REPRESENTACION DE ONTOLOGIAS EN LA WEB SEMANTICA

Lydia Silva Muñoz
Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
lydia@inf.ufrgs.br

1 Introducción

Las ontologías proveen una comprensión *compartida y cocensuada* del conocimiento de un dominio que puede ser comunicada entre personas y sistemas heterogéneos. Fueron desarrolladas en el área de Inteligencia Artificial (IA) para facilitar el intercambio y reuso del conocimiento.

Actualmente, los computadores han dejado de ser dispositivos aislados y se han convertido en puntos de entrada en la red mundial de intercambio de información y transacciones de negocio. Se ha vuelto un asunto clave contar con apoyo en el intercambo de datos, información y conocimiento

Emplear el poder de razonamiento automático para guiar el acceso a orígenes de información requiere de representación de la semántica de tales orígenes de manera procesable por computador, en consecuencia, se requieren metadatos que describan de una manera computable, dichos orígenes.

En este trabajo se hace un estudio parcial de los avances realizados hasta el momento por agregar semántica a la web, como lo son la introducción de XML, XML Schemas, RDF y RDF Schemas, y reconociendo el aporte que las ontologías pueden brindar en tal sentido se focaliza en determinar la posibilidad de representarlas con estos lenguajes. Para aportar una referencia en ontologías, se describe brevemente el modelo ontológico OIL, concebido para la Web.

Algunos problemas surgen cuando se trata con grandes cantidades de información semiestructurada:

- Los actuales buscadores basados en palabras clave suelen devolver información irrelevante que usa una cierta palabra con un significado diferente del que se pretende en la búsqueda, y pierden información cuando no reconocen palabras diferentes pero con el mismo significado que la buscada.
- Actualmente se requiere lectura humana para extraer información relevante de un origen, desde que agentes automáticos no tienen el sentido común requerido para reconocer dicha información en representación textual.
- Mantener orígenes textuales débilmente estructurados representa una tarea dificultosa, y consumidora de tiempo cuando tales orígenes aumentan considerablemente de tamaño. Mantener esas colecciones consistentes y al dia requiere de representación interpretable por computador, de semántica que ayude a detectar anomalías automáticamente.
- La utilidad de sitios web adaptativos que permitan su reconfiguración dinámica de acuerdo al perfil del usuario u otros aspectos relevantes, requiere una representación computable de la semántica involucrada.

En general, se dispone de dos estrategias complementarias para resolver estos problemas.
Se pueden enriquecer los orígenes de información de manera declarativa, con comentarios (*annotations*) que provean de su semántica de manera interpretable para un computador, y se pueden escribir programas tales como filtros, wrappers y programas de extracción que proceduralmente extraigan la semántica de los recursos web.
En el presente trabajo se hace énfasis en la primera de las estrategias.

En la sección 2 se describen brevemente XML, y RDF, la sección 3 presenta un modelo de ontología diseñado para ser usado en el intercambio de información en la web: OIL y en la sección 4 se presentan las posibilidades encontradas para representar ontologías en XML-DTD, XML Schema y RDF Schema, finalmente en la sección 5 se hace un resumen de las posibilidades encontradas.

2 Lenguajes para la Web

Recomendados como estándares por la W3C, o candidatos a serlo, algunos de las cuales han ganado mucha popularidad, se describen a continuación XML, XML Schema, RDF y RDF Schema.

2.1 XML

Uno de los resultados del un empuje general hacia una estrucura mas semántica en la Web, fue el desarrollo del lenguaje de demarcación XML, que permite que los creadores de páginas web, usen su propio conjunto de etiquetas de demarcación (markup-tags). Esas etiquetas pueden ser elegidas

de manera tal que reflejen la semántica específica del dominio tratado, en lugar de ocuparse meramente de la posición y formato de la información que comprenden.
Ejemplo:

```
<BODY>
  Esta página fue escrita por
  <AUTOR>  Pedro  </AUTOR>
    <DIRECCION>
      <CALLE>  García de Zúñiga  </CALLE>
      <NUMERO>  1234  </NUMERO>
       <APTO>  505  </APTO>
       <TEL> 34567 </TEL>
    </DIRECCION>
</BODY>
```

Mientras <AUTOR> y </AUTOR> son marcas que denotan comienzo y fin del dato almacenado entre ellas, el string “Pedro” es el dato propiamente dicho.

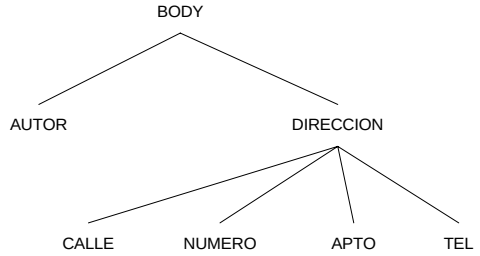

fig. 2.1 estructura del texto XML propuesto como ejemplo

2.2 RDF

XML provee información semántica como un sub-producto de definir la estructura del documento, ya que define una estructura en árbol para un documento de manera que las hojas del mismo contienen la información. Se puede observar entonces que la estructura y la semántica de un documento XML están entrelazadas.
El Resource Description Framework RDF [1] provee un medio de agregar semántica a un documento sin referirse a su estructura. RDF es una aplicación XML recomendada como estándar por la W3C.
El modelo de datos de RDF provee tres tipos de objetos: recursos, propiedades y sentencias

- Un recurso es una entidad que puede ser referenciada por un Identificador Unico de Recursos (URI) .
- Una propiedad define una relación binaria entre recursos y/o valores atómicos de los tipos de datos primitivos provistos por XML.
- Una sentencia especifica un valor en una propiedad para un determinado recurso.

