CHAPTER 3

THE XML MODEL

Programming is a way to translate concrete real-world desires into computer-world operations and programming languages are tools that allow communication between a computer and a developer [46]. Development of programming languages is not a static process. They are created constantly and disappear over the course of their use. The requirements and needs of the users force evolution of new languages. Each new programming language extends the number of operations a programmer can perform and raises the level of abstraction that programmers require to implement an idea. It is when a programmer finds difficulty in implementing his/her idea, a new programming language is developed or the present one is extended.

3.1 Development of Programming Languages and their evolution

Programming languages have been evolving ever since Ada Lovelace and Babbage were writing programs for the project of "difference engine", and then the "analytical engine". In 1945, the german inventor K. Zuse [47] of the Z3 computer worked on defining an evolving language for this engine (with arrays and records). Few documents of the epoch about this language exist. Timeline of general-purpose programming languages is shown in Fig. [48]

Then came Assembly Language which was written for the Assemblers which associated a symbolic name to the Machine-Language code. After that came the IPL (Information Processing Language) in 1956, developed by A Newell, H Simon and JC Shaw. It was a low level list processing language which implemented the recursivity. Then John Backus developed FORTRAN (FORmula TRANslation Language) during 1954-58. It was dedicated to mathematical calculations and introduced SUBROUTINES, FUNCTIONS and LOOPS and FOR Control structure.
Mac Carthy (1958-60) developed Lisp (List Structure Processing) which was purely recursive and there was no difference between the data and code. Algol (1960) was the first machine independent language developed by a consortium of computer specialists. It introduced the concept of BLOCKS of STAMENTS, LOCAL VARIABLES, DYNAMIC ARRAYS, BEGIN....END, GOTO etc.
COBOL (Common Business Oriented Language) was developed in 1960 by CODASYL. It is a classical procedural language which introduced the concept of RECORDS. BASIC (Beginner's All Purpose Symbolic Instruction Code) was introduced by John Kemeney and Thomas Kurtz in 1964. Here each line has a number to allow GOTO statements and Sub programs were called by GOSUB to line numbers.

In 1969 Charles Goldfarb, Edward Mosher and Raymond Lorie developed The GML (Generalized Markup Language) to identify the components of a document with a specified nested syntax [49]. In 1978 the ANSI (American National Standards Institute) set up a standards group to merge the concepts of GenCode and GML. This developed into SGML which was standardized in 1986.

Then came Simula, SNOBOL, PL/1, and Pascal (1970). Pascal (by Nikalaus Wirth) introduced the concept of structured programming and then Turbo Pascal (1983) was developed by Anders Hejlsberg, which was fast having complete IDE. Smalltalk developed by Alan Kay in 1972 was fully object oriented language which ran inside graphical environment with windows, mouse etc.

C was developed by Dennis Ritchie in 1973 to program the UNIX environment system but was later accepted universally due to its portability and speed. Prolog introduced logic programming and then SQL introduced language of query for relational databases.

During the 80s C++ combined object oriented programming and systems programming.

However, one important new trend in language design was an increased focus on programming for large-scale systems through the use of modules, or large-scale organizational units of code. Other important languages developed in the 80s were Ada, Erlang, Perl, Tcl, FL etc.

The 1990s saw the coming of the internet age. Many RAD (Rapid Application Development) languages emerged, which usually came with IDE (Information Development environment). More radical and innovative than the RAD languages were the scripting languages which became more prominent because of their use in
connection with the Web. Other important languages developed during the period were Visual Basic, Python, Java, Javascript, PHP etc.

3.2 The development of the WWW

The NCSA (National Centre for Supercomputer Applications) released the Mosaic browser in 1993[50]. This made internet accessible to all. The Telnet, SMTP and FTP protocols were developed as there was widespread interest to easily transfer files over long distances. The WWW was developed as an additional protocol to the three. Physicists at CERN developed the concept of semantic description where one server would display a document and a client program, called a browser, would read and understand the displayed document. The power of this system was that the displayed document contained pointers (hypertext) to other documents. The protocol developed was the HTTP (Hyper Text Transport Protocol) and each pointer was known as a URL (Uniform Resource Locator). By clicking on the URLs embedded within the displayed document as hypertext links to locate other documents, the user could navigate from Web server to Web server. Hyperlinks can be created in a document via the HTTP protocol or any of the other protocols, namely - FTP, Gopher, Mailto, File, News etc. If a new protocol is developed, then each browser will have to be updated to handle this new feature. Given the amorphous and sometimes chaotic growth of internet, it is very difficult to update all the browsers.

Java was developed to handle this problem. As HTML is a passive language, the browser simply displays web pages in HTML and uses embedded URLs on that page to navigate from one HTML file to another displaying information or pictures. Web services have additional features which permit a more active role for the server. It can acquire information from the client browser and provide for alternative behaviour. Here the web server is used to execute programs and the web pages are programmed. These executable web pages are called applets or mini applications. The existing web structure with HTML documents being read by web browsers has some limitations which are [51]:

1. For each change or addition to a protocol, all browsers need to be updated. It does not help much to just update the server to display the new format because the display process resides in the browser client.
2. The amount of information an HTML process can access is limited to the few simple keystrokes generated by the `<form>` elements of a document.

3. Control of the interaction is handled by the server with the embedded HTML commands in the web pages. This slows down the server, especially at sites that are popular with many “hits” per day. It would be more efficient if control could be managed by the client browser with the server being in more of a supporting role.

4. Displaying web pages is limited by the transmission speed of the communication line. Typically, the user client machine is idle, waiting for information, while the data are being downloaded to a user’s machine. With an increase in the large video, picture and audio components to the web pages, this time delay increases.

