

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA



UNIVERSIDAD DE MÁLAGA

TRABAJO FINAL

GRADO EN INGENIERÍA DE LA SALUD

MÁLAGA, 2019

Proyecto Estándares de Datos Abiertos e Integración de Datos
Ingeniería de la Salud

**Formato de Publicación de la Escuela Técnica Superior de Ingeniería
Informática de Málaga**

Autores:

José Rodríguez Maldonado
Naira María Chiclana García

Resumen

En este documento se trata un proyecto de la asignatura de Estándares de Datos Abiertos e Integración de Datos que realizaron alumnos del grado de Ingeniería de la Salud de la Escuela Técnica Superior de Ingeniería Informática de Málaga.

En el se hace una breve introducción a la Web Semántica, y se tratan algunas de las herramientas principales: La estructuración de los datos en XML, el estandar RDF y el estandar de transferencia de datos clínicos HL7.

Indice de contenidos

1. Introducción 1
 - 1.1. Contexto 1.1
 - 1.2. Motivación 1.2
 - 1.3. Objetivos del proyectos 1.3
 - 1.4. Estructura del documento 1.4
2. Web Semántica 2
 - 2.1. Que es una Ontología 2.1
3. Base de Datos 3
 - 3.1. Creación de la Base de datos 3.2
4. XML 4
 - 4.1. Transformación datos BD a XML 4.1
 - 4.2. XMLSchema 4.2
 - 4.3. XSL 4.3
 - 4.3.1. XSLT 4.3.1
5. HL7 5
 - 5.1. Implementación HL7 5.1
6. RDF 6
 - 6.1. Modelado de la ontología de nuestra BD en RDF 6.1
7. Conclusiones y trabajos futuros 7
8. GitHub 8
9. Webgrafía 9

Índice de ilustraciones

- Figura 1: Mapa conceptual simplificado de la Web Semántica.
- Figura 2: Esquema de la base de datos usada *HOSPITAL*.
- Figura 3: Ejemplo de W3S de estructura de árbol para documento XML.
- Figura 4: Estructura de árbol de la BD.
- Figura 5: Datos XML con Schema en navegador. Figura 6: Datos XML con Schema y XSLT en navegador.
- Figura 7: Ejemplo terminal cliente.
- Figura 8: Ejemplo terminal servidor.
- Figura 9: Resultado mensaje HL7.
- Figura 10: Codificación mensaje HL7.
- Figura 11: Esquema tripleta RDF.
- Figura 12: Clases de la ontología.
- Figura 13: Propiedades de la ontología.
- Figura 14: Individuos de la ontología.
- Figura 15: Estructura del repositorio..
- Figura 16: Readme del repositorio.
- Figura 17: Características de la licencia *MIT*.

1. Introducción: Definición inicial del proyecto

En este capítulo se va a proporcionar al lector una visión general sobre el proyecto desarrollado. Para ello se ha dividido en los siguientes apartados: Contexto, motivación, objetivos establecidos y estructura de la memoria.

1.1. Contexto

Este proyecto se enmarca dentro de la asignatura de Estándares de datos abiertos e integración de datos.

1.2. Motivación

El objetivo de este proyecto es el de reforzar todos los conocimientos impartidos en clase de una forma práctica, además implementarlos para profundizar en como funcionan realmente.

1.3. Objetivos del proyecto

Reforzar de una forma práctica todos los conocimientos impartidos en clase. El objetivo será hondar en los contenidos expuestos a continuación:

Tras ver brevemente lo que es la Web Semántica, trabajaremos con algunas de sus herramientas. Para hemos creado una Base de datos sobre un hospital, de la cual modularemos los datos en formato XML, con su correspondiente XMLSchema para asegurar que su formato es correcto y una hoja de estilos XSLT para su visualiación. Además, modularemos la ontología con la que la hemos descrito en Protegé para crear tripletas RDF. También veremos el estandar de datos clínico HL7 y explicaremos como hemos creado un conexión a través de sockets para intercambiar mensajes tanto de texto como de HL7.

1.4. Estructura del documento

- **CAPÍTULO 1-INTRODUCCIÓN.**

En este capítulo se va a proporcionar al lector una visión general sobre el proyecto desarrollado. Para ello se ha dividido en los siguientes apartados: Contexto, motivación, objetivos establecidos y estructura de la memoria.

- **CAPÍTULO 2- WEB SEMÁNTICA**

Breve introducción a la Web Semántica, sus utilidades y objetivos.

- **CAPÍTULO 3- BASE DE DATOS**

Explicación de la creación y la estructura de la Base de datos usada para tratar en el resto de capítulos.

- **CAPÍTULO 4-XML**

En este capítulo se introducen los conceptos de XML, sus usos, herramientas y utilidades. Además, modelamos nuestra Base de datos como un archivo XML, y le aplicamos un XML Schema y una hoja de estilos XSLT.

- **CAPÍTULO 5-HL7**

Tras una breve introducción a los conceptos de HL7, mostraremos como hemos codificado un mensaje con HL7 y como enviarlo a través de sockets.

- **CAPÍTULO 6-RDF**

Breve introducción a RDF, su relación con la Web Semantica y sus utilidades. Modularemos un RDF de la ontología nuestra Base de datos usando la herramienta Protégé.

- **CAPÍTULO 7-CONCLUSIONES Y TRABAJOS FUTUROS**

Durante este capítulo se aportaran las conclusiones obtenidas a lo largo del desarrollo del proyecto, teniendo en cuenta los objetivos y el todo el proceso de desarrollo. Además, se describen los conocimientos aprendidos a lo largo del proyecto y algunos aspectos a tener en cuenta en trabajos futuros.

- **CAPÍTULO 8-WEBGRAFÍA**

Referencias a documentos y webs que se han usado para realizar la redacción de este documento.

2. Web Semántica

[1] Es una gran colección formal y heterogénea, procesable por máquinas, accesible por la web, basada esencialmente en ontologías (sobre recursos web y otras entidades) y expresada en sintaxis basada en XML. 4

La tercera década de la Web en la que nos encontramos se caracteriza por un enriquecimiento de la estructura de esta.

Las tecnologías semánticas tienen un papel fundamental en transformar la Web desde los sistemas de ficheros de Bases de Datos.

Gracias a la Web Semántica tenemos: entre otros, Mejor búsqueda, anuncios más dirigidos, Colaboración más inteligente, integración más profunda, Contenido más tico y Mejor personalización.

