Knowledge Engineering
CSU44D02

Take-Home Assessment

Aaryaman Saini
19323373
4th Year
Computer Engineering
D Stream

Trinity College Dublin

Question 1

(A) The XML Vocabulary I have chosen to develop a DTD is about a School Magazine. It contains articles and information about writers, editors, head, etc. Here, I have created an **external DTD**.

```
<!— File: school_magazine.dtd —>
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT school_magazine (article+)>
<!ENTITY % detail "title, writer_name, head, paragraph, image?, tag">
<!ELEMENT article (%detail)>
<!ATTLIST article author CDATA #REQUIRED>
<!ATTLIST article editor CDATA #IMPLIED>
<!ATTLIST article publish_date CDATA #REQUIRED>
<!ATTLIST article magazine_edition CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT writer_name (#PCDATA)>
<!ATTLIST writer_name gender (male | female) #REQUIRED>
<!ELEMENT head (#PCDATA)>
<!ATTLIST head id CDATA #REQUIRED>
<!ELEMENT paragraph (#PCDATA)>
<!ELEMENT image EMPTY>
<!ATTLIST image src CDATA #REQUIRED>
<!ELEMENT tag (#PCDATA)>
```

As we can see, I have used nesting in school_magazine element and article element. The article element has cardinality of one or more. I have used parameter entity, detail, in the article element. I have used other features of DTD in elements and attributes.

(B) Here is the **XSD** version of DTD that I created in part A.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="school_magazine">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="article" maxOccurs="unbounded" >
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="title" type="xsd:string" />
     <xsd:element name="writer_name">
      <xsd:complexType>
       <xsd:simpleContent>
        <xsd:extension base="xsd:string">
         <xsd:attribute name="gender" use="required">
          <xsd:simpleType>
           <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="male"/>
            <xsd:enumeration value="female"/>
           </xsd:restriction>
          </xsd:simpleType>
         </xsd:attribute>
        </xsd:extension>
       </xsd:simpleContent>
      </xsd:complexType>
     </xsd:element>
     <xsd:element name="head">
      <xsd:complexType>
       <xsd:simpleContent>
        <xsd:extension base="xsd:string">
         <xsd:attribute name="id" type="xsd:string" use="required" />
        </xsd:extension>
       </xsd:simpleContent>
      </xsd:complexType>
     </xsd:element>
```

```
      <xsd:element name="paragraph" type="xsd:string" />
      <xsd:element name="image" minOccurs="0">
       <xsd:complexType>
        <xsd:attribute name="src" type="xsd:string" use="required"/>
       </xsd:complexType>
      </xsd:element>
      <xsd:element name="tag" type="xsd:string" />
     </xsd:sequence>
     <xsd:attribute name="author" type="xsd:string" use="required"/>
     <xsd:attribute name="editor" type="xsd:string"/>
     <xsd:attribute name="publish_date" type="xsd:string" use="required"/>
     <xsd:attribute name="magazine_edition" type="xsd:string" use="required"/>
    </xsd:complexType>
     </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

XSD is more potent than DTD as we can use many of its features to validate our XML document. It supports a lot of data types to check the correctness of the content. It is written in XML. With XSD, we can apply constraints to elements. I have set minimum and maximum occurrences of an element. I have used named types, enumerations, restrictions, cardinalities, sequence, attributes, and empty elements in my XSD for more rigorous validation of an XML document.