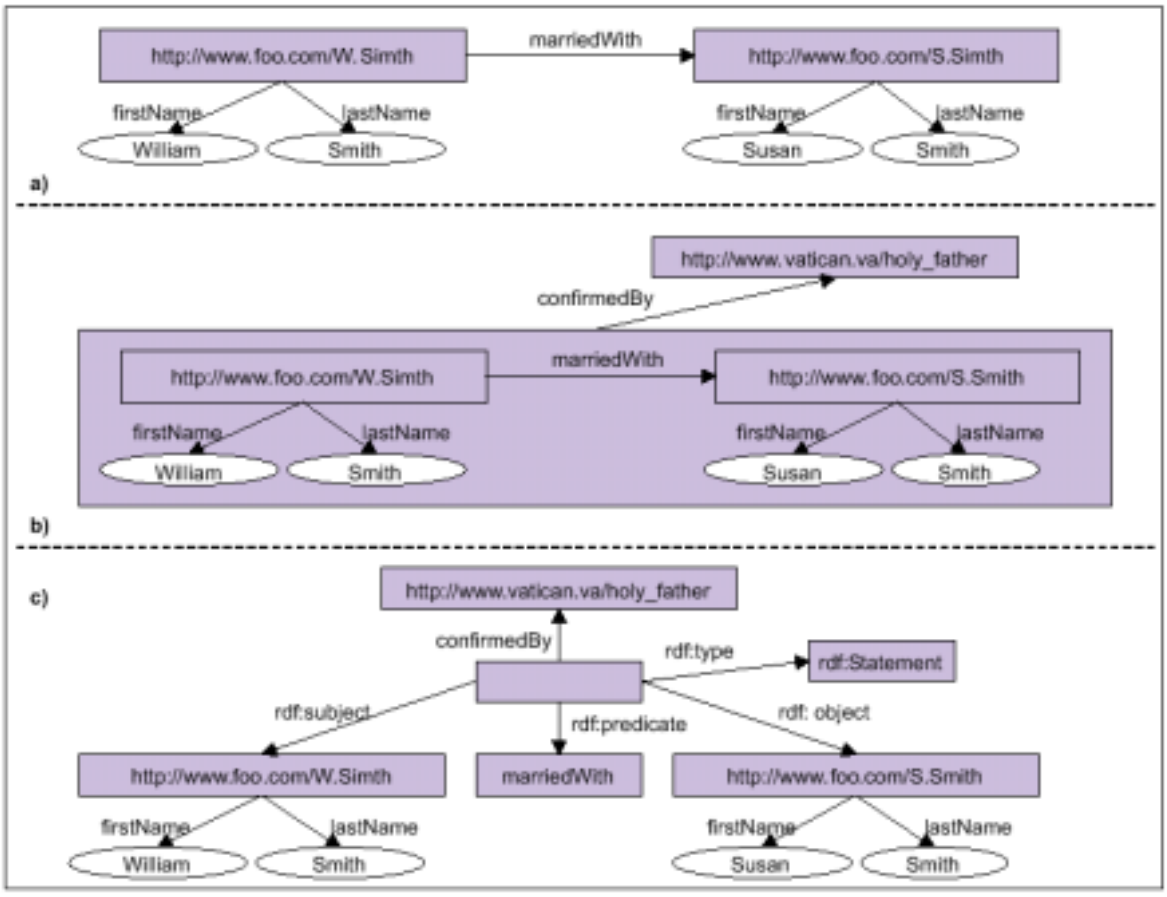

fig. 2.2 Ejemplo de modelo de datos en RDF [5]

Todo lo expresable en RDF, es expresable en sintaxis lineal de XML, podría surgir la pregunta entonces de porque es necesario RDF si todo metadato representado en RDF puede también ser representado en XML. La razón es que RDF provee un modo *estándard* de representar metadatos en XML.Usando directamente XML para representar metadatos, podrían obtenerse varias representaciones diferentes.

En [5] los autores proponen el ejemplo presentado en figura 2.2 donde se han representado recursos en rectángulos sombreados, valores literales en óvalos y propiedades por arcos dirigidos.
En la parte (a) de la figura 2.2 se definen dos recursos, cada uno asociando su propiedad `firstName` y `lastName` con valores literales, identificando los recursos como William y Susan Smith, respectivamente. Esos dos recursos tienen un URI (Uniform Resource Identifier) como su único y global identificador y están relacionados entre ellos por la propiedad `marriedWith`, que dice que William está casado con Susan.
En la parte (b) de la figura se ve un atajo conveniente para expresar una sentencia mas compleja, esto es, reificar una sentencia y definir una propiedad para el nuevo recurso obtenido. El ejemplo denota que el matrimonio entre William y Susan fue confirmado por el recurso que representa el Santo Padre en Roma.
El modelo de datos RDF ofrece el recurso predefinido `rdf: statement` y las propiedades predefinidas `rdf:subject`, `rdf:predicate`, y `rdf:object` para reificar una sentencia como un recurso. El modelo de la parte (b) se muestra en la parte (c) usando estas propiedades y recurso.

