Extensible Markup Language (XML)

- XML is a markup language that was defined by the World Wide Web Consortium (W3C) for general use on the Web. In general, the term *markup language* refers to a textual encoding that represents both a text and details as to its structure or its appearance. Both XML and HTML were derived from SGML (Standardized Generalized Markup Language) [ISO 8879], a very complex markup language. HTML (see Section 1.6) was designed for defining the appearance of web pages. XML was designed for writing structured documents for the Web.
- XML data items are tagged with ‘markup’ strings. The tags are used to describe the logical structure of the data and to associate attribute-value pairs with logical structures. That is, in XML, the tags relate to the structure of the text that they enclose, in contrast to HTML, in which the tags specify how a browser could display the text.

Extensible Markup Language (XML)

- XML is used to enable clients to communicate with web services and for defining the interfaces and other properties of web services. However, XML is also used in many other ways, including in archiving and retrieval systems – although an XML archive may be larger than a binary one, it has the advantage of being readable on any computer. Other examples of uses of XML include for the specification of user interfaces and the encoding of configuration files in operating systems.
- XML is *extensible* in the sense that users can define their own tags, in contrast to HTML, which uses a fixed set of tags. However, if an XML document is intended to be used by more than one application, then the names of the tags must be agreed between them. For example, clients usually use SOAP messages to communicate with web services. SOAP is an XML format whose tags are published for use by web services and their clients.

Extensible Markup Language (XML)

- Some external data representations (such as CORBA CDR) do not need to be self describing, because it is assumed that the client and server exchanging a message have prior knowledge of the order and the types of the information it contains. However, XML was intended to be used by multiple applications for different purposes. The provision of tags, together with the use of namespaces to define the meaning of the tags, has made this possible. In addition, the use of tags enables applications to select just those parts of a document it needs to process: it will not be affected by the addition of information relevant to other applications.

Extensible Markup Language (XML)

- XML documents, being textual, can be read by humans. In practice, most XML documents are generated and read by XML processing software, but the ability to read XML can be useful when things go wrong. In addition, the use of text makes XML independent of any particular platform. The use of a textual rather than a binary representation, together with the use of tags, makes the messages large, so they require longer processing and transmission times, as well as more space to store.
- However, files and messages can be compressed – HTTP version 1.1 allows data to be compressed, which saves bandwidth during transmission.

XML elements and attributes

- Figure shows the XML definition of the *Person* structure that was used to illustrate marshalling in CORBA CDR and Java.

XML definition of the *Person* structure

```
<person id="123456789">
        <name>Smith</name>
        <place>London</place>
        <year>1984</year>
        <!-- a comment -->
</person >
```

- It shows that XML consists of tags and character data. The character data, for example *Smith* or *1984*, is the actual data. As in HTML, the structure of an XML document is defined by pairs of tags enclosed in angle brackets.
- In figure, <*name*> and <*place*> are both tags. As in HTML, layout can generally be used to improve readability. Comments in XML are denoted in the same way as those in HTML.

Elements

- An element in XML consists of a portion of character data surrounded by matching start and end tags. For example, one of the elements in previous figure consists of the data *Smith* contained within the <*name*> ... </*name*> tag pair. Note that the element with the <*name*> tag is enclosed in the element with the <*person id="123456789"*> ...</*person* > tag pair. The ability of an element to enclose another element allows hierarchic data to be represented – a very important aspect of XML. An empty tag has no content and is terminated with /> instead of >. For example, the empty tag <*european*/> could be included within the <*person*> ...</*person*> tag.

Attributes

- A start tag may optionally include pairs of associated attribute names and values such as *id="123456789",* as shown above. The syntax is the same as for HTML, in which an attribute name is followed by an equal sign and an attribute value in quotes. Multiple attribute values are separated by spaces.
- It is a matter of choice as to which items are represented as elements and which ones as attributes. An element is generally a container for data, whereas an attribute is used for labelling that data. In our example, *123456789* might be an identifier used by the application, whereas *name*, *place* and *year* might be displayed. Also, if data contains substructures or several lines, it must be defined as an element. Attributes are for simple values.

Names:

- The names of tags and attributes in XML generally start with a letter, but can also start with an underline or a colon. The names continue with letters, digits, hyphens, underscores, colons or full stops. Letters are case-sensitive. Names that start with *xml* are reserved.

