

Introduction to the Semantic Web, XML & RDF(S)

Πληροφοριακά Συστήματα Διαδικτύου

How the Web is Today?

- WWW is an impressive success:
 - amount of available information (> 1 Giga pages)
 - number of human users (> 200 Mega users)
- Information and its presentation are mixed up in the form of HTML documents
 - all intended for human consumption
 - many generated automatically by applications
- Easy to fetch any Web page, from any server, any platform
 - access through a uniform interface

Semantic Web: the vision

- We've only seen **two generations**:
 - handwritten HTML
 - dynamically generated pages
- The real power will come with the **3rd generation**:
 - machine accessible semantics
 - machine-accessible meaning of information (reasoning services)

Semantic Web: the vision

- The “Next Generation Web” aims to provide infrastructure for expressing information in a **precise, human-readable, and machine-interpretable** form
- Enable both **syntactic** and **semantic interoperability** among independently-developed Web applications, allowing them to efficiently perform sophisticated tasks for humans
- Enable Web resources (data & applications) to be **accessible by their meaning** rather than by keywords and syntactic forms
 - Conceptual Navigation & Querying
 - Inference Services

Semantic Web: the vision

- The aim of the Semantic Web is to allow much more **advanced knowledge management systems**:
 - Knowledge will be organised in conceptual spaces according to its meaning.
 - Automated tools will support maintenance by checking for inconsistencies and extracting new knowledge.
 - Keyword-based search will be replaced by query answering: requested knowledge will be retrieved, extracted, and presented in a human friendly way.
 - Query answering over several documents will be supported.
 - Definition of views on certain parts of information (even parts of documents) will be possible.

Impossible (?) using the Syntactic Web...

- Complex queries involving **background knowledge**
 - Find information about “animals that use sonar and are either bats or dolphins”
- Locating information in **data repositories**
 - Travel enquiries
 - Prices of goods and services
 - Results of human genome experiments
- Finding and using “**web services**”
 - Visualise surface interactions between two proteins
- Delegating complex tasks to web “**agents**”
 - Book me a holiday next weekend somewhere warm, not too far away, and where they speak French or English

What is the Problem?

- Consider a typical web page
 - Markup consists of:
 - rendering information (e.g., font size and colour)
 - Hyper-links to related content
 - Semantic content is accessible to humans but not (easily) to computers...

What information can we see...

WWW2002

The eleventh international world wide web conference

Sheraton waikiki hotel

Honolulu, hawaii, USA

7-11 may 2002

Registered participants coming from

australia, canada, chile denmark, france, germany, ghana, hong kong, india, ireland, italy, japan, malta, new zealand, the netherlands, norway, singapore, switzerland, the united kingdom, the united states, vietnam, zaire

On the 7th May Honolulu will provide the backdrop of the eleventh international world wide web conference. This prestigious event ...

Speakers confirmed

Tim berners-lee

Tim is the well known inventor of the Web, ...

Ian Foster

Ian is the pioneer of the Grid, the next generation internet ...

But What About...

<conf>☎☎☎📁📁📁

※ℳ ℳ•ℳ❖ℳ■◆ℳ ✕■◆ℳ□■🔍◆✕□■🔍● ◆□□●☎ ◆✕☎ℳ ◆ℳ🌀ℳ□■</conf>

<place>🔍ℳ□🔍□■ ◆🔍✕✕✕ ✕□◆ℳ●

☎□□●◆●◆📁 ℳ🔍◆🔍✕✕📁 †🔍🌀</place>

<date>📁📁📁📁 ○🔍📁 📁📁📁📁</date>

<participants>☎ℳ🌀✕◆◆ℳ□ℳ☎ □🔍□◆✕ℳ✕□🔍◆◆ ℳ□○✕■🌀 ✕□□○

🔍◆◆◆□🔍●✕🔍📁 ℳ🔍🔍🔍☎🔍📁 ℳℳ✕●ℳ ☎ℳ■○🔍□&📁 ✕□🔍■ℳ📁

🌀ℳ□○🔍📁📁 🌀ℳ🔍🔍🔍🔍 ℳ□■🌀 &□■🌀📁 ✕■☎✕🔍📁

✕□ℳ●🔍🔍📁 ✕◆🔍●📁📁 📁🔍🔍🔍📁 ○🔍●◆🔍📁 ■ℳ◆ ✕ℳ🔍●🔍🔍📁

◆ℳℳ ■ℳ◆ℳℳ□●🔍🔍📁📁 ■□□◆🔍📁◆ ◆✕■🌀🔍🔍🔍📁

◆◆✕◆✕ℳ□●🔍🔍📁◆ ◆ℳℳ ◆■✕◆ℳ☎ &✕■🌀☎□○📁◆ ◆ℳℳ ◆■✕◆ℳ☎

◆◆🔍◆ℳ◆📁 ◆✕ℳ◆■🔍○📁 ℳ🔍✕□ℳ</participants>

<introduction>☎ℳ🌀✕◆◆ℳ□ ■□◆

📁 ■ ◆ℳℳ 📁◆ℳ ☎🔍📁 ☎□□□●◆●◆ ◆✕●● □□□❖✕☎ℳ ◆ℳℳ

🌀🔍ℳ&☎□□□ □✕ ◆ℳℳ ℳ•ℳ❖ℳ■◆ℳ ✕■◆ℳ□■🔍◆✕□■🔍● ◆□□●☎

◆✕☎ℳ ◆ℳ🌀 ℳ□■✕ℳ□ℳ■ℳ📁📁 ※ℳ✕ □□ℳ◆✕🌀✕□◆◆ ℳ❖ℳ■◆ ⑤

☎□ℳ🔍&ℳ□◆ ℳ□■✕✕□○ℳ☎</introduction>

<speaker>※✕○ 🌀ℳ□■ℳ□◆📁●ℳℳ</speaker>

<bio>※✕○ ✕◆ ◆ℳℳ ◆ℳ●● &■□◆■ ✕■❖ℳ■◆□□ □✕ ◆ℳℳ ✕ℳ🌀📁...

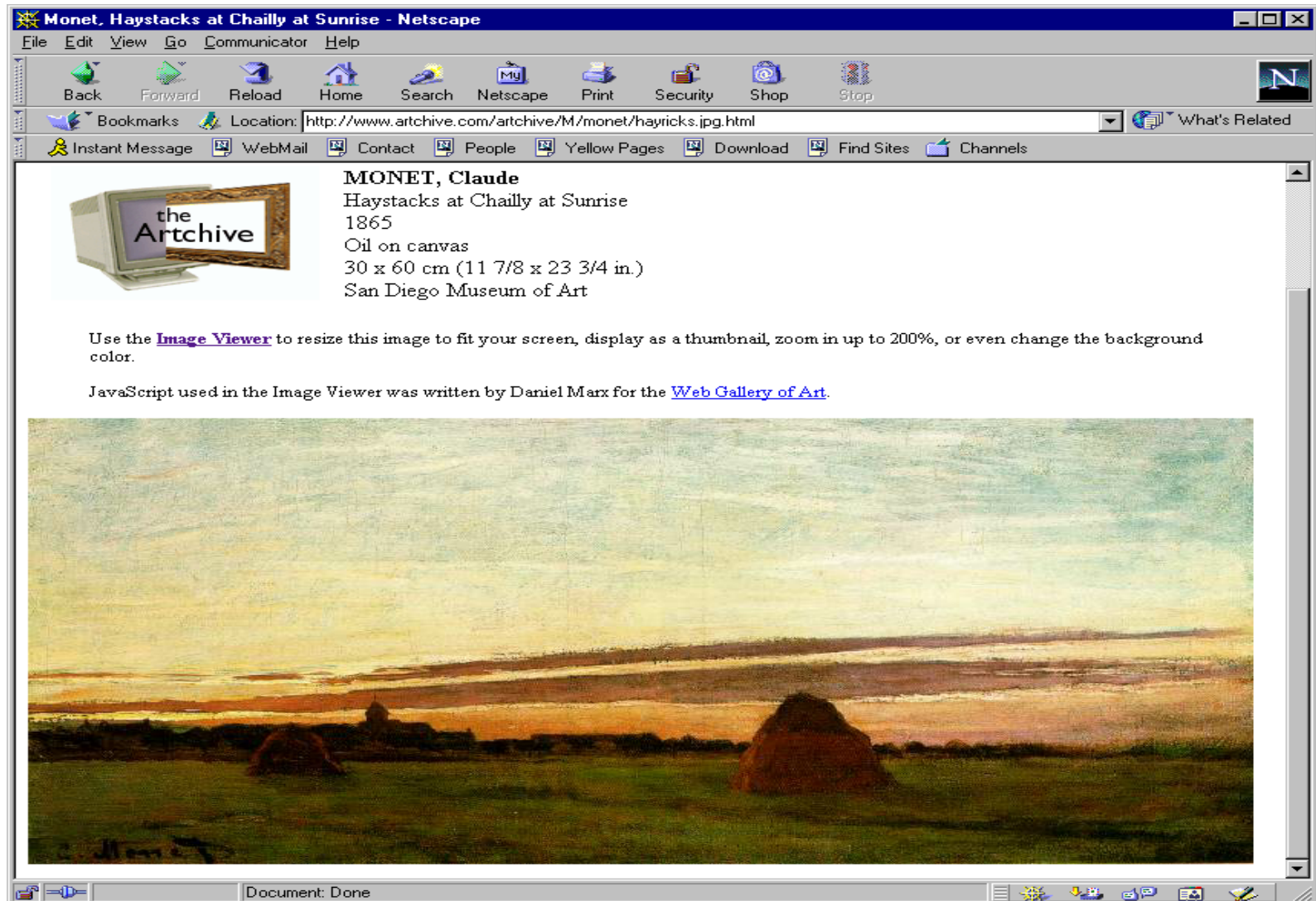
Need to Add “Semantics”