En la figura 2.3 se muestra el modelo RDF que corresponde a la siguiente aseveración [1]:
The individual referred to by employee id 85740 is named Ora Lassila and has the email address lassila@w3.org. The resource http://www.w3.org/Home/Lassila was created by this individual

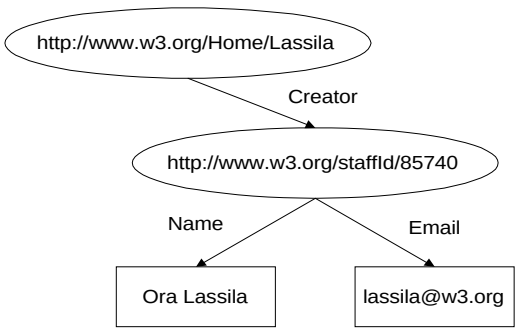

fig. 2.3 representación de un valor estructurado en RDF [1]

Para procesar modelos RDF, éstos deben ser serializados en XML previamente, de esta forma RDF explota la denotación de XML para permitir que diferentes orígenes de información puedan intercambiar el conocimiento que expresa. A continuación se aporta la serialización en XML correspondiente al ejemplo de la figura 2.3.

```
<rdf:RDF>
 <rdf:Description about="http://www.w3.org/Home/Lassila" >
  <s:Creator>
   <rdf:Description about="http://www.w3.org/staffId/85740" >
   <v:Name>Ora Lassila </v:Name>
   <v:Email>lassila@w3.org </v:Email>
   </rdf:Description >
 </s:Creator>
</rdf:RDF>
```

3 OIL: Una Ontología para la Web

Las ontologías son un tópico común de investigación en en varias comunidades, tales como ingeniería del conocimiento, procesamiento de lenguaje natural, sistemas de información cooperativos, integración inteligente de información y gestión del conocimiento, proveen una comprensión *compartida y cocensuada* del conocimiento de un dominio que puede ser comunicada entre personas y sistemas heterogéneos. Fueron desarrolladas en el área de Inteligencia Artificial (IA) para facilitar compartir y reusar el conocimiento.
Las ontologías son teorías formales acerca de un dominio de discurso y por eso requieren de un lenguaje lógico formal para ser expresadas. En el área de IA se han desarrollado muchos lenguajes para este fin. Algunos basados en Lógica de predicados de primer orden, como KIF y Cycl que proveen poderosas primitivas de modelado (en particular CycL con varios cuantificadores) y la posibilidad de reificar fórmulas que les permite convertirse en términos de otras fórmulas. Otros lenguajes son basados en frames, con mas poder expresivo pero menos capacidad de inferencia, como Ontolingua y Frame Logic, otros orientados a ser robustos en el razonamiento que provee Description Logic, como Loom y Classic.

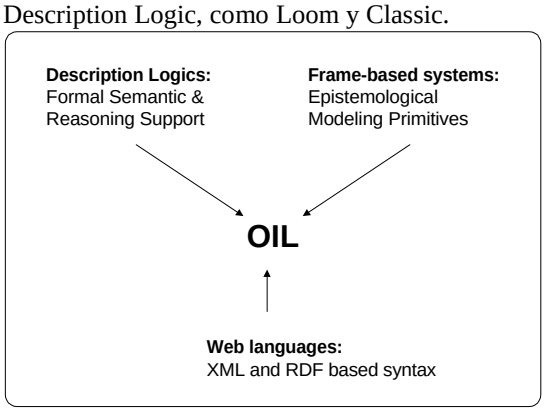

fig. 3.1 Las tres vertientes de lenguajes en OIL

```
class-def animal                                        % animals are a class
class-def plant                                         % plants are a class
        subclass-of NOT animal                          % that is disjoint from animals
class-def tree
        subclass-of plant                               % trees are a type of plants
class-def branch
        slot-constraint is-part-of                      % branches are parts of trees
                has-value tree
class-def leaf
        slot-constraint is-part-of                      %leafs are parts of branches
                has-value branch
class-def defined carnivore                             %carnivores are animals
        subclass-of animal
        slot-constraint eats                            %that eat only other animals
                value-type animal
class-def defined herbivore                             %herbivores are animals
        subclass-of animal
        slot-constraint eats                            %that eat only parts of plants
                value-type plant OR (slot-constraint is-part-of has-value plant)
class-def giraffe                                       %giraffes are animals
        subclass-of animal
        slot-constraint eats                            %that eat leafs
                value-type leaf
class-def lion                                          %lions are animals
        subclass-of animal
        slot-constraint eats                            %that eat herbivores
                value-type herbivore
class-def tasty-plant                                   %tasty plants are plants that are eaten by
        subclass-of plant                               %both herbivores and carnivores
        slot-constraint eaten-by
                value-type herbivore, carnivore
```

fig. 3.2 Un ejemplo de ontología en OIL [10]

En su concepción Oil unifica tres importantes aspectos provistos por diferentes comunidades (ver figura 3.1): la semántica formal y el soporte eficiente para razonamiento provisto por Description Logics, el enriquecimiento epistemológico de las primitivas provistas por los Frames y una propuesta estándar para notación de intercambio propuesta por la comunidad Web [6].