Mapa conceptual simplificado de la web semántica:

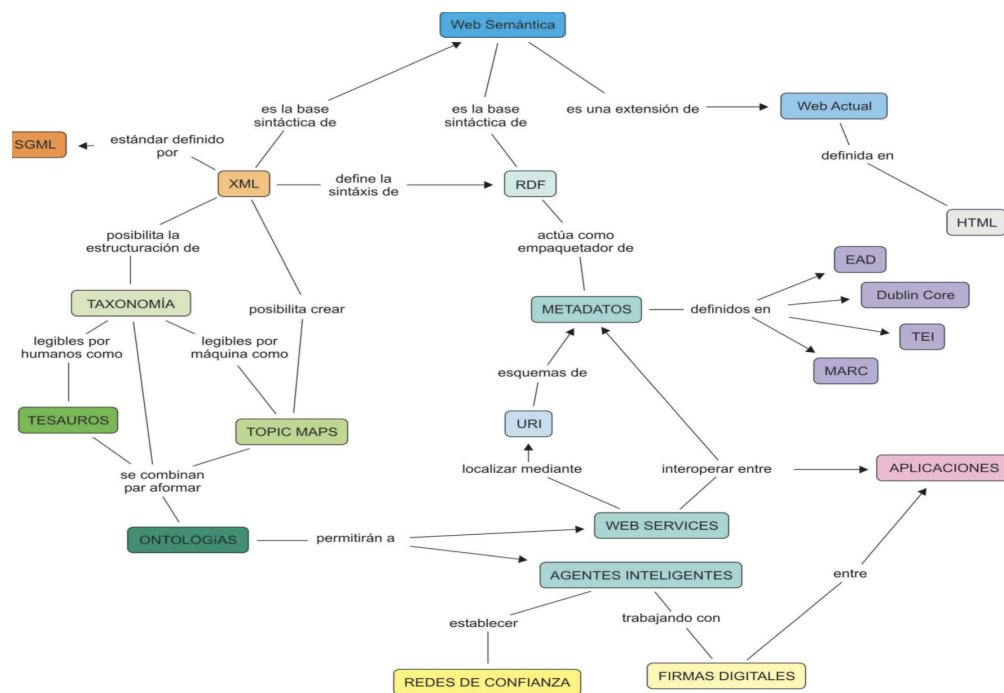


Figura 1: Mapa conceptual simplificado de la Web Semántica

Tenemos dos **caminos hacia la semántica**:

- "Bottom-Up": La clásica, se basa en añadir metadatos semánticos a las páginas y bases de datos de toda la web. Todas las páginas Web se vuelven semánticas y todo el mundo tendría que aprender RDF/OWL.
- "Top-Down": El contemporáneo, se basa en generar automáticamente metadatos semánticos para dominios verticales, crear servicios que lo proporcionan para cubrir la Web no semántica y nadie tendría que aprender RDF/OWL.

La Web Semantica es un **Key Enabler**:

- Mueve la inteligencia desde las aplicaciones hasta los datos.
- Los datos se vuelven autodescritos.
- El significado forma parte de los datos.
- Las aplicaciones pueden ser más inteligentes con menos esfuerzo porque los datos portan conocimiento a cerca de lo que son y como se usan.
- Los datos pueden ser enlazados y compartidos más facilmente.

También es una **capa de bases de datos abiertas para la web**:

- Interfaces de *Querys* abiertas.
- Mapeados de datos abiertos.
- Reglas abiertas.
- Ontologias abiertas.

Estándares abiertos de la Web Semántica:

- *RDF* 6 Almacena os datos en tripletas.
- *OWL* Define ontologias (sistemas de conceptos)
- *SPARQL* Lenguaje para hacer consultas en datos en *RDF*.
- *SWRL* Define reglas.
- *GRDDL* Transforma los datos en GRDDL.

El objetivo es que "La web sea una Base de datos".

2.1. Ontología

Mencionada numerosas veces en la descripción de web semántica: Una Ontología es una especificación explícita (de conceptos, propiedades, funciones y axiomas), formal(entendible por las máquinas) de una conceptualización común (modelo abstracto de algún fenómeno en el mundo y conocimiento consensuado). Define términos básicos y relaciones que conforman el vocabulario de un area de interés, además de las reglas para combinar los términos y las relaciones para definir extensiones del vocabulario.

3. Base de Datos

La Base de datos que hemos usado para definirla como ontología con sus correspondientes XML y RDF se trata de un Hospital al que acuden pacientes. La creación de los datos ha sido en parte de datos estraidos de Kaggle [2] y el resto creados por nosotros.

3.1. Estructura de la base de datos

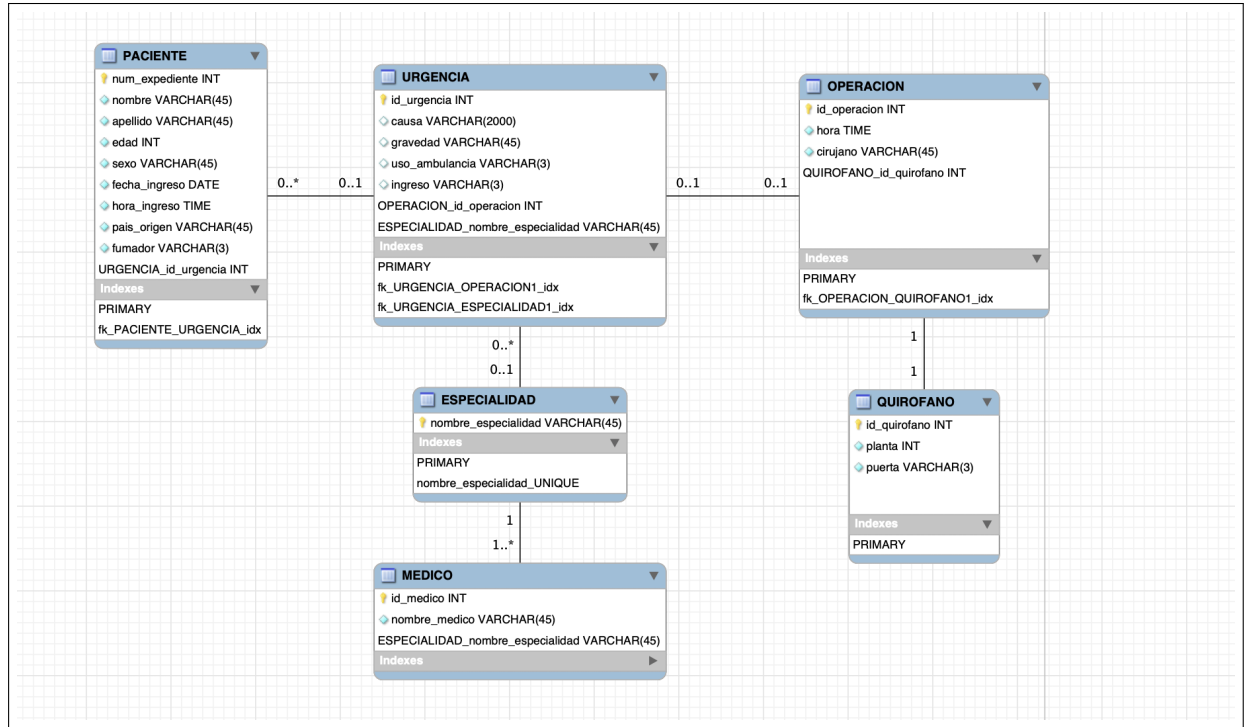


Figura 2: E
squema de la base de datos usada *HOSPITAL*.

Cada *PACIENTE* puede tener una *URGENCIA*. La urgencia pertenece a una *ESPECIALIDAD*, donde hay uno o más *MEDICOS*. Además de pertenecer a una especialidad, la *URGENCIA* puede requerir una *OPERACION*, que se llevará a cabo en un *QUIROFANO*.