- **External agreement** on meaning of annotations
 - E.g., Dublin Core
 - Agree on the meaning of a set of annotation tags
 - Problems with this approach
 - Inflexible
 - Limited number of things can be expressed
- Use **Ontologies** to specify meaning of annotations
 - Ontologies provide a vocabulary of terms
 - New terms can be formed by combining existing ones
 - Meaning (**semantics**) of such terms is formally specified
 - Can also specify relationships between terms in multiple ontologies

A Semantic Web – First Steps

- Make web resources more accessible to **automated processes**
- Extend existing rendering markup with **semantic markup**
 - Metadata annotations that describe content/function of web accessible resources
- Use Ontologies to provide **vocabulary** for annotations
 - “Formal specification” is accessible to machines
- A prerequisite is a standard web ontology language
 - Need to agree common **syntax** before we can share semantics
 - Syntactic web based on **standards** such as **HTTP** and **HTML**

HTML Document Presentation




Monet, Haystacks at Chailly at Sunrise - Netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop


Location: <http://www.artchive.com/artchive/M/monet/hayricks.jpg.html> What's Related

Instant Message WebMail Contact People Yellow Pages Download Find Sites Channels

 **MONET, Claude**
Haystacks at Chailly at Sunrise
1865
Oil on canvas
30 x 60 cm (11 7/8 x 23 3/4 in.)
San Diego Museum of Art

Use the [Image Viewer](#) to resize this image to fit your screen, display as a thumbnail, zoom in up to 200%, or even change the background color.

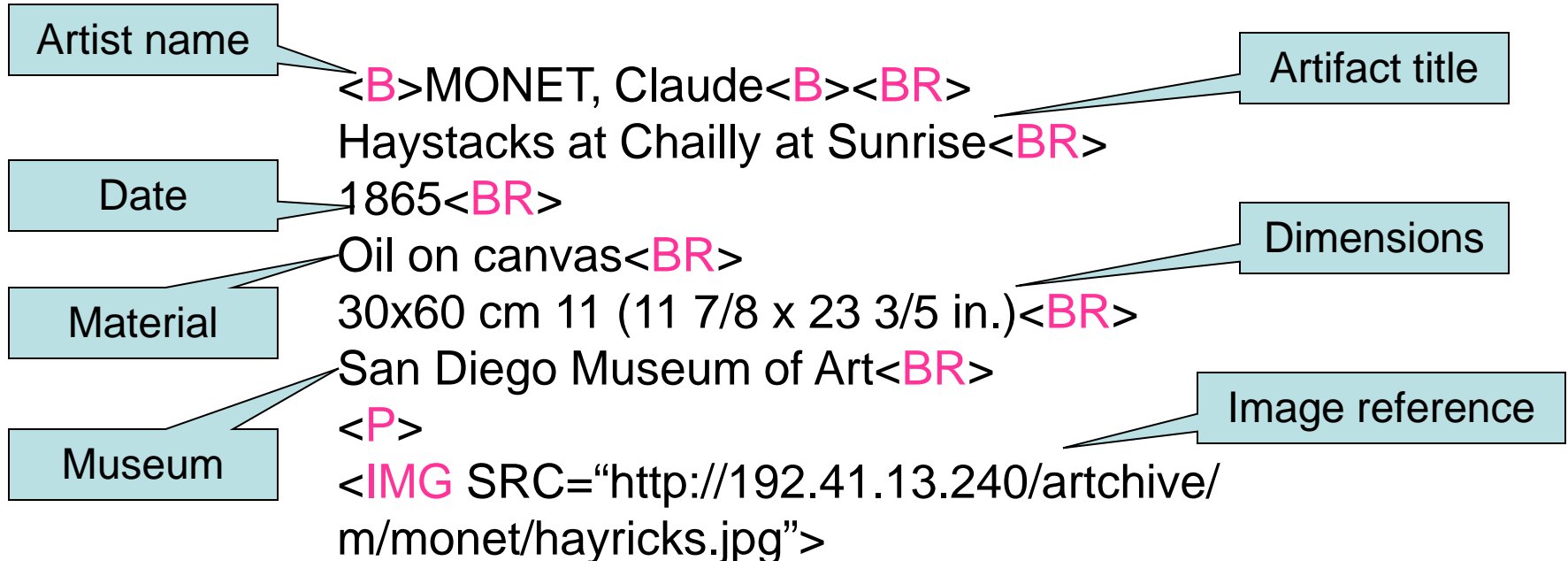
JavaScript used in the Image Viewer was written by Daniel Marx for the [Web Gallery of Art](#).



Document: Done

What's wrong with HTML

- HTML may reflect document presentation, but it cannot adequately represent semantics & structure of data.



But Modern Web Applications Need More!

- **Infomediaries:**
 - Community Web Portals
 - Digital Museums & Libraries
- **Electronic commerce:**
 - On-line Catalogs & Procurement
 - Comparison Shoppers
 - Market Places
 - Virtual Enterprises
- **Scientific applications:**
 - E-learning
 - Data & Knowledge Grids
- **Advanced Information Management**
 - finding,
 - extracting,
 - representing,
 - interpreting,
 - maintaining
- **Flexible, Quick Interoperation:** the ability to uniformly **share**, **interpret** and **manipulate heterogeneous** information
 - applications cannot consume HTML

XML Data Representation

- A possible XML markup of the same information will retain the structure (and the semantics) of the various data objects

```
<ARTIST>
  <NAME><FIRST>Claude</FIRST><LAST>Monet</LAST></NAME>
  <ARTWORK>
    <ARTIFACT>
      <TITLE>Haystacks at Chailly at Sunrise</TITLE>
      <DATE>1865</DATE>
      <MATERIAL>Oil on canvas</MATERIAL>
      <DIM Metric='cm'>
        <HEIGHT>30</HEIGHT><WIDTH>60</WIDTH></DIM>
      <DIM Metric='in'>
        <HEIGHT>11 7/8</HEIGHT><WIDTH>23 3/4</WIDTH></DIM>
      <LOCATION>San Diego Museum of Art</LOCATION>
      <IMAGE File='http://192.41.13.240/artchive/m/monet/hayricks.jpg'/>
    </ARTIFACT>
  </ARTWORK>
</ARTIST>
```

Introduction to XML

What is XML?

- XML stands for **E**Xtensible **M**arkup **L**anguage
- XML is a **markup language** much like HTML
- XML was designed to **describe data**
- XML tags are not predefined. You must **define your own tags**
- XML uses a **Document Type Definition (DTD)** or an **XML Schema** to describe the data
- XML with a DTD or XML Schema is designed to be **self-descriptive**

The main difference between XML and HTML

- XML was designed to carry data.
- XML is not a replacement for HTML. XML and HTML were designed with different goals:
 - XML was designed to describe data and to focus on what data is.
 - HTML was designed to display data and to focus on how data looks.
- HTML is about displaying information, while XML is about describing information.