La figura 3.2 muestra un pequeño ejemplo parcial de una ontología expresada en OIL [10]

Una ontología expresada en OIL consiste de una lista de definiciones de clases ("class-def") y definiciones de slots ("slot-def") (las definiciones de slots han sido omitidas). Puede verse el concepto de slot como análogo al de atributo existente en el paradigma de orientación a objetos o modelo entidad relación.
Por claridad, primero se introducirá el concepto de **class-expression**. Una class-expression (expresión de clase) puede ser el nombre de una clase, una *slot-constraint*, o cualquier combinación compleja de expresiones de clase usando los operadores lógicos AND, OR y NOT.
Una definición de clase asocia un nombre de clase con una descripción y consta de los siguientes componentes, (algunos se han omitido por simplicidad):

- El **type** de la definición, que puede ser *primitive* o *defined*, por defecto se asume primitivo. Cuando una definición de clase es de tipo *primitive* las restricciones que delimitan su definición (*slot-constraints*) son consideradas necesarias pero no suficientes en el sentido de la pertenencia de un miembro a dicha clase. En caso de tipo *defined*, las restricciones tienen carácter de necesarias y suficientes. Ej: Si se define la clase primitiva `elefante` como una sub-clase de `animal` con una restricción (*slot-constraint*) estableciendo que su `color` debe ser `gris`, entonces todas las instancias de `elefante` deben ser necesariamente de `color gris`, pero pueden existir animales grises que no sean instancias de `elefante`. Si se define una clase `carnivoro` como *defined*, y como sub-clase de `animal`, con la restricción de que `come carne`, entonces todas las instancias de `carnivoro` serán necesariamente animales que comen carne, pero además todo animal que coma carne, será una instancia de `carnivoro`.
- **subclass-of** - una lista de una o mas expresiones de clase (*class expressions*), de manera que la clase en definición, debe ser una subclase de cada una de las expresiones de la lista.
- **slot-constraint** - una lista de cero o mas (*slot-constraints*), de manera que la clase en definición, debe ser una subclase de cada una de las clases definidas por los *slot-constraints* existentes en la lista.
  Una *slot-constraint* es una lista de restricciones que se aplican a un slot. Un slot es una relación binaria, esto es que sus instancias son pares de individuos, pero también es implícitamente una definición de clase, cuyas instancias son aquellos individuos que satisfacen la restricción establecida.

Los principales componentes de una *slot-constraint* son:

- **name** – un string indicando el nombre del slot
- **has-value** – una lista de una o mas expresiones de clase, de manera que cada instancia de la clase definida por la *slot-constraint* debe estar relacionada por la relación *slot* a una instancia de cada una de las expresiones de clase presentes en la lista.
- **value-type** – una lista de una o mas expresiones de clase. Si una instancia de la clase definida por la slot-constraint está relacionada por la relación slot a algún individuo x, entonces x debe ser una instancia de cada expresión de clase de la lista.

Para ejemplificar que una *slot-constraint* define una clase podemos considerar que si el par (Leo, Gace) es una instancia del slot eats perteneciente a la clase Leon, Leo es una instancia de la clase Leon y Gace es una instancia de la clase Gacela, entonces Leo es también una instancia de la *slot-constraint* `has-value gacela`

El modelado de primitivas y su semántica es uno de los aspectos a tratar en la definición de OIL, respecto de su sintaxis, se atendió a los estándares que actualmente posee la web.
Oil tiene una sintaxis bien definida en XML basada en DTD's y XML Schema. Por otra parte OIL es una extensión de RDF y RDFS. Con respecto a ontologías, RDFS provee dos importantes contribuciones: una sintaxis estandarizada para denotar ontologías y un conjunto estándar de primitivas de modelado tales como las relaciones *instance-of* y *subclass-of*.

3.1 La arquitectura en capas de OIL

Un lenguaje ontológico monolítico probablemente no pueda colmar las necesidades de la diversa clase de usuarios de la web semántica. Esa es la base para organizar OIL en una serie de capas de sublenguajes. Cada capa adiciona funcionalidad y complejidad a su previa. Agentes (humanos o máquinas) que pueden procesar solamente una capa baja, pueden aún parcialmente comprender y beneficiarse de lo expresado por la ontología [8].

Como se ve en la figura 3.3, la capa del núcleo: *core* OIL, coincide ampliamente con RDFS (exceptuando la capacidad de reificación de RDF), esto significa que aún simples RDFS agentes pueden procesar ontologías OIL y levantar tanto de su significado como sea posible de acuerdo al límite de sus capacidades.

Standard OIL tiene como objetivo capturar las primitivas del modelo que proveen un adecuado poder expresivo a la ontología, especificando su semántica y haciendo viable la inferencia.

