CS 580 Client-Server Programming
Spring Semester, 2010
Doc 20 UDP & Distributed Computing
April 20, 2010

References

Java Networking Programming, 3rd Ed., Harold, O'Reilly, 2005, Chapters 13

JSON - JavaScript Object Notation,
    Main JSON web site, http://www.json.org/
    JSON in Java, http://www.json.org/java/index.html

Java's Object Streams
    Java 6 API documentation, http://java.sun.com/javase/6/docs/api/
    Object Stream Spec, http://java.sun.com/javase/6/docs/platform/serialization/spec/protocol.html

UDP

UDP Client

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class UDPClient {
    public static void main(String[] args) throws IOException {
        InetAddress serverAddress = InetAddress.getByName("localhost");
        DatagramSocket socket = new DatagramSocket();
        socket.connect(serverAddress, 8888);

        byte[] message = "Hello World".getBytes();
        DatagramPacket output = new DatagramPacket(message, message.length);
        socket.send(output);
        byte[] buffer = new byte[65507];
        DatagramPacket response = new DatagramPacket(buffer, buffer.length);
        socket.receive(response);
        String answer = new String(response.getData(), 0, response.getLength());
        System.out.println("cleint" + answer);
    }
}
```

UDPEchoServer

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;

public class UDPEchoServer extends Thread {

    private int bufferSize = 8192;
    protected DatagramSocket socket;

    public UDPEchoServer(int port) throws SocketException {
        this.socket = new DatagramSocket(port);
    }
```

UDPEchoServer

```
public void run() {
    byte[] buffer = new byte[bufferSize];
    while (true) {
        DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
        try {
            socket.receive(incoming);
            byte[] data = incoming.getData();
            System.out.println("Server " + new String(data));
            DatagramPacket outgoing = new DatagramPacket(
                    incoming.getData(), incoming.getLength(), incoming
                            .getAddress(), incoming.getPort());
            socket.send(outgoing);
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

Packet Size

IPv4 protocol supports UDP packets up to 65,507 bytes

In practice 8,192 bytes is limit

Larger packets may be

- trucated
- split
- dropped

Connectionless

```
DatagramSocket socket = new DatagramSocket();
socket.connect(serverAddress, 8888);

byte[] message = "Hello World".getBytes();
DatagramPacket output = new DatagramPacket(message, message.length);
socket.send(output);
```

connect method restricts socket to sending only to one machine and port. However after using the disconnect method the socket can be used to connect to any machine or port.

Connectionless

```
DatagramSocket socket = new DatagramSocket();

byte[] message = "Hello World".getBytes();
DatagramPacket output =
     new DatagramPacket(message, message.length, serverAddress, 8888);
socket.send(output);
```

One datagram socket can send to multiple addresses

Distributed Computing

Related Terms

Concurrent Computing

Simultaneous execution of multiple interacting computational tasks

Networking

Multiple computers interacting via network

Do not share single program

Distributed Computing

Different parts of a program run on multiple computers

Parts communicate via network

Parallel computing

Different parts of a program run on multiple processors in same computer

Some Motivation

Basic Communication Steps

Design protocol

Create domain objects

Extract protocol string from select domain objects

Convert protocol string to domain object

Vote Example

```
public class Vote {
    static String CR = "\r";
    int id;
    String option;

    public Vote(int pollId, String optionVote) {
        id = pollId;
        option = optionVote;
    }

    public int id() {return id;}

    public String option() {return option;}
```

```
public String toString() {
     StringBuffer protocol = new StringBuffer();
     protocol.append("command:VOTE");
     protocol.append(CR);
     protocol.append("poll-id:");
     protocol.append(id);
     protocol.append(CR);
     protocol.append("option:");
     protocol.append(option);
     protocol.append(CR);
     protocol.append(CR);
     return protocol.toString();
 }
```

Vote Example - Converting