XML does not do anything on its own

- XML was not designed to do anything on its own.
- Maybe it is a little hard to understand, but XML does not do anything. XML was created to structure information.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>George</to>
  <from>Marios</from>
  <heading>Reminder</heading>
  <body>Don't forget your appointment!</body>
</note>
```

- The note has a header and a message body. It also has sender and receiver information. But still, this XML document does not do anything. It is just pure information wrapped in XML tags. Someone must write a piece of software to send, receive or display it.

XML is free and extensible

- XML tags are not predefined. You must "invent" your own tags.
- The tags used to mark up HTML documents and the structure of HTML documents are predefined. The author of HTML documents can only use tags that are defined in the HTML standard (like `<p>`, `<h1>`, etc.).
- XML allows the author to define his own tags and his own document structure.
- The tags in the example above (like `<to>` and `<from>`) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

XML is used to Exchange, Store and Share Data

- With XML, data can be exchanged between incompatible systems.
- In the real world, computer systems and databases contain data in incompatible formats. One of the most time-consuming challenges for developers has been to exchange data between such systems over the Internet.
- Converting the data to XML can greatly reduce this complexity and create data that can be read by many different types of applications.
- Since XML data is stored in plain text format, XML provides a software- and hardware-independent way of sharing and storing data.

XML was not designed for the Web

- XML is in no way a successor to HTML
- XML is not tied to the Web – it should not be considered as a Web technology
- XML is primarily used for information exchange
- Its true power lies in its flexibility and portability (platform-independence)

XML is everywhere

- How does a technology spread?
 - By gaining wide acceptance, becoming a standard
- In what way is XML everywhere?
 - XML parsers exist for most programming languages and software technologies
- Built-in support for XML makes it very easy to use it for data exchange

Technologies using XML

- **Semantic Web** – introducing semantics to the Web
- **Web Services** – distributed computing
- **The Grid** – distributed computing
- **VRML** – creating virtual worlds
- **SVG** – image exchange format
- **Ant** – creating build files for programs
- **XRL** – composing workflows
- Web/Application server configuration files
- . . .

XML Syntax

- The **first line** in the document - the XML declaration - defines the XML version and the character encoding used in the document (such as ISO-8859-1, UTF-8 etc). The character encoding is not mandatory.

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

- The next line describes the **root element** of the document:

```
<note>
```

- The syntax for writing **comments** in XML is similar to that of HTML.

```
<!-- This is a comment -->
```

XML elements

- An element consists of an **opening tag**, its **content**, and a **closing tag**. For example:

```
<heading>Reminder</heading>
```
- Tag names can be chosen almost freely, there are very few restrictions. The most important ones are that the first character must be a letter, an underscore or a colon; and that no name may begin with the string “xml” in any combination of cases (such as “Xml” and “xML”).

XML elements

- The **content** may be text, or other elements, or nothing. It is illegal to omit the closing tag (e.g. like `<p>` in HTML). Unlike HTML, XML tags are case sensitive.

```
<Name>  
    <First>Marios</First>  
    <Last>Marios</Last>  
</Name>
```

- If there is no content then the element is called **empty**.

```
<heading></heading> OR <heading/>
```

Attributes

- An empty element is not necessarily meaningless, because it may have some properties in terms of **attributes**. An attribute is a name-value pair inside the opening tag of an element.

```
<heading title="Reminder"/>
```

- The same information could be written replacing attributes by nested elements.

Attributes

- As in HTML, in XML attributes provide additional information about elements:

```

```

```
<a href="demo.html">
```

- Attribute values must always be enclosed in quotes, but either single or double quotes can be used.
- **Note:** If the attribute value itself contains double quotes it is necessary to use single quotes, like in this example:

```
<pilot name='John "The Fox" Carter'>
```


Avoid using attributes?

- Here are some of the **problems** using attributes:
 - attributes cannot contain multiple values (child elements can)
 - attributes are not easily expandable (for future changes)
 - attributes cannot describe structures (child elements can)
 - attributes are more difficult to manipulate by program code
 - attribute values are not easy to test against a Document Type Definition (DTD) - which is used to define the legal elements of an XML document
- If you use attributes as containers for data, you end up with documents that are difficult to read and maintain. Try to use **elements** to describe data. Use attributes only to provide information that is not relevant to the data.

The “correct” way ?

- A date attribute is used in the first example:

```
<note date="12/11/2002">
```

- A date element is used:

```
<note>  
  <date>12/11/2002</date>
```

- An expanded date element is used:

```
<note>  
  <date>  
    <day>12</day>  
    <month>11</month>  
    <year>2002</year>  
  </date>
```

Example

- Imagine that this XML document describes the book:

```
<book>
  <title>My First XML</title>
  <prod id="33-657" media="paper"></prod>
  <chapter>Introduction to XML
    <para>What is HTML</para>
    <para>What is XML</para>
  </chapter>
  <chapter>XML Syntax
    <para>Elements must have a closing tag</para>
    <para>Elements must be properly nested</para>
  </chapter>
</book>
```

- Book is the **root element**. Title, prod, and chapter are **child elements** of book. Book is the **parent element** of title, prod, and chapter. Title, prod, and chapter are **siblings** (or **sister elements**) because they have the same parent.

Well-formed XML documents

- An XML document is **well-formed** if it is syntactically correct. Some syntactic rules are:
 - There is only one outermost element in the document (root element).
 - Each element contains an open and a corresponding closing tag.
 - Tags may not overlap, as in

```
<author><name>John Smith</author></name>
```
 - Attributes within an element have unique names.
 - Element and tag names must be permissible.

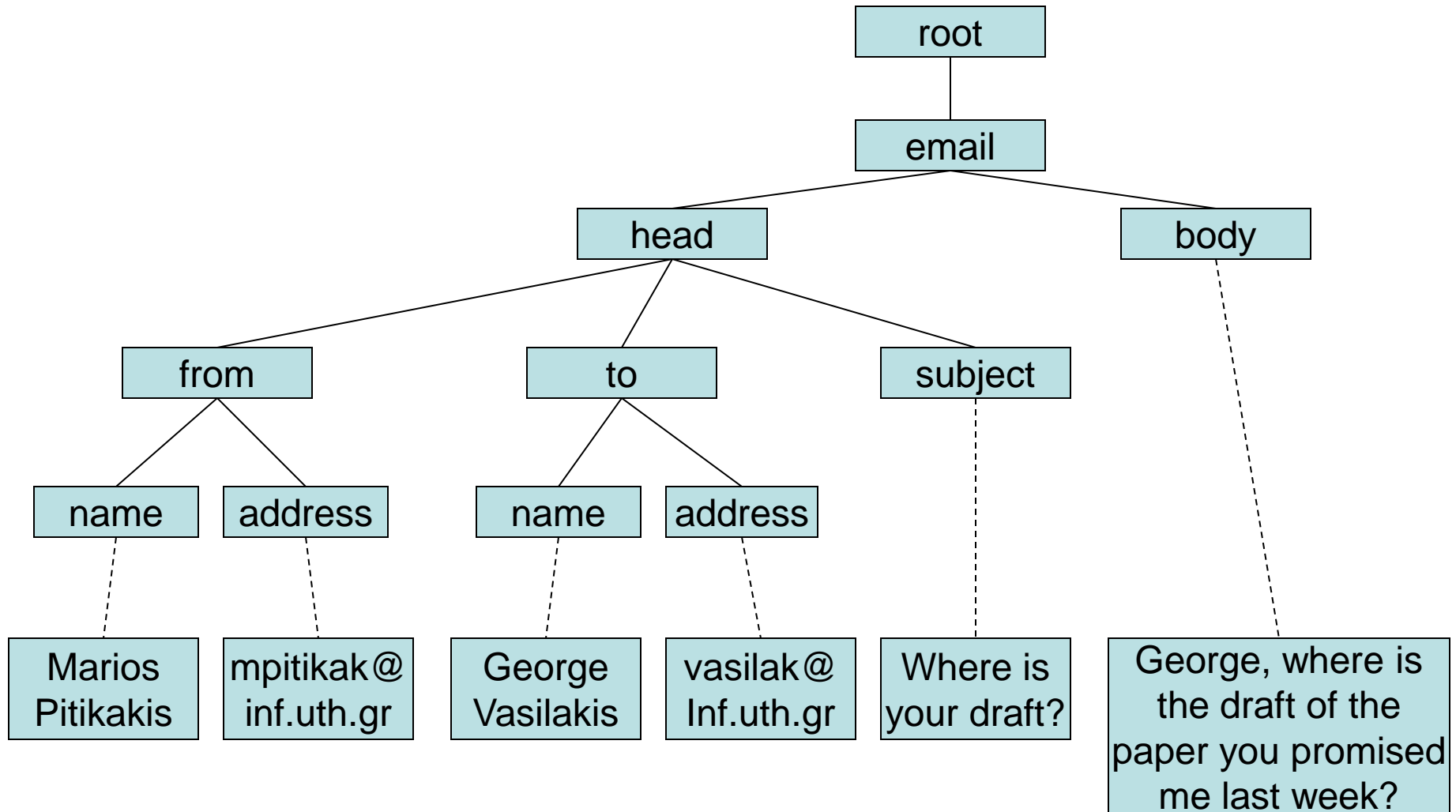
The tree model of XML documents

- It is possible to represent well-formed XML documents as trees, thus trees provide a formal data model for XML.
- This representation is often instructive. As an example, consider the following document:

The tree model of XML documents