(C) Two XML documents that are valid against my XSD are as follows:

```
<!-- Example 1 -->
<?xml version="1.0" encoding="UTF-8"?>
<school_magazine>
	<article author="ron" editor="hermione"  publish_date ="21/11/19"
magazine_edition="M1">
		<title>HelloWorld</title>
		<writer_name gender="male">Ron</writer_name>
		<head id="h1">Snape</head>
		<paragraph>This is a para &amp;amp; it is inside the article.</paragraph>
		<image src="image1.png" />
		<tag>funny</tag>
	</article>
	<article author="harry" publish_date ="21/11/19" magazine_edition ="M1" >
		<title>Hi World</title>
		<writer_name gender="male">Harry</writer_name>
		<head id="h1">Snape</head>
		<paragraph>This is a new para.</paragraph>
		<image src="image2.png" />
		<tag>suspence</tag>
	</article>
</school_magazine >

<!-- Example 2 -->
<?xml version="1.0" encoding="UTF-8"?>
<school_magazine>
	<article author="belatrix" editor="lucius"  publish_date ="21/11/20"
magazine_edition="M2">
		<title>Spell Crucio</title>
		<writer_name gender="female">Belatrix</writer_name>
		<head id="v1">Tom</head>
		<paragraph>&amp;quot; i killed sirius black &amp;quot;</paragraph>
		<tag>horror</tag>
	</article>
```

```
        <article author="remus" publish_date ="21/11/20" magazine_edition ="M2" >
                <title>Save the world</title>
                <writer_name gender="male">Remus</writer_name>
                <head id="v1">Tom</head>
                <paragraph>Voldemort has Dumbledore&amp;apos;s Wand.</paragraph>
                <image src="image3.png" />
                <tag>hope</tag>
        </article>
        <article author="peter" publish_date ="21/11/20" magazine_edition ="M2" >
                <title>Resurrection</title>
                <writer_name gender="male">Peter</writer_name>
                <head id="v1">Tom</head>
                <paragraph>My Lord, I am your most loyal servant.</paragraph>
                <tag>thiller</tag>
        </article>
</school_magazine >
```

Here are **4 partial examples** of XML documents that my XSD labelled as invalid.

```
<!--
Element image can not have character or child items because it is of EMPTY content
type.
-->
<school_magazine>
        <article author="abc" editor="xyz"  publish_date ="christmas"
magazine_edition="first">
                <title>knowledge</title>
                <writer_name gender="female">ABC</writer_name>
                <head id="none">editor and head is same</head>
                <paragraph>some text</paragraph>
                <image src="home">flower.png</image>
                <tag>horror</tag>
        </article>
</school_magazine >
```

<!--
The school magazine expects at least one element article.
-->
<?xml version="1.0" encoding="UTF-8"?>
<school_magazine>
</school_magazine >

<! --
Value ‘boy' of the attribute gender is not valid with respect to enumeration '[male, female]'. It must be a value from the enumeration.
-->
<school_magazine>
	<article author="abc" editor="xyz" publish_date ="christmas" magazine_edition="first">
		<title>knowledge</title>
		<writer_name gender="boy">ABC</writer_name>
		<head id="none">editor and head is same</head>
		<paragraph>some text</paragraph>
		<tag>horror</tag>
	</article>
</school_magazine >

<! -- Attribute publish_date is always required on the element article.-->
<school_magazine>
	<article author="abc" editor="xyz" magazine_edition="first">
		<title>knowledge</title>
		<writer_name gender="male">ABC</writer_name>
		<head id="none">editor and head is same</head>
		<paragraph>some text</paragraph>
		<tag>horror</tag>
	</article>
</school_magazine >

(D) I have used XML document in Example 2 of part C for XPATH expressions.

school_magazine//article

Displays all article elements that are descendants of school_magazine element.

```
Element='<article author="belatrix"
          editor="lucius"
          magazine_edition="M2"
          publish_date="21/11/20">
           <title>Spell Crucio</title>
           <writer_name gender="female">Belatrix</writer_name>
           <head id="v1">Tom</head>
           <paragraph>" i killed sirius black "</paragraph>
           <tag>horror</tag>
      </article>'
Element='<article author="remus" magazine_edition="M2"
publish_date="21/11/20">
           <title>Save the world</title>
           <writer_name gender="male">Remus</writer_name>
           <head id="v1">Tom</head>
           <paragraph>Voldemort has Dumbledore's Wand.</paragraph>
           <image src="image3.png"/>
           <tag>hope</tag>
      </article>'
Element='<article author="peter" magazine_edition="M2"
publish_date="21/11/20">
           <title>Resurrection</title>
           <writer_name gender="male">Peter</writer_name>
           <head id="v1">Tom</head>
           <paragraph>My Lord, I am your most loyal servant.</
paragraph>
           <tag>thiller</tag>
      </article>’
```

//@author

Displays all author attributes.

```
Attribute='author=belatrix'
Attribute='author=remus'
Attribute=‘author=peter'
```

/school_magazine/article[last()-1]