Programming language evolution continues in Industry as well as Research. Recent developments include Internet languages, Scripting languages and Markup languages. In the markup languages, the latest trend is to turn XML (Extensible Markup Language) documents into executables.

### 3.3 Development of XML

Web based technologies (on Internet) are extensively being used in almost every field for Web-based documentation and publishing - right from entertainment (music, videos, games) to scientific fields (maths, chemistry). Historically, SGML (Standard Generalized Markup Language) was developed by IBM professionals in 1974 for defining a list of specifications for documents transmitted over the internet. Many application languages were developed based on SGML, out of which HTML (Hyper Text Markup Language) became most popular [52]. In the mid 90’s, due to the rapid growth in the field of web development, there were demands to support new features like images, animations, multimedia, e-commerce etc. on web pages. Thus HTML grew in an unorganized and uncontrolled fashion to cater to the ever-increasing requirements. HTML became display-centric, a deviation from the original intent to keep content and presentation facets of a markup language separate from each other. A working group was formed under the auspices of the World Web Consortium
(W3C) to develop a specification language that could provide the necessary flexibility and scalability to support all necessary features, while keeping the content completely separate from the display and presentation. Thus XML (Extensible Markup Language) was developed. XML is not itself a markup language, but instead a specification for defining a markup language [53]. Extensible language is any High Level Language that allows a user to define syntax in which other Domain Specific languages can be written.

XML is a text and data-formatting language that has a tag-based syntax very similar to the Hyper Text Markup Language (HTML) syntax, but much more capable and much more flexible. It not only prescribes text styles but also defines data types for cross platform communication. XML documents contain only data, so applications that process XML documents must decide how to display the document's data. For example a Personal Digital Assistant (PDA) may render an XML document differently than a wireless phone or a desktop computer would render that document.

XML permits document authors to create “markup” for virtually any type of information. Markup is notation that provides information to an XML parser on how to parse, or read, an XML document; which parts of a document to skip; and which parts of a document to hand over to another application. Consequently, this extensibility enables document authors to create entirely new markup languages for describing specific types of data, including mathematical formulas, chemical molecular structures, music etc.

XML is being used by organizations for representing content online and simplifying data storage and sharing. It simplifies sharing, data transport, platform changes, makes handling of data more easier and efficient for describing document content, validating the correctness, defining data facets and converting data between different data types [53]. XML has been used to create new Internet languages. The inevitable evolution of these languages for domain specific purposes is called versioning. Some of these are XHTML (Extensible Hyper Text Markup Language), WSDL (Web Service Definition Language), WAP (Wireless Application Protocol) & WML (Wireless Markup Language, RSS (Really Simple Syndication), RDF (Resource Description Framework), OWL (Web Ontology Language) and SMIL (Synchronized Multimedia Integration Language) [54].
XML is non proprietary with no copyrights, patents, trade secrets or Intellectual Property restrictions involved [54]. Domain specific languages developed by extending XML enable users to exchange information within and outside their fields. It is expressed in an EBNF and has six markups, namely elements, entities, references, comments, processing instructions, marked sections and document type declarations (DTDs).

Moreover, the XML is integrated and is improved in a new technology called XML Web Services. The real power of the XML Web Services is that it lets applications share data and invoke capabilities from other applications without regard to how those applications were built. XML is also independent and can work with any operating system or platform they run on and different devices are used to access them. While XML Web Services remain independent of each other, they can loosely link themselves into a collaborating group that performs a particular task.

XML has many additional features when we compare it with HTML. HTML has a fixed tag set and tag semantics and does not provide for arbitrary structure. XML, on the other hand allows unlimited flexibility and freedom to the author of the document in terms of structure and content. It allows self defining tags and markups. It allows them to set standards defining the information that should appear in a document, and in the sequence desired. XML, in combination with other standards, makes it possible to define the content of a document separately from its formatting, making it easy to reuse that content in other applications or for other presentation environments. In XML, the content is developed first, and then instructions about how to display the content are provided in a separate Stylesheet attached to the document. This technology enables authors, programmers, designers to work more independently with each other. Multiple stylesheets can be applied to a single document and multiple styles can be applied to a single element. XML provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations without needing to pass through many layers of conversion. XML documents are read in Unicode which is a platform-independent character set that includes almost all character sets from most of the languages being used across the world. Unicode assigns each character set a unique code.
XML makes it possible to create new systems for data management and organization without many of the incompatibilities and complexities that were associated with the older systems [55]. Data embedded within HTML pages needs to be reprocessed by special purpose, page specific parsers before meaningful queries can be posed. If the same information is provided in XML, the XML encodings are compatible. In this case the structure and the meaning of the data, as well as data itself is readily parsable and sets the stage for powerful queries [54].

3.3.1 Definition of Some Terms

1. XML

XML is a subset of the Standard Generalized Markup Language (SGML) defined in ISO standard 8879(1986) that is designed to make it easy to interchange structured documents over the Internet [55]. XML files always clearly mark the beginning and end of each of the elements of an interchanged document. XML restricts the use of the SGML constructs to ensure that fallback options are available when access to certain components of the document are not currently possible over the Internet. With XML, a developer can use any tags he wants, and the browser does not automatically understand the meaning of these tags. For example the tag <body> could mean an HTML text body or perhaps the human body in a medical article. Because of the nature of XML, no standard way to display an XML document exists. It is flexible depending upon the requirements.

