SEMANTIC WEB

Unit-I

Introduction: Introduction to Semantic Web, the Business Case for the Semantic Web, XML and Its Impact on the Enterprise

Introduction to Semantic Web

The initial step in Tim Berner's LEE vision present information on the web in a way that machine can easily understand. This involves either placing data in a format that machine can naturally understand or converting existing data into a such a format. The outcome is what Berner's Lee's terms a semantic web.

What Is the Semantic Web?

Tim Berner's Lee has a two part visison for the future web

1. **To make web a collaborative medium**
2. **Machnine understandable**

So, how do we create a web of data that machines can process? The first step is a paradigm shift in the way we think about data. Historically, data has been locked away in proprietary applications. Data was seen as secondary to processing the data. This incorrect attitude gave rise to the expression "garbage in, garbage out," or GIGO. GIGO basically reveals the flaw in the original argument by establishing the dependency between processing and data. In other words, useful software is wholly dependent on good data.

Computing professionals began to realize that data was important, and it must be verified and protected. Programming languages began to acquire object-oriented facilities that internally made data first-class citizens. However, this "data as king" approach was kept internal to applications so that vendors could keep data proprietary to their applications for competitive reasons.

With the Web, Extensible Markup Language (XML), and now the emerging Semantic Web, the shift of power is moving from applications to data. This also gives us the key to understanding the Semantic Web. The path to machine-processable data is to make the data smarter.

Figure 1.2 shows four stages of the smart data continuum; however, there will be more fine-grained stages, as well as more follow-on stages. The four stages in the diagram progress from data with minimal smarts to data embodied with enough semantic information for machines to make inferences about it. Let's discuss each stage:

Text and databases (pre-XML). The initial stage where most data is proprietary to an application. Thus, the "smarts" are in the application and not in the data.

XML documents for a single domain. The stage where data achieves application independence within a specific domain. Data is now smart enough to move between applications in a single domain. An example of this would be the XML standards in the healthcare industry, insurance industry, or real estate industry.

Taxonomies and documents with mixed vocabularies. In this stage, data can be composed from multiple domains and accurately classified in a hierarchical taxonomy. In fact, the classification can be used for discovery of data. Simple relationships between categories in the taxonomy can be used to relate and thus combine data. Thus, data is now smart enough to be easily discovered and sensibly combined with other data.

Activate V

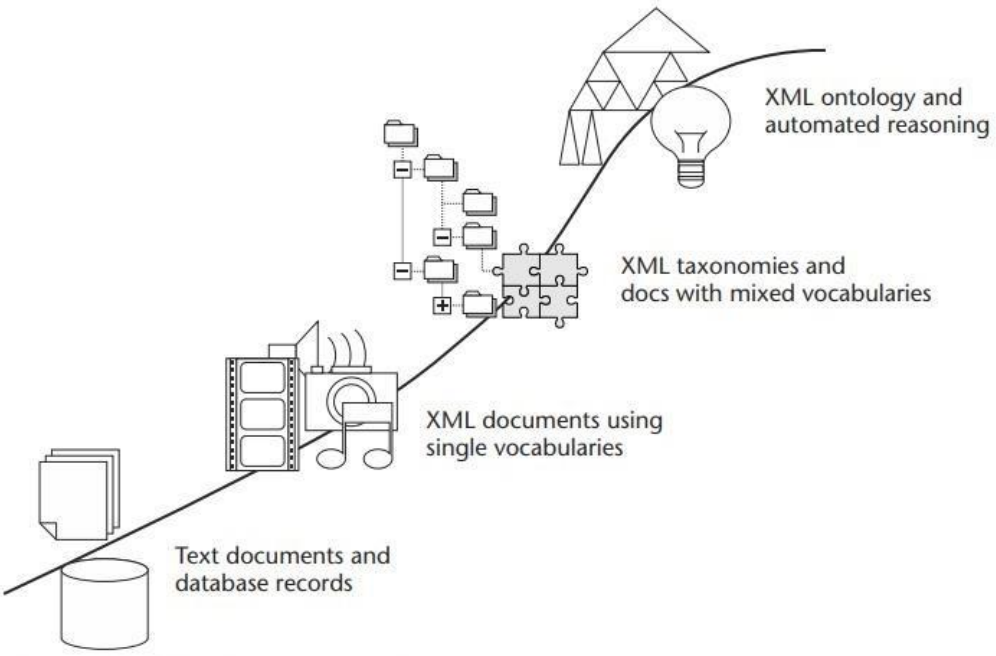

Figure 1.2 The smart data continuum.

Ontologies and rules. In this stage, new data can be inferred from existing data by following logical rules. In essence, data is now smart enough to be described with concrete relationships, and sophisticated formalisms where logical calculations can be made on this "semantic algebra." This allows the combination and recombination of data at a more atomic level and very fine-grained analysis of data. Thus, in this stage, data no longer exists as a blob but as a part of a sophisticated microcosm. An example of this data sophistication is the automatic translation of a document in one domain to the equivalent (or as close as possible) document in another domain

Why Do We Need the Semantic Web?

The Semantic Web is not just for the World Wide Web. It represents a set of technologies that will work equally well on internal corporate intranets. This is analogous to Web services representing services not only across the Internet but also within a corporation's intranet. So, the Semantic Web will resolve several key problems facing current information technology architectures.

Information Overload

Information overload is the most obvious problem in need of a solution, and technology experts have been warning us about it for 50 years. In the article "Overcoming Information Overload," Paul Krill states, "This condition results from having a rapid rate of growth in the amount of information available, while days remain n 24 hours long and our brains remain in roughly the same state of development as they were when cavemen communicated by scrawling messages in stone."1 Of course, it is generally acknowledged that this problem has grown worse with the propagation of the Internet, email, and now instant messaging. Unfortunately, our bias toward production over reuse of knowledge has left this problem unresolved until it has finally hit tragic proportions. A glaring reminder of our failure to make progress on this issue is Vannevar Bush's warning in 1945 when he said, "There is a growing mountain of research. But there is increased evidence that we are being bogged down today as specialization extends. The investigator is staggered by the findings and conclusions of thousands of other workers—conclusions which he cannot find time to grasp, much less to remember, as they appear. Yet specialization becomes increasingly necessary for progress, and the effort to bridge between disciplines is correspondingly superficial.

Stovepipe Systems

A *stovepipe system* is a system where all the components are hardwired to only work together. Therefore, information only flows in the stovepipe and cannot be shared by other systems or organizations that need it. For example, the client can only communicate with specific middleware that only understands a single database with a fixed schema Breaking down stovepipe systems needs to occur on all tiers of enterprise information architectures; however, the Semantic Web technologies will be most effective in breaking down stove piped database systems.

Recently, manual database coordination was successful in solving the Washington sniper case. Jonathan Alter of *Newsweek* described the success like this: "It was by matching a print found on a gun catalog at a crime scene in Montgomery, Ala., to one in an INS database in Washington state that the Feds cracked open the case and paved the way for the arrest of the two suspected snipers.

Poor Content Aggregation

Putting together information from disparate sources is a recurring problem in a number of areas, such as financial account aggregation, portal aggregation, comparison shopping, and content mining. Unfortunately, the most common technique for these activities is screen scraping.

How Does XML Fit into the Semantic Web

XML is the syntactic foundation layer of the Semantic Web. All other technologies providing features for the Semantic Web will be built on top of XML. Requiring other Semantic Web technologies (like the Resource Description Framework) to be layered on top of XML guarantees a base level of interoperability.

The technologies that XML is built upon are Unicode characters and Uniform Resource Identifiers (URIs). The Unicode characters allow XML to be authored using international characters. URIs are used as unique identifiers for concepts
in the Semantic Web.