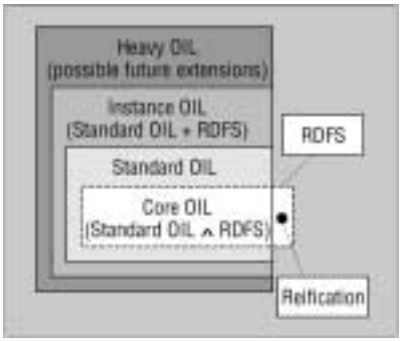

figura 3.3 Modelo de lenguaje en capas de OIL [8]

Instance OIL inlcuye una minuciosa integración de instancias. Aunque la capa previa – Standard OIL – incluye constructores de modelo para especificar individuos que cumplen condiciones (definiciones por intensión), Instance OIL incluye capacidades de base de datos.

Heavy OIL incluirá capacidades adicionales de representación y razonamiento. Un lenguaje de reglas mas expresivo y facilidades de metaclases serán adicionadas. Se definirán esas extensiones en cooperación con la iniciativa DAML (DARPA Agent Markup Language; *www.daml.org*) .

Tres ventajas importantes de la arquitectura en capas de OIL son:

- La aplicación que use la ontología no es forzada a trabajar con un lenguaje que ofrece mucha mas expresividad y complejidad que la que ella necesita.
- Las aplicaciones que solo pueden procesar un bajo nivel de complejidad, pueden igualmente beneficiarse de la obtención de algunos aspectos de la ontología.
- Una aplicación que está capacitada para tratar con un nivel de complejidad alto, puede aún entender ontologías expresadas por otra aplicación en un nivel mas bajo de compejidad.

Haber definido un lenguaje ontológico como una extensión de RDFS significa que cada contología expresada en RDFS es una ontología válida en el nuevo lenguaje, y definir una extensión a OIL tan cercanamente posible a RDFS permite un reuso maximal de aplicaciones existentes basadas en RDFS.
Hay que notar sin embargo, que como el lenguaje de la ontología contiene mas vocabulario que RDFS, un 100 por ciento de compatibilidad no resultará posible.
En la figura 3.4 se muestra un ejemplo tendiente a la visualización de la segunda de las ventajas enumeradas anteriormente, en ella se define

herbivore como una clase, la cual es una subclase de **animal** y disjunta de **carnivore**.

```
<rdfs:Class rdf:ID="herbivore">
 <rdf:type
   rdf:resource="http://www.
     ontoknowledge.org/oil(RDFS-
       chema/#DefinedClass"/>
 <rdfs:subClassOf
rdf:resource="#animal"/>
 <rdfs:subClassOf>
   <oil:NOT>
          <oil:hasOperand rdf:resource="
          #carnivore"/>
  </oil:NOT>
 </rdfs:subClassOf>
<rdfs:Class>
```

fig. 3.4 ejemplo de expresión OIL [8]

Una aplicación limitada a RDFS puro, puede aún capturar algunos aspectos de esta definición. Como se muestra en la figura 3.5, encontrará que **herbivore** es una subclase **animal** y que es una subclase de una segunda clase que él no puede comprender apropiadamente (que no puede ser una subclase de **carnívore**).

```
<rdfs:Class rdf:ID="herbivore">
 <rdf:type
   rdf:resource="http://www.
     ontoknowledge.org/oil(RDFS-
       chema/#DefinedClass"/>
 <rdfs:subClassOf rdf:resource="#animal"/>
 <rdfs:subClassOf>
      ….
 </rdfs:subClassOf>
<rdfs:Class>
```

fig. 3.5 Lo comprensible por una aplicación simple[8].

3.2 Limitaciones actuales de OIL

Según los autores en [6], algunas de las limitaciones de OIL son:

Default reasoning. Oil provee el mecanismo de heredar valores de superclases, pero tales valores no pueden ser sobrescritos en una especialización.

Reglas/Axiomas. Solo un número fijo de propiedades algebraicas de slots pueden ser expresadas en OIL. No existe la facilidad de describir axiomas que se deban cumplir para determinados ítems de la ontología.

Propiedades algebraicas La carencia de un lenguaje para expresar axiomas podría ser parcialmente compensada por la extensión del conjunto de propiedades que pueden ser especificadas para relaciones en OIL. Actualmente el conjunto contiene inversa, transitiva y simétrica. Candidatas a agregar podrían ser reflexiva, irreflexiva, antisimetría, asimetría, conectividad (aRb or a=b or bRa para todo par a,b) orden parcial y orden total.

Uso de instancias en definiciones de clases Investigaciones en lógicas modales demuestran que la complejidad computacional aumenta considerablemente si se permite la representanción de instancias del dominio como parte de la definición de clases, por eso OIL no permite aún por ejemplo definir una clase por enumeración de sus elementos (extensión). Esta no es una limitación que pueda considerarse claramente como tal en la medida en que individuos del dominio pueden ser identificados como clases. Sin embargo, como se vió con anteriormente esta será una de los aspectos a considerar en la capa *Heavy* OIL.

En resumen, OIL tiene algunas ventajas, está apropiadamente asentado sobre lenguajes de la Web, como XML Schemas y RDFS (ver fig. 3.6) y ofrece diferentes niveles de complejidad. Atendiendo a sus primitivas de modelado, OIL refleja cierto concenso entre áreas como Description Logics y sistemas basados en frames. OIL es también un importante punto de inspiración para el lenguaje DAML+OIL (www.cs.man.ac.uk/~horrocks/DAML+OIL) , de la iniciativa DAML.