3.2. Creación de la Base de datos

Para ello hemos usado principalmente la libreria *Pandas*[3]. Hemos creado cada una de las tablas como un dataframe de *Pandas* y listas con su correspondiente tipo para cada columna de las tablas. Para los id hemos generado números aleatorios en el rango [1, 999999] usando *randrange*, y hemos comprobado que fueran valores únicos y no nulos. Para los posibles valores de tipo *VARCHAR* también se han elegido aleatoriamente de enumerados usando la función *choice*.

Hemos añadido algunas especificaciones concretas para que los valores para un mismo paciente no contuvieran incoherencias.

Como veremos más adelante en la Figura 4 con la estructura de árbol definida, los valores están creados acorde, es decir, un solo paciente tendrá los datos acordes a su urgencia que tendrá su especialidad que tendrá su médico...

4. XML

Extensible Markup Language es un lenguaje de etiquetas que define unas reglas para codificar documentos en un formato entendible por humanos y máquinas gracias a etiquetas con significado. Está definido por estándares abiertos, entre ellos destacan las especificaciones de *W3C*. [4]

Es la base sintáctica de la Web Semántica.

Los documentos *XML* son contenedores de información estructurada. Después de describir su esquema, se puede utilizar el lenguaje *XSL* que definirá su presentación.

Algunas de las **características principales**:

- Al ser un lenguaje de etiquetas, todas las etiquetas tienen que encontrarse entre los símbolos menor y mayor y estar correctamente anidadas.
- Las etiquetas son " case sensitive ".

Algunos *naming styles* son:

`<firstname>`, `<FIRSTNAME>`, `<first_name>`, `<FirstName>`, `<firstName>`.

- Las etiquetas están jerarquizadas en estructura de árbol. Hay un único nodo raíz (*HOSPITAL* en nuestro caso) en el que se encuentran anidados todos los demás. Las anidaciones particulares las explicaremos más adelante.
- Pueden tener documentos *DTD* (para describir los datos) o esquemas *XML*, aunque no es necesario.
- Los espacios se preservan.
- Los símbolos reservados por el lenguaje (entidades) hay que escribirlos de forma especial, como por ejemplo:

```
&lt; <
&gt; >
&amp; &
&apos; '
&quot; "
```

Forma de la **estructura de árbol**:

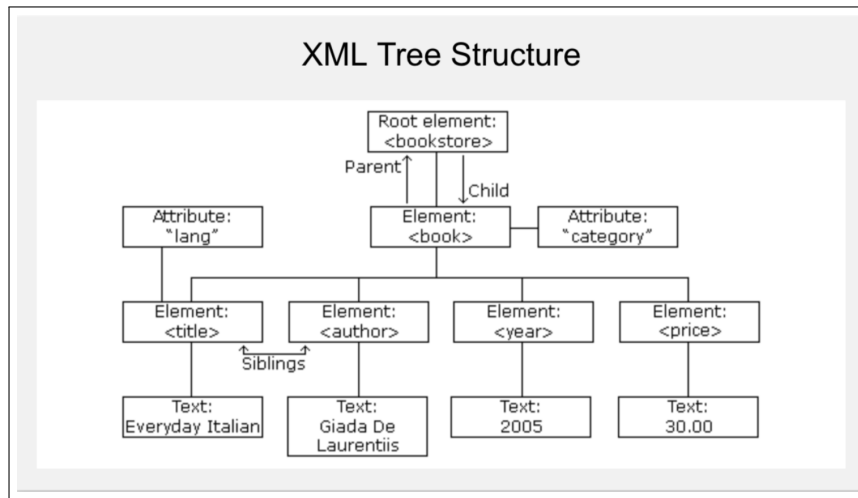


Figura 3: Ejemplo de W3S de estructura de árbol para documento XML.

Lo que en lenguaje XML vemos como:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <child (category="name" or 'name' or 'name with "name" inside')>
    <subchild (category="name") >value </subchild >
  </child>
</root>
```

Los **tipos de objetos** que podemos encontrar son:

- Elementos: Pueden contener texto, otros elementos o estar vacíos.
- Atributos: Proporcionan información extra sobre los elementos.
- Entidades: Caracteres con un significado especial en XML, se tienen que describir de forma especial.
- PCDATA: parsed character data.
- CDATA: character data.

Algunas de las **herramientas** disponibles son:

- **XML DOM** (Document Object Model): es una plataforma e interfaz de lenguaje neutral que permite a los programas y scripts acceder dinámicamente y actualizar el contenido, estructura y estilo de un documento XML.
- **XPATH**: Usa expresiones path para seleccionar nodos o set de nodos en un documento XML. Algunos ejemplos son:

```
root/child[1] (first child) [...last()], [last()-1], [position()<3]
subchild[@atributo], subchild[@atributo=value]
root/child[subchild1 with condition]/subchild2
```

Para seleccionar nodos: \ root, . current node, .. parent of current node,...

Axes: ancestor, child, descendant, following, parent, preceding, self...

Operadores: + ,-, div ,=, !=, < ,..., or ,and,mod

- **XLink:** Crea hyperlinks en documentos XML.
- **Documentos de XML válidos.** Un documento tiene que estar bien formado (cumplir las características descritas anteriormente) y cumplir un documento de definición de tipos. Estos pueden ser:
 - DTD: Document Type Definition: define la estructura y atributos y elementos legales de un documento XML.
 - XML Schema 4.2: Describe la estructura que ha de tener un documento XML.

4.1. Transformación datos BD a XML

Acorde al esquema de la Figura 4, hemos pasado los datos organizados en dataframes con listas al lenguaje XML estructurados en etiquetas jerarquizadas.

Para ello hemos hecho uso de la librería `lxml.tree` [5] para definir la jerarquía. Primero hemos importado todos los datos de nuestra base de datos local con una consulta SQL, la cual tenía un *join* que relacionaba todas las tablas con todas. Una vez teníamos los datos, hemos definido *HOSPITAL* como el elemento root `ET.Element('HOSPITAL')` (ET es el nombre con el que hemos importado la librería). En el resto de tablas hemos indicado su posición en la jerarquía de la forma:

```
tabla_actual = ET.SubElement(tabla_padre, tabla_actual).
```

Y dentro de cada tabla, cada columna de la forma:

```
ET.SubElement(tabla, nombre_columna).
```

Para recoger los datos que hemos introducido en el XML, hemos

Una vez tenemos los datos de las tablas modulados en XML según la jerarquía, definiremos un esquema para asegurar que tienen el formato correcto y una hoja de estilos para visualizarlos.

4.2. XMLSchema

Jerarquía utilizada:

Hemos puesto las tablas que contienen FK dentro de las tablas que contienen las PK a las que apuntan. Así, cada tabla "hija" estará en el nivel de etiquetas de el resto de columnas de su tabla "padre", y así sucesivamente.

El primer elemento define el nodo raíz que contendrá todas las tablas, en nuestro caso *HOSPITAL*, el resto lo hemos construido en base a la estructura la hemos visto en 3.1.

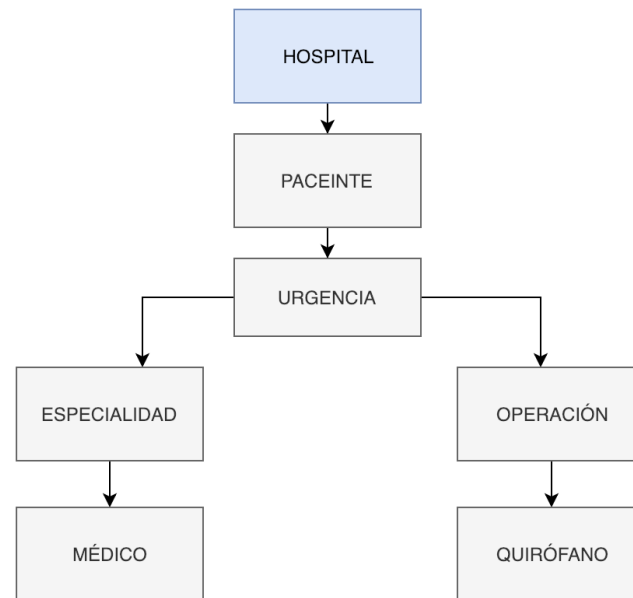


Figura 4: E
estructura de árbol de la BD.

Cada tabla de nuestra base de datos 3 la hemos modulado como un `xs:element` de tipo `xs:complexType`. Dentro de cada tabla, en una `xs:sequence` hemos definido cada una de sus columnas como un `xs:element` con su correspondiente `type`.

Una visión general de la jerarquía de etiquetas sin incluir las columnas:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

<xs:element name="HOSPITAL"> <!--ROOT-->
  <xs:complexType>
    <xs:sequence>
      <xs:element name="PACIENTE">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="nombre" type="xs:string"/>
            <xs:element name="apellido" type="xs:string"/>

```



```

<xs:element name="edad" type="xs:nonNegativeInteger"/>
<xs:element name="fecha_ingreso" type="xs:date"/>
<xs:element name="hora_ingreso" type="xs:time"/>
<xs:element name="pais_origen" type="xs:string"/>
<!--Urgencia anidada en Paciente-->
<xs:element name="URGENCIA">
  <xs:complexType>
    <xs:sequence>
      <!--Especialidad anidada en Urgencia anidada en Paciente-->
      <xs:element name="ESPECIALIDAD">
        <xs:complexType>
          <xs:sequence>
            <!--Columnas de especialidad-->
            <!--Medico anidado en Especialidad anidado en Urgencia anidado en Paciente-->
            <xs:element name="MEDICO">
              <xs:complexType>
                <xs:sequence>
                  <!--Columnas de medico-->
                </xs:element> <!--close medico-->
              </xs:sequence>
            </xs:complexType>
          </xs:element> <!--close especialidad-->
          <xs:element name="OPERACION">
            <xs:complexType>
              <xs:sequence>
                <!--Columnas de operacion-->
                <!--Quirófano nested in Operación nested in Urgencia nested in Paciente-->
                <xs:element name="QUIROFANO">
                  <xs:complexType>
                    <xs:sequence>
                      <!--Columnas de quirofono-->
                    </xs:sequence>
                  </xs:complexType>
                </xs:element> <!--close quirofono-->
              </xs:sequence>
            </xs:complexType>
          </xs:element> <!--close operacion-->
        </xs:sequence>
      </xs:complexType>
    </xs:element><!--close Urgencia-->
  </xs:sequence>
</xs:complexType>
</xs:element> <!--close Paciente-->
</xs:sequence>
</xs:complexType>

```

```
</xs:element><!--close HOSPITAL-->
</xs:schema>
```

Para algunas columnas hemos definido restricciones tipos especiales de elementos. Algunos de ellos han sido:

- Enumerados:

De múltiples valores, por ejemplo *gravedad*

```
<xs:element name="gravedad">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="BAJA"/>
      <xs:enumeration value="ALTA"/>
      <xs:enumeration value="MUY ALTA"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Con solo dos opciones, por ejemplo *uso_ambulancia*

```
<xs:element name="uso_ambulancia">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="NO|YES"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- Los id: cadenas de números no negativos contenidos en el rango [0,9]

```
<xs:element name="id_urgencia">
  <xs:simpleType>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:pattern value="([0-9])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Podemos hacer numerosas combinaciones de patrones:

[a-z], [A-Z] [A-Z] [A-Z], [a-zA-Z], [a-zA-Z], [xyz], [a-zA-Z0-9]{8},...

Si desde el archivo que contiene los datos hacemos referencia al esquema podemos verlos en el navegador estructurados en el árbol definido:

```

- <note xsi:schemaLocation="mydbXSLT.xsd">
- <HOSPITAL>
- <PACIENTE>
  <num_expediente>1116330</num_expediente>
  <nombre>AGUSTIN</nombre>
  <apellido>GARCIA</apellido>
  <edad>20</edad>
  <sexo>HOMBRE</sexo>
  <fecha_ingreso>2009-06-01</fecha_ingreso>
  <hora_ingreso>12:00:00</hora_ingreso>
  <pais_origen>Federación Rusa</pais_origen>
  <fumador>NO</fumador>
- <URGENCIA>
  <id_urgencia>5062</id_urgencia>
  <causa>TIENE SOLO CINCO PIEZAS DENTARIAS</causa>
  <gravedad>ALTA</gravedad>
  <uso_ambulancia>NO</uso_ambulancia>
  <ingreso>YES</ingreso>
- <ESPECIALIDAD>
  <nombre_especialidad>Medicina preventiva y salud pública</nombre_especialidad>
- <MEDICO>
  <id_medico>30</id_medico>
  <nombre_medico>ARZANI Daniel</nombre_medico>
  </MEDICO>
</ESPECIALIDAD>
- <OPERACION>
  <id_operacion>5039344</id_operacion>
  <hora>1:00:00</hora>
  <cirujano>TOLISSO Corentin</cirujano>
- <QUIROFANO>
  <id_quirofano>17</id_quirofano>
  <alanto>2</alanto>

```

Figura 5: Datos XML con Schema en navegador.

4.3. XSL

Una vez sabemos que nuestro documento es correcto ya que encaja en el XMLSchema definido, podemos definir como queremos que se muestren los datos.

[6] XSL es un lenguaje de hojas de estilo ampliable para mostrar la información de un documento *XML*. Podríamos verlo como el equivalente a CSS para HTML.

La familia de recomendaciones para definir la transformación y presentación de documentos XML tiene 3 partes:

- XSLT (eXtensible Stylesheet Language for Transformations): Lenguaje para transformar *XML* a *XML*, *HTML* o algún otro lenguaje basado en texto.
- XPath (XML Path Language) : Lenguaje de expresión usado por *XSLT* para acceder o referirse a partes de un documento *XML*.
- XSL-FO (eXtensible Stylesheet Language – Formatting Objects): Conversión del *XML* en un formato Imprimible y legible por una persona, como por ejemplo PDF.

4.3.1. XSLT

Transformaciones de lenguaje de hojas de estilo ampliable es un subconjunto del lenguaje XSL que permite mostrar los datos XML en una página Web y "transfor-

marlos"(junto con los estilos XSL) en información legible y con estilos en formato HTML.

Para obtener los datos del XML que queremos mostrar, tenemos los operadores básicos:

- **value-of** Extrae el valor de un elemento concreto.
- **for-each** Permite seleccionar cada uno de los elementos de un nodo específico.
- **sort** Ordenar los datos de salida.
- **if** Podemos filtrar el output. Algunos de los operadores condicionales disponibles son: `=`, `!=`, `<`, `>`;
- **choose** Condicional múltiple `<xsl:choose>`, `<xsl:when>`, `<xsl:otherwise>`

En nuestro caso, hemos enseñado cada tabla en formato de tabla. Para cada tabla usamos un **for-each** e indicamos su ruta relativa de la forma

`<xsl:for-each select="note/HOSPITAL/PACIENTE">`. Una vez dentro de la tabla, usaremos un **value-of** para extraer el valor de cada columna de la forma `<td><xsl:value-of select="num_expediente"/></td>`.

Una vez tenemos definido el XMLSchema y la hoja de estilos XSLT, las referenciamos desde el archivo con los datos:

```
<?xml version="1.0" encoding = "UTF-8"?>
<?xml-stylesheet type='text/xsl' href='mydbXSLT.xsl'?>
<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="mydbXSLT.xsd">
```

Lo que nos permite poder verlos en el navegador como vemos en la Figura 6.

localhost:63342/HospitalDEWeb/XML/CLData_Output.xml?_ijt=e6tjn3siofc5jm6n2p84p4mpjs 50%

PACIENTE

num.identificac	nombre	apellido	edad	sexo	fecha ingreso	hora ingreso	país origen	estado
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY
11111111	AGUSTIN	CLARICA	20	HOMBRE	2009-08-01	12:00:00	Estados Unidos	NY

Figura 6: Datos XML con Schema y XSLT en navegador

5. HL7

HL7 es un **estándar de interoperabilidad para sistemas de información sanitaria**.

Muy brevemente lo podemos definir como **aplicación de protocolo para el intercambio de datos electrónico en entornos de salud**.

Se fundó en 1987 para promover la comunicación entre sistemas de datos en los hospitales. Está acreditado por los estándares ANSI y funciona sin ánimo de lucro. Han desarrollado una organización dedicada a proveer un framework comprensivo y estándares para el **intercambio, integración, compartimiento y recuperación de información clínica electrónica** que soporte la práctica clínica y el **manejo, envío y evaluación de los servicios clínicos**.

Su objetivo es proveer estándares para interoperabilidad que mejoren el envío y workflow, reduzca la ambigüedad de la transferencia de conocimiento para todos los stakeholders (vendedores, medicos, pacientes,...).

Algunas de las funcionalidades que NO contiene son:

- Seguridad y Control de acceso.
- Privacidad y confidencialidad.
- Responsabilidad y auditoría de transacciones.

5.1. Implementación HL7

En primer lugar vamos a montar un servicio de cliente/servidor, para el cual nos serviremos del módulo *socket*[7] que podemos encontrar en python. Los **sockets** los utilizaremos para comunicar el programa de *cliente (envía mensajes)* y el *servidor (espera mensajes)*, incluso podremos hacerlo desde máquinas distintas. Aunque teóricamente parece algo un tanto complejo, a la hora de la practica resulta bastante intuitivo el funcionamiento de los mismos.

Para poder establecer la comunicación necesitaremos:

- Un **servidor**, que iniciamos con la dirección a la que queremos estar escuchando.
- Un **cliente** solo tendrá que conectarse a ese puerto.

El servidor quedaría tal que así:

```
host = socket.gethostname()
port = 5000 # initiate port no above 1024
server_socket = socket.socket() # get instance
# look closely. The bind() function takes tuple as argument
```

```

server_socket.bind((host, port)) # bind host address and port together
# configure how many client the server can listen simultaneously
server_socket.listen(2)
conn, address = server_socket.accept()
client = conn.recv(7000).decode()

```

Mientras que el cliente se implementa de la siguiente forma:

```

host = socket.gethostname() # as both code is running on same pc
port = 5000 # socket server port number
client_socket = socket.socket() # instantiate
client_socket.connect((host, port)) # connect to the server
client_socket.send(client.encode())

```

Teniendo ya los sockets preparados para enviar, recibir y procesar mensajes solo queda integrar HL7. Para ello nos hemos servido de una librería llamada `HL7apy`[8]. Esta librería que se utiliza para codificar y parsear mensajes en HL7. Podemos encontrar la función que codifica los HL7 en un script aparte.

De entre todos los mensajes que se pueden hacer con HL7, nosotros nos hemos querido centrar en transmitir mensajes del tipo *ORU_R01*. Estos mensajes son muy comunes en la práctica médica puesto que se utilizan para transmitir resultados electrónicos, bien sean PDFs, un archivo *DICOM* o alguna otra clase de archivo.

La codificación del mensaje HL7 podríamos englobarla en 2 partes:

- La cabecera MSH

Dentro definiremos el propósito del mensaje, el destino, delimitadores especiales que podamos encontrar, ... Podemos tener 19 campos dentro de la cabecera, pero de entre ellos se requiere como mínimo:

- Separador de campos.
- Conjunto de caracteres de codificación.
- Tipo de mensaje.
- ID de control del mensaje.
- ID de procesamiento.
- ID de versión.

- El cuerpo

El resto de parámetros que le podemos añadir son opcionales. En nuestro caso, tenemos definidos los siguientes parámetros:

```

hl7 = core.Message("ORU_R01")
hl7.msh.msh_3 = "SendingApp"

```



```

hl7.msh.msh_4 = "SendingFac"
hl7.msh.msh_5 = "ReceivingApp"
hl7.msh.msh_6 = "ReceivingFac"
hl7.msh.msh_9 = "ORU^R01^ORU_R01"
hl7.msh.msh_10 = "168715"
hl7.msh.msh_11 = "P"

```

En propio mensaje irá dentro del cuerpo y tendrá incluido un poco de información y el archivo.