For example, if I label something a <price> $12.00 </price> and you label that field on your invoice <cost> $12.00 </cost>, there is no way that a machine will know those two mean the same thing unless Semantic Web technologies like
ontologies are added

How Do Web Services Fit into the Semantic Web?

The semantic web can make web services "smarter" in three ways

1. **More "smart data":** by using more structured data like XML, web services become more understandable for computers.

2. **Easier discovery:** Semantic Web technologies will be necessary to solve the Web service discovery problem.

3. **Better interaction:** is in enabling Web services to interact with other Web services. Advanced Web service applications involving comparison, composition , or orchestration of Web services will require Semantic Web technologies for such interactions to be automated. In short web services are like building blocks and the semantic web is the glue that helps us connect them and use them effectively.

Figure 1.3 demonstrates the various convergences that combine to form Semantic Web services.

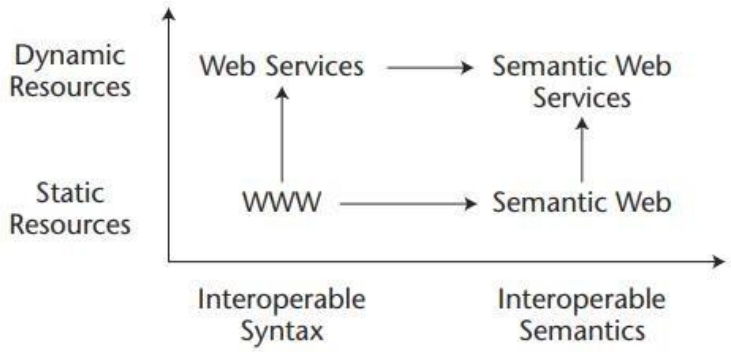

Figure 1.3 Semantic Web services.

Derived in part from two separate presentations at the Web Services One Conference 2002 by Dieter Fensel and Dragan Sretenovic.

What's after Web Services

Web services complete a platform-neutral processing model for XML. The step after that is to make both the data and the processing model smarter. In other words, continue along the **"smart-data continuum**.". This will move along five axes:

1. logical assertions
2. classification
3. formal class models
4. rules, and
5. trust.

Logical Assertions

An assertion is the smallest expression of useful information. How do we make an assertion? One way is to model the key parts of a sentence by connecting a subject to an object with a verb. Resource Description Framework (RDF), which captures these associations between subjects and objects. The importance of this cannot be understated. As Tim Berners-Lee states, "*The philosophy was: What matters is in the connections. It isn't the letters, it's the way they're strung together into words. It isn't the words, it's the way they're strung together into phrases. It isn't the phrases, it is the way they're strung together into a document."* Agreeing with this sentiment, Hewlett-Packard Research has developed open source software to process RDF called Jena.

Classification

We classify things to establish groupings by which generalizations can be made. Just as we classify files on our personal computer in a directory structure, we will continue to better classify resources on corporate intranets and even the Internet. The concepts for classification have been around a long time. Carolus Linnaeus developed a classification system for biological organisms in 1758. An example is displayed in Figure 1.4. The downside of classification systems is evident when examining different people's filesystem classification on their personal computers. Categories (or folder names) can be arbitrary, and the membership criteria for categories are often ambiguous. Thus, while taxonomies are extremely useful for humans browsing for information, they lack rigorous logic for machines to make inferences from.

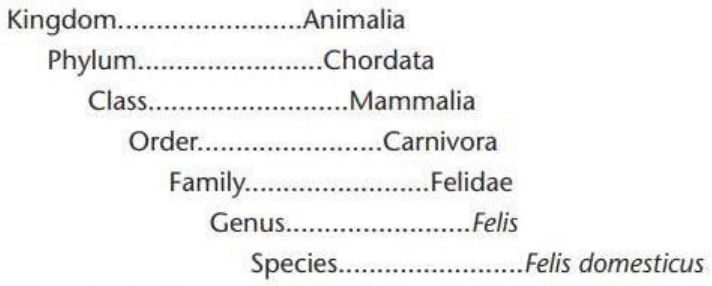

Figure 1.4 Linnaean classification of a house cat.

Formal class models. A formal representation of classes and relationships between classes to enable inference requires rigorous formalisms even beyond conventions used in current object-oriented programming languages like Java and C#. Ontologies are used to represent such formal class hierarchies, constrained properties, and relations between classes. The W3C is developing a Web Ontology Language (abbreviated as OWL)

Figure 1.5 shows several classes (Person, Leader, Image, etc.), a few properties of the class Person (birthdate, gender), and relations between classes (knows, is-A, leads, etc.). Again,

while not nearly a complete ontology, the purpose of Figure 1.5 is to demonstrate how an ontology captures logical information in a manner that can allow inference. For example, if John is identified as a Leader, you can infer than John is a person and that John may lead an organization. Additionally, you may be interested in questioning any other person that "knows" John. Or you may want to know if John is depicted in the same image as another person (also known as co-depiction). It is important to state that the concepts described so far (classes, subclasses, properties) are not rigorous enough for inference. To each of these basic concepts, additional formalisms are added. For example, a property can be further specialized as a symmetric property or a transitive property. Here are the rules that define those formalisms:

If x = y, then y = x. (symmetric property)
If x = y and y = z, then x = z. (transitive property)

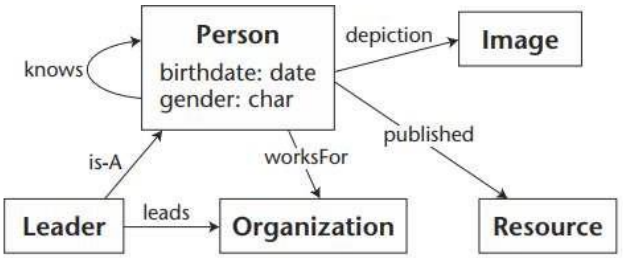

Figure 1.5 Key ontology components.

An example of a transitive property is "has Ancestor." Here is how the rule applies to the "has Ancestor" property:

If Joe hasAncestor Sam and Sam hasAncestor Jill, then Joe hasAncestor Jill.

Rules. With XML, RDF, and inference rules, the Web can be transformed from a collection of documents into a knowledge base. An inference rule allows you to derive conclusions from a set of premises. A well-known logic rule called "modus ponens" states the following:

If P is TRUE, then Q is TRUE.

P is TRUE.

Therefore, Q is TRUE.

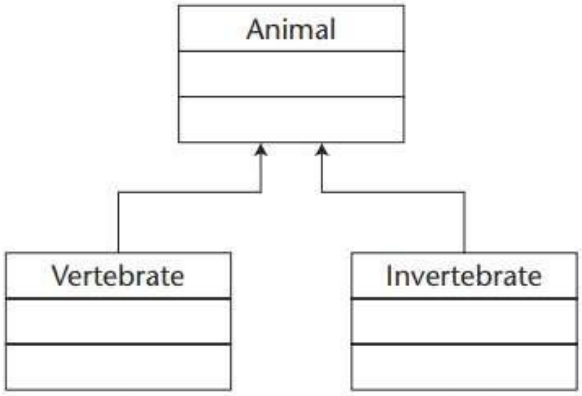

Figure 1.6 UML presentation of ontology class and subclasses.

An example of modus ponens is as follows:

An apple is tasty if it is not cooked. This apple is not cooked. Therefore, it is tasty.

The Semantic Web can use information in an ontology with logic rules to infer new information. Let's look at a common genealogical example of how to infer the "uncle" relation as depicted in Figure 1.7:

If a person C is a male and childOf a person A, then person C is a "sonOf" person A.