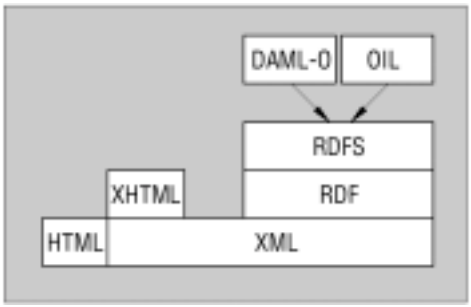

fig. 3.6 El modelo de implementación en capas [8]

4 Representación de ontologías en lenguajes para la Web.

Para definir una manera estándar de representar ontologías, debemos decidir acerca de dos cuestiones:

- Cuales son las primitivas epistemológicas usadas para representar una ontología (esto es, cosas como clases , relación is-a, relación element-of, atributos, dominio y restricciones de rango, etc) Básicamente esas son consideraciones acerca de la meta-ontologia usada para representar ontologías.
- Como podrían esos conceptos ser representados en la sintaxis del lenguaje en consideración.

XML – DTD

Una DTD es la descripción en un formalismo gramatical de que anidamientos, secuencias, atributos y entidades se pueden usar en el documento al que se refiere la DTD.
Ejemplo:

```
<?xml version=”1.0” standalone=”yes”>
<!DOCTYPE AGENDA [
<!ELEMENT AGENDA (CONTACTO)*>
<!ELEMENT CONTACTO (NOMBRE, EMAIL)
<!ELEMENT NOMBRE (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
]>
<AGENDA>
<CONTACTO>
<NOMBRE> Germán Cuello </NOMBRE>
<EMAIL> german@serv.com </EMAIL>
</CONTACTO>
<CONTACTO>
<NOMBRE> Germán Cuello </NOMBRE>
<EMAIL> german@chasque.net </EMAIL>
</CONTACTO>
</AGENDA>
```

Según [11], existen importantes razones que hacen poco recomendable el uso de DTDs para representar ontologías:

- Una DTD especifica el correcto anidamiento léxico en un documento, que puede coincidir o no con una jerarquía ontológica. No existe nada en una DTD que se corresponda con la relación “is a” entre clases, que es usualmente central en una ontología.
- Las DTDs carecen de la noción de herencia. En una ontología las subclases heredan atributos definidos en sus superclases y las superclases heredan instancias definidas para sus subclases. Ambos mecanismos de herencia no existen en una DTD.
- Las DTDs proveen una forma muy pobre de definir semántica de etiquetas elementales. Basicamente una etiqueta puede ser definida como estando compuesta de otras etiquetas o como siendo un string. Usualmente las ontologías proveen mucho mas riqueza en la definición de sus elementos.
- Las DTDs definen el orden en cual las etiquetas aparecen en el documento, contrariamente a lo que ocurre en las ontologías, donde el orden de los atributos no es relevante.

XML Schema

Un XML Schema es un medio de definir restricciones de la sintaxis y la estructura de documentos XML, tiene el mismo propósito que una DTD, pero significativas ventajas:

- Definiciones realizadas en un XML Schema son ellas mismas documentos XML, no es necesario un segundo lenguaje como se debe usar en las DTDs, y una ventaja es que todo lo desarrollado para documentos XML puede usarse para documentos de tipo XML Schema.
- Proveen un conjunto de tipos de datos mucho mas rico que el que puede ser definido actualmente en una DTD.
- Permiten definir anidamientos en la estructura de una manera mucho mas rica que las DTDs.
- Usan el mecanismo de espacios de nombres de XML para combinar documentos XML provenientes de orígenes heterogéneos.

Las ontologías y los XML Schemata sirven para muy diferentes propósitos. Los lenguajes ontológicos se destinan a especificar teorías de dominio, y los XML Schemata son una forma de proporcionar restricciones de integridad para orígenes de información (documentos y datos semiestructurados).

En [7] los autores encuentran que la relación existente entre una ontología y un XML Schema es equivalente a la existente entre el modelo Entidad Relación extendido y el esquema relacional de una base de datos.
El modelo relacional provee una descripción de las bases de datos orientada a la implementación, en tanto el modelo Entidad Relación provee un marco para moledar orígenes de información requeridos para una aplicación.

Puede resumirse entonces que expresar una ontología en XML Schema es posible, pero su definición debería ser previamente realizada en un lenguaje ontológico y luego trasladada a XML Schema, algunas razones se dan a continuación[7]:

- Una ontología debería contar con tipos definidos por intensión a través de axiomas, y los XML Schema carece actualmente de la posibilidad de tal tipo de definición, sin embargo, tipos definidos por intensión, pueden mapearse perfectamente a los tipos definidos en XML Schema toda vez que sea necesario.
- A pesar de que XML Schema incorpora la noción de *type-derivation*, ésto solo puede ser parcialmente comparado con lo que puede ser provisto por herencia en un lenguaje ontológico. En primer lugar, en XML Schema toda herencia tiene que ser modelada explícitamente, no existe la posibilidad de que se derive automáticamente. En segundo lugar, XML Schema no permite la posibilidad de herencia múltiple. Respecto de la relación “is-a”, la herencia top-down de atributos de superclases a subclases debe ser modelada de manera artificial, ya que los type-derivation solo pueden extender o restringir el tipo base, por tanto un tipo intermediario “dummy” tiene que ser introducido para permitir tal herencia.