2. Metadata

A common short definition of metadata is that it is “data about data”[55]. Actually, metadata can be data about almost anything. Its purpose and usage make it metadata, rather than its content or structure. Most often, metadata is designed to support people or programs in locating and retrieving information resources. A piece of data can be metadata to one application, and just data to another. Metadata is relatively short, has a simple structure, and is so familiar that one may not realize that he or she is using metadata every day. The most common example is when using a collection of tapes or CDs. If all of them were in blank boxes, and one had to play each of them to see what was recorded on it, then finding a particular film or track would be a long and difficult job. However, if one places a label with the name of the tape or the CD and a list of
the songs on each of them, then picking out the right one immediately becomes very much easier. Extending this example to a large library or the World Wide Web, the problem of finding exactly what you are looking for is clearly enormous, but the basic concept of the solution remains the same. Make some simple, relevant, searchable information available, together with location information, and searching and retrieval becomes much easier.

Unfortunately, when the information pool is very large, a large number of relevant results can be found, which must be filtered and ranked according to the specific user’s needs. XML metadata can take many different forms. It may be embedded in an XML document alongside the information it is about, or it may be held in a separate XML document. In the second case the XML metadata could identify which information it is about. This is accomplished by using, for example, a Unified Resource Identifier (URI). In order to understand better the breadth of metadata some examples of different kinds are presented below:

**Annotation:** These are side notes added to a document for a specific purpose, and they will be read by some or by all of the readers but at different times and for different purposes.

**Cataloguing and Identification:** In this kind of metadata there is an association between specific properties and their values with whatever the metadata is about. For example, a music store catalogue record for a music CD, gives its title, singer and publisher. In XML applications, this metadata is usually information about the properties of information resources, leading to the general name “resource based metadata.”

**Subject Indexes:** This refers to metadata that represents subjects and their interrelationships, and also usually designates specific information resources as belonging to these subjects. A closely related kind of metadata is the already mentioned example with the tapes and CDs. Those kinds of metadata are usually referred to as “subject based metadata.”

**Cross-references:** These are an important kind of metadata for long-term collections of documents where documents are often cross referenced in ways their original authors did not foresee. In complex independent collections, such as legal codes and
cases, cross-references become structural mapping. In practice, most technologies include aspects of both resource based and subject based metadata. This is partly because if the author wants to represent a subject inside a computer, he will probably end up doing so either by identifying the subject with an information resource such as a thesaurus entry or by using a name for the subject as a value of a property. In the music store example, the catalogue record can also give extra information for each song contained in it.

3. XML Schema, XSL and XLink

XML Schema

Traditionally DTDs have been used to define data models. However, the facilities available in the DTD syntax to express the properties of the data model are limited. For example, it is not possible to declare a data type or value space of an attribute. Also, there is no way to define a cardinality to 2n which is very common in agricultural spatial applications. In order to overcome these limitations, the W3C has developed a more powerful XML-based schema definition language, named XML Schema [56]. The XML Schema specification is currently divided into two separate documents, Structures and Data Types. The structures specification defines all the facilities for constraining the logical structure of an XML dataset, providing mechanisms to constrain the hierarchical relationships between the elements, allowed attributes and their types, attribute value spaces, default values etc. The XML Schema Data Type specification defines all the built-in primitive data types of the XML schema language and establishes the mechanisms for constructing new application-specific data types.

W3C XML Schema [56] provides a mechanism called a wildcard, <xs:any>, for controlling where elements from certain namespaces are allowed. The wildcard indicates that elements in specified namespaces are allowed in instance documents where the wildcard occurs. This allows for later extension of a schema in a well-defined manner. Consumers of extended documents can identify and, depending upon the processing model, safely ignore the extensions they don't understand.

<xs:any> uses the namespace attribute to control what namespaces extension elements can come from. The most interesting values for this attribute are: "##any", which means one can extend the schema using an element from any possible
namespace; "##other", which only allows extension elements from namespaces other than the target namespace of the schema; and ##targetnamespace, which only allows extension elements from the target namespace of the schema. `<xs:any>` also uses the processContents attribute to control how a XML parser validates extended elements. Permissible methods include “lax” - validate any elements from supported namespaces but ignore all other elements, “strict” – validate all elements, and “skip” – validate no elements. “Lax” validation is recommended by a majority of XML developers, as it is the most flexible and is the typical choice for web services specifications. The main goal of the "Must Ignore" pattern of extensibility is to allow backward and forward compatible changes to documents.

In general, an extension can be defined by a new specification that makes a normative reference to the earlier specification and then defines the new element. No permission should be needed from the authors of the specification to make such an extension. In fact, the major design point of XML namespaces is to allow decentralized extensions. The corollary is that permission is required for extensions in the same namespace. A namespace has an owner; non-owners changing the meaning of something can be harmful.

Attribute extensions do not have non-determinism issues because the attributes are always unordered and the model group for attributes uses a different mechanism for associating attributes with schema types than the model group for elements.

Ideally, producers should be able to extend existing XML documents with new elements without consumers having to change existing implementations. Extensibility is one step toward this goal, but achieving compatibility also requires a processing model for the extensions. The behavior of software when it encounters an extension should be clear.

Each schema language has its own capabilities and limitations, hence multiple schema languages may be required to express all the additional constraints ex. Schematron does not have loops and variables. Using a Programming language to code is a good option but there are other XML technologies that could be used to express the additional constraints in a declarative manner, without going through the compiling, linking and executing effort as is required while using a programming language. Using XSLT/XPath Stylesheet, application specific constraint checking can be achieved.
XSL

In order to display XML documents, having a mechanism to describe how the document should be displayed is necessary. The extensible Stylesheet Language (XSL) is such a mechanism, which is a W3C standard specification based on XML [52]. XSL is actually a language for translating content from one application of XML to another. XSL consists of two parts, a method for transforming XML documents and a method for formatting XML documents. As an example, we can think of XSL as a language that can transform XML into HTML, a language that can filter and sort XML data, and a language that can format XML data, based on the data value, like displaying negative numbers in red. XSL can be used to define how an XML file should be displayed by transforming the XML file into a format that is recognizable to a browser. Normally XSL does this by transforming each XML element into an HTML element because HTML is a browser recognizable format. XSL can also add completely new elements into the output file, or can remove elements. It can rearrange and sort the elements and test and make decisions about which elements to display, and a lot more. It uses templates that match elements in the input document, changing them to different elements in the output document. XSL also includes many procedural constructs, such as IF statements, to control the conditional output content. The advantage of XSL is that it can process XML without knowing the significance or nature of the content.