Binary data:

- All of the information in XML elements must be expressed as character data. But the question is: how do we represent encrypted elements or secure hashes The answer is that they can be represented in *base64* notation [Freed and Borenstein 1996], which uses only the alphanumeric characters together with +, / and =, which has a special meaning.

Parsing and well-formed documents

- An XML document must be well formed – that is, it must conform to rules about its structure. A basic rule is that every start tag has a matching end tag. Another basic rule is that all tags are correctly nested – for example, <*x*>..<*y*>..</*y*>..</*x*> is correct, whereas <*x*>..<*y*>..</*x*>..</*y*> is not. Finally, every XML document must have a single root element that encloses all the other elements. These rules make it very simple to implement parsers for XML documents. When a parser reads an XML document that is not well formed, it will report a fatal error.

Remote object references

- When a client invokes a method in a remote object, an invocation message is sent to the server process that hosts the remote object. This message needs to specify which particular object is to have its method invoked. A *remote object reference* is an identifier for a remote object that is valid throughout a distributed system. A remote object reference is passed in the invocation message to specify which object is to be invoked. Remote object references are also passed as arguments and returned as results of remote method invocations, that each remote object has a single remote object reference and that remote object references can be compared to see whether they refer to the same remote object. Here, we discuss the external representation of remote object references.

Remote object references (cont.)

- Remote object references must be generated in a manner that ensures uniqueness over space and time. In general, there may be many processes hosting remote objects, so remote object references must be unique among all of the processes in the various computers in a distributed system. Even after the remote object associated with a given remote object reference is deleted, it is important that the remote object reference is not reused, because its potential invokers may retain obsolete remote object references. Any attempt to invoke a deleted object should produce an error rather than allow access to a different object.
- There are several ways to ensure that a remote object reference is unique. One way is to construct a remote object reference by concatenating the Internet address of its host computer and the port number of the process that created it with the time of its creation and a local object number. The local object number is incremented each time an object is created in that process.

Remote object references (cont.)

- The port number and time together produce a unique process identifier on that computer. With this approach, remote object references might be represented with a format such as that shown in Figure.

Representation of a remote object reference

32 bits	*32 bits*	*32 bits*	*32 bits*	
Internet address	port number	time	object number	interface of remote object

- In the simplest implementations of RMI, remote objects live only in the process that created them and survive only as long as that process continues to run. In such cases, the remote object reference can be used as the address of the remote object. In other words, invocation messages are sent to the Internet address in the remote reference and to the process on that computer using the given port number. To allow remote objects to be relocated into a different process on a different computer, the remote object reference should not be used as the address of the remote object.

Multicast communication

- The pairwise exchange of messages is not the best model for communication from one process to a group of other processes, which may be necessary, for example, when a service is implemented as a number of different processes in different computers, perhaps to provide fault tolerance or to enhance availability.
- A *multicast operation* is more appropriate – this is an operation that sends a single message from one process to each of the members of a group of processes, usually in such a way that the membership of the group is transparent to the sender.
- There is a range of possibilities in the desired behaviour of a multicast. The simplest multicast protocol provides no guarantees about message delivery or ordering.

Multicast communication (cont.)

- Multicast messages provide a useful infrastructure for constructing distributed systems with the following characteristics:

1. ***Fault tolerance based on replicated services*:** A replicated service consists of a group of servers. Client requests are multicast to all the members of the group, each of which performs an identical operation. Even when some of the members fail, clients can still be served.
2. ***Discovering services in spontaneous networking*:** Multicast messages can be used by servers and clients to locate available discovery services in order to register their interfaces or to look up the interfaces of other services in the distributed system.

Multicast communication (cont.)

3. ***Better performance through replicated data*:** Data are replicated to increase the performance of a service – in some cases replicas of the data are placed in users' computers. Each time the data changes, the new value is multicast to the processes managing the replicas.
4. ***Propagation of event notifications*:** Multicast to a group may be used to notify processes when something happens. For example, in Facebook, when someone changes their status, all their friends receive notifications. Similarly, publish subscribe protocols may make use of group multicast to disseminate events to subscribers

IP multicast – An implementation of multicast communication

- **IP multicast** • *IP multicast* is built on top of the Internet Protocol (IP). Note that IP packets are addressed to computers – ports belong to the TCP and UDP levels. IP multicast allows the sender to transmit a single IP packet to a set of computers that form a multicast group. The sender is unaware of the identities of the individual recipients and of the size of the group. A *multicast group* is specified by a Class D Internet address- – that is, an address whose first 4 bits are 1110 in IPv4.
- Being a member of a multicast group allows a computer to receive IP packets sent to the group. The membership of multicast groups is dynamic, allowing computers to join or leave at any time and to join an arbitrary number of groups. It is possible to send datagrams to a multicast group without being a member.