Displays one before the last article element that is the child of school_magazine element.

```
Element='<article author="remus" magazine_edition="M2"
publish_date="21/11/20">
          <title>Save the world</title>
          <writer_name gender="male">Remus</writer_name>
          <head id="v1">Tom</head>
          <paragraph>Voldemort has Dumbledore's Wand.</paragraph>
          <image src="image3.png"/>
          <tag>hope</tag>
     </article>’
```

//image[@*]

Displays all image elements that has at least one attribute of any kind.

```
Element='<image src="image3.png"/>'
```

/school_magazine/article[1]/head | //title

Displays head element of first article element of school_magazine element AND all title elements in the document.

```
Element='<title>Spell Crucio</title>'
Element='<head id="v1">Tom</head>'
Element='<title>Save the world</title>'
Element=‘<title>Resurrection</title>’
```

***/*/child::title**

Displays text of title element that is the grandchild of current element.

```
Spell Crucio
Save the world
Resurrection
```

//article[@author="remus"]/attribute::*

Displays the text of attributes of the article element whose author attribute has value "remus".

```
remus
21/11/20
M2
```

//paragraph//text()

Displays text value of paragraph elements in the document.

```
" i killed sirius black "
Voldemort has Dumbledore's Wand.
My Lord, I am your most loyal servant.
```

(E) The XSLT that can transform XML documents corresponding to my XSD to HTML is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes" />
  <xsl:template match="school_magazine">
    <html>
      <body>
        <h2>The School Magazine
        </h2>
        <ol>
          <xsl:for-each select="article">
            <li>
              <xsl:text>Title "
              </xsl:text>
              <xsl:value-of select="title"/>
              <xsl:text>"
              </xsl:text>
            </li>
            <br>
            <xsl:text>Date:
            </xsl:text>
            <xsl:value-of select=“@publish_date"/>
            </br>
            <br>
            <xsl:text>By:
            </xsl:text>
            <xsl:value-of select="writer_name"/>
            </br>
          <br>
          <xsl:text>Head:
          </xsl:text>
          <xsl:value-of select="head"/>
          </br>
```

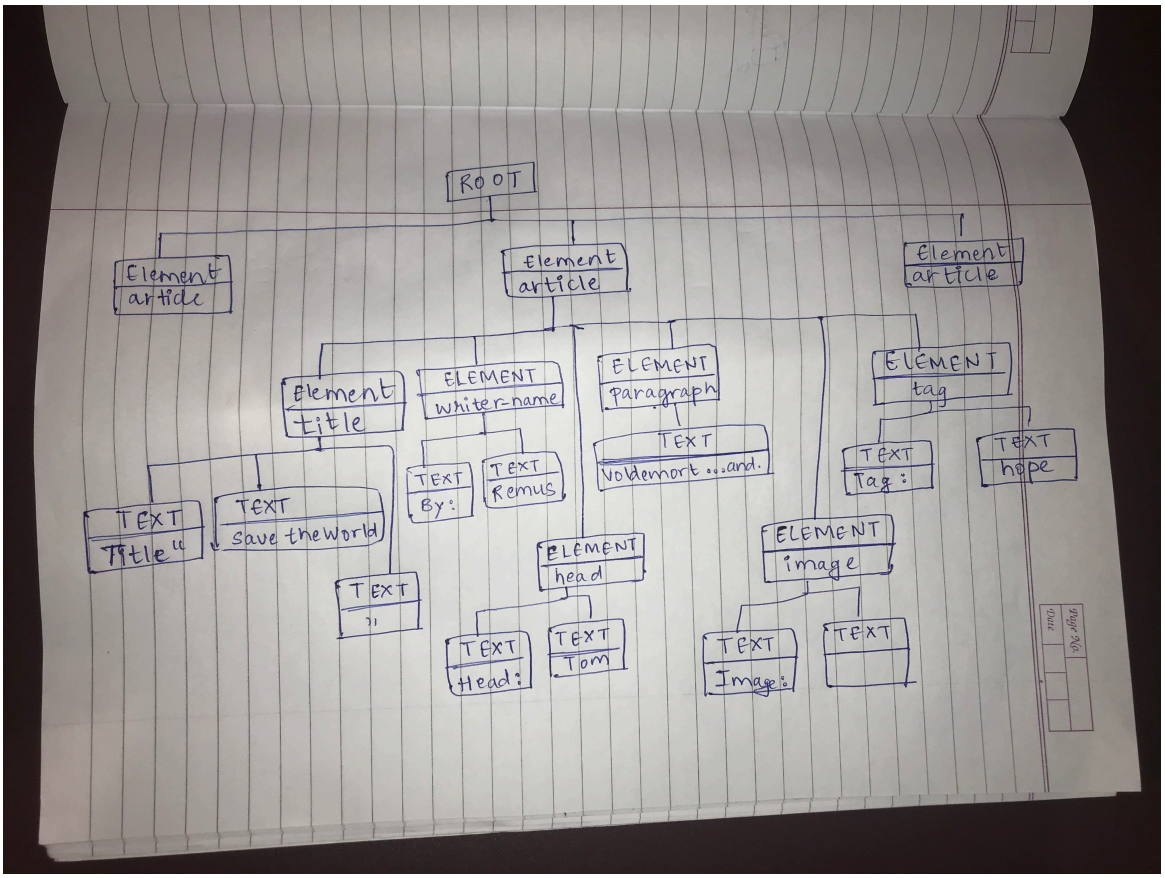
```
      <br>
      <xsl:value-of select="paragraph"/>
      </br>
    <br>
    <xsl:text>Image:
    </xsl:text>
    <xsl:value-of select="image/@src"/>
    </br>
  <br>
  <xsl:text>Tag:
  </xsl:text>
  <xsl:value-of select="tag"/>
  </br>
 </xsl:for-each>
</ol>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

