Algorithms and Programming IV
Interprocess Communication (cont.)

Summer Term 2021 | 14.06.2021
Barry Linnert
Recap: Architectural Model

Architectural elements

Communicating entities
- Processes
- Objects
- Components
- Web Services

Communication paradigm

Inter-process communication
- UDP sockets
- TCP sockets
- Multi-cast

Indirect communication
- Remote invocation

Roles and responsibilities

Architectural styles
- Client-server
- Peer-to-peer

Placement
- Multiple server
- Proxy/Cache
- Mobile code

Architectural patterns
Our topics today

Interprocess Communication
• External Data Representation and Marshalling
• Multicast Communication

Remote Invocation
• Request-and-Reply Protocols
• Remote Procedure Call
Interprocess Communication

EXTERNAL DATA REPRESENTATION
AND MARSHALLING
What is the Challenge?

Messages consist of sequences of bytes.

Interoperability problems
- Big-endian, little-endian byte ordering
- Floating point representation
- Character encodings (ASCII, UTF-8, Unicode, EBCDIC)

So, we must either:
- Have both sides agree on an external representation or
- Transmit in the sender’s format along with an indication of the format used. The receiver converts to its form.
External Data Representation and Marshalling

**External data representation**
- An agreed standard for the representation of data structures and primitive values

**Marshalling**
- The process of taking a collection of data items and assembling them into a form suitable for transmission in a message

**Unmarshalling**
- Is the process of disassembling them on arrival into an equivalent representation at the destination

http://www.breti.org/tech/files/b400feb80f01f69e5cafca5160be5d65-67.html
Approaches for External Data Representation

Java’s object serialization

XML (Extensible Markup Language)

Protocol buffer (protobuf)

JSON (JavaScript Object Notation)
Java Object Serialization

public class Person implements Serializable {
    private String name;
    private String place;
    private int year;
    public Person(String aName, String aPlace, int aYear) {
        name = aName;
        place = aPlace;
        year = aYear;
    }
    // followed by methods for accessing the instance variables
}
Example: Java Object Serialization

```java
Person p = new Person("Smith", "London", 1984);
```

Serialized values

<table>
<thead>
<tr>
<th>Person</th>
<th>8-byte version number</th>
<th>h0</th>
<th>h1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>int year</td>
<td>java.lang.String name:</td>
<td>java.lang.String place:</td>
</tr>
<tr>
<td>1984</td>
<td>5 Smith</td>
<td>6 London</td>
<td></td>
</tr>
</tbody>
</table>

The true serialized form contains additional type markers; h0 and h1 are handles.
Extensible Markup Language (XML)

- XML is a markup language that was defined by the World Wide Web Consortium (W3C) for general use 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. For a specification of XML, see the pages on XML provided by W3C [www.w3.org VI].

- XML is used to enable clients to communicate with web services and for defining the interfaces and other properties of web services.
Example: XML definition with namespace

<person pers:id="123456789" xmlns:pers = "http://www.nonsense.net/person">
  <pers:name> Smith </pers:name>
  <pers:place> London </pers:place>
  <pers:year> 1984 </pers:year>
  <!-- a comment -->
</person>
Example: XML schema

```
<xsd:schema xmlns:xsd = URL of XML schema definitions  >

<xsd:element name= "person" type ="personType" /

<xsd:complexType name="personType">

<xsd:sequence>

<xsd:element name = "name"  type="xs:string"/>
<xsd:element name = "place"  type="xs:string"/>
<xsd:element name = "year"  type="xs:positiveInteger"/>

</xsd:sequence>

<xsd:attribute name= "id"   type = "xs:positiveInteger"/>

</xsd:complexType>

</xsd:schema>
```
Google Protocol Buffer

• Google Protocol Buffer (protobuf) is a common serialization format for storing and interchanging all kinds of structured information. It serves as a basis for a remote procedure call (RPC) system that is used for nearly all inter-machine communication at Google.