IP multicast (cont.)

- At the application programming level, IP multicast is available only via UDP. An application program performs multicasts by sending UDP datagrams with multicast addresses and ordinary port numbers. It can join a multicast group by making its socket join the group, enabling it to receive messages to the group. At the IP level, a computer belongs to a multicast group when one or more of its processes has sockets that belong to that group. When a multicast message arrives at a computer, copies are forwarded to all of the local sockets that have joined the specified multicast address and are bound to the specified port number. The following details are specific to IPv4:

IP multicast (cont.)

The following details are specific to IPv4:

- ***Multicast routers*:**
  - IP packets can be multicast both on a local network and on the wider Internet. Local multicasts use the multicast capability of the local network, for example, of an Ethernet. Internet multicasts make use of multicast routers, which forward single datagrams to routers on other networks, where they are again multicast to local members.
  - To limit the distance of propagation of a multicast datagram, the sender can specify the number of routers it is allowed to pass – called the *time to live*, or TTL for short.

IP multicast (cont.)

- ***Multicast address allocation*:** Class D addresses (that is, addresses in the range 224.0.0.0 to 239.255.255.255) are reserved for multicast traffic and managed globally by the Internet Assigned Numbers Authority (IANA). The management of this address space is reviewed annually, with current practice documented in RPC 3171 [Albanna *et al.* 2001]. This document defines a partitioning of this address space into a number of blocks, including:
- Local Network Control Block (224.0.0.0 to 224.0.0.225), for multicast traffic within a given local network.
- Internet Control Block (224.0.1.0 to 224.0.1.225).
- Ad Hoc Control Block (224.0.2.0 to 224.0.255.0), for traffic that does not fit any other block.
- Administratively Scoped Block (239.0.0.0 to 239.255.255.255), which is used to implement a scoping mechanism for multicast traffic (to constrain propagation).

IP multicast (cont.)

- Multicast addresses may be permanent or temporary. Permanent groups exist even when there are no members – their addresses are assigned by IANA and span the various blocks mentioned above. For example, 224.0.1.1 in the Internet block is reserved for the Network Time Protocol (NTP), and the range 224.0.6.000 to 224.0.6.127 in the ad hoc block is reserved for the ISIS project. Addresses are reserved for a variety of purposes, from specific Internet protocols to given organizations that make heavy use of multicast traffic, including multimedia broadcasters and financial institutions.
- The remainder of the multicast addresses are available for use by temporary groups, which must be created before use and cease to exist when all the members have left. When a temporary group is created, it requires a free multicast address to avoid accidental participation in an existing group. The IP multicast protocol does not directly address this issue. If used locally, relatively simple solutions are possible – for example setting the TTL to a small value, making collisions with other groups unlikely.
- A client-server solution is adopted whereby clients request a multicast address from a multicast address allocation server (MAAS), which must then communicate across domains to ensure allocations are unique for the given lifetime and scope.

Failure model for multicast datagrams

- Datagrams multicast over IP multicast have the same failure characteristics as UDP datagrams – that is, they suffer from omission failures. The effect on a multicast is that messages are not guaranteed to be delivered to any particular group member in the face of even a single omission failure. That is, some but not all of the members of the group may receive it. This can be called *unreliable* multicast, because it does not guarantee that a message will be delivered to any member of a group.

Java API to IP multicast

- The Java API provides a datagram interface to IP multicast through the class *MulticastSocket*, which is a subclass of *DatagramSocket* with the additional capability of being able to join multicast groups. The class *MulticastSocket* provides two alternative constructors, allowing sockets to be created to use either a specified local port (6789, in next Figure), or any free local port.
- A process can join a multicast group with a given multicast address by invoking the *joinGroup* method of its multicast socket. Effectively, the socket joins a multicast group at a given port and it will receive datagrams sent by processes on other computers to that group at that port. A process can leave a specified group by invoking the *leaveGroup* method of its multicast socket.

