site Summary 1.0 Modules. 2001-03-20. http://purl.org/rss/1.0/modules/
- Rael Dornfest. Writing RSS 1.0. 2000-08-25. O'Reilly Network, www.oreillynet.com
- Rael Dornfest. RSS: Lightweight Web Syndication. 2000-07-17. XML.com
- RSS Syndication and Aggregation. 2000-03-28. WebReference.com
- Jonathan Eisenzopf. Making Headlines with RSS. 2000-02. www.webtechniques.com
- Dave Winer. A Bright Future for Syndication. 1999-09. DaveNet, davenet.userland.com
- Mark Nottingham. RSS Tutorial for Content Publishers and Webmasters. 2002-10-21. http://www.mnot.net/rss/tutorial/
- Kevin Werbach. The Web Goes Into Syndication. 1999-07. Release 1.0, http://www.edventure.com/release1/release1.html
- Leigh Dodds. RSS Modularization. 2000-07-05. XML.com
- Edd Dumbill. RSS moves forward. 2000-08-14. O'Reilly Network, www.oreillynet.com
- Edd Dumbill. XML in News Syndication. 2000-07. XML.com
- Bryan Caporlette. XML Portal Content Aggregation. 2000-05-15. XML.com
- Peter Wiggin. RSS Delivers the XML Promise. 1999-10-29. webreview.com
- David Megginson. SAX2-Java The Simple API for XML. 2000-04-19. www.megginson.com
- Mark Johnson. Programming XML in Java. 2000-03. www.javaworld.com
- Dennis M. Sosnoski. XML documents on the run. 2002-02-08. www.javaworld.com
- Robert Hustead. Mapping XML to Java. 2000-08. www.javaworld.com
- Andrew King. The Evolution of RSS. 2001-05-14. WebReference.com
- Megginson Technologies Ltd. Specifications. 2000. XMLNews.org
- Norman Walsh. A Technical Introduction to XML. 2000. XML.com
- Norman Walsh. Understanding XML Schemas. 1999-07. XML.com
- Mark Walter. Namespaces in XML Adopted by W3C. 1999-01. XML.com
- Eric Miller. An Introduction to the Resource Description Framework. 1998-05. D-Lib Magazine. ISSN 1082-9873