En [7] Los autores proponen un procedimiento para expresar en XML Schema una ontología definida en OIL. En dicho procedimiento semi automático, se materializan en una primera etapa las jerarquías y luego se trasladan conceptos y slots en elementos.

RDF Schema - RDFS

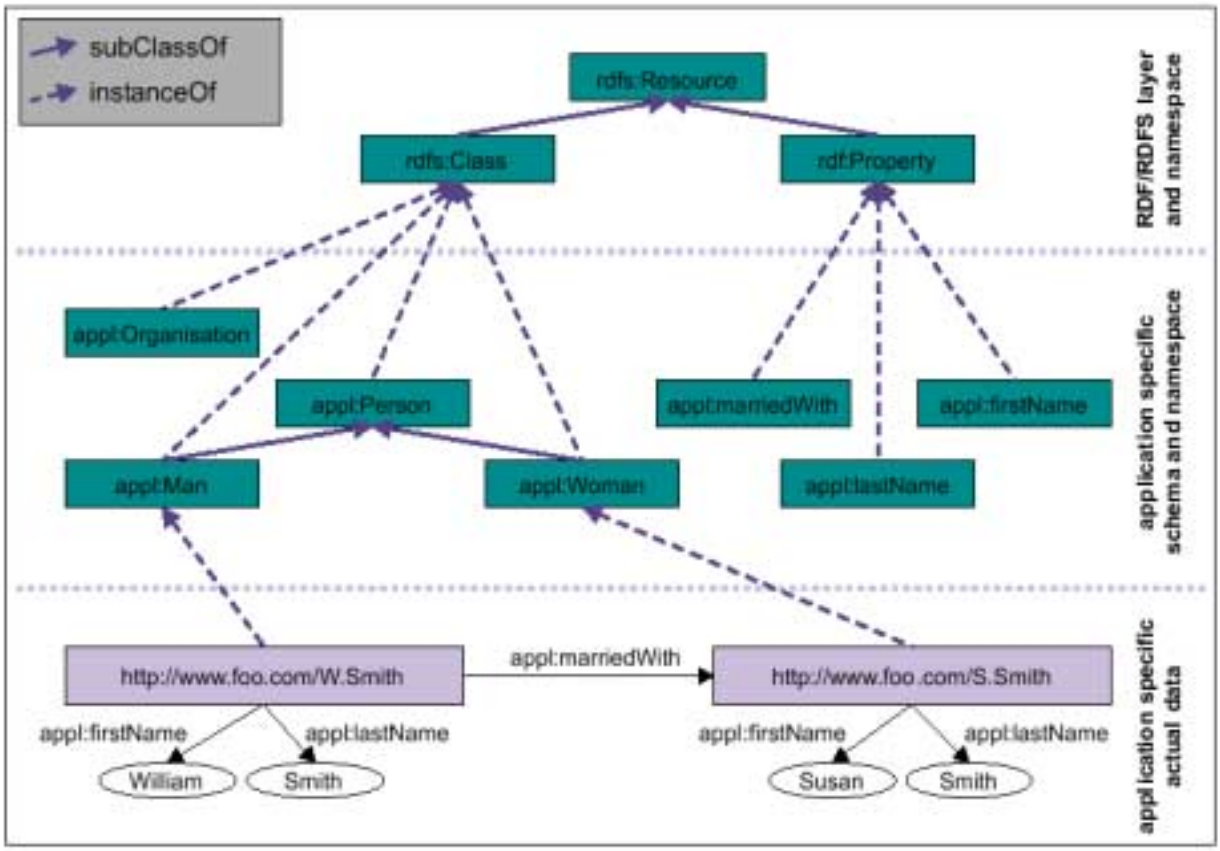

fig 4.2 Un ejemplo de RDF Schema [5]

RDFS fue definido sobre el lenguaje RDF para ofrecer un vocabulario particular para modelar clases y jerarquías de propiedades y otras primitivas básicas que puedan ser referenciadas desde modelos RDF.
Según [5] el rol de RDFS en la terminología de la ingeniería del conocimiento es definir una ontología simple que documentos RDF particulares puedan chequear, para decidir su consistencia.

La siguiente lista pretende reseñar los términos mas relevantes del RDFS

- La clase mas general en RDFS es `rdfs:Resource`. Tiene dos subclases llamadas `rdfs:Class` y `rdfs:Property` (ver figura 4.2). Cuando se especifia un esquema específico para un dominio en RDFS, las clases y propiedades definidas en ese esquema se vuelven instancias de estos dos recursos.
- El recurso `rdfs:class` denota el conjunto de todas las clases en el sentido de orientación a objetos. Esto significa que clases como `appl:Person` o `appl:Organization` son instancias de la metaclase `rdfs:Class`.
  Se cumple lo mismo para propiedades, esto es que cada propiedad definida en un RDFS específico de una aplicación, es una instancia de `rdf:Property`, por ejemplo `appl:marriedWith`.
- RDFS define la propiedad especial `rdfs:subClassOf` que define la relación subclase entre clases. Desde que `rdfs:subClassOf` es transitiva, las

definiciones son heredadas por las clases mas específicas desde las mas generales, y recursos que son instancias de una clase son automáticamente instancias de todas las superclases de esa clase. En RDFS está prohibido que cualquier clase sea una `rdfs:subClassOf` de si misma o de una de sus subclases.