```
 public static Vote fromString(String voteString) {
      String[] lines = voteString.split(CR);
      HashMap<String, String> data = new HashMap<String, String>();
      for (int k = 0; k < lines.length;k++) {
         String[] keyValue = lines[k].split(":");
         data.put(keyValue[0].toLowerCase(), keyValue[1]);
      }
      return fromMap(data);
   }

  public static Vote fromMap(Map<String,String> voteData ){
      String option = voteData.get("option");
      Integer id = Integer.valueOf(voteData.get("poll-id"));
      return new Vote(id.intValue(), option);
   }
}
```

Repeat

Repeat for each Command

Repeat for each client-server project

Some Ways to Automate the Work

JSON
ObjectStreams

JSON

http://www.json.org/

JavaScript Object Notation

data-interchange format

rfc 4627

Maps to/from strings

- null
- true, false
- number
- string
- array
- objects

Implementations in

- C, C++, C#, D, E, Java, Objective C
- Cold Fusion, Delphi, Erlang, Haskell
- JavaScript, Lisp, LotusScript, Perl,
- PHP, Pike, Prolog, Python, Ruby, Smalltalk

Examples

Java Structure

JSON Representation

```
Vector array = new Vector();
array.append(new Integer(12));
array.append("Egypt");
array.append(new Boolean(false));
array.append(new Integer(-31));
```

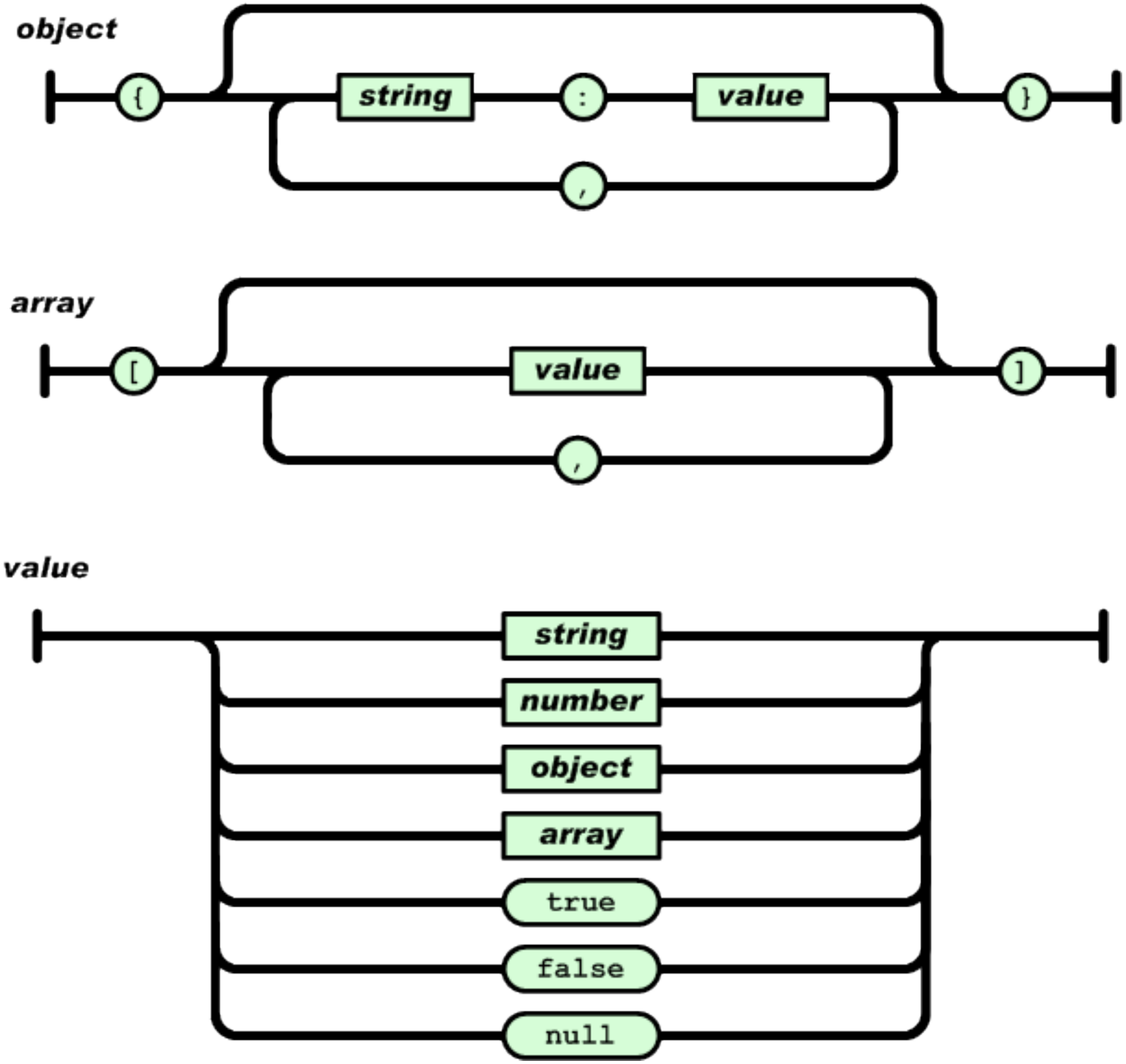
```
[12,"Egypt",false,-31]
```