```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE email SYSTEM "email.dtd">
<email>
  <head>
    <from name="Marios Pitikakis" address="mpitikak@inf.uth.gr"/>
    <to name="George Vasilakis" address="vasilak@inf.uth.gr"/>
    <subject>Where is your draft?</subject>
  </head>
  <body>
    George, where is the draft of the paper
    you promised me last week?
  </body>
</email>
```

The tree model of XML documents



The tree model of XML documents

- It is an ordered labeled tree. So:
 - There is exactly one root.
 - There are no cycles.
 - Each node, other than the root, has exactly one parent.
 - Each node has a label.
 - The order of elements is important.
- However we should note that while the order of elements is important, the order of attributes is not. So, the following two elements are equivalent:
 - `<person lastname="Smith" firstname="John"/>`
 - `<person firstname="John" lastname="Smith"/>`

XML Data Representation

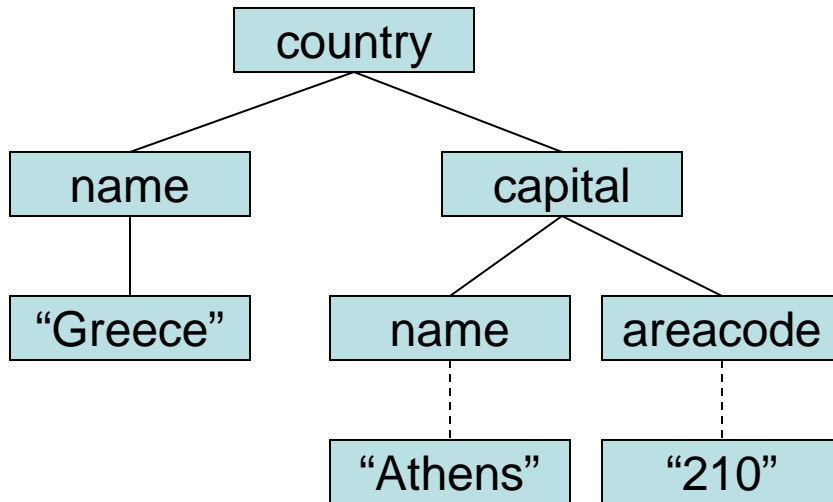
```
<country name="Greece">  
  <capital name="Athens">  
    <areacode>210</areacode>  
  </capital>  
</country>
```

- **Syntax:**

- angle brackets, elements and attributes, etc.

- **Data model:**

- ordered, labeled tree



Structuring

- No agreement on:
 - structure
 - is **country** a:
 - object?
 - class?
 - attribute?
 - relation?
 - something else?
 - what does nesting mean?
 - vocabulary
 - is **country** the same as **nation** ?

```
<country name="Greece">  
  <capital name="Athens">  
    <areacode>210</areacode>  
  </capital>  
</country>
```

```
<nation>  
  <name>Greece</name>  
  <capital>Athens</capital>  
  <capital_areacode>  
    210  
  </capital_areacode>  
</nation>
```

- Are the above XML documents the same?
- Do they convey the same information?
- Is that information machine-accessible?

Structuring: DTDs and XML Schema

- An XML document is well-formed if it respects certain syntactic rules. However those rules say nothing specific about the **structure** of the document.
- Now imagine two applications which try to communicate, further suppose they wish to use the same vocabulary. For this purpose it is necessary to **define all the element and attribute names** that may be used. Moreover their **structure should also be defined**: what values an attribute may take, which elements may, or must, occur within other elements etc.

Structuring: DTDs and XML Schema

- In the presence of such structuring information we have an enhanced possibility of document validation. We say that an XML document is *valid* if it is well-formed, uses structuring information, and respects that structuring information.
- There are two ways of defining the structure of XML documents: **DTDs (Document Type Definitions)**, the older and more restricted way, and **XML Schema**, which offers extended possibilities, mainly for the definition of data types.

XML Structuring: DTDs

DTDs

- The components of a DTD can be defined in a separate file (external DTD), or within the XML document itself (internal DTD). Usually it is better to use external DTDs, because their definitions can be used across several documents.

DTD Elements

- Consider the element:

```
<person>  
  <firstname>John</firstname>  
  <lastname>Smith</lastname>  
  <phone>+30 2810 223344</phone>  
</person>
```

- A DTD for this element type looks as follows:

```
<!ELEMENT person (firstname,lastname,phone)>  
<!ELEMENT firstname (#PCDATA)>  
<!ELEMENT lastname (#PCDATA)>  
<!ELEMENT phone (#PCDATA)>
```

DTD Elements

- The meaning of this DTD is as follows:
 - The element types *person*, *firstname*, *lastname* and *phone* may be used in the document.
 - A *person* element contains a *firstname* element, a *lastname* element and a *phone* element, **in this order**.
 - A *name* element and a *phone* element may have any content. In DTDs, *#PCDATA* is the **only** atomic type for elements.

DTD Elements

- We express that a *person* element contains either a *firstname* element or a *lastname* element as follows:

```
<!ELEMENT person (firstname | lastname)>
```

- It gets more difficult when we wish to specify that a *person* element contains a *firstname* element and a *lastname* element **in any order**. We can only use the trick:

```
<!ELEMENT person ((firstname,lastname) | (lastname,firstname))>
```

- However this approach suffers from practical limitations (imagine ten elements in any order!).

DTD Attributes

Consider the element:

```
<order orderNo="23456"  
  customer="John Smith"  
  date="October 15, 2002">  
  <item itemNo="a528"  
    quantity="1"/>  
  <item itemNo="c817"  
    quantity="3"/>  
</order>
```

A DTD for it looks as follows:

```
<!ELEMENT order (item+)>  
<!ATTLIST order  
  orderNo ID #REQUIRED  
  customer CDATA #REQUIRED  
  date CDATA #REQUIRED>  
<!ELEMENT item EMPTY>  
<!ATTLIST item  
  itemNo ID #REQUIRED  
  quantity CDATA #REQUIRED  
  comments CDATA #IMPLIED>
```

DTD Attributes

- Compared to the previous example, a new aspect is that the item element type is defined to be empty. Another new aspect is the appearance of + after item in the definition of the order element type. It is one of the **cardinality operators**. These are:
 - ?: appears zero times or once
 - *: appears zero or more times
 - +: appears one or more times
 - No cardinality operator means exactly once.
- In addition to defining elements, we have to define attributes, too. This is done in an **attribute list**.

DTD Attributes Types

- They are similar to predefined data types, but the selection is very limited. The most important types are:
 - **CDATA**: a string (sequence of characters).
 - **ID**: a name that is unique across the entire XML document.
 - **IDREF**: a reference to another element with an ID attribute carrying the same value as the IDREF attribute.
 - **IDREFS**: A series of IDREFs.
 - **(v₁ | . . . | v_n)**: an enumeration of all possible values.
- The selection is indeed not satisfactory. For example, dates and numbers cannot be specified, they have to be interpreted as strings (CDATA); Thus their specific structure cannot be enforced.

DTD Value types

- There are four value types:
 - **#REQUIRED**: the attribute must appear in every occurrence of the element type in the XML document. In our example above, itemNo and quantity must always appear within an item element.
 - **#IMPLIED**: the appearance of the attribute is optional. In our example above, comments are optional.
 - **#FIXED "value"**: every element must have this attribute, which has always the value given after #FIXED in the DTD. A value given in an XML document is meaningless because it is overridden by the fixed value.
 - **"value"**: it specifies the default value for the attribute. If a specific value appears in the XML document, it overrides the default value. For example, the default encoding of the email system may be mime, but binhex will be used if specified explicitly by the user.

DTD Referencing

Here is an example for the use of **IDREF** and **IDREFS**.