If a person B is a male and siblingOf a person A, then person B is a "brotherOf" person A.

If a person C is a "sonOf" person A, and person B is a "brotherOf" person A, then person B is the "uncleOf" person C.

Trust. Instead of having trust be a binary operation of possessing the correct credentials, we can make trust determination better by adding semantics. For example, you may want to allow access to information if a trusted friend vouches (via a digital signature) for a third party. Digital signatures are crucial to the "web of trust" and are discussed in Chapter 4. In fact, by allowing anyone to make logical statements about resources, smart applications will only want to make inferences on statements that they can trust. Thus, verifying the source of statements is a key part of the Semantic Web.

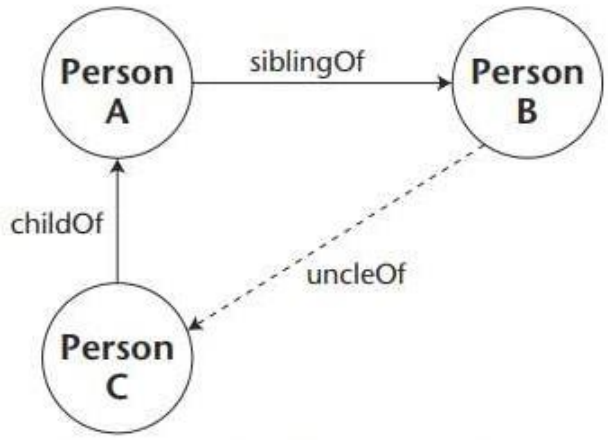

Figure 1.7 Using rules to infer the uncleOf relation.

Activate Wir

What Do the Skeptics Say about the Semantic Web?

Every new technology faces skepticism: some warranted, some not. The skepticism of the Semantic Web seems to follow one of three paths

1. Bad precedent
2. Fear, uncertainty, and doubt (FUD)
3. Status quo

Bad precedent

The paragraphs discuss how skeptics often use the failure of early artificial intelligence predictions to discredit the Semantic Web. They mention a famous prediction from 1957 by Herbert Simon and Allen Newell about a computer beating a human at chess within 10 years, which didn't come true. Tim Berners-Lee, the inventor of the World Wide Web, responds to these comparisons by clarifying that the Semantic Web is not the same as artificial intelligence. He explains that the Semantic Web focuses on making documents understandable to machines, but it doesn't involve creating magical AI that understands

human language effortlessly. Instead, it requires people to structure data in a specific way to make it easier for machines to process and solve well-defined problems.

Fear, uncertainty, and doubt (FUD). This is skepticism "in the small" or nitpicking skepticism over the difficulty of implementation details. The most common FUD tactic is deeming the Semantic Web as too costly. Semantic Web modeling is on the same scale as modeling complex relational databases. Relational databases were costly in the 1970s, but prices have dropped precipitously (especially with the advent of open source). The cost of Semantic Web applications is already low due to the Herculean efforts of academic and research institutions. The cost will drop further as the Semantic Web goes mainstream in corporate portals and intranets within the next three years.

Status quo.

Some people don't like change (status quo!). They think the current way of doing things is fine and the Semantic Web is unnecessary.

a. Just like people initially doubted the World Wide Web, they doubt the Semantic Web's usefulness.
b. Back then, people used clunky systems just for phone numbers. Tim Berners-Lee showed them a simple web-based solution, but they didn't see the bigger picture.
c. They preferred isolated "stovepipe" solutions for each task, instead of a more flexible web architecture.
d. Why? They couldn't understand the power of sharing information on a global network. Basically: Some people resist change and don't see the potential benefits of the Semantic Web, just like they initially doubted the World Wide Web

Why the Skeptics Are Wrong!

1. ***We have the computing power.***

The increasing computing power available due to advancements in technology. The progression from cell phones to personal computers to servers to mainframes, which has led to a significant increase in computing power. This increase allows for the creation of more complex systems and layers, such as virtual machines like Java and C#. These virtual machines were conceptualized decades ago but became practical with the computing power available in the 1990s. While the underpinnings are being standardized now, the Semantic Web will be practical, in terms of computing power, within three years.

2. ***Consumers and businesses want to apply the network effect to their information***.

Average people see and understand the network effect and want it applied to their home information processing. Average homeowners now have multiple computers and want them networked. Employees understand that they can be more effective by capturing and

leveraging knowledge from their coworkers. Businesses also see this, and the smart ones are using it to their advantage. Many businesses and government organizations see an opportunity for employing these technologies (and business process reengineering) with the deployment of enterprise portals as natural aggregation points.

3. ***Progress through combinatorial experimentation demands it.***
An interesting brute-force approach to research called combinatorial experimentation is at work on the Internet.. This method involves trying out different combinations of research findings, leveraging the fact that research is instantly accessible worldwide. It emphasizes the importance of the network effect in research, where the ability to access and combine findings globally leads to progress. The text suggests that effective combinatorial experimentation relies on the Semantic Web, a structured way of organizing data so computers can understand it better. It concludes by stating that progress will drive the development of the Semantic Web, as predicted by Vannevar Bush in 1945.

The Business Case for the Semantic Web

Main Point: Organizations that can find and use information effectively have a competitive advantage. The Semantic Web can help with this.

Key points:

Knowledge is Power: Having the right information and being able to use it quickly is crucial for success. It's not just about having lots of data anymore; it's about knowing how to turn that data into useful knowledge for making decisions.

2. **Challenges of Traditional Knowledge Management**: With the internet, we're flooded with information, making it hard to find what we need. Keyword searches are not always efficient, and there's often a lack of trustworthy information. Also, computers struggle to understand natural language.

3. **Role of Semantic Web**: The Semantic Web can help bring order to this chaos by organizing information in a structured way. It's not just about storing data; it's about tagging it in a way that computers can understand and making sure it's reliable.

4. **Key Concepts of Semantic Web**: It's not enough to just gather information; we need to tag it with meaningful labels and ensure its reliability. We also need tools to analyze and use this information effectively.

5.**Benefits for Organizations**: By implementing Semantic Web principles, organizations can avoid duplicating efforts, share lessons learned, and save time and money. Having a centralized knowledge base that software can analyze can lead to efficient web-based applications.

In simple terms, the message is: to succeed in today's information-rich world, organizations need to move beyond just collecting data. They need to organize it in a meaningful way, ensure its reliability, and have tools to turn it into useful knowledge for making decisions and improving efficiency.

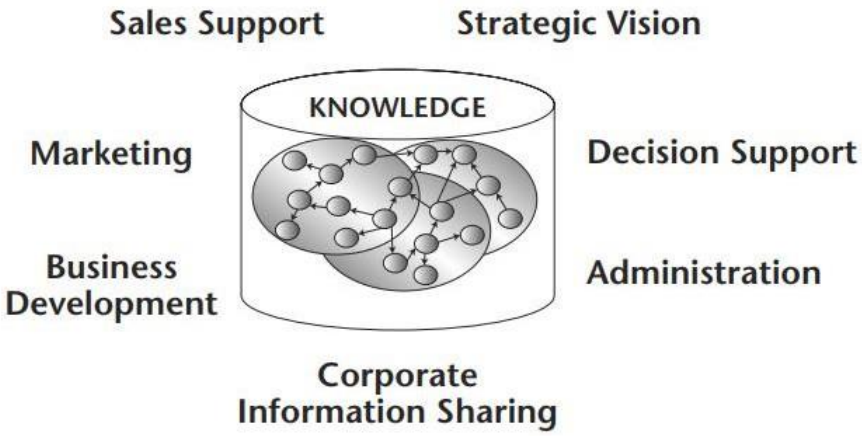

Figure 2.1 Uses of the Semantic Web in your enterprise.