When I gave XML document in example 2 of part C to the XSLT, it gave the following result:

```
<html>
   <body>
      <h2>The School Magazine</h2>
      <ol>
         <li>Title "Spell Crucio"</li>
         <br>Date: 21/11/20</br>
         <br>By: Belatrix</br>
         <br>Head: Tom</br>
         <br>" i killed sirius black "</br>
         <br>Image: </br>
         <br>Tag: horror</br>
         <li>Title "Save the world"</li>
         <br>Date: 21/11/20</br>
         <br>By: Remus</br>
         <br>Head: Tom</br>
         <br>Voldemort has Dumbledore's Wand.</br>
         <br>Image: image3.png</br>
         <br>Tag: hope</br>
```

```
            <li>Title "Resurrection"</li>
            <br>Date: 21/11/20</br>
            <br>By: Peter</br>
            <br>Head: Tom</br>
            <br>My Lord, I am your most loyal servant.</br>
            <br>Image: </br>
            <br>Tag: thiller</br>
        </ol>
    </body>
</html>
```

First, the XSL stylesheet and namespace are applied. The output is defined as HTML. A template pattern is matched with element school_magazine. The generic HTML syntax is used which contains <html> and <body>. Data nodes like <h2>, <ol>, <li>,
 are copied to the result tree. A for-loop is used to get a hold of every article element in the document with select instruction. The attribute publish_date and child elements of article like title, writer_name, head, paragraph, image, and tag, are selected by xsl:value-of. In this way, it produces the above HTML as output. In the following diagram, I have shown a step-by-step illustration of the second article in the XML document.



Here is the HTML page created from the transformation.

The School Magazine

1. Title "Spell Crucio"

   Date: 21/11/20

   By: Belatrix

   Head: Tom

   " i killed sirius black "

   Image:

   Tag: horror
2. Title "Save the world"

   Date: 21/11/20

   By: Remus

   Head: Tom

   Voldemort has Dumbledore's Wand.

   Image: image3.png

   Tag: hope
3. Title "Resurrection"

   Date: 21/11/20

   By: Peter

   Head: Tom

   My Lord, I am your most loyal servant.

   Image:

   Tag: thiller

(F) RDF uses URIs to identify resources. These resources are described with properties and property values by RDF. This triplet combination forms a Statement. The structure of the statement consists of a subject, predicate, and object. The same subject's predicate and object are grouped in a block. Here is an example:

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ar="http://www.school_magazine.fake/article#">
<rdf:Description
rdf:about="http://www.school_magazine.fake/article/HelloWorld">
  <ar:writer_name>Bob</ar:writer_name>
  <ar:head>Alice</ar:head>
  <ar:paragraph>a new day</ar:paragraph>
  <ar:image></ar:image>
  <ar:tag>science</ar:tag>
</rdf:Description>
</rdf:RDF>
```

XSLT facilitates the transformation of XML documents to RDF. No base URI will be generated by default in the transformation. Hence, URIs of every resource is relative. Attributes are transformed into predicates and objects with parent elements as subjects. XML text is transformed into literals that serve as objects. The predicate is rdf:value. The URIs of child elements are constructed from a sequence of ancestor elements. Predicates are the parent element. This is how XSLT may be used to transform documents into RDF.

Question 2

Information retrieval (IR) is finding unknown information by describing it. It can be thought of as a paradox. After 1975, information retrieval started to make a clear picture by introducing technologies like the WWW search engine. There are some challenges in information retrieval. Researches have found ways to combat them, but it is still prevalent. They are looking for better ways to organise information objects and filter them according to users' needs. These hardships exist since data is volatile, abundant, distributed, unstructured, and heterogeneous. IR is different from data access. SQL retrieves data, not information. So, this is why IR is hard to tackle. We need information on a thing that we are not sure about. It might seem like we have come a long way in IR over the past 20 years, but researchers claim that the problem of IR is only 10% solved.

Web search is on a gigantic increase since its invention. Many people have attempted to help users with a web search, and now they have made businesses out of it, such as Google and Bing. They are leading in the market of web search. Google has 86% web search followed by Yahoo and Bing with 6% and 4% respectively. The pace at which web pages are being created and updated creates a problem that many sites are not indexed, a phenomenon known as the invisible web. Web search is also influenced by monetary contributions and spamming. Google is leading with a massive margin in web search because no one can buy a higher page rank. It aims to provide high-quality websites with information relevant to our needs. The algorithms used in web search to deliver the most relevant information to the user are Hyperlink analysis, HITS, and PageRank. Hyperlink algorithm is a type of recursive algorithm that measures popularity and relevance based on link measures on a site. HITS is an algorithm that checks pages by their importance. Popular pages that become hubs and authorities, both are checked for relevance. Then there is the PageRank algorithm. It is Google's basis for ranking pages. In the PageRank algorithm, if the sum of ranks of pages pointing at it is high, then the page's rank is high. It measures a link score and looks at the anchor text around the links. PageRank is a complex algorithm as it incorporates over 100 factors for ranking and relevance. Google uses a combination of HITS and anchor text. It aims for high precision than recall. Thus, it seems like hubs, authorities and link texts, and identifying scammers are the areas to improve web search.

Semantics is defined as a well-structured meaning that computers can understand. Syntax and semantics are different things. We need the syntax to express semantics. RDF is one of the syntaxes for the Semantic Web. Information on the web is unstructured, which is a limitation in current web search. Making interconnections between different documents based on the context of the query is hard. The Semantic Web will be the future. The current Web 3.0 evolved from the PC Era, Web 1.0, and Web 2.0. Web 4.0 will be the next generation of the World Wide Web. It will be the WebOS. The traditional web was a web of documents. Limitations of current web search are:

- Low precision web search results
- Results being susceptible to vocabulary
- Single web page results
- Unstructured publishing content

Data on the web is readily available, and now we rely on this data for our applications. The next step in the evolution of the web is semantic interoperation that is understanding and linking the data and developing intelligent applications that automate support for data integration. So, Semantic Web technologies enter the "big game" to provide an infrastructure for the data on the web. The semantic-based approach is so important because the boundaries between data from multiple sectors are becoming a blur. In layman's language, it is a web of data that machines too can understand. As simple as this approach may sound, it is way more complicated in implementation. Machines have trouble combining several sources of information. They can read information but not understand it. Semantic Web has several technologies to make integration of Web Data possible.

1. Structured Web Documents: XML provides domain-specific markup that helps to encode documents in a serialised structured manner.
2. Describe Web Documents: RDF is the framework for interchanging metadata. It can integrate information from multiple resources and provides machine-understandable semantics. RDF uses URIs to identify web resources. URIs which are used for accessing resources on WWW are URLs. RDFs can not constrain the graphs too much because it is an untyped mechanism. This leads to difficulties in interpretation and scaling. RDFS is an extension of RDF with schema vocabulary. RDFS also has problems. It is too weak to describe resources in sufficient detail, and second, it does not have native reasoners for nonstandard semantics.