```
<!ELEMENT family (person*)>
<!ELEMENT person (name)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST person
  id ID #REQUIRED
  mother IDREF #IMPLIED
  father IDREF #IMPLIED
  children IDREFS #IMPLIED>
```

An XML element that respects this DTD is the following:

```
<family>
  <person id="bob" mother="mary" father="peter">
    <name>Bob Marley</name>
  </person>
  <person id="bridget" mother="mary">
    <name>Bridget Jones</name>
  </person>
  <person id="mary" children="bob bridget">
    <name>Mary Poppins</name>
  </person>
  <person id="peter" children="bob">
    <name>Peter Marley</name>
  </person>
</family>
```

DTD Example

```
<!ELEMENT email (head,body)>
<!ELEMENT head (from,to+,cc*,subject)>
<!ELEMENT from EMPTY>
<!ATTLIST from
  name CDATA #IMPLIED
  address CDATA #REQUIRED>
<!ELEMENT to EMPTY>
<!ATTLIST to
  name CDATA #IMPLIED
  address CDATA #REQUIRED>
<!ELEMENT cc EMPTY>
<!ATTLIST cc
  name CDATA #IMPLIED
  address CDATA #REQUIRED>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (text,attachment*)>
<!ELEMENT text (#PCDATA)>
<!ELEMENT attachment EMPTY>
<!ATTLIST attachment
  encoding (mime|binhex) "mime"
  file CDATA #REQUIRED>
```

XML Structuring: XML Schema

XML Schema

- XML Schema offers a significantly richer language for defining the structure of XML documents. One of its characteristics is that its syntax is based on XML itself! This design decision provides a significant improvement in readability but more importantly, it also allows significant reuse of technology.
- It is no longer necessary to write separate parsers, editors, etc. for a separate syntax, as was required for DTD's.

XML Schema

- An even more important improvement is the possibility to reuse and refine schemas. XML Schema allows to define new types by extending or restricting already existing ones.
- Finally, XML Schema provides a sophisticated set of datatypes that can be used in XML documents (DTD's were limited to strings only).
- An XML schema is an element with an opening tag like:

```
<xsd:schema
    xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
    version="1.0">
```

Elements types

- Syntax:

<element name=". . ."/>

- Optional attributes

– type:

`type` = ". . ." (more on types later)

– cardinality constraints:

- `minOccurs`="x", where x may be any natural number (including zero)
- `maxOccurs`="x", where x may be any natural number (including zero), or *unbounded*.

Elements types

- minOccurs and maxOccurs are obviously generalizations of the cardinality operators ?, *, and +, offered by DTDs. When cardinality constraints are not provided explicitly, minOccurs and maxOccurs have value 1 by default.
- Here are a few examples
 - `<element name="email"/>`
 - `<element name="head" minOccurs="1" maxOccurs="1"/>`
 - `<element name="to" minOccurs="1"/>`

Attribute types

- Syntax:

```
<attribute name="..." />
```

- Optional attributes

- type:

```
type = "..."
```

- existence (corresponds to #OPTIONAL and #IMPLIED in DTDs):

```
use="x", where x may be: optional or required
```

- default value (corresponds to #FIXED and default values in DTDs):

```
use="x" value="...", where x may be default or fixed.
```

- Here are a few examples:

```
<attribute name="id" type="ID" use="required" />
```

```
<element name="speaks" type="Language" use="default"  
value="en" />
```

Data types

- A key weakness of DTDs is the very limited data types. XML Schema provides powerful capabilities for defining data types.
- A few built-in data types:
 - Numerical data types: `integer`, `Short`, `Byte`, `Long`, `Decimal`, `Float` etc.
 - String data types: `string`, `ID`, `IDREF`, `CDATA`, `Language` etc.
 - Date and time data types: `time`, `Date`, `Month`, `Year` etc.

Data types

- User-defined data types:
 - simple data types which cannot use elements or attributes
 - complex data types which can use elements and attributes.
- Complex types are defined from already existing data types by defining some attributes (if any), and by using:
 - **sequence**: a sequence of existing data type elements, the appearance of which in a predefined order is important.
 - **all**: a collection of elements that must appear, but the order of which is not important.
 - **choice**: a collection of elements, of which one will be chosen.

Complex data type example

```
<complexType name="lecturerType">
  <sequence>
    <element name="firstname" type="string"
      minOccurs="0" maxOccurs="unbounded"/>
    <element name="lastname" type="string"/>
  </sequence>
  <attribute name="title" type="string" use="optional"/>
</complexType>
```

- The meaning is that an element in an XML document that is declared to be of type *lecturerType* may have a *title* attribute, it may also include any number of *firstname* elements, and must include exactly one *lastname* element.

Data type extensions

- An existing data types can be extended by new elements or attributes. As an example, we extend the *lecturerType* data type:

```
<complexType name="extendedLecturerType">
  <extension base="lecturerType">
    <sequence>
      <element name="email" type="string"
        minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="rank" type="string" use="required"/>
  </extension>
</complexType>
```

- In this example, *lecturerType* is extended by an email element and a rank attribute.

Data type restriction

- An existing data type may also be restricted by adding constraints on certain values. For example, new **type** and **use** attributes may be added, or the numerical constraints of **minOccurs** and **maxOccurs** tightened.
- It is important to understand that restriction is not the opposite process from extension. Restriction is not achieved by deleting elements or attributes.

Data type restriction

```
<complexType name="restrictedLecturerType">  
  <restriction base="lecturerType">  
    <sequence>  
      <element name="firstname" type="string"  
        minOccurs="1" maxOccurs="2"/>  
    </sequence>  
    <attribute name="title" type="string" use="required"/>  
  </restriction>  
</complexType>
```

- The tightened constraints are highlighted

Data type restriction

- Simple data types can also be defined by restricting existing data types. For example, we can define a type `dayOfMonth` which admits values from 1 to 31 as follows:

```
<simpleType name="dayOfMonth">  
  <restriction base="integer">  
    <minInclusive value="1"/>  
    <maxInclusive value="31"/>  
  </restriction>  
</simpleType>
```

Data type restriction

- Also it is possible to define a data type by listing all the possible values. For example, we can define a data type `dayOfWeek` as follows:

```
<simpleType name="dayOfWeek">  
  <restriction base="string">  
    <enumeration value="Mon"/>  
    <enumeration value="Tue"/>  
    <enumeration value="Wed"/>  
    <enumeration value="Thu"/>  
    <enumeration value="Fri"/>  
    <enumeration value="Sat"/>  
    <enumeration value="Sun"/>  
  </restriction>  
</simpleType>
```

DTD and XML Schema example

- XML documents can have a reference to a DTD or an XML Schema. This is a simple XML document called "note.xml" with a DTD reference:

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

```
<!DOCTYPE note SYSTEM "note.dtd">
```

```
<note>
```

```
  <to>George</to>
```

```
  <from>Marios</from>
```

```
  <heading>Reminder</heading>
```

```
  <body>Don't forget your appointment!</body>
```

```
</note>
```

DTD and XML Schema example

- This is a simple XML document called "note.xml" with a XML Schema reference:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="note.xsd">
  <to>George</to>
  <from>Marios</from>
  <heading>Reminder</heading>
  <body>Don't forget your appointment!</body>
</note>
```

- The line `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` tells the XML parser that this document should be validated against a schema. The line: `xsi:noNamespaceSchemaLocation="note.xsd"` specifies WHERE the schema resides (here it is in the same folder as "note.xml").

DTD and XML Schema example

- This is a simple DTD file called "`note.dtd`" that defines the elements of the XML document "`note.xml`":

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

- Line 1 defines the **note** element to have four elements: "to, from, heading, body". Line 2-5 defines the **to** element to be of the type "#PCDATA", the **from** element to be of the type "#PCDATA", and so on...