XLink

Some information, which is critical for a farmer, and which may change periodically (example nitrogen concentration in a fertilizer, or contents of a pesticide) are copied directly into XML instances by various farm management systems (FMS). If, at any time, the contents of the fertilizer or pesticide has been changed by the company and the same have not been updated into the FMS, the farmer could use them in a strength which is quite different from the recommended one, which may spoil his crop. Hence, in the system in the present study, such vital information would be left at the place it is produced. What would be done to ensure framers getting the latest information is to use XLink Standard to provide an “external” link to reference the information. Uniform Resource Identifiers (URIs) references offer an excellent mechanism to establish a link from a data item into an additional resource available
on the network. It provides globally unique identifiers. The mechanism is the same that is used in hyperlinks of HTML documents. This would enable real distributed data storage and also link data to build real knowledge bases.

The XLink specification defines an element called “extended link”, which enables several resources to be listed as participating in the link [51], [52]. A “locator” element is used for this purpose. An individual relationship between two resources is defined by a separate “arc” element. An arbitrary number of relationships can thus be defined among the locators. A specified link construct innovates a notion of a link for which all the locators are remote. Dedicated link databases are supposed to be established to store these links. Two separate specifications, namely XPath and XPointer are available as flexible addressing mechanism to refer to external links.

The extended links can include multidirectional links between many documents. An extended link consists of a set of resources and a set of the connections between them. The resources could be local (part of the extended link element) or remote (generally in another document).

**XPath and XPointer**

XPath is a fourth generation declarative language for locating nodes in XML documents. An XPath location path says which nodes from the document are required. A user simply requires to pass an XPath statement to a method, and the XPath engine is responsible for figuring out how to find all the nodes satisfying that expression. The user is not concerned about what algorithm is used to search the XML document. XPath is much more robust than writing the detailed search and navigation code using DOM or SAX [51].

Traditional URLs point to a single, complete document. XPointer serves a very important purpose if we want to link to a particular element or group of elements on a page without having to change the document to which the link is desired. So XPointers can address individual elements. Using XPointer a given element can be targeted by number, name, type or relation to other elements in the document.
3.3.2 XML Data storage

XML can handle all kinds of data, including text, images and sound, and is user-extensible to handle anything special. One of the main tasks is that how XML-tagged data can be managed. A suitable and convenient solution is to use databases to store, to retrieve and to manipulate XML. The primary advantage of databases is that users are familiar with them and their behavior, so combining XML with a database context seems natural.

Storing XML documents in a relational or object-oriented database can be done in different ways. First, one could extract the data elements in an XML document and store them as data rows and columns in an SQL database. This technique is called “element storage.” Given an example of an XML document, a number of SQL tables could be created with columns for the individual elements of document. Then, those kinds of data can be managed in SQL with normal SQL operations. By retrieving the data, an XML document can be produced and published or transferred via the Web. In case there are new data that must be stored, an “UPDATE” operation, using common SQL commands, must be done. “Element storage” has the advantage that all of the data from the XML document is available to SQL as normal SQL data so that it can be queried and updated with SQL operations. However, element storage has the disadvantage of the extra overhead of assembling and disassembling the XML documents for interchange.

One can also store the XML document in a single SQL column. Since XML is primarily a file format, a natural storage mechanism is simply a flat file[52]. It is usually referred to as “document storage.” This approach has some drawbacks, such as data isolation problems, integrity checks and concurrent-access problems. However, the wide availability of XML tools that work on file data makes it relatively easy to access and to query XML data stored in files. Thus, this storage format may be sufficient for some kinds of applications. Using any example, a table can be created having a column for the specific category of system’s document. The data-type of that column could be SQL text, or a Java class designed for XML documents in general, or a Java class designed specifically for each specific type of the XML document. “Document storage” eliminates the need for assembling and disassembling the data for interchange. However, there is a need to use Java (or any other language)
methods to reference or to update the elements of the XML documents while they are in SQL, which could be slower and less convenient than the direct SQL access of element storage.

Finally, third option is to use a combination of both the above mentioned approaches, called “hybrid storage” that exploits the advantages of both [57]. This mainly stores an XML document in an SQL column, but at the same time extracts some of its data elements into separate columns for faster and more convenient access. Given a previous example, one can create SQL tables for document storage, and then one can include (or later add) just one or more columns to contain elements extracted from the same type of documents for element storage. Hybrid storage balances the advantages of element storage and document storage, but has the cost and complexity of redundant storage of the extracted data.

### 3.3.3 XML Parsers

Since XML documents are character data, they can be analyzed and their data can be extracted using SQL character string operations. This process can sometimes become very complicated and tedious. Using XML with SQL is greatly facilitated by using XML tools written in Java (or C++,,) which are called “parsers,” and their main job is to analyze and to validate XML documents [57]. An XML can be integrated into the Java applications with the Java API for XML Processing (JAXP). JAXP allows applications to parse and transform XML documents using an API that is independent of any particular XML processor implementation. Through a plug-in scheme, developers can change XML processor implementations without altering their applications.