Decision Support

Having good, connected information (knowledge) is key to better decisions.

- The example used is the 9/11 attacks, where FBI Director Mueller wished for a system that could connect different data sources and identify hidden relationships.
- This is exactly what the Semantic Web aims to do - connect and understand information beyond just keywords.
- It allows software agents (like intelligent assistants) to find hidden connections and patterns in data across different systems, something traditional methods struggle with.
- This is beneficial for both large organizations like governments and businesses because:
  - Information from different departments/groups can be combined and analyzed, revealing hidden insights.
  - Relationships between projects or data points can be identified, giving a broader picture.
  - Better decisions can be made based on this richer understanding of information.

Examples include:

- Government: Automatically identifying suspicious patterns in financial documents (like SEC filings).
- Businesses: Creating decision support systems that provide relevant information and alerts based on specific needs.

  Overall: The Semantic Web helps organizations unlock the true power of their information by connecting it and making it more meaningful, leading to better decisions.

Business Development

1. **Importance of Up-to-Date Information**: It's crucial for organizations to have the latest information to seize business opportunities. However, it's not feasible to physically bring all experts to every sales meeting.

2. **Example Scenario:** Imagine a salesperson learns during a meeting that a potential customer is interested in a specific topic, like building an e-commerce system with biometric identification. If the organization has a quick way to access its knowledge base, the salesperson can respond effectively by mentioning relevant recent developments and offering a demonstration.

3. **Competitive Proposals**: A well-organized knowledge base can also help in creating competitive proposals. By understanding customer needs, past proposals, and competitive intelligence, organizations can tailor their bids better and increase their chances of winning projects.

4. **Customer Relationship Management (CRM)**: CRM systems provide personalized information from various data sources within an organization to improve customer relationships. The challenge lies in integrating legacy data sources quickly and comparing information across different areas of the business.

5. **Semantic Web in E-commerce Matchmaking**: The Semantic Web can automate matchmaking processes in e-business, connecting businesses with potential partners or customers more efficiently than traditional methods.

   In simple terms, having access to the right information at the right time can significantly enhance business opportunities and improve customer relationships. Semantic Web technologies play a crucial role in organizing and utilizing this information effectively.

Information Sharing and Knowledge Discovery

Information sharing and communication are paramount in any organization, but as most organizations grow and collect more information, this is a major struggle. We all understand the importance of not reinventing the wheel, but how many times have we unintentionally duplicated efforts? When organizations get larger, communication gaps are inevitable. With a little bit of effort, a corporate knowledge base could at least include a registry of descriptions of projects and what each team is building. Imagine how easy it would be for your employees to be able to find relevant information. Using Semantic Web-enabled Web services can allow us to create such a registry.

Administration and Automation

Up to this point, we've discussed the somewhat obvious examples based on sharing knowledge within an organization. A side effect of having such a knowledge base is the ability of software programs to automate administrative tasks. Booking travel, for example, is an example where the Semantic Web and Web services could aid in making a painful task easy. Making travel arrangements can be an administrative nightmare. Everyone has personal travel preferences and must take items such as the following into consideration:

Activate

- Hotel proximity to meeting places
- Hotel room preferences (nonsmoking, king, bar, wireless network in lobby)
- Rental car options and associated rewards
- Price (lodging and transportation per diem rates for your company)

Creating a flowchart of your travel arrangement decisions can be a complex process. Say, for example, that if the trip is less than 100 miles, you will rent a car. If the trip is between 100 miles and 300 miles, you will take the train or bus. If the trip is above 300 miles, you will fly. If you fly, you will look for the cheapest ticket, unless you can get a first-class seat with your frequent flyer miles from American Airlines. If you do book a flight, you want a vegetarian meal. You want to weigh the cost of your hotel against the proximity to your meeting place, and you have room preferences, and so on. As you begin mapping out the logic for simply booking travel, you realize that this could be a complex process that could take a few hours.

Is the Technology for the Semantic Web "There Yet"?

- The Semantic Web is still a dream, but the groundwork is being laid with new technologies.
- Standards are being developed by organizations like W3C to make data work across different systems.
- Technologies like XML and RDF are being used to add meaning to information online.
- Web services are being built to allow software programs to talk to each other.
- Companies like Adobe and IBM are investing in Semantic Web technologies.

In short, the internet is getting smarter and more connected, paving the way for a Semantic Web.

Why is XML success?

The primary use of XML is for data exchange between internal and external organizations (interoperability).

XML has four primary accomplishments,

1. **XML creates application-independent documents and data.**

Think of XML as plain English for computers**.** It's easy for anyone to read and understand, unlike some software's specific file formats.
Imagine you have a document:

a. **Normal document:** You can easily read and understand it.
b. **Software-specific document:** It's like a secret code, only the software that created it can understand.

XML is like the normal document. It's readable by any program, making it easier to share and use data.

2. **It has a standard syntax for meta data.**

The second key accomplishment is that XML provides a simple, standard syntax for encoding the meaning of data values, or meta data. **An often-used definition of meta data is "data about data."** XML standardizes a simple, text-based method for encoding meta data. In other words, XML provides a simple yet robust mechanism for encoding semantic information, or the meaning of data. Table 3.1 demonstrates the difference between meta data and data. It should be evident that the data is the raw context-specific values and the meta data denotes the meaning or purpose of those values.

Table 3.1 Comparing Data to Meta Data

DATA	META DATA
Joe Smith	Name
222 Happy Lane	Address
Sierra Vista	City
AZ	State
85635	Zip code

3. **It has a standard structure for both documents and data.**

The third major accomplishment of XML is standardizing a structure suitable to express semantic information for both documents and data fields. The structure XML uses is a hierarchy or tree structure. A good common example of a tree structure is an individual's file system on a computer, as shown in Figure 3.3. The hierarchical structure allows the user to decompose a concept into its component parts in a recursive manner.

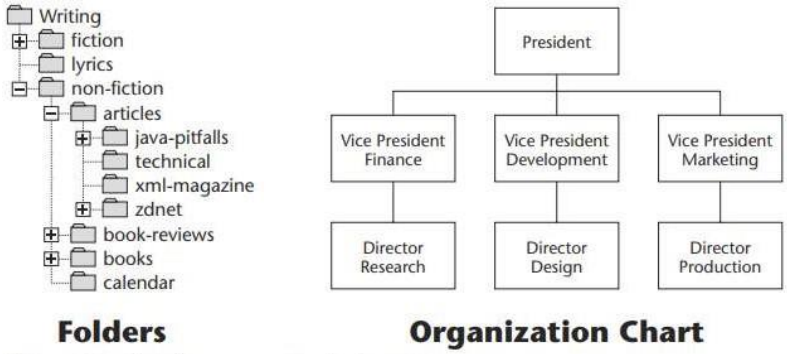

Figure 3.3 Sample trees as organization structures.

4. **XML is not a new technology (not a 1.0 release)**

The last accomplishment of XML is that it is not a new technology. XML is a subset of the Standardized Generalized Markup Language (SGML) that was invented in 1969 by Dr. Charles Goldfarb, Ed Mosher, and Ray Lorie. So, the concepts for XML were devised over 30 years ago and continuously perfected, tested, and broadly implemented.

What is XML?

XML is not a language; it is actually a set of syntax rules for creating semantically rich markup languages in a particular domain. In other words, you apply XML to create new languages. Any language created via the rules of XML, like the Math Markup Language (MathML), is called an application of XML. A markup language's primary concern is how to add semantic information about the raw content in a document; thus, the vocabulary of a markup language is the external "marks" to be attached or embedded in a document.