3. Web Ontology Languages: Ontology is a specification of a conceptualisation. It represents an area of knowledge by describing vocabulary and relationships between concepts. Ontologies are essential for search, exchange, and discovery. OWL is an ontology web language. Ontology is different from relational schema because it defines relationships that are interpretable by both humans and programs. It is a graph of explicitly mentioned domains.
4. Logic is reasoning, and drawing conclusions is making inferences. Logic has a vital role in the statement of queries. It is used to trace, evaluate rules and infer facts. Rule languages like RIF, RuleML, SWIRL are used for expressing rules. Reasoning tools like Jena are needed to handle rules and reasons about the data.
5. Searching Query Language, for example, SPARQL, is a declarative query language used to extract information based on pattern matching from RDF graphs.
6. Repositories: The massive amount of data the Semantic Web creates, semantic repositories are used to manipulate Semantic Web data efficiently. Sesame is a Semantic Repository Engine that supports RDF(S).
7. Semantic Web Services: A web service is a network-accessible interface. It is implemented using WSDL, REST. They connect devices and computers using the internet to exchange data. Semantic Web Services are web services that have been automated using semantic web technology.
8. Intelligent Software Agents: Agents are capable of interacting with other agents to exchange data. They designed to do some autonomous action. Semantics are needed to seek information, represent logic, and support agent communication and negotiation.
9. Trust and Proof: Trust is confined to identity established in the form of digital certificates and authentication. We refer to the answer from querying in the semantic web as to whether correct or not, which we refer to as proof.
10. Social Web: It allows people to connect over the internet by providing a layer of abstractions.
11. Applications: Examples include eBusiness, eCommerce, Supply Chain Management

There are semantic technologies for unstructured data that are related to NLP. These include entity extraction, cluster analysis, classification, dependency identification, coreference resolution, and automatic summarisation. GATE is a tool that is used for deploying those technologies.

At the heart of the Semantic Web is where Linked Data is situated. It aims to expose datasets on the Web and set links among data items from different datasets. RDF is a technology that can realise the principles in Linked Data. Linked Data is different from Open Data. Linked Data is 5 stars. On the other hand, Open Data is three stars. The data in Semantic Database is Linked Data. The datasets are read-only. Semantic Web technologies, Linked Data principles, and practices should play a significant role in publishing and using Data on the Web.

- Despite having so many advantages, the Semantic Web still has a few problems.
- Little availability of Semantic content.
- Extensive efforts are needed for the creation of ontologies used in the Semantic Web.
- Necessary mechanisms are needed to store and access content in a scalable manner.
- Multilingualism is a challenge in the current Web and has to be tackled in Semantic web as well.
- New visualisations are needed to analyse information, something that is increasingly overloading properly.
- Stability of Semantic Web Technologies.

The Semantic Web has an economic impact. There can be many types of products and services that can only be possible with Semantic Web technologies. One service can be content providers upgrading their existing content to the Semantic Web. However, the feasibility of such commercial scenarios is limited because delivering such services can be done with sector-specific ontologies. Their development is an ongoing process, and improvements can be expected in the next 5 years. The Semantic annotation will be incorporated in all types of end user applications. Also, the first commercial application of Semantic Web technology is foreseen in intranets. However, many issues have to be solved before Semantic Web starts getting exploited at the same level as the current web. Companies running on HTML will face a risk as Semantic Web takes over. The migration to the semantic web is inevitable, and we will know the migration is complete when the semantic web is simple called the web. When we have an entire Semantic web, we will find direct answers to questions and not the relevant documents where we may or may not find our answers. I believe this reduces the spread of fake news. A future where web information has an exact meaning and computers can understand that, and computers can integrate information from the web is a semantic web vision.

Bibliography

1. https://www.w3schools.com/xml/
2. Richard Benjamins, Jesus Contreras, Oscar Corcho. Six challenges for the Semantic Web. April 2002. URL: https://www.researchgate.net/publication/49911494_Six_challenges_for_the_Semantic_Web
3. F. Breitling. A standard transformation from XML to RDF via XSLT. August 2009. URL: https://www.researchgate.net/publication/45856423_A_standard_transformation_from_XML_to_RDF_via_XSLT
4. The Semantic Web - Trinity College Dublin. URL: https://www.scss.tcd.ie/Owen.Conlan/CS7063/Semantic%20Web%20-%20extended.pdf