Specifically, XML parsers provide many capabilities, including the following:

- Checking that a document is well formed and valid
- Handling character set issues
- Generating a Java representation of a document’s parse tree
- Building or modifying a document’s parse tree
- Generating a document’s text from its parse tree

The XML parsing process operates in three phases:
1. **XML input processing.** In this stage, the application parses and validates the source document; recognizes and searches for relevant information based on its location or its tagging in the source document; extracts the relevant information when it is located; and, optionally, maps and binds the retrieved information to business objects.

2. **Business logic handling.** This is the stage in which the actual processing of the input information takes place. It might result in the generation of output information.

3. **XML output processing.** In this stage, the application constructs a model of the document to be generated with the Document Object Model (DOM). It then either applies XSLT style sheets or directly serializes to XML.

   SAX (the Simple API for XML) and DOM are the most common processing models [52]. If a SAX-based parser is used to process an XML document, the method has to be coded to handle events thrown by the parser as it encounters the different tokens of the markup language. Because a SAX parser generates a transient flow of events, the XML input processing steps described above (parsing, recognizing, extracting, and mapping) must be performed in a single cycle: Each event encountered is handled immediately and the relevant information is passed on with the event. SAX-based parsers fall into the category of “push parsers”. A push parser reads through an entire XML document. As it encounters various parts of the document, it notifies a listener object. SAX provides facilities for specifying input sources, character sets, and routines to handle external references. It generates events during the parse so that user routines can process the document incrementally, and it returns a DOM object that is the parse tree of the document.

   If a DOM-based parser is used, the code has to be written to walk through the tree-like data structure that the parser will create from the source document. With DOM, the XML input processing is done in at least two cycles. First, the DOM parser creates a tree-like data structure, called a DOM tree, that models the XML source document; then the application walks through the DOM tree, searching for relevant information to extract and further process. This last cycle can be repeated as many times as necessary, since the DOM tree persists in memory. DOM-based parsers fall
into the category of “model parsers”. A model parser reads an entire document and creates a representation of the document in memory.

There are many XML parsers available in Java, often with a free license or public domain. Most XML parsers implement two standard interfaces, the Simple API for XML (SAX) and the Document Object Model (DOM). Applications that use the implementations of the SAX and DOM interfaces of an XML parser can be portable across XML parsers.

3.3.4 Mapping issues

Another important issue about the XML storage is where the mapping should be done [58]. Almost all the languages that can manipulate XML can run in both the client and the server. For example, Java methods can be executed in either the client or server environment, giving us a choice of which environment to map an XML document to or from SQL. This is a consideration only for element storage and hybrid storage, since document storage involves little or no processing of the document. If the main priority is the efficiency, then mapping should be done in the client and only the SQL data should be transferred between the client and the server. This approach cannot provide users with a high level of security [58]. When security is the priority, the entire process must be done in the server where the appropriate security policy can be enforced.

In general, there are two common approaches to mapping XML to databases: a table-based mapping and an object-relational (object-based) mapping. Both are commonly used as the basis for software that transfers data between XML documents and databases, especially relational databases. An important characteristic in this respect is that both methods are bidirectional, allowing XML documents to be both stored and retrieved.

Table-based mapping is the most obvious mapping. It views the document as a single table or a set of tables, and column data can be represented either as PCDATA-only elements or as attributes. Table-based mapping uses a document structure similar to this:
This approach models pure data without a document and is similar to the relational model. However, it works only with a limited subset of XML documents.

With object-based mapping, XML-DBMS maps the XML document to the database according to an object-relational scheme in which element types are generally viewed as classes, and attributes and PCDATA are treated as properties of those classes. This models the XML document as a tree of objects that are specific to the data in the document and then maps these objects to the database.

The intrinsic flexibility of XML provides a number of tangible benefits to the users. XML allows the integration of data from completely different sources, with middleware application extracting data from different databases and translating it into neutral XML for client side processing [57]. Such XML documents can contain description of data, which can be used in multiple applications, each using specific tags, and delivering appropriate views to the users for their particular tasks. The data delivered to the client can be manipulated in a variety of ways without having to be re-processed by the server into another format or another view. Standard web protocols such as HTTP are used to accomplish such display.

The XML specification fixes only the logical structure and exact syntax of the language and not the individual tags used in the markup. This fact has led to the development of a number of XML vocabularies, adapted to various fields. These vocabularies enable the semantics of the data to be indicated, and making the web a vast repository of self-describing and smart data. In the agricultural application
domain, the user of XML syntax for transmission of data-sets has also been discussed. XML provides a neutral support for the hierarchical data structure present in the data of agricultural domain.

3.3.5 Multichannel access

XML documents aren’t just for web pages. They can be shown on TV screens, printed on paper, bound in book form, read by speech synthesizers, beamed to Palm Tops and projected on to movie screens [51], [52], [59]. All the style information is placed in a separate document called the style sheet rather than being stored as part of the document itself. Hence a single XML document can be formatted in many different ways just by changing the style sheet. Different style sheets can be designed for different purposes – for print, for the Web, for presentations and other uses – all with the styles appropriate for the specific medium, and all without changing any of the content in the document itself. Cascading Style sheets (CSS) allow a developer to vary styles to match the medium in which the content is displayed. A CSS style sheet contains a list of rules. Each rule gives the name of the elements it applies to and the styles to be applied to those elements.

Fig. 3.2 Multichannel access through XSLT

In the XSL Transformation process, XML combines with CSS to create a document that is rendered on a web browser or other user agent (UA)[52]. The process of styling requires an XML source document that contains the information to be
displayed and a style sheet that describes how the document should be presented. The series of steps involved in the process are[51]:

i) An XML parser interprets the XML document and forms a tree

ii) The tree is handed off to an XSLT processor

iii) The XSLT processor compares the nodes in the tree with the instructions contained in the referenced style sheet (XSL)

iv) When the XSLT processor finds the match, it outputs the resulting tree fragment

v) The tree is output to a UA in a format that could be HTML, speech or text, depending upon the processing instructions (PIs) given in the XML document.