Markup is separate from content.

So, the first key principle of XML is *markup is separate from content*. A corollary to that principle is that markup can surround or contain content. Thus, a markup language is a set of words, or marks, that surround, or "tag," a portion of a document's content in order to attach additional meaning to the tagged content. The mechanism

invented to mark content was to enclose each word of the language's vocabulary in a less-than sign (<) and a greater-than sign (>) like this:

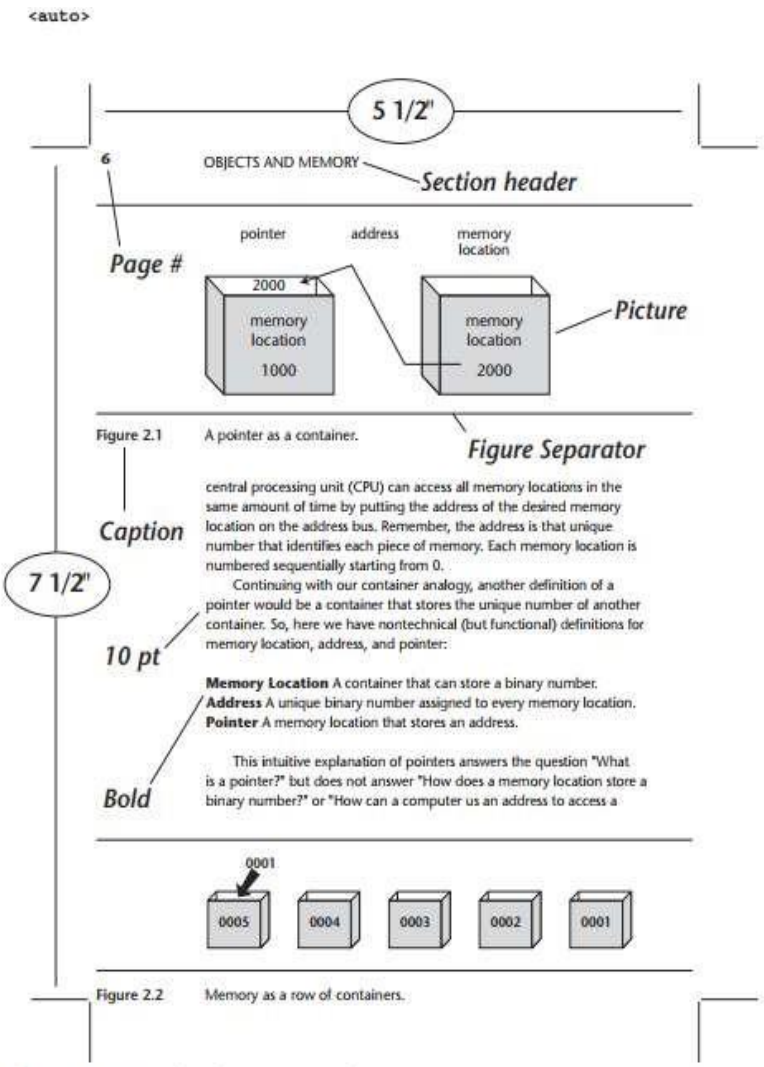

Figure 3.4 Manual markup on a page layout.

To use the < and > characters as part of a tag, these characters cannot be used in content, and therefore they are replaced by the special codes (called entities) > (for greater than) and < (for less than). This satisfies our requirement to separate a mark from content but does not yet allow us to surround, or contain, content. Containing content is achieved by wrapping the target content with a start and end tag. Thus, each vocabulary word in our markup language can be expressed in one of three ways: a start tag, an end tag, or an empty tag. Table 3.2 demonstrates all three tag types.

The start and end tags are used to demarcate the start and end of the tagged content, respectively. The empty tag is used to embed semantic information that does not surround content. A good example of the use of an empty tag is the image tag in HTML, which looks like this: . An image tag does not need to surround content, as its purpose is to insert an image at the place where the tag is. In other words, its purpose is to be embedded at a specific point in raw content and not to surround content. Thus, we can now extend our first principle of XML to this: *Markup is separate from content and may contain content*.

An XML element is an XML container consisting of a start tag, content (contained character data, sub elements, or both), and an end tag—except for empty elements, which use a single tag denoting both the start and end of the element. The content of an element can be other elements. Following is an example of an element:

<footnote>
<author> Michael C. Daconta </author>, <title> Java Pitfalls </title>
</footnote>

Here we have one element, called "footnote," which contains character data and two subelements: "author" and "title."

Table 3.2 Three Types of XML Tags

TAG TYPE	EXAMPLE
Start tag	<author>
End tag	</author>
Empty tag	<img />

Another effect of tagging content is that it divides the document into semantic parts. For example, we could divide this chapter into <chapter>, <section>, and <para> elements. The creation of diverse parts of a whole entity enables us to classify, or group, parts, and thus treat them differently based on their membership in a group. In XML, such classification begins by constraining a valid document to be composed of a single element, called the *root*. In turn, that element may contain other elements or content. Thus, we create a hierarchy, or tree structure, for every XML document. Here is an example of the hierarchy of an XHTML document (see sidebar on XHTML):

```
<html>
   <head>
      <title> My web page </title>
   </head>
   <body>
      Go Semantic Web!!
   </body>
</html>
```

Listing 3.3 A single HTML root element.

The second key principle of XML is this: *A document is classified as a member of a type by dividing its parts, or elements, into a hierarchical structure known as a tree*. In Listing 3.3, an HTML document starts with a root element, called "html." which contains a "head" element and a "body" element. The head and body element can contain other subelements and content as specified by the HTML specification. Thus, another function of XML is to classify all parts of a document into a single hierarchical set of parts.

Activate Win
Go to Settings to

Why Should Documents Be Well-Formed and Valid?

The XML specification defined two levels of conformance for XML documents: *well-formed* and *valid*. Well-formedness is mandatory, while validity is optional. A well-formed XML document complies with all the W3C syntax rules of XML (explicitly called out in the XML specification as well-formedness constraints) like naming, nesting, and attribute quoting. This requirement guarantees that an XML processor can parse (break into identifiable components) the document without error. If a compliant XML processor encounters a well-formedness violation, the specification requires it to stop processing the document and report a fatal error to the calling application.

A valid XML document references and satisfies a schema. A schema is a separate document whose purpose is to define the legal elements, attributes, and structure of an XML instance document. In general, think of a schema as defining the legal vocabulary,

number, and placement of elements and attributes in your markup language. Therefore, a schema defines a particular type or class of documents. The markup language constrains the information to be of a certain type to be considered "legal."

What Is XML Schema?

XML Schema is a definition language that enables you to constrain conforming XML documents to a specific vocabulary and a specific hierarchical structure. The things you want to define in your language are element types, attribute types, and the composition of both into composite types (called complex types). XML Schema is analogous to a database schema, which defines the column names and data types in database tables. XML Schema became a W3C Recommendation (synonymous with standard) on May 5, 2001. XML Schema is not the only definition language, and you may hear about others like Document Type Definitions (DTDs), RELAX NG, and Schematron (see the sidebar titled "Other Schema Languages").

As shown in Figure 3.5, we have two types of documents: a schema document (or definition document) and multiple instance documents that conform to the schema. A good analogy to remember the difference between these two types of documents is that a *schema* definition is a blueprint (or template) of a type and each *instance* is an incarnation of that template. This also demonstrates the two roles that a schema can play:

- Template for a form generator to generate instances of a document type
- Validator to ensure the accuracy of documents

Both the schema document and the instance document use XML syntax (tags, elements, and attributes). This was one of the primary motivating factors to replace DTDs, which did not use XML syntax. Having a single syntax for both definition and instance documents enables a single parser to be used for both.