```
HashMap<String,Integer> object = new HashMap<String,Integer>();
object.put("lowerBound", 18);
object.put("upperBound", 139);
```

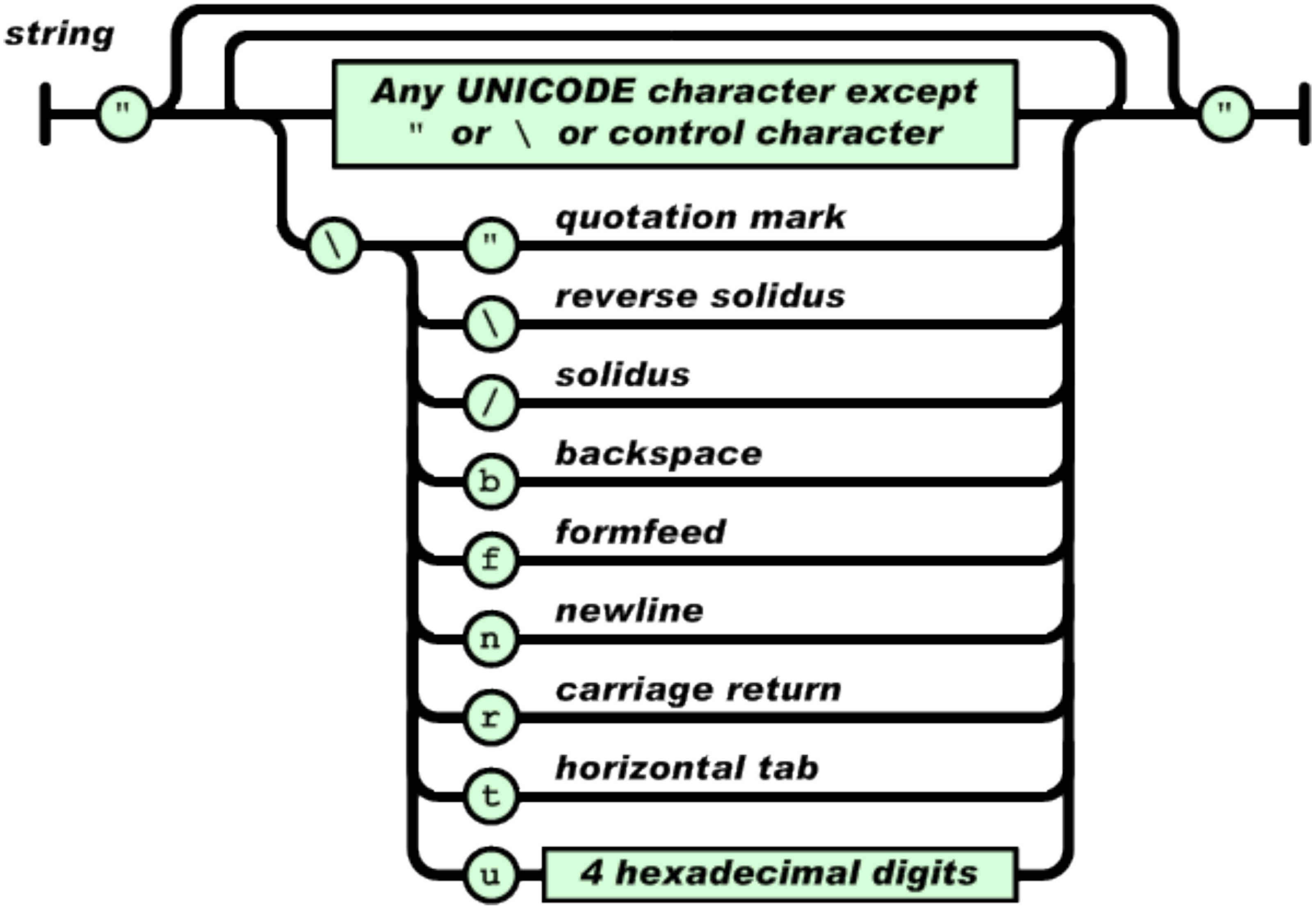
```
{"lowerBound":18,"upperBound":139}
```

JSON Definition

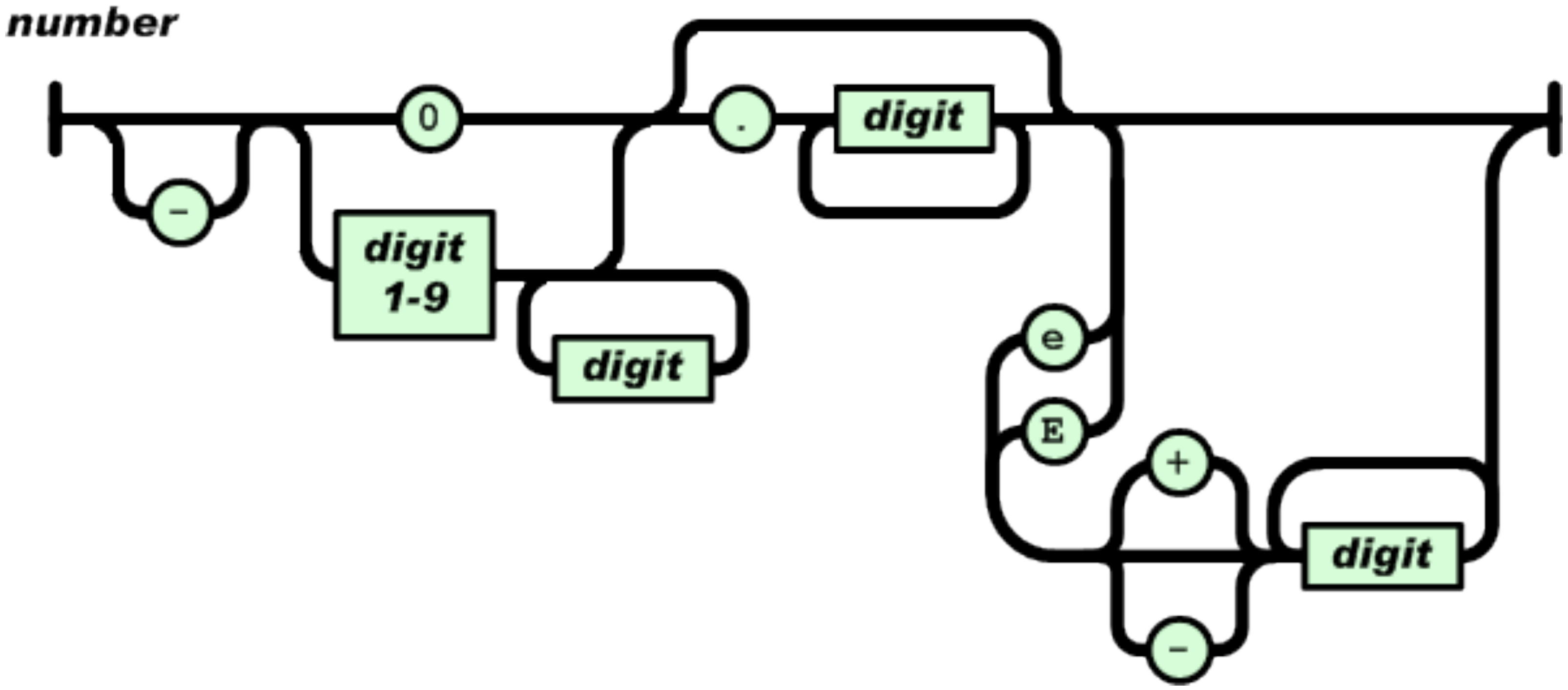
Source: http://www.json.org/



String


string
"
Any UNICODE character except
" or \ or control character
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits
"

Number



Possible Client-Server Usage

Use JSON as protocol syntax

Use JSON libraries to

- Generate client-server messages
- Parse messages from network

JSON in Java

http://www.json.org/java/index.html

A Java JSON library

JSONObject

- Constructs JSON strings
- Parses JSON strings