• The goal of Protocol Buffer is to provide a language- and platform-neutral way to specify and serialize data, it has been released as open source.

• Protobuf is 3-10 times smaller than an XML and 10-100 times faster than an XML.

http://code.google.com/apis/protocolbuffers
JSON (JavaScript Object Notation)

- JavaScript Object Notation (JSON) is a language-independent data format.

- It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data.

- Example:
  ```json
  {
    "firstName": "John",
    "lastName": "Smith",
    "birthyear": "1984",
    "address": {
      "city": "New York",
      "state": "NY"
    }
  }
  ```
## Comparison of Data-Serialization Formats

<table>
<thead>
<tr>
<th></th>
<th>Standardized</th>
<th>Binary</th>
<th>Human-Readable</th>
<th>Standard-API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>XML</td>
<td>Yes</td>
<td>Partial</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>protobuf</td>
<td>No</td>
<td>Yes</td>
<td>Partial</td>
<td>For example, C++, Java, C#, Python, Ruby, C, PHP, R</td>
</tr>
<tr>
<td>JSON</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Partial (JSON-LD)</td>
</tr>
</tbody>
</table>

Recap: Architectural Model

Architectural elements

- Communicating entities
  - Processes
  - Objects
  - Components
  - Web Services

Communication paradigm

- Inter-process communication
  - UDP sockets
  - TCP sockets
  - Multicast
- Indirect communication
  - Remote invocation

Roles and responsibilities

- Architectural styles
  - Client-server
  - Peer-to-peer

Placement

- Multiple server
- Proxy/Cache
- Mobile code

Architectural patterns
Interprocess Communication

MULTICAST COMMUNICATION
Possibilities to Communicate

• Connectionless 1:1
  UDP

• Connection-oriented 1:1
  TCP

• Connectionless 1:n
  Multicast
Multicast Communication

Efficient group communication is becoming increasingly important in modern multimedia applications such as video conferencing or joint editing of documents.

The standard solution is called multicast and provides 1-to-n communication:

• The application only needs to manage one connection per group.
• The resources in the network are used more efficiently.
Effect of using Multicast for building Distributed Systems

- Fault tolerance based on replicated services
- Discovering services in spontaneous networking
- Better performance through replicated data
- Propagation of event notifications
Multicast Sockets

1. Participants bind socket
2. Participants join group
3. Participants receive messages from sender
4. Participants leave group and release socket
IP Multicast

- Is built on top of the Internet Protocol (IP) and allow the sender to transmit a single IP packet to a set of computers that form a multicast group.

- Multicast group is specified by a Class D Internet Address. Every IP datagram whose destination address starts with "1110" (in IPv4) is an IP Multicast datagram.

- IP packets can be multicast on a local and wider network. In order to limit the distance of operation, the sender can specify the number of routers that can be passed (i.e. time to live, or TTL)

- Multicast addresses can be permanent (e.g. 224.0.1.1 is reserved for the Network Time Protocol (NTP))
Java API: java.net.MulticastSocket

```java
public class MulticastSocket extends DatagramSocket {
    public MulticastSocket()...
    public MulticastSocket(int port)...
    // create socket and select port number explicitly or implicitly

    public void joinGroup(InetAddress mcastaddr) throws ...
    // join group under the address mcastaddr
    public void leaveGroup(InetAddress mcastaddr) throws ...
    // leave group
    public void setTimeToLive(int ttl) ...
    // define Time to Live – default is 1!
}
```

Please note: send, receive, ... are inherited from class DatagramSocket
Failure Model of \textit{(unreliable)} Multicast

- 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.

- This model can be extended by a reliable multicast that considers:
  - \textit{Integrity}: the message received is the same as the one sent, and no messages are delivered twice
  - \textit{Validity}: any outgoing message is eventually delivered
  - \textit{Agreement}, stating that if the message is delivered to one process, then it is delivered to all processes in the group
Middleware Layers

Applications, services

Remote invocation, indirect communication

Underlying inter-process communication primitives:
  Sockets, multicast support

UDP and TCP