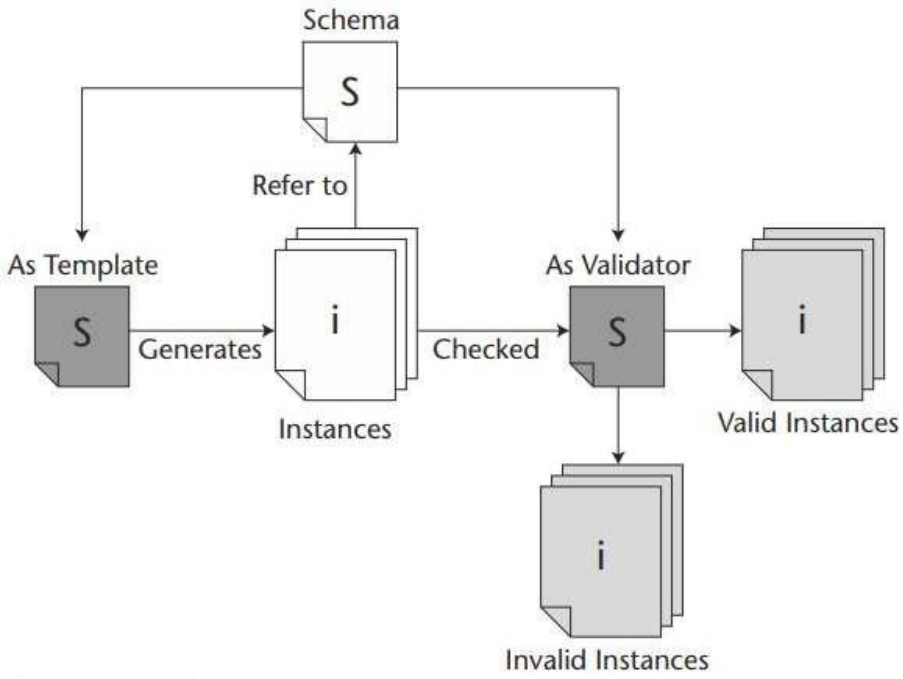

Figure 3.5 Schema and instances.

XML Schemas allow validation of instances to ensure the accuracy of field values and document structure at the time of creation. The accuracy of fields is checked against the type of the field; for example, a quantity typed as an integer or money typed as a decimal. The structure of a document is checked for things like legal element and attribute names, correct number of children, and required attributes. All XML documents should be checked for validity before they are transferred to another partner or system.

Activate \
Go to Settinc

What Do Schemas Look Like?

An XML Schema uses XML syntax to declare a set of simple and complex type declarations. A *type* is a named template that can hold one or more values. Simple types hold one value. Complex types are composed of multiple simple types. So, a type has two key characteristics: a name and a legal set of values. Let's look at examples of both simple and complex types.

A simple type is an element declaration that includes its name and value constraints. Here is an example of an element called "author" that can contain any number of text characters:

```
<xsd:element name="author" type="xsd:string" />
```

The preceding element declaration enables an instance document to have an element like this:

```
<author> Mike Daconta </author>
```

Notice that the type attributed in the element declaration declares the type to be "xsd:string". A *string* is a sequence of characters. There are many built-in data types defined in the XML Schema specification. Table 3.4 lists the most common. If a built-in data type does not constrain the values the way the document designer wants, XML Schema allows the definition of custom data types.

Table 3.4 Common XML Schema Primitive Data Types

DATA TYPE	DESCRIPTION
string	Unicode characters of some specified length.
boolean	A binary state value of true or false.
ID	A unique identifier attribute type from the 1.0 XML Specification.
IDREF	A reference to an ID.
integer	The set of whole numbers.
long	long is derived from integer by fixing the values of maxInclusive to be 9223372036854775807 and minInclusive to be - 9223372036854775808.
int	int is derived from long by fixing the values of maxInclusive to be 2147483647 and minInclusive to be -2147483648.
short	short is derived from int by fixing the values of maxInclusive to be 32767 and minInclusive to be -32768.
decimal	Represents arbitrary precision decimal numbers with an integer part and a fraction part.
float	IEEE single precision 32-bit floating-point number.
double	IEEE double-precision 64-bit floating-point number.
date	Date as a string defined in ISO 8601.
time	Time as a string defined in ISO 8601.

A *complex type* is an element that either contains other elements or has attached attributes. Let's first examine an element with attached attributes and then a more complex element that contains child elements. Here is a definition for a book element that has two attributes called "title" and "pages":

Activate
Go to Settin

```
<xsd:element name="book">
    <xsd:complexType>
      <xsd:attribute name="title" type="xsd:string" />
      <xsd:attribute name="pages" type = "xsd:int" />
    </xsd:complexType>
</xsd:element>
```

An XML instance of the book element would look like this:

```
<book title = "More Java Pitfalls" pages="453" />
```

Now let's look at how we define a "product" element with both attributes and child elements. The product element will have three attributes: id, title, and price. It will also have two child elements: description and categories. The categories child element is mandatory and repeatable, while the description child element will be optional:

```
<xsd:element name="product">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="description" type="xsd:string"
         minOccurs="0" maxOccurs = "1" />
         <xsd:element name="category" type="xsd:string"
         minOccurs = "1" maxOccurs = "unbounded" />
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" />
      <xsd:attribute name="title" type="xsd:string" />
      <xsd:attribute name="price" type="xsd:decimal" />
   </xsd:complexType>
</xsd:element>
```

Here is an XML instance of the product element defined previously:

```
<product id="P01" title="Wonder Teddy" price="49.99">
     <description>
     The best selling teddy bear of the year.
     </description>
     <category> toys </category>
     <category> stuffed animals </category>
</product>
```

An alternate version of the product element could look like this:

```
<product id="P02" title="RC Racer" price="89.99">
     <category> toys </category>
     <category> electronic </category>
     <category> radio-controlled </category>
</product>
```

What Are XML Namespaces?

Namespaces are a simple mechanism for creating globally unique names for the elements and attributes of your markup language. This is important for two reasons:

1. To deconflict the meaning of identical names in different markup languages and
2. To allow different markup languages to be mixed together without ambiguity.

Unfortunately, namespaces were not fully compatible with DTDs, and therefore their adoption has been slow. The current markup definition languages, like XML Schema, fully support namespaces.

Namespaces are implemented by requiring every XML name to consist of two parts: a prefix and a local part. Here is an example of a fully qualified element name:

```
<xsd:integer>
```

The local part is the identifier for the meta data (in the preceding example, the local part is "integer"), and the prefix is an abbreviation for the actual namespace in the namespace declaration. The actual namespace is a unique Uniform Resource Identifier (URI; see sidebar). Here is a sample namespace declaration:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

The preceding example declares a namespace for all the XML Schema elements to be used in a schema document. It defines the prefix "xsd" to stand for the namespace "http://www.w3.org/2001/XMLSchema". It is important to understand that the prefix is not the namespace. The prefix can change from one instance document to another. The prefix is merely an abbreviation for the namespace, which is the URI. To specify the namespace of the new elements you are defining, you use the targetNamespace attribute:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.mycompany.com/markup">
```

There are two ways to apply a namespace to a document: attach the prefix to each element and attribute in the document or declare a default namespace for the document. A default namespace is declared by eliminating the prefix from the declaration:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head> <title> Default namespace Test </title> </head>
<body> Go Semantic Web!! </body>
</html>
```

Here is a text representation of what the preceding document is internally translated to by a conforming XML processor (note that the use of braces to offset the namespace is an artifice to clearly demarcate the namespace from the local part):

```
<{http://www.w3.org/1999/xhtml}html>
<{http://www.w3.org/1999/xhtml}head>
<{http://www.w3.org/1999/xhtml}title> Default namespace Test
</{http://www.w3.org/1999/xhtml}title> </head>
<{http://www.w3.org/1999/xhtml}body> Go Semantic Web!!
</{http://www.w3.org/1999/xhtml}body>
</{http://www.w3.org/1999/xhtml}html>
```

This processing occurs during parsing by an application. *Parsing* is the dissection of a block of text into discernible words (also known as tokens). There are three common ways to parse an XML document: by using the Simple API for XML (SAX), by building a Document Object Model (DOM), and by employing a new technique called *pull parsing*. SAX is a style of parsing called *event-based parsing* where each information class in the instance document generates a corresponding event in the parser as the document is traversed. SAX parsers are useful for parsing very large XML documents or in low-memory environments.