```
JSONObject json = new JSONObject();
json.put("lowerBound", 18);
json.put("upperBound", 139);

String objectString = json.toString();        //"{"lowerBound":18,"upperBound":139}"

JSONObject newJson = new JSONObject(objectString);
int bound = newJson.getInt("lowerBound");        // 18
```

Vote Example

```
import org.json.JSONObject;
import org.json.JSONException;

public class Vote implements Serializable {
    int id;
    String option;

    public Vote(int pollId, String optionVote) {
        id = pollId;
        option = optionVote;
    }

    public int id() {return id;}
    public String option() {return option;}

    public String toJson() throws JSONException {
        JSONObject json = new JSONObject();
        json.put("command", "VOTE");
        json.put("poll-id", id);
        json.put("option", option);
        return json.toString();
    }
```

```
public static Vote fromJson(String jsonString)
                      throws JSONException {
    JSONObject json = new JSONObject(jsonString);
    int id = json.getInt("poll-id");
    String option = json.getString("option");
    return new Vote(id, option);
}
```

Using Java JSON library from
http://www.json.org/java/index.html

Using JSON Strings

```
Vote cat = new Vote(1,"cat{:}dog");
String json = cat.toJson();

System.out.println(json);  //{"command":"VOTE","option":"cat{:}dog","poll-id":1}

Vote jsonVote = Vote.fromJson(json);
assertEquals(jsonVote.id(), 1);
```

Consequences

Benefits

Parsing and generation of protocol simplified

- No need to escape special characters

Define protocol in terms of

- maps (key-value pairs)
- arrays
- basic types (string, number, boolean)

Cross language support

Drawbacks

Nested { } and [] complicate parsing

No general end of message sequence

Limited support for primitive types

Object Streams

ObjectOutputStream

- Serializes objects
- Converts objects to bytes

ObjectInputStream

- Deserializes objects
- Converts DataOutputStream byte back into objects

"Requires" writer and reader to be Java programs

Possible Client-Server Usage

Create Message and Response classes

Send message objects to server

Message object

- Contains needed data
- Possibility executes methods on server

Vote Example

```
import java.io.Serializable;

public class Vote implements Serializable, Message {
    int id;
    String option;

    public Vote(int pollId, String optionVote) {
        id = pollId;
        option = optionVote;
    }

    public int id() {return id;}

    public String option() {return option;}

        public Response execute(VoteServer aServer ) {
                boolean succeeded  = aServer.addVote(id, option);
                if (succeeded)
                        return new SuccessResponse();
                return new FailedResponse();
}
```

Using ObjectStreams

Writing the Object

```
Vote cat = new Vote(1,"cat");
FileOutputStream catBytes = new FileOutputStream("cat");
ObjectOutputStream out = new ObjectOutputStream(catBytes);
out.writeObject(cat);
out.close();
```

Reading the Object

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("cat"));
Vote result = (Vote)in.readObject();
```

Output File

```
¨ÌsrVoteªl\€ä…áIidLoptiontLjava/lang/String;xptcat
```

Sample Client Usage

```
Socket connection = new Socket(server, port);
OutputStream rawOut = connection.getOutputStream();
ObjectOutputStream out = new ObjectOutputStream(rawOut);
InputStream rawIn = connection.getInputStream();
ObjectInputStream in = new ObjectInputStream(rawIn);

Vote forCat = new Vote(1,"cat");
out.writeObject(forCat);
out.flush();
Response answer =(Response) in.readObject();

out.close();
in.close();
```