DTD and XML Schema example

- This is a simple XML Schema file called "note.xsd" that defines the elements of the XML document "note.xml":

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="note">
    <complexType>
      <sequence>
        <element name="to" type="string"/>
        <element name="from" type="string"/>
        <element name="heading" type="string"/>
        <element name="body" type="string"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

XML Namespaces

- One of the main advantages of using XML is that information from various sources may be accessed; in technical terms, an XML document may use more than one DTD or schema.
- But since each structuring document was developed independently, name clashes appear inevitable. If DTD A and DTD B define an element type **e** in different ways, a parser that tries to validate an XML document in which an **e** element appears must be told which DTD to use for validation purposes.
- The technical solution is simple: disambiguation is achieved by using a different prefix for each DTD or schema. The prefix is separated from the local name by a colon:

prefix:name

XML Namespaces

- Namespaces are declared within an element, and can be used in that element and any of its children (elements and attributes). A namespace declaration has the form:

`xmlns:prefix="location"`

where location is the address of the DTD or schema. If a prefix is not specified, as in

`xmlns="location"`

then the location is used by default.

XML Namespaces example

```
<?xml version="1.0" encoding="UTF-8"?>
<vu:instructors
  xmlns:vu="http://www.vu.com/empDTD"
  xmlns="http://www.gu.au/empDTD"
  xmlns:uky="http://www.uky.edu/empDTD">
  <uky:faculty
    uky:title="assistant professor"
    uky:name="John Smith"
    uky:department="Computer Science"/>
  <academicStaff
    title="lecturer"
    name="Mate Jones"
    school="Information Technology"/>
</vu:instructors>
```

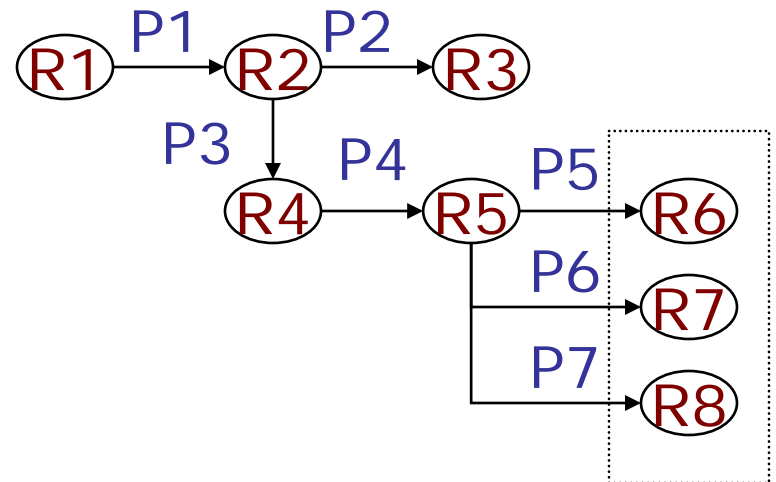
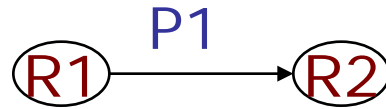
Introduction to RDF(S)

What is RDF

- RDF stands for **Resource Description Framework**
- RDF is for describing resources on the web
- RDF is designed to be read by computers
- RDF is not designed for being displayed to people
- RDF uses **URIs** (Uniform Resource Identifier) to identify web resources
- RDF uses property values to describe web resources
- RDF is essentially a **data-model**.
- RDF is written in XML
- RDF is a web standard - became a **W3C** (World Wide Web Consortium) Recommendation in February 2004

The Core RDF Data Model

- **RDF**: enables communities to describe their resources in a quite natural and flexible way
 - **Data Model**: Directed Labeled Graphs
 - **Nodes**: Resources (URIs) or Literals
 - **Edges**: Properties – Attributes or Relationships
 - **Statement**: assertion of the form *resource, property, value*
 - **Description**: set of statements concerning a resource
 - XML syntax



RDF: Basic Ideas

- **Resources:** We can think of a resource as an object; a “thing” we want to talk about. Resources may be authors, books, publishers, places, people, hotels, rooms etc. Every resource has a **URI**, a Universal Resource Identifier. A URI can be a **URL** (Unified Resource Locator, or Web address), or some other kind of unique identifier; note that an identifier does not necessarily enable **access** to a resource.

RDF: Basic Ideas

- **Properties:** They are special kinds of resources, and describe relations between resources, for example “written by”, “age”, “title” etc. Properties in RDF are also identified by URIs (and in practice by URLs). The value of using URIs to identify “things” and the relations between them should not be underestimated. This choice gives us in one stroke a global, worldwide unique naming scheme. The use of such a scheme greatly reduces the homonym problem that has plagued distributed data-representation until now.

RDF: Basic Ideas

- **Statements**, which assert the properties of resources. A statement is an object-attribute-value **triple**, consisting of
 - a Resource
 - a Property
 - a Value
- Values can either be resources, or **literals**. Literals are atomic values (strings).

RDF: Basic Ideas

- *statements* are (subject, predicate, object) *triples*:
(Greece, hasCapital, Athens)



- statements describe *properties* of *resources*
- a resource is any object that can be pointed at by a **URI** :
 - a document, a picture, a paragraph on the Web
 - <http://www.inf.uth.gr>
 - a book in the library, 'real-world' objects
 - isbn://5031-4444-3333

RDF syntax: XML

- An RDF document is represented by an XML element with tag `rdf:RDF`. The content of this element is a number of descriptions, which use `rdf:Description` tags. Every description makes a statement about a resource which is identified in one of three different ways:
 - an `about` attribute, referencing an existing resource.
 - an `ID` attribute, creating a new resource.
 - without a name, creating an anonymous resource.

- Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:mydomain="http://www.mydomain.org/my-rdf-ns">
  <rdf:Description rdf:about="http://www.mysite.edu/~smith">
    <mydomain:site-owner> John Smith </mydomain:site-owner>
  </rdf:Description>
</rdf:RDF>
```

RDF syntax: XML

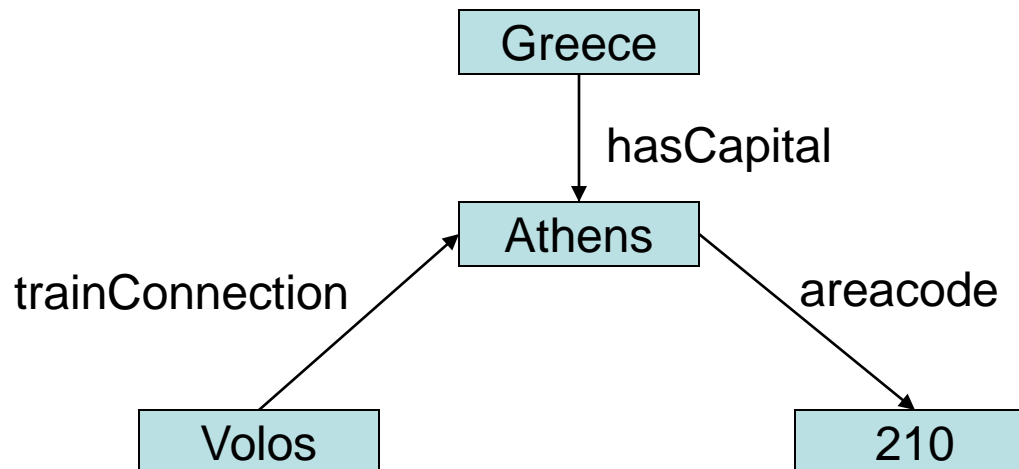
- RDF has an XML syntax that has a specific meaning:
 - every Description element describes a resource
 - every attribute or nested element inside a Description is a property of that resource

```
<Description about="http://www.countries.org/countries#Greece">  
  <hasCapital resource="http://www.cities.org/cities#Athens"/>  
</Description>  
<Description about="http://www.cities.org/cities#Athens">  
  <areacode>210</areacode>  
</Description>
```

- Keep in mind that the order of descriptions (or resources) is not significant according to the abstract model of RDF.

Linking statements

- The subject of one statement can be the object of another
- Such collections of statements form a directed, labeled graph



RDF/XML syntax: just a syntax

- Different ways to write down the same model

```
<Description about="http://www.countries.org/countries#Greece">  
  <hasCapital resource="http://www.cities.org/cities#Athens"/>  
</Description>
```