What Is the Document Object Model (DOM)?

The *Document Object Model (DOM)* is a language-neutral data model and application programming interface (API) for programmatic access and manipulation of XML and HTML. Unlike XML instances and XML schemas, which reside in files on disk, the DOM is an in-memory representation of a document. The need for this arose from differences between the way Internet Explorer (IE) and Netscape Navigator allowed access and manipulation of HTML documents to support Dynamic HTML (DHTML). IE and Navigator represent the parts of a document with different names, which made cross browser scripting extremely difficult. Thus, out of the desire for cross-browser scripting came the need for a standard representation for document objects in the browser's memory. The model for this memory representation is object oriented programming (OOP). So , by turning around the title, we get the definition of a DOM: a data model, using objects, to represent an XML or HTML document.

Object-oriented programming introduces two key data modeling concepts classes and objects. A *class* is a definition or template describing the *characteristics* and *behaviors* of a real-world entity or concept. From this description, an in memory instance of the class can be constructed, which is called an object. So,an *object* is a specific instance of a class. The key benefit of this approach to modeling program data is that your programming language more closely resembles the problem domain you are solving. Real-world entities have characteristics and behaviors. Thus, programmers create classes that model realworld entities like "Auto," "Employee," and "Product." Along with a class name, a class has characteristics, known as *data members*, and behaviors, known as *methods*. Figure 3.6 graphically portrays a class and two objects.

The simplest way to think about a DOM is as a set of classes that allow you to create a tree of objects in memory that represent a manipulable version of an XML or HTML document. There are two ways to access this tree of objects:

1. A generic way and
2. a specific way.

The generic way (see Figure 3.7) shows all parts of the document as objects of the same class, called Node. The generic DOM representation is often called a "flattened view" because it does not use class inheritance. Class inheritance is where a child class inherits characteristics and behaviors from a parent class just like in biological inheritance.

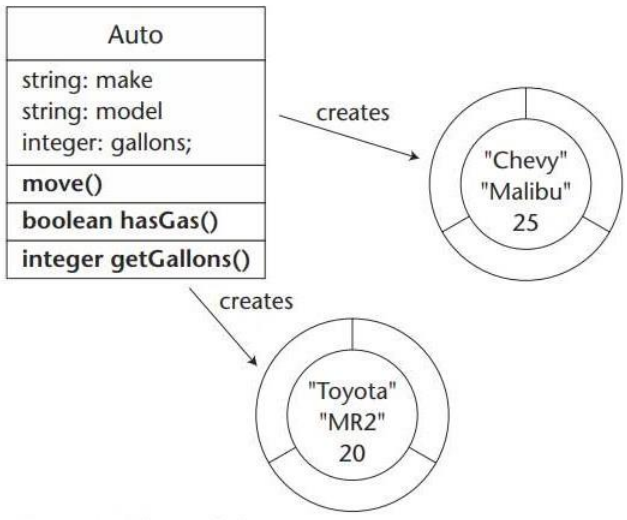

Figure 3.6 Class and objects.

The DOM in Figure 3.7 can also be accessed using specific subclasses of Node for each major part of the document like Document, DocumentFragment, Element, Attr (for attribute), Text, and Comment. This more object-oriented tree is displayed in Figure 3.8.

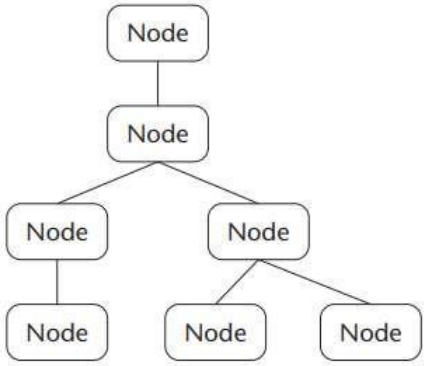

Figure 3.7 A DOM as a tree of nodes.

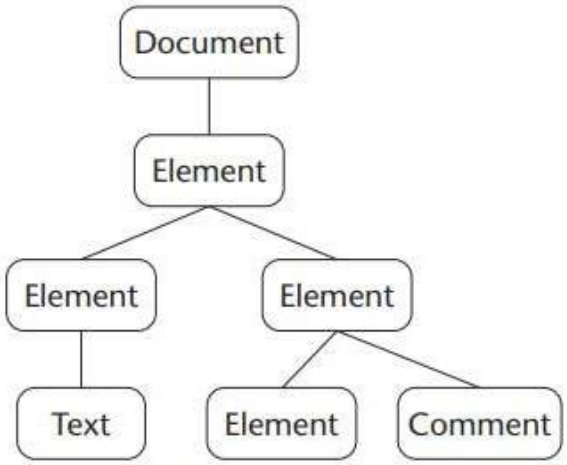

Figure 3.8 A DOM as a tree of subclasses.

There are currently three DOM levels:

DOM Level 1. This set of classes represents XML 1.0 and HTML 4.0 documents.

DOM Level 2. This extends Level 1 to add support for namespaces; cascading style sheets, level 2 (CSS2); alternate views; user interface events; and enhanced tree manipulation via interfaces for traversal and ranges. Cascading style sheets can be embedded in HTML or XML documents in the <style> element and provide a method of attaching styles to selected elements in the document. Alternate views allow alternate perspectives of a document like a new DOM after a style sheet has been applied. User interface events are events triggered by a user, such as mouse events and key events, or triggered by other software, such as mutation events and HTML events (load, unload, submit, etc.). Traversals add new methods of visiting nodes in a tree—specifically, NodeInterator and TreeWalker—that correspond to traversing the flattened view and traversing the hierarchical view.

DOM Level 3. This extends Level 2 by adding support for mixed vocabularies (different namespaces), XPath expressions load and save methods, and a representation of abstract schemas (includes both DTD and XML Schema). XPath is a language to select a set of nodes within a document. Load and save methods specify a standard way to load an XML document into a DOM and a way to save a DOM into an XML document. Abstract schemas provide classes to represent DTDs and schemas and operations on the schemas.

Impact of XML on Enterprise IT

XML is pervading all areas of the enterprise, from the IT department to the intranet, extranet, Web sites, and databases. The adoption of XML technology has moved well beyond early adopters into mainstream use and has become integrated with the majority of commercial products on the market, either as a primary or enabling technology.

1. **Data exchange and interoperability.**

XML has become the universal syntax for exchanging data between organizations. By agreeing on a standard schema, organization can produce these text documents that can be validated, transmitted, and parsed by any application regardless of hardware or operating system. The government has become a major adopter of XML and is moving all reporting requirements to XML. Companies report financial information via XML, and local governments report regulatory information. XML has been called the next Electronic Data Interchange (EDI) system, which formerly was extremely costly, was cumbersome, and used binary encoding. Easy data exchange is the enabling technology behind the next two areas:

a. ebusiness and
b. Enterprise Application Integration.