```

hl7.ORU_R01_PATIENT_RESULT.ORU_R01_PATIENT.PID.pid_3 = "15"
hl7.ORU_R01_PATIENT_RESULT.ORU_R01_PATIENT.PID.pid_5 = "MAYA"
hl7.ORU_R01_PATIENT_RESULT.ORU_R01_ORDER_OBSERVATION.OBR.obr_4 =
patients_detail
hl7.ORU_R01_PATIENT_RESULT.ORU_R01_ORDER_OBSERVATION.
ORU_R01_OBSERVATION.OBX.obx_1 = "1"
hl7.ORU_R01_PATIENT_RESULT.ORU_R01_ORDER_OBSERVATION.
ORU_R01_OBSERVATION.OBX.obx_2 = "ED"
hl7.ORU_R01_PATIENT_RESULT.ORU_R01_ORDER_OBSERVATION.
ORU_R01_OBSERVATION.OBX.obx_3 = "OWL"
hl7.ORU_R01_PATIENT_RESULT.ORU_R01_ORDER_OBSERVATION.
ORU_R01_OBSERVATION.OBX.obx_5 = mri
hl7.ORU_R01_PATIENT_RESULT.ORU_R01_ORDER_OBSERVATION.
ORU_R01_OBSERVATION.OBX.obx_11 = "F"
# Observ Result Status -- "F" meaning 'Final result'

```

Nosotros vamos a transmitir 2 tipos de archivos dentro de este tipo de mensajes, que incluiremos en el campo *hl7.ORU_R01_PATIENT_RESULT.ORU_R01_ORDER_OBSERVATION.OBU.obx_5* como vemos en el código anterior.

- Imágenes de todo tipo.
- Archivos de texto, scripts, ...

Pese a que se pueden enviar muchos mas tipos de archivos con el código que tenemos generado nos parece importante tratar de adaptar este ejercicio a la práctica médica y enviar archivos parecidos a los que podríamos encontrar para un caso real.

Una pega que tiene nuestra codificación de los mensajes HL7 es que hemos dejado muchos parámetros por defecto. En la práctica real, cada uno de estos parámetros debe estar correctamente rellenado. Hemos decidido dejarlos por defecto para que a la hora de la demostración del funcionamiento, resulte mucho mas fácil y dinámico.

Para comprobar que nuestro mensaje HL7 se ha codificado correctamente podemos recurrir a una función muy útil que nos proporciona esta libreria que es

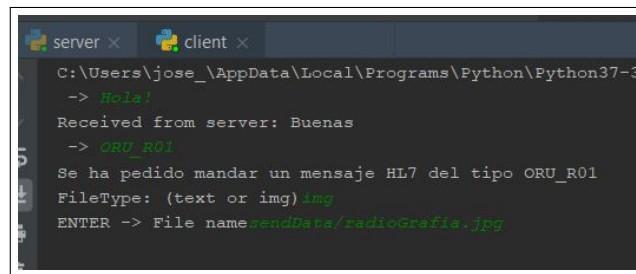
```
hl7.validate()
```

que simplemente nos comprueba si está bien codificado el mensaje.

Una vez enviado y codificado nuestro mensaje HL7 resulta francamente sencillo parsearlo. Podríamos decir que lo que se envía es una especie de *objeto* que la librería HL7apy nos permite parsear con facilidad. Así como decodificar y guardar la información transmitida. Por ejemplo, recuperar y guardar el archivo transmitido:

```
file_like = m.ORU_R01_PATIENT_RESULT.ORU_R01_ORDER_OBSERVATION
            .ORU_R01_OBSERVATION.OBX.obx_5.value
f1 = open("recivedData/hl7image.jpg", "wb")
f1.write(base64.b64decode(str.encode(file_like)))
f1.close()
```

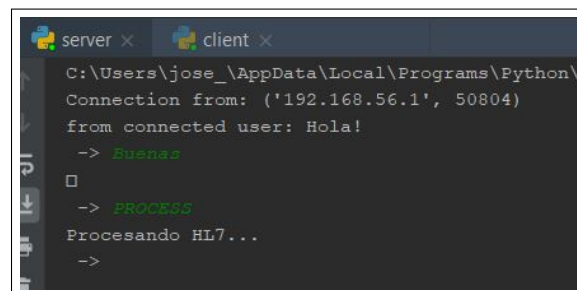
Para intercambiar un mensaje HL7 cuando ejecutamos el cliente y el servidor solo tenemos que enviar como si fuera un mensaje ORU_R01, como vemos en la Figura 7.



```
server x client x
C:\Users\jose\AppData\Local\Programs\Python\Python37-3
-> Hola!
Received from server: Buenas
-> ORU_R01
Se ha pedido mandar un mensaje HL7 del tipo ORU_R01
FileType: (text or img)img
ENTER -> File name sendData/radioGrafia.jpg
```

Figura 7: Ejemplo terminal cliente

Una vez enviado el mensaje HL7, los podremos procesar y guardar la información escribiendo en el servidor **PROCESS**. Tal como se muestra en la Figura 8.



```
server x client x
C:\Users\jose\AppData\Local\Programs\Python\
Connection from: ('192.168.56.1', 50804)
from connected user: Hola!
-> Buenas
-> PROCESS
Procesando HL7...
->
```

Figura 8: Ejemplo terminal servidor

Si vemos los archivos que se han generado en la carpeta **recivedData**, podemos encontrar tanto el archivo transmitido como el mensaje HL7, como vemos en la Figura 9.

Podemos abrir el bloc de notas y comprobar como se ha codificado (Figura 10) el mensaje HL7.



Figura 9: Resultado mensaje