An XML document optionally can reference another document which defines that particular XML document’s structure. This referenced document is either a Document Type Definition (DTD) or a Schema. A DTD expresses the set of rules for the document structure using an Extended Backus-Naur Form (EBNF) grammar. Unlike DTDs, Schemas do not use EBNF grammar. Instead they use XML syntax and are actually XML documents that programs can manipulate like other XML documents. Schemas are more flexible and more powerful than DTDs, and many researchers in the XML community believe that Schemas will prevail over DTDs [52]. When an XML document references a DTD or Schema, some parsers, which are also called validating parsers, can read the DTD or the Schema and check the XML document according to the structure that the DTD or Schema defines. If the XML document conforms to the DTD/Schema, then the XML document is “valid.” Those parsers that cannot check for document conformity against DTD/Schemas are called non-validating parsers. If an XML document can be processed successfully by a validating or a non-validating parser, which means that the document is syntactically correct, then that document is called “well formed.” By definition, a valid XML document is also well formed.

Such improvements in IT provide new opportunities to improve the utilization and performance of livelihood technologies such as education, agriculture, health and medical services. The research efforts for utilizing IT in agriculture have been described in Chapter 2.
3.3.6 Interoperability and Extensibility

Interoperability and extensibility are the two components which make XML a very powerful medium for sharing different applications and adding instances to include additional features in a system which are compatible.

**Interoperability**

Interoperability is defined as the ability of software and hardware on different machines from different vendors to share data [51],[60]. Interoperable applications can share data and invoke capabilities from other applications without regard to how those applications were built. XML is also independent and can work with any operating system or platform they run on and any devices used to access them. While XML Web Services remain independent of each other, they can loosely link themselves into a collaborating group that performs a particular task.

**Extensibility**

Extensibility allows authors to change instances without going through a centralized authority, and may allow the centralized authority greater opportunities for versioning [61]. The common characteristic of a compatible change is the use of extensibility. Designing extensibility into languages typically results in systems that are more loosely coupled. A fundamental requirement for extensibility is to be able to determine the language of elements and attributes. XML namespaces provide a mechanism for associating a URI with an XML element or attribute name, thus specifying the language of the name. This also serves to prevent name collisions. HTML did not have the ability to distinguish between the namespaces of extensions. This meant that authors could produce the same element name but with different interpretations, and software had no way of determining which interpretation was applicable. This motivated authors to move from HTML to the XML vocabulary.

3.4 Vocabulary

One way in which comprehension of standards may be improved, is a controlled vocabulary. Such a controlled vocabulary contains all the terms which are required and provides a unique opportunity to define each term. In order to process the
standard, it is then necessary to translate each term into the vocabulary internally used (could be natural language or database schema). It should be necessary to perform this step once for each controlled vocabulary, after which any document written in the controlled vocabulary may be interpreted uniquely.

A common way of doing this is the namespace and qualified name system used in XML. In the simplest form, a controlled vocabulary may be expressed as a content list, which is an enumeration of the allowable terms. An extension of the content list is the dictionary where each term is accompanied by a definition, and to provide the definition of terms, an ontology is used.

Use of an existing basic agricultural vocabulary, such as AGROVOC (FAO, 2009), should be considered. The criteria for the content of the vocabulary are the same as the information required as well as the types of the required rules. Main considerations for the vocabulary should be interoperability between the rules of different organisations. An existing vocabulary can also be used either as a base or together with an additional vocabulary.

Two main standards of metadata description exist. One of these and most widely used is the "Dublin Core" standard. The second one is the ISO19100-series, which includes the ISO19115, ISO 19119 and ISO19139 standards.

3.4.1 Dublin Core

"The Dublin Core Metadata Element Set is a vocabulary of fifteen properties for use in resource description. The fifteen element 'Dublin Core' described in this standard is part of a larger set of metadata vocabularies and technical specifications maintained by the Dublin Core Metadata Initiative (DCMI)” (DCMI, 2004). The Dublin Core metadata elements which have been accepted as ISO15836(2003), allow a very generic description of any resource (typically used for webpages), including title, publisher, date, subject, language, contributors, resource type, spatio-temporal coverage, etc.

3.4.2 ISO19115

ISO19115 applies to the full description of datasets, dataset series, and individual geographic features and feature properties. A minimum set of metadata is defined, together with optional elements and extension points. The metadata model bases on the model of geographic information developed in the ISO19100-series standards. As well as basic metadata (identification, description, origin, actuality, usage, extent) as is included in Dublin Core, additional information regarding data quality, spatial representation and reference system, portrayal and application schema may be recorded.

ISO19119 (2005) describes a taxonomy of geographic web services and both platform-neutral and platform-specific service specifications. In particular, the metadata relevant needed for such services is specified. ISO19139 (2007) provides an XML schema for encoding of the ISO19100-series metadata, allowing metadata for geographic information and geographic web services to be represented in machine-readable form for transfer between systems.

ISO19115 (encoded using ISO19139) allows a much more in-depth and formalized description of datasets than Dublin Core. However, due to its greater simplicity, Dublin Core is more preferable and more widely used in practice. But where more precision is required, the ISO19100-series is being used.

3.5 Ontology

Ontology is defined as an “explicit specification of a conceptualization”[7]. Ontologies provide formal semantic representation that describes a particular, real world domain. Two strongly-linked languages for representing ontologies are in use, RDF and OWL. Both of these are recommendations from the W3C and are widely used.