- De manera similar a `rdfs:subClassOf` , que define una jerarquía de clases, otro especial tipo de relación `rdfs:subPropertyOf` define una jerarquía de propiedades, por ejemplo podemos expresar que `fatherOf` es una `rdfs:subPropertyOf` de `parentOf.`
- RDFS permite definir restricciones de rango y dominio asociadas con propiedades. Por ejemplo, esas restricciones permiten que personas y solo personas puedan estar `marriedWith` con otras personas.

Como se muestra en la capa central de la figura 4.2 las clases específicas del dominio considerado `appl:Person`, `appl:Man`, y `appl:Woman` están definidas como instancias de `rdfs:class`. De la misma forma las propiedades específicas del dominio están definidas como instancias de `rdf:Property`.

En resumen, según [5] RDFS carece de capacidades para describir la semántica de conceptos y relaciones mas allá de aquella provista por los mecanismos de herencia, eso lo hace un lenguaje muy débil aún para el mas austero de los sistemas basados en conocimiento. RDFS provee solo las mas básicas primitivas para modelado de ontologías.
Frente al mantenimiento de un deseable equilibro entre tratabilidad y expresividad de un lenguaje, RDFS se coloca en el extremo de la mínima expresividad, porque fue concebido para ser aplicable a toda la variada gama de recursos de la web. En contraste con los típicos lenguajes de representación de conocimiento, RDFS no ha sido concebido para ser una respuesta definitiva en la representación de conocimiento en un dominio particular, sino un núcleo pasible de ser extendido (*extensible core language*).

Aunque RDFS provee soporte para el modelado de conceptos ontológicos y relaciones, no lo provee para los axiomas, que son un ingrediente clave en la definición de una ontología. Además provee una poderosa reificación, que es descriptivamente conveniente, pero dificulta cualquier servicio de razonamiento.

Con todas estas consideraciones, vale decir que RDFS puede ser usado directamente para describir una ontología.

3 Conclusiones

Como resumen se concluye que las DTDs no resultan adecuadas para representar ontologías, Los XML Schemata pueden representarlas, pero dificultosamente, Los autores de [7] proponen un procedimiento semi-automático de traducción de una ontología representada en OIL a una representación en XML Schema, y RDF Schema resulta adecuado para representación de ontologías, con la aclaración de que por estar diseñado para estándar de la web, no provee primitivas suficientes como para resultar un lenguaje ontológico aceptablemente expresivo, sino mas bien un núcleo concebido para extender.

4 Referencias

[1] Ora Lassila and Ralph R. Swick. Resource description framework (RDF) model and syntax specification. *Technical report, W3C, 1999*. W3C Recommendation. http://www.w3.org/TR/REC-rdf-syntax.

[2] Dan Brickley and R.V. Guha. Resource description framework (RDF) schema specification. *Technical report, W3C, 1999. W3C Proposed Recommendation*. http://www.w3.org/TR/PR-rdf-schema/.

[3] David C. Fallside: XML-Schema Part 0: Primer, *W3C Working Draft, 7 April 2000*. http://www.w3.org/TR/2000/WD-xmlschema-0-20000407/

[4] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn: XML Schema Part 1: Structures, *W3C Working Draft, 7 April 2000*. http://www.w3.org/TR/2000/WD-xmlschema-1-20000407/

[5] Steffen Staab, Michael Erdmann, Alexander Maedche, Stefan Decker. An Extensible Approach for Modeling Ontologies in RDF(S) *ECDL Workshop on the Semantic Web. 2000*

[6] Horrocks et al., *The Ontology InferenceLayer OIL*, tech. report, Vrije Universiteit Amsterdam; www.ontoknowledge.org/oil/
TR/oil.long.html (current 9 Mar. 2001).

[7] M. Klein et al., "The Relation between Ontologies and Schema-Languages: Trans-lating OIL Specifications to XML Schema,"
*Proc. Workshop on Applications of Ontolo-gies and Problem-Solving Methods, 14th European Conf. on Artificial Intelligenc*e, Berlin, 2000.

[8] Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah Mc.Guinness, Peter F. Patel-Schneider. Oil: An Ontology infrastructure for the Semantic Web In *IEEE Intelligent Systems* March/April 2001, pp 38-45.

[9] Frank van Harmelen, Dieter Fensel. Practical Knowledge Representation for the Web, *Proceedings of*

the IJCAI-99 workshop on Intelligent Information Integration Stockholm, Sweden, 1999.

[10] S. Decker, F. van Harmelen, J. Broekstra, M. Erdmann, D. Fensel, I. Horrocks , M. Klein, and S. Melnik.The semantic Web – on the respective Roles of XML and RDF, *IEEE Internet Computing, September/October 2000*

[11] D. Fensel: Relating Ontology Languages and Web Standards. *In J. Ebert et al (eds), Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000, St. Goar, April 5-7, 2000, Folbach Verlag, Koblenz, 2000*