```

message.hl7: Bloc de notas
Archivo Edición Formato Ver Ayuda
MSH|^~\&|img|SendingFac|ReceivingApp|ReceivingFac|20190204214723||ORU^R01^ORU_R01|168715|P|2.5
PID|||15||JOSE
OBR|||10000
OBX|1|ED|OWL||/9j/4AAQSkZJRgABAQAAQABAAD/2wCEAAkGBxITEBITExIVFhUXFRUvGBUwEBUvFxcZFxyXfHuVFXUYHSggGBolGxUVITEhJSkrLi4
SswPmsjScgFY2lZOKtDYQwM9rPipS2Ku1lam0jCXGQATvAnunvUwHVSF0X1wXvabtIDxrui19UHK6RbmNNrKsauLT357lxDwqY+kCia101XZE061NXP9X
9zeBI8jCmJ931KTg4tPHTco+v66X0rOLRHAM8gTKDLs4MNHVM/Xktlf9q/9bVbPtVqQ8g4/JeOz0YbA3CB4LyX/UP2Ro/3p8mH90HMHZbIE9o4lYiVut
erygsdoV67sqQc143r02H5oJM2WtQho1GuSkq14aZDSswCAN+U5mc1C1ywt10CFMGztsP22hblWpgYmkyY0DhxGSD0Waq5wI3eawWHK0du4+0JE5A6Z
SvbRMK+pYyDoqNYg7TowvLq7e1pOVRpZ4nNvqFNyC+crkthVVqdTe1zXeRB+S+imVAQ07iJCDiPsZZbc1o0cT6yF3YwotlgHXGpHTACy3jL9ltm6BAXnt
C2N23wgUajsg40aToLZ9Qgnd9oBdrkPivU3QKPKN9HrcQqNNMkZg5d/eurst6GphIkM3cTG88Ag3SKjTKqgiIiIy6Ybsc6k1zTMAYIzjfmvnmw8KeFy+x

```

Figura 10: Mensaje HL7

6. RDF

Una gran parte de los datos existentes en la web están almacenados en bases de datos relacionales, pero la Web Semántica (2) necesita una nueva estructuración de los mismos. Uno de los objetivos es reestructuralos en la forma de RDFs.

RDF es una manera de expresar relaciones entre conceptos sintácticamente bien definidos y no una manera de definir nuevos elementos sintácticos.

Es un dialecto de XML (define su sintaxis) para especificar, describir y procesar metadatos de recursos (entes que se identifican con URIs) y describir los recursos en función de propiedades simples y sus valores. Ofrece una convención sintáctica y un modelo de datos simple para representar la semántica de los datos procesable por un computador. Permite crear metadatos sobre un documento como un ente individual. Sus datos pueden ser consultados en el lenguaje *SPARQL*.

Es un modelo para representar propiedades etiquetadas y valores de propiedades:

- Recursos: Objetos.
- Propiedades: Relaciones entre objetos.

También están jerarquizados. Teniendo una sintaxis y estructura similar a la de los lenguajes orientados a objetos tiene clases y subclases. El elemento `<rdf:Description>` contiene un atributo `df:about` que se refiere al recurso que se está describiendo. Estas subclases son todas las propiedades del recurso que se está describiendo, y pueden heredar propiedades de las clases (es posible la herencia múltiple). El conjunto de las clases que definen un dominio es un esquema.

Componentes de una tripleta:

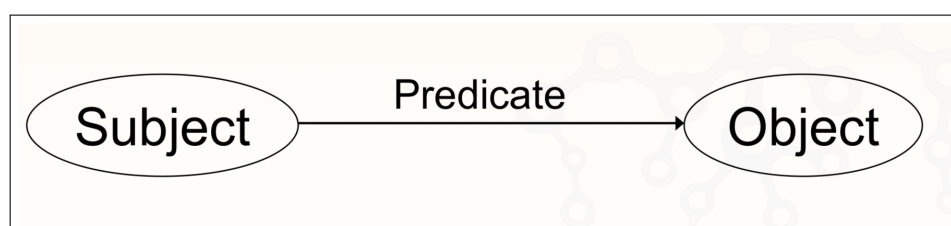


Figura 11: Esquema tripleta RDF

- Sujeto: Referencia RDF URI o nodo vacío.
- Predicado: Referencia RDF URI.
- Objeto: Referencia RDF URI, un literal, o un nodo vacío.

6.1. Modelado de la ontología de nuestra BD en RDF

Para modelar la ontología descrita 3.1 en RDF siguiendo la estructura 4 hemos usado Protégé [9].

1. Crear Clases:

Definimos una clase para cada tabla siguiendo la jerarquía descrita.

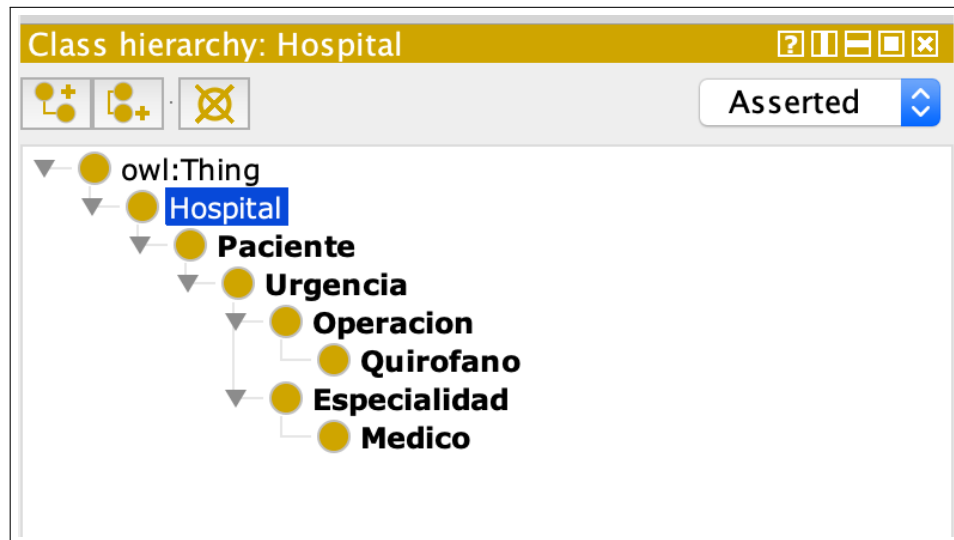


Figura 12: Clases de la ontología.

2. Definir propiedades:

Definimos las propiedades que pueden tener las clases. En nuestro caso la relación de las clases del punto anterior con las propiedades serían: Al Hospital *AcudePaciente* Paciente. Paciente *TieneUrgencia* Urgencia. Urgencia *NecesitaOperacion* Operacion. Operacion *SeRealizaEnQuirofano* Quirofano. Urgencia *PerteneceAEspecialidad* Especialidad.

3. Crear individuos y aplicar el razonador:

Algunos de los individuos que hemos creado usando las propiedades han sido:

MartaPerez (entidad clase Paciente) *tieneUrgencia* (propiedad) *DerrameCerebral* (entidad clase Urgencia). *DerrameCerebral* *PerteneceAEspecialidad* (propiedad) *Neurologia* (entidad clase Especialidad). *Neurologia* *TieneMedico* (propiedad) *MedicoCamachoQuijote* (entidad clase Medico).

JuanRodriguez *tieneUrgencia*. *UrgenciaPorDesangramiento*. *Si NecesitaOperacion*. *SeRealizaEnQuirofano* 34B.

Después de crear los individuos de las clases y relacionarlos adecuadamente usando las propiedades, hemos aplicado el razonador *HermiT* [10] el cual ha

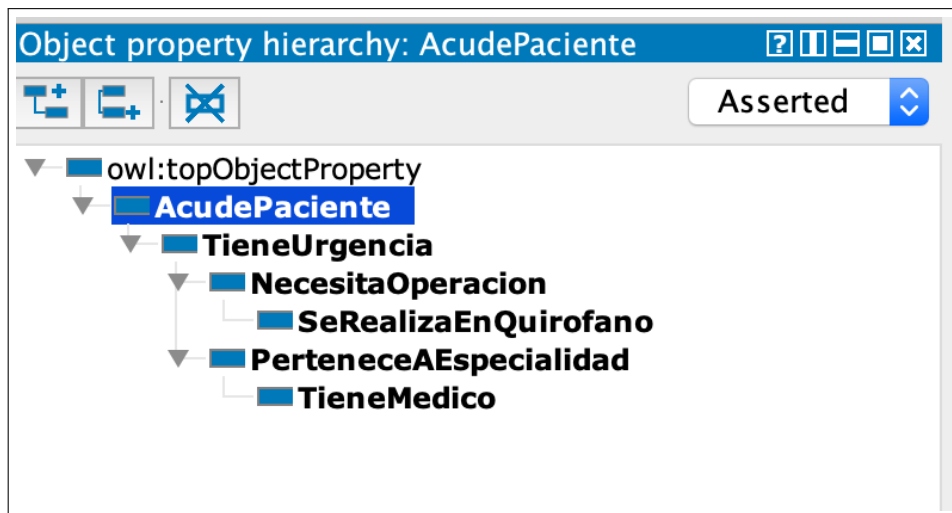


Figura 13: Propiedades de la ontología.

deducido razonamientos coherentes para todos los objetos. Algunos de ellos los podemos ver en pantalla.