3.5.1 RDF

Resource Description Framework (RDF) is a family of specifications intended for the modelling of metadata. RDF is an abstract model based on triplets of subject, predicate and object, therefore it has several serialization formats of which two are currently in common use. RDF is intended for automated processing and there exists query and inference languages for RDF of which SPARQL is currently dominant.
RDF schemata are specified by RDFS, which is a knowledge representation language for specifying RDF vocabularies (ontologies) [51].

### 3.5.2 OWL

Web Ontology Language (OWL) is a family of extension to RDFS intended for the definition of ontologies. OWL retains the functionality of RDFS and three levels of OWL exist. OWL Lite is the most simple implementation of OWL and is restricted to simple classification tasks, OWL DL is the compromise between expressiveness and computability, and retains the computation power of description logics. OWL Full is the most expressive version of OWL[63]. OWL is commonly serialized to RDF/XML syntax which makes automated handling of OWL ontologies feasible with existing tools. A plain XML schema (without RDF) is also available for OWL, which is also compatible with many tools.

### 3.6 SOAP (Simple Object Access Protocol)

SOAP is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It employs XML for its message format, and usually relies on other Application Layer protocols, most notably Remote Procedure Call (RPC) and Hypertext Transfer Protocol (HTTP), for message negotiation and transmission. SOAP can form the foundation layer of a web services protocol, and provide a basic messaging framework upon which web services can be built. This XML based protocol consists of three parts: an envelope, which defines what is in the message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing procedure calls and responses.

As an example of how SOAP procedures can be used, a SOAP message could be sent to a web-service-enabled web site, for example, a real-estate price database, with the parameters needed for a search. The site would then return an XML-formatted document with the resulting data, e.g., prices, location, features. Because the data is returned in a standardized machine-parseable format, it could then be integrated directly into a third-party web site or application.
3.7 Analysis and Requirements of DBMS Architecture

In this section we discuss the basic requirements that users expect in the performance of DBMS. Some DBMS architectures are also discussed.

3.7.1 Basic Requirements

The DBMS that are built to manage a large amount of documents, like those in large organization must, at least, include a number of capabilities that will assure an acceptable level of functionality[63]. First of all, the DBMS has to be “scalable”, in both performance capacity and incremental data volume growth. The proposed solution scales in a near - linear fashion and behaves consistently as the database increases in size, the number of concurrent users and the complexity of queries. DBMS must also have a “powerful” design in order to support complex decisions with multi-users and a mixed workload. The optimizer should be mature enough to support every type of query with good performance, and must also determine the best execution plan based on the changing data demographics. The optimizer must also check on the conditional parallelism. Finally, it must check on the dynamic and controllable prioritization of resources for queries.

“Manageability” through minimal System Administrator intervention is another feature that must be present in DBMS architecture. System administration can be simpler if the DBMS provides a single point of control, which will allow one to create and implement the new tables at any time. In order to support the critical applications of a large-scale organization’s mission, the DBMS must have a high “availability” level. Any down time and any issues that might deny or degrade service to end-users must be completely transparent to the system administrators. These down time requirements could include batch load times, software/hardware upgrades, severe system performance issues and system maintenance outages.

The design and the system architecture should also be “flexible” and “extensible” to keep pace with evolving business requirements and to improve the value of the existing investment in hardware and applications. The impact of repartitioning tables and the addition or deletion of columns must be minimal. The design should also provide optimal performance across the full range of normalized, star and hybrid data schemas with large numbers of tables.
Nowadays, “interoperability” through the web or internal networks has also become a major factor when deciding on a DBMS. Its ability to support multiple applications from different business units, leveraging data that is integrated across business functions and subject areas is considered a critical feature of the DBMS.

As more business transactions are conducted over the Web, securing data in motion and user identities is a growing concern. User management and deploying secure infrastructures have become one of the top priorities of the database administrators. The architecture of the DBMS should protect the data stored in the database, when transferred from unauthorized access, and from malicious destruction or alteration and accidental introduction of inconsistency. Encryption forms the basis for secure authentication of users. Even though absolute protection of the database from malicious abuse is not possible, a sufficient increase in the required cost to the perpetrator can be a deterrent [59]. In order for the DBMS to be protected against malicious or unauthorized access, several forms of authentication and authorization must be enforced. The system should also allow the users to grant some forms of authority to other users ensuring at the same time that this authorization can be revoked at some future time. Rules help to assign specific sets of privileges to different groups of users inside an organization. When the stored data are highly sensitive, the various authorizations provided in the database may not be sufficient. In such cases, data must be encrypted. Only the user who knows how to decrypt and possesses the necessary decryption key should be allowed to read the data. Compatibility with the modern techniques of encryption-decryption, Public Key Infrastructure (PKI), and Digital Signatures are highly required when deciding on a DBMS for today or for the future world.

3.7.2 XML and DBMS Architectures

Extensible Markup Language (XML) is emerging as the format of choice for a variety of types of data, especially documents. Providing its ability to tag different fields, XML makes searching simpler and more dynamic. It is also ideal for organizations trying to integrate incompatible systems because it can serve as a common transport technology for moving data around in a system-neutral format. In addition, XML can handle all kinds of data, including text, images and sound, and is user-extensible to handle anything special. One of the main concerns until now has been to manage the
XML-tagged data. A suitable and convenient solution is to use databases to store, to retrieve and to manipulate XML. The idea is to place the XML-tagged data in a framework where searching, analysis, updating and output can proceed in a more manageable, systematic and well-understood environment. The primary advantage of databases is that users are familiar with them and their behavior, so combining XML with a database context seems natural.