```
<Description about="http://www.cities.org/cities#Athens">  
  <areacode>210</areacode>  
</Description>
```

```
<Description about="http://www.countries.org/countries#Greece">  
  <hasCapital resource="http://www.cities.org/cities#Athens"/>  
</Description>  
<Description about="http://www.cities.org/cities#Athens" areacode="210"/>  
</Description>
```

```
<Description about="http://www.countries.org/countries#Greece">  
  <hasCapital resource="http://www.cities.org/cities#Athens"/>  
    <areacode>210</areacode>  
  </hasCapital>  
</Description>
```

Namespaces

- Like in 'normal' XML, you can define namespaces to disambiguate elements and attributes:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/TR/2004/REC-rdf-primer-20040210/"
  xmlns:geo="http://www.geography.org/schema.rdf#">
  <rdf:Description rdf:about="http://www.countries.org/countries#Greece">
    <geo:hasCapital rdf:resource="http://www.cities.org/cities#Athens"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.cities.org/cities#Athens">
    <geo:areacode>210</geo:areacode>
  </rdf:Description>
</rdf:RDF>
```


RDF Container Elements

- The `rdf:Bag` element contains an unordered list of value elements:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:cd="http://www.recshop.fake/cd">
  <rdf:Description rdf:about="http://www.recshop.fake/cd/Beatles for Sale">
    <cd:artist>
      <rdf:Bag>
        <rdf:li>John</rdf:li>
        <rdf:li>Paul</rdf:li>
        <rdf:li>George</rdf:li>
        <rdf:li>Ringo</rdf:li>
      </rdf:Bag>
    </cd:artist>
  </rdf:Description>
```

- The `rdf:Seq` element contains an ordered list of value elements,

So what can we use this for?

- We can:
 - make explicit statements about web resources
 - have the machine
 - know that these are statements
 - know how the statements relate
 - compare values
- BUT
 - we still miss a way to define a vocabulary:
 - Should we use 'country' or 'nation'?
 - Is Greece a country? Are there more countries? What properties can countries have?

RDF Schema

- *RDF Schema* defines a set of modeling primitives for structured vocabularies for machine-processable semantics of information.
 - Two crucial RDF Schema constructions are *subClassOf* and *subPropertyOf* allowing hierarchical structured vocabularies.

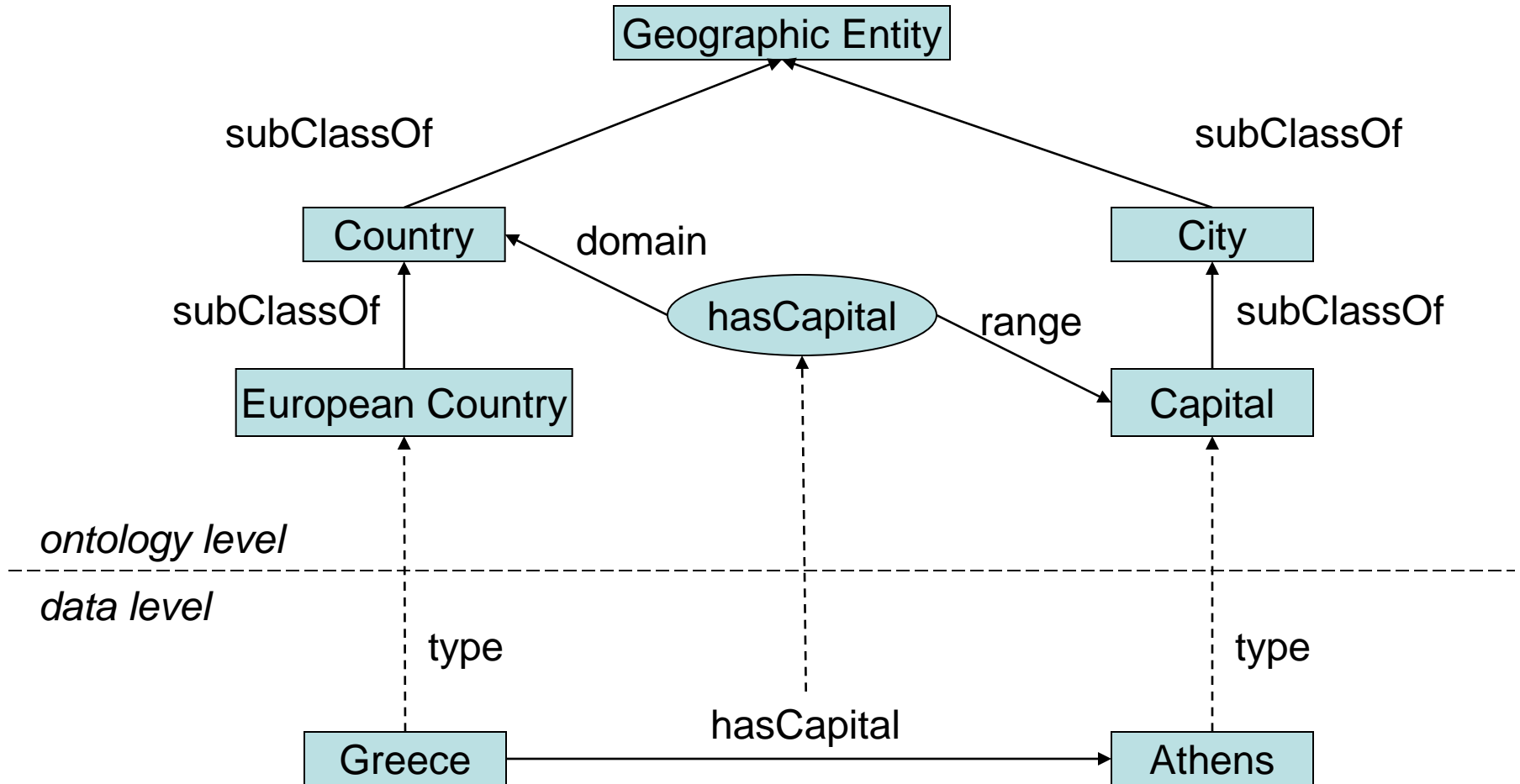
RDF Schema

- RDF gives a data model for metadata annotation, and a way to write it down in XML, but it can not define the vocabulary for a domain.
- **RDF Schema** allows you to define vocabulary terms and the relations between these terms
 - It gives 'extra meaning' to particular RDF predicates and resources
 - this 'extra meaning', or semantics, define how a term should be interpreted

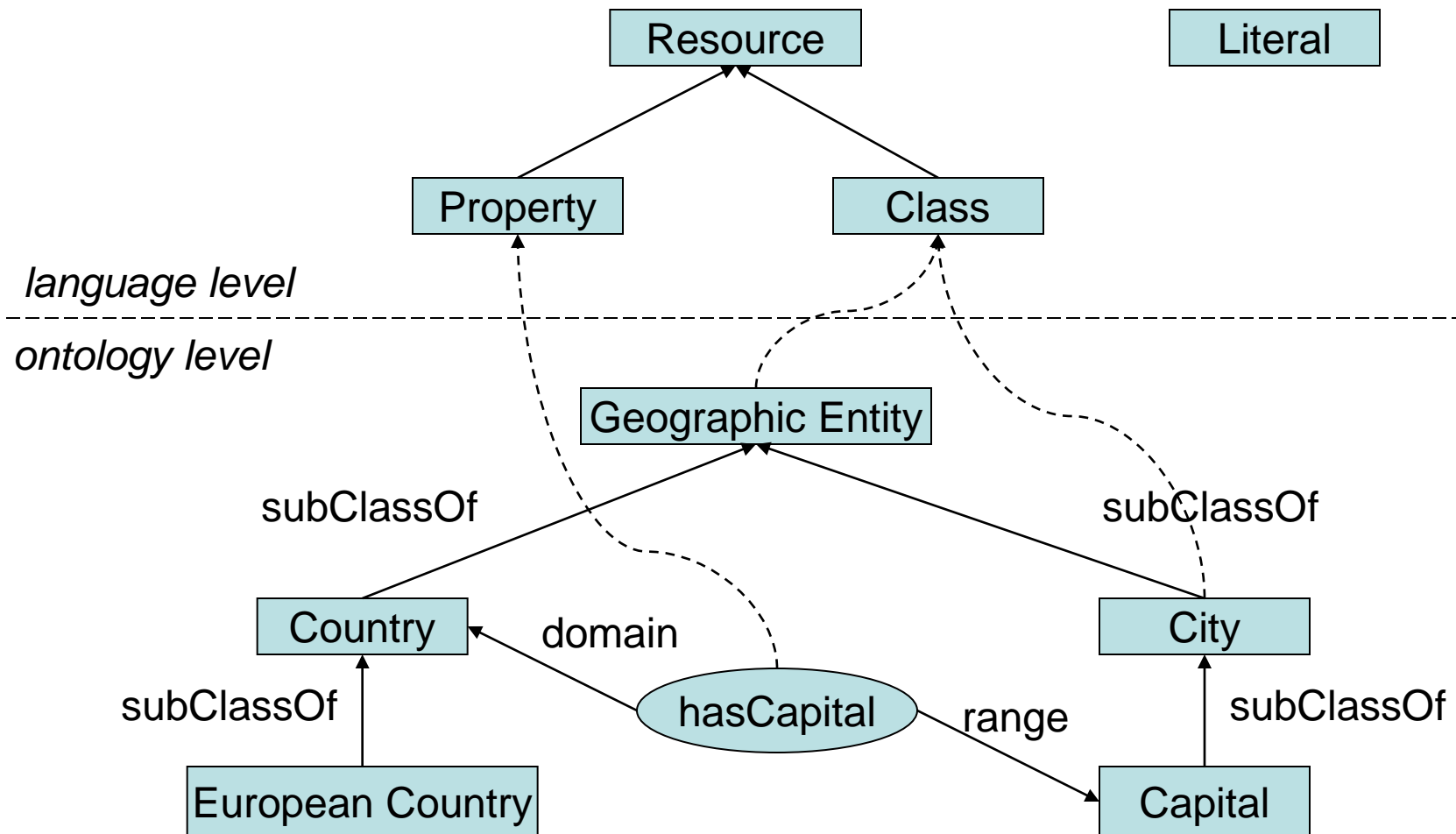
RDF Schema

- RDF Schema does not provide **actual** application-specific classes and properties.
- Instead RDF Schema provides the framework to **describe** application-specific classes and properties
- Classes in RDF Schema are much like classes in object oriented programming languages. This allows resources to be defined as instances of classes, and subclasses of classes.

RDF Schema Example



RDF Schema Example



Some observations

- Classes and properties are modeled separately!
 - this is different from 'normal' Object-Oriented modeling where properties (attributes) are part of a class.
 - Because of this, domain/range statements become very restrictive
- Again: RDF Schema is 'just' RDF, but with some added meaning to particular terms.

RDF Schema syntax

- **Class definition**