2. **Ebusiness**

Business-to-business (B2B) transactions have been revolutionized through XML. B2B revolves around the exchange of business messages to conduct business transactions. There are dozens of commercial products supporting numerous business vocabularies developed by RosettaNet, OASIS, and other organizations. Coca-Cola, IBM, and others have seen major benefits from using XML to streamline their B2B transactions. This has helped them cut costs and boost profits. The future of B2B communication is even brighter with web services, which are like little software applications that businesses can easily connect to and use. This will make it even easier for companies to work together electronically.

3. **Databases and data mining.**

XML has had a greater effect on relational database management systems (DBMS) than object-oriented programming (which created a new category of database called object-oriented database management systems, or OODBMS). XML has even spawned a new category of databases called native XML databases exclusively for the storage and retrieval of XML. All the major database vendors have responded to this challenge by supporting XML translation between relational tables and XML schemas. Additionally, all of the database vendors are further integrating XML into their systems as a native data type. This trend toward storing and retrieving XML will accelerate with the completion of the W3C XQuery specification.

4. **Customer relationship management (CRM).**

CRM systems enable an organization's sales and marketing staff to understand, track, inform, and service their customers. CRM involves many of the other systems we have discussed here, such as portals, content management systems, data integration, and databases, where XML is playing a major role. XML is becoming the glue to tie all these systems together to enable the sales force or customers (directly) to access information when they want and wherever they are (including wireless).

XML is pervading all areas of the enterprise, from the IT department to the intranet, extranet, Web sites, and databases. The adoption of XML technology has moved well beyond early adopters into mainstream use and has become integrated with the majority of commercial products on the market, either as a primary or enabling technology.

5. **Data exchange and interoperability.**

XML has become the universal syntax for exchanging data between organizations. By agreeing on a standard schema, organization can produce these text documents that can be validated, transmitted, and parsed by any application regardless of hardware or operating system. The government has become a major adopter of XML and is moving all reporting requirements to XML. Companies report financial information via XML, and local governments report regulatory information. XML has been called the next Electronic Data Interchange (EDI) system, which formerly was extremely costly, was cumbersome, and used binary encoding. Easy data exchange is the enabling technology behind the next two areas:

   c. ebusiness and
   d. Enterprise Application Integration.

6. **Ebusiness**

Business-to-business (B2B) transactions have been revolutionized through XML. B2B revolves around the exchange of business messages to conduct business transactions. There are dozens of commercial products supporting numerous business vocabularies developed by RosettaNet, OASIS, and other organizations. Coca-Cola, IBM, and others have seen major benefits from using XML to streamline their B2B transactions. This has helped them cut costs and boost profits. The future of B2B communication is even brighter with web services, which are like little software applications that businesses can easily connect to and use. This will make it even easier for companies to work together electronically.

7. **Databases and data mining.**

XML has had a greater effect on relational database management systems (DBMS) than object-oriented programming (which created a new category of database called object-oriented database management systems, or OODBMS). XML has even spawned a new category of databases called native XML databases exclusively for the storage and retrieval of XML. All the major database vendors have responded to this challenge by supporting XML translation between relational tables and XML schemas. Additionally, all of the database vendors are further integrating XML into their systems as a native data type. This trend toward storing and retrieving XML will accelerate with the completion of the W3C XQuery specification.

8. **Customer relationship management (CRM).**

CRM systems enable an organization's sales and marketing staff to understand, track, inform, and service their customers. CRM involves many of the other systems we have discussed here, such as portals, content management systems, data integration, and databases, where XML is playing a major role. XML is becoming the glue to tie all these systems together to enable the sales force or customers (directly) to access information when they want and wherever they are (including wireless).

Why Meta Data Is Not Enough

XML meta data is a form of description. It describes the purpose or meaning of raw data values via a text format to more easily enable exchange, interoperability, and application independence. As description, the general rule applies that "more is better." Meta data increases the fidelity and granularity of our data. The way to think about the current state of meta data is that we attach words (or labels) to our data values to describe it. How could we attach sentences? What about paragraphs? While the approach toward meta data evolution will not follow natural language description, it is a good analogy for the inadequacy of words alone. The motivation for providing richer data description is to move data processing from being tediously preplanned and mechanistic to dynamic, just-in-time, and adaptive.

For example, you may be enabling your systems to respond in real time to a location-aware cell phone customer who is walking by one of your store outlets. If your system can match consumers' needs or past buying habits to current sale merchandise, you increase revenue. Additionally, your computers should be able to support that sale with just-in-time inventory by automating your supply chain with your partners. Finally, after the sale, your systems should perform rich customer relationship management by allowing transparency of your operations in fulfilling the sale and the ability to anticipate the needs of your customers by understanding their life and needs. The general rule is this: The more computers understand, the more effectively they can handle complex tasks

Semantic Levels

Figure 3.9 shows the evolution of data fidelity required for semantically aware applications. Instead of just meta data, we will have an information stack composed of semantic levels. We are currently at Level 1 with XML Schema, which is represented as modeling the properties of our data classes. We are capturing and processing meta data about isolated data classes like purchase orders, products, employees, and customers. On the left side of the diagram we associate a simple physical metaphor to the state of each level. Level 1 is analogous to describing singular concepts or objects. In Level 2, we will move beyond data modeling (simple meta data properties) to knowledge modeling. Knowledge modeling enables us to model statements both about the relationships between Level 1 objects and about how those objects operate. This is diagrammed as connections between our objects in Figure 3.9. Beyond the knowledge statements of Level 2 are the superstructures or "closed world modeling" of Level 3. The technology that implements these sophisticated models of systems is called ontologies.

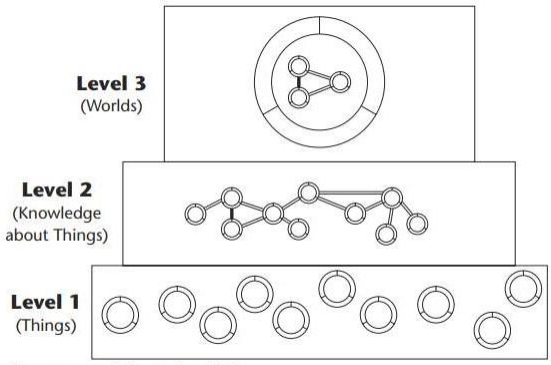

Figure 3.9 Evolution in data fidelity.

Rules and Logic

The semantic levels of information provide the input for software systems. The operations that a software system uses to manipulate the semantic information will be standardized into one or more rule languages. In general, a rule specifies an action if certain conditions are met. The general form is this: if (x) then y. Current efforts on rule languages are discussed in Chapters 5 and 8.

Figure 3.10 Data fidelity evolution in video games.
SpaceWar by Stern, from the Spacewar emulator at the MIT Media Lab

Activa
Go to S

Inference Engines

Applying rules and logic to our semantic data requires standard, embeddable inference engines. These programs will execute a set of rules on a specific instance of data using an ontology. An early example of these types of inferencing engines is the open source software Closed World Machine (CWM). CWM is an inference engine that allows you to load ontologies or closed worlds (Semantic Level 3), then it executes a rule language on that world.

So, meta data is a starting point for semantic representation and processing. The rise of meta data is related to the ability to reuse meta data between organizations and systems. XML provides the best universal syntax to do that. With XML, everyone is glimpsing the power of meta data and the limitations of simple meta data. The following chapters examine how we move beyond meta data toward knowledge.