Multicast peer joins a group and sends and receives datagrams

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
    public static void main(String args[]){
      // args give message contents & destination multicast group (e.g. "228.5.6.7")
       MulticastSocket s =null;
       try {
          InetAddress group = InetAddress.getByName(args[1]);
          s = new MulticastSocket(6789);
          s.joinGroup(group);
          byte [] m = args[0].getBytes();
          DatagramPacket messageOut =
                new DatagramPacket(m, m.length, group, 6789);
          s.send(messageOut);
          byte[] buffer = new byte[1000];
          for(int i=0; i< 3; i++) {    // get messages from others in group
                DatagramPacket messageIn =
                      new DatagramPacket(buffer, buffer.length);
                s.receive(messageIn);
                System.out.println("Received:" + new String(messageIn.getData()));
          }
          s.leaveGroup(group);
       } catch (SocketException e){System.out.println("Socket: " + e.getMessage());
       } catch (IOException e){System.out.println("IO: " + e.getMessage());
       } finally { if(s != null) s.close();}
    }
}
```

Java API to IP multicast (cont.)

- In the example, the arguments to the *main* method specify a message to be multicast and the multicast address of a group (for example, *"228.5.6.7").* After joining that multicast group, the process makes an instance of *DatagramPacket* containing the message and sends it through its multicast socket to the multicast group address at port 6789. After that, it attempts to receive three multicast messages from its peers via its socket, which also belongs to the group on the same port. When several instances of this program are run simultaneously on different computers, all of them join the same group, and each of them should receive its own message and the messages from those that joined after it.

Java API to IP multicast (cont.)

- The Java API allows the TTL to be set for a multicast socket by means of the *setTimeToLive* method. The default is 1, allowing the multicast to propagate only on the local network.
- An application implemented over IP multicast may use more than one port. For example, the MultiTalk application, which allows groups of users to hold text based conversations, has one port for sending and receiving data and another for exchanging control data.

Reliability and ordering of multicast

- A datagram sent from one multicast router to another may be lost, thus preventing all recipients beyond that router from receiving the message. Also, when a multicast on a local area network uses the multicasting capabilities of the network to allow a single datagram to arrive at multiple recipients, any one of those recipients may drop the message because its buffer is full.
- Another factor is that any process may fail. If a multicast router fails, the group members beyond that router will not receive the multicast message, although local members may do so.
- Ordering is another issue. IP packets sent over an internetwork do not necessarily arrive in the order in which they were sent, with the possible effect that some group members receive datagrams from a single sender in a different order from other group members. In addition, messages sent by two different processes will not necessarily arrive in the same order at all the members of the group.

Some examples of the effects of reliability and ordering

***1. Fault tolerance based on replicated services*:** Consider a replicated service that consists of the members of a group of servers that start in the same initial state and always perform the same operations in the same order, so as to remain consistent with one another.

- This application of multicast requires that either all of the replicas or none of them should receive each request to perform an operation – if one of them misses a request, it will become inconsistent with the others. In most cases, this service would require that all members receive request messages in the same order as one another.

Some examples of the effects of reliability and ordering (cont.)

***2. Discovering services in spontaneous networking*:** One way for a process to discover services in spontaneous networking is to multicast requests at periodic intervals, and for the available services to listen for those multicasts and respond. An occasional lost request is not an issue when discovering services. In fact, Jini uses IP multicast in its protocol for discovering services.

***3. Better performance through replicated data*:** Consider the case where the replicated data itself, rather than operations on the data, are distributed by means of multicast messages. The effect of lost messages and inconsistent ordering would depend on the method of replication and the importance of all replicas being totally up-to-date.

***4. Propagation of event notifications*:** The particular application determines the qualities required of multicast. For example, the Jini lookup services use IP multicast to announce their existence.

Some examples of the effects of reliability and ordering (cont.)

- These examples suggest that some applications require a multicast protocol that is more reliable than IP multicast. In particular, there is a need for *reliable multicast*, in which any message transmitted is either received by all members of a group or by none of them.
- The examples also suggest that some applications have strong requirements for ordering, the strictest of which is called *totally ordered multicast*, in which all of the messages transmitted to a group reach all of the members in the same order.