```
<rdf:Description rdf:about="http://www.geography.org/schema.rdf#Country">  
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>  
  <rdfs:subClassOf  
    rdfs:resource="http://www.geography.org/schema.rdf#GeographicEntity"/>  
</rdf:Description>
```

```
<rdfs:Class rdf:about="http://www.geography.org/schema.rdf#Country">  
  <rdfs:subClassOf  
    rdfs:resource="http://www.geography.org/schema.rdf#GeographicEntity"/>  
</rdfs:Class>
```

- **Property definition**

```
<rdf:Property rdf:about="http://www.geography.org/schema.rdf#hasCapital">  
  <rdfs:domain rdfs:resource="http://www.geography.org/schema.rdf#Country"/>  
  <rdfs:range rdfs:resource="http://www.geography.org/schema.rdf#Capital"/>  
</rdf:Property>
```

Ontology language?

- Ontology: a formal specification of a shared conceptualization
- RDF Schema allows:
 - specification (we have just seen that)
 - sharing (because it is an open, web-based standard)
 - formality?
- Is RDF Schema expressive enough?

What is still missing?

- Cardinality constraints
 - “a country can have exactly *one* capital”
- Conjunction, disjunction, negation, equivalence
 - “countries and cities are disjoint: something can not be both a city and a country”
- Localized constraints
 - “when the property 'population' is used on a city, its value must be between 20.000 and 10 million”
- A way to access this information!
 - having it written down is nice and all, but if you want to use it for question answering you need a **query language**
- A way to define rules relating concepts and properties

A motivating example

- Definitions:
 - Rule1: To walk through a door a VH's height must be less than that of the door's
 - Rule2: To walk through a locked door a VH must have the key
 - Rule3: A VH can walk from room A to room B if
 - there is a door between room A and B
 - the VH is short enough
 - the VH has the key to the door
 - Rule4: If a VH can walk from room A to room B and from room B to room C, then the VH can walk from room A to room C (transitive)
 - . . .

A motivating example

- Facts (metadata):
 - This VH's name is *John*
 - Door with id *D2* has a key with id *K2*
 - Door *D8* is locked
 - The VH with name *Marios* has height *178 cm*
 - The VH with name *John* has key *K3*
 - The VH with name *John* is in room *A*
 - Door *D5* connects rooms *B* and *C*

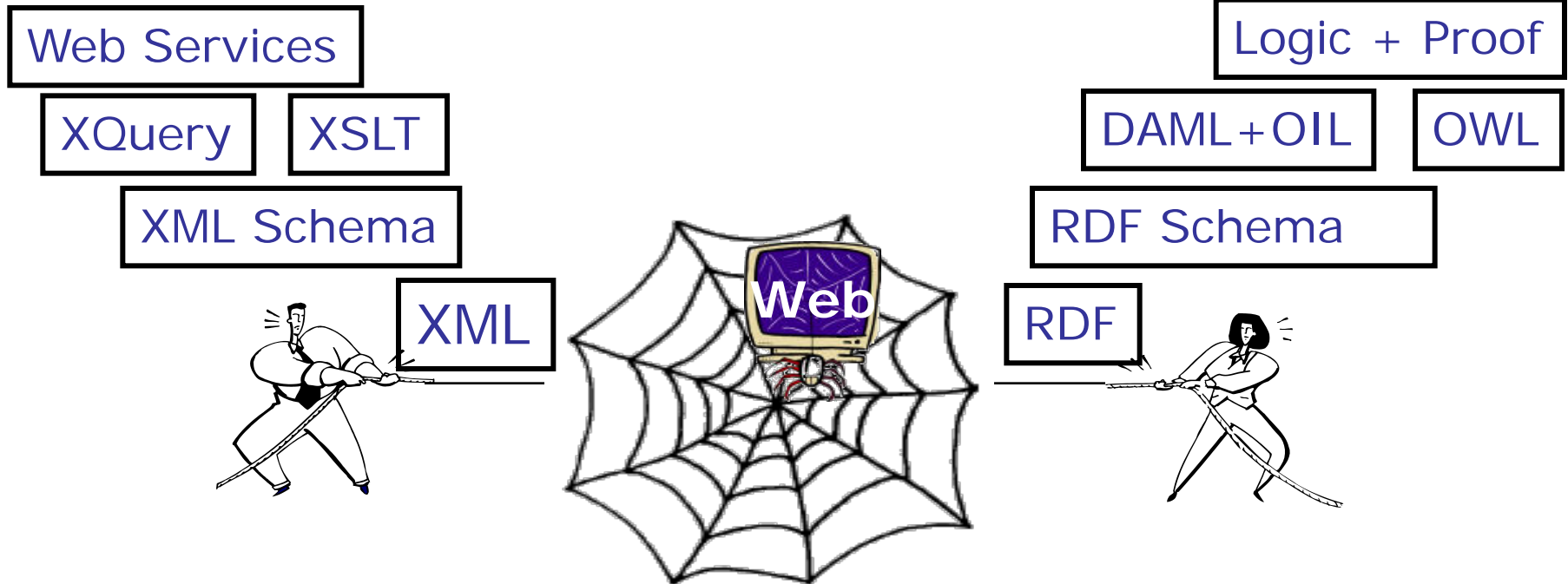
A motivating example

- Questions:
 - Find all VHs who can walk from room A to room B
 - Deduce a path from A to B
 - Check which doors in the path are locked
 - Find a VH who has the keys for all locked doors in the path
 - Find a VH short enough to walk through all doors in the path

Two Cultures on the Future Web

- DB Community focus on:
 - XML Data Semantics
 - XML Data Manipulation Languages (Querying, Views, Programming)

- KR Community focus on:
 - Ontology Languages
 - Reasoners and Theorem Provers



Tools, tools, tools

- Metadata annotations
- Ontologies
- Repositories
- Languages
- Search engines
- Inference