A few different approaches exist regarding the use of XML in a database. The main categorization is according to the format that the DBMS uses to store the XML document [63]. Many categories store XML in its native format, while others transform it and store it in a common relational or object-oriented database. There are a number of reasons to use existing database types and existing database products to store XML, even if it is not in its native form. First, ordinary relational and object-oriented databases are well known, while native XML databases are new. Second, as a result of familiarity with relational and object-oriented databases, users understand their behavior, especially regarding performance. Many businesses are reluctant to move to a native XML database whose characteristics, especially scalability, have not been tested [63].

Finally, relational and object-oriented databases are safe choices in the corporate mind. On the other hand, one of XML’s attractive features is its hierarchical organization. Relational databases must map XML to relational tables and therefore convert XML structures into rows and columns each time data is needed. In addition, translating XML to and from the database requires considerable processing, especially for large or complex documents. This performance factor may be important when dealing with web pages through a dial-up Internet connection, but at the same time this performance factor may be of slight importance if it is used in an Intranet or through a high-speed Internet connection.

3.7.3 XML and Java

Java revolutionized the programming world by providing a platform-independent programming language. XML takes the revolution a step further with a platform-independent language for interchanging data. Java and XML can be put together to build real-world applications in which both the code and the data are truly portable.
3.8 Different Layered Architectures

Layer

A layer [64] is a fraction of code that can be used again and which performs a particular function. In the network environment, a layer is typically a set of connections as a project that symbolizes this particular purpose. This particular layer works with other layers to carry out some particular aim.

Single tier structure

If we have the code dealing with presentation logic (production of HTML documents), business logic (the making of rules) and data access logic (the production and completing of DML statements) into a single module then it is a single tier structure. All processing is done in single computer and all the resources are attached to a single computer. This structure puts a high load on the single computer.

Two tier structure

In the above mentioned architecture, if we divide the code that deals with the communication with physical database to separate component, it is called a 2 tier architecture.

Three tier structure

If we divide further the presentation logic and business logic into separate components then it becomes a 3 tier architecture. In this structure there is no direct communication between the presentation and data access layers. Everything must pass through the business layer which is in the middle. Many components in the presentation layer share the same parts in the business layer and all parts in the business layer share the same parts in the data access layer.

Many of today’s applications can be divided into three distinct areas:

i. The Presentation Logic: the user interface that displays data to the user or accepts input from the user.

ii. The Business Logic: validates and processes the data, ensuring that it is consistent and in accordance with the requirements and the specifications before being added to the database.
iii. The Data Access Logic: communicates with the database and provides access to the tables and indices. It also packs and unpacks the data.

Advantages of 3 tier architecture

- The advantage of a 3-tier system is that the contents of any of the tiers/layers can be replaced without making any resultant changes in any of the others. For example:
  - A change from one DBMS to other will only involve a change to the part in the data access layer.
  - A change in the Use Interface (from desktop to the web), will need only some changes in the components of the presentation layer.
  - The benefit of writing the presentation and business layered architecture in different languages is that it is feasible to use different developer teams to work on each. It means that only PHP skills are required for data access layer and business layer, and HTML, CSS and XLS skills for presentation layers. It is easier to find a developer with skills in one of these rather having a kind of developer having all of the skills.
  - Another main advantage of using XML/XLS in the presentation layer is that the output can be changed from HTML to WML or PDF or any other format using a different XLS style sheet. XLS files can be used to change XML documents into a number of formats and not only HTML.
  - The entire work load is divided.
  - Security polices can be imposed without effecting the clients.

The user interface in those applications is usually an HTML, XML or XHTML file, either dynamically generated for each case, or a saved, static one. Furthermore, the front-tier, user interface can also contain client-side scripts and sometimes Java applets.

But when portability and interoperability is the main concern, the XHTML is the preferred mechanism for the user’s representation. Almost all the browsers support XHTML; so designing the user interface to be accessible through a Web browser guarantees a portability and interoperability across most platforms. The user interface communicates with the middle-tier business logic by using the networking features that the browser provides automatically.
When the middle-tier receives a request from the user interface, it processes the request according to the business logic and then accesses the database to manipulate any data required. Generally, in today's multi-tier architectures, Web servers are increasingly used to build the middle tier. They can efficiently provide the business logic that manipulates data from the databases and that also communicates with the client Web browsers. This request-response model of communication between the client-browser and the server is accomplished by using a specific Java programming called a servlet. A servlet extends the functionality of a server and is based on the `javax.servlet` and `javax.servlet.http` packages that provide the necessary classes and interfaces.

The middle-tier servlet interacts with most of the third-tier database systems through the Java Database Connectivity (JDBC), which provides all the means required for communication. Developers need not be familiar with the specifics of each database system. They use common SQL-based queries and the JDBC driver handles the specifics of interacting with each database system. Moreover, other technologies like ODBC (Object Database Connectivity), developed by Microsoft, also provide generic access to disparate database systems on the Windows platform (and some UNIX platforms). In those cases, Java enables the JDBC - to-ODBC driver to allow any Java program to access any ODBC data source[65].

Organizations that divide their applications according to the above three distinct and separate areas can gain many advantages. For example, component roles are specialized, improving maintainability; networking and I/O overheads. They are also clearly defined within a 3-tier framework, which provides a good basis for component-based development and reusability. Components in the business layer can be shared by any number of components in the presentation layer. Furthermore, using a 3-tier architecture enhances infrastructure independence. This is because presentation and data access areas that are often infrastructure-dependent are separated from the application's business logic. Finally, a specific set of skills are required to develop each tier, so tiers can be developed independently of one another. For example, the thin presentation tier allows front-end experts to do their work without being affected by developments occurring in the business logic tier.