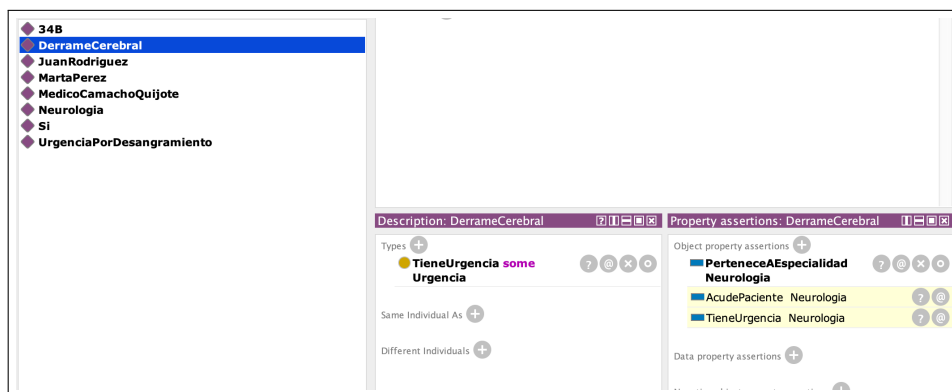


Figura 14: Individuos de la ontología.

7. Conclusiones y trabajos futuros

Como conclusión comentaremos las tareas que hemos realizado así como los trabajos futuros que ofrecen.

■ Conclusiones

- **XML** Para esta tarea además de la parte obligatoria, que consistía en añadir a un XML, un XMLSchema y un XSLT. Hemos generado un base de datos usando pandas y MySQL, reutilizando datasets de *Kaggle*. Con `lxml.etree` hemos automatizado la generación del script de XML a partir de una base de datos. Por lo que consideramos que para esta tarea hemos cumplido correctamente los objetivos propuestos.
- **HL7** En esta segunda parte hemos creado una conexión socket que nos permite intercambiar mensajes. Hemos integrado el mensaje `ORU_R01` de HL7. Como se muestra, no solo funciona correctamente, sino que le hemos añadido funcionalidades extra para que sea capaz de enviar diferentes tipos de archivos, texto, scripts e incluso imágenes. Por lo que consideramos que los objetivos propuestos para esta tarea se han cumplido sobradamente.
- **RDF** Para última tarea hemos definido con *Protégé* la ontología de nuestra base de datos, y hemos incluido algunos individuos para usar el razonador *HermiT*. Aunque el razonador ha inferido razonamientos lógicos y con sentido, no son especialmente interesantes, como los que se ven en las ontologías de referencia de: *pizza* o *HappyFamily*.

■ Trabajos futuros

- **XML** Creemos que se puede mejorar la forma de mostrar los datos. Desde el diseño, utilizando algún template en el XSLT, hasta la forma de estructurarlos. Por ejemplo, que en una tabla apareciesen todos los pacientes, y al hacer click sobre uno, te redirige a otro XML con la información que contiene ese paciente.
- **HL7** Como mejoras para esta tarea creemos que las que más valor aportarían sería generar una interfaz desde la que sea mucho más fácil tratar y comunicarse que la terminal. Además, podríamos aumentar los tipos de mensajes HL7 que se puedan enviar.
- **RDF** Creemos que resultaría muy interesante generar un proceso automatizado para modelar nuestra base de datos completa en RDF, mediante el uso de herramientas como JENA [11].

8. Github

<https://github.com/nairachiclana/HospitalDBWeb>

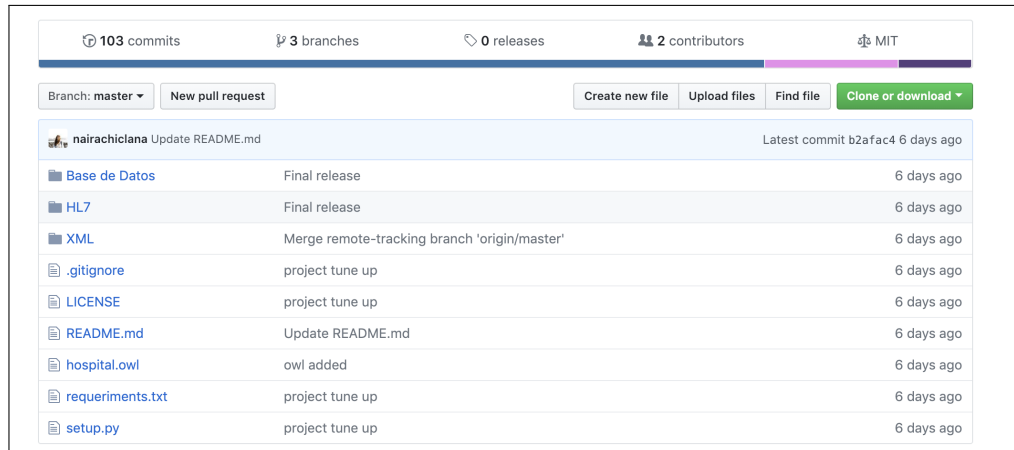


Figura 15: Estructura del repositorio.

Como vemos en la Figura 15, hemos organizado nuestro repositorio en diferentes carpetas para las diferentes secciones del trabajo.

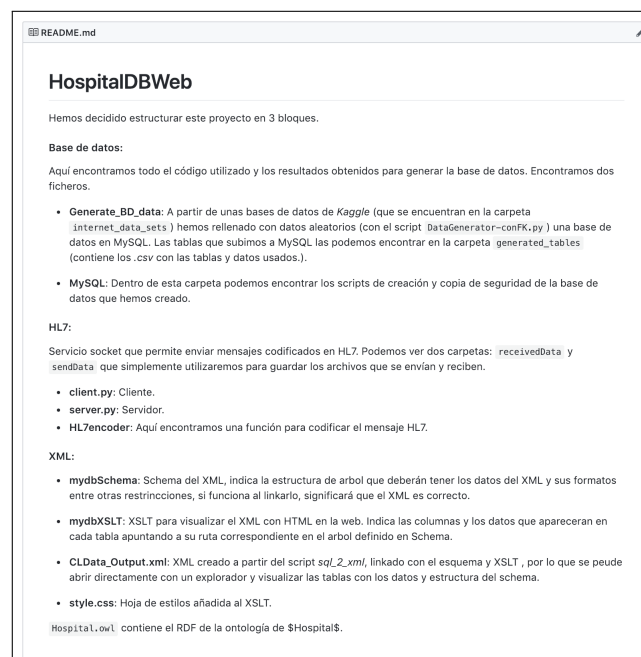


Figura 16: R readme del repositorio.

El *README* (Figura 16) contiene la explicación de que hay en cada carpeta y su función.

También hemos añadido una licencia *MIT* [12] (Figura 17) , un `setup.py` y un archivo `requeriments.txt` con las librerías necesarias para la ejecución del programa.


 nairachiclana/HospitalDBWeb is licensed under the MIT License A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.	Permissions <ul style="list-style-type: none">✓ Commercial use✓ Modification✓ Distribution✓ Private use	Limitations <ul style="list-style-type: none">✗ Liability✗ Warranty	Conditions <ul style="list-style-type: none">④ License and copyright notice
This is not legal advice. Learn more about repository licenses.			

Figura 17: Características de la licencia *MIT*.

9. Webgrafía

Referencias

- [1] Web Semantica. *Introducción a la Web Semántica* https://informatica.cv.uma.es/pluginfile.php/296975/mod_resource/content/5/Semantic%20Web%20and%20Web%202.0.pdf
- [2] Kaggle
- [3] Librería Pandas. <https://pandas.pydata.org>
- [4] W3S. *XML Tutorial* . https://www.w3schools.com/xml/xml_what_is.asp
- [5] LXML Tree Library. `lxml.etree`
- [6] Diapositivas XSL/T Campus virtual. *XSLT* <https://informatica.cv.uma.es/course/view.php?id=3532§ion=1>
- [7] Librería sockets. <https://docs.python.org/3/library/socket.html>
- [8] Librería HL7apy. <https://hl7apy.readthedocs.io/en/latest/>
- [9] Protégé. <https://protege.stanford.edu/>
- [10] HermiT Reasoner. <http://www.hermit-reasoner.com/>
- [11] JENA. <https://jena.apache.org/documentation/>
- [12] MIT License <https://opensource.org/licenses/MIT>