Sample Server

```
ServerSocket input = new ServerSocket( port );
while (true) {
  Socket client = input.accept();

     OutputStream rawOut = client.getOutputStream();
     ObjectOutputStream out = new ObjectOutputStream(rawOut);
     InputStream rawIn = client.getInputStream();
     ObjectInputStream in = new ObjectInputStream(rawIn);

     Message request = (Message) in.readObject();
     Response answer = request.execute(this);
     out.writeObject(answer);
     out.flush();
  client.close();
 }
```

Consequences

Benefits

No need for a text protocol - just send objects

Protocol is just objects one can send

No need for parsing - just read objects

Drawbacks

Client and server need to be in Java

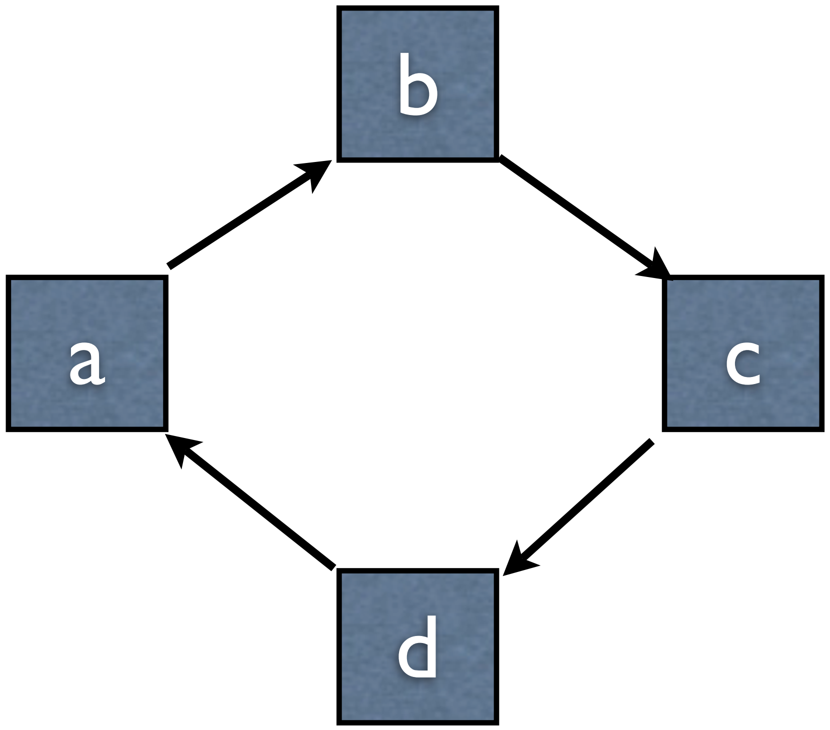
Client & server must have same classes

Modifications of a class can cause problems

Circular References

Care is needed when serializing/deserializing objects with circular references

ObjectStream does it correctly



Object Stream Protocol

The protocol used by Object Stream is documented at:

http://java.sun.com/javase/6/docs/platform/serialization/spec/protocol.html

At least one Lisp implementation of this protocol exists

RPC - Remote Procedure Call

Client "directly" calls a function on the server

Issues

Cross platform

Marshalling/unmarshalling of parameters and results

How can one handle pointers as parameters?

Different contexts of client and server

Registering and finding servers

XML-RPC

RPC using

- HTTP as transport layer and
- XML to encode request/response
- Language and platform independent

Started by Userland (http://frontier.userland.com/) in 1998

Languages/Systems with XML-RPC implementations

- Java, Perl, Python, Tcl, C, C++, Smalltalk
- ASP, PHP, AppleScript, COM
- Zope, WebCrossing

Led to the development of SOAP

Example - Add Server

Client can access all public instance methods in AddServer

```
import org.apache.xmlrpc.*;

public class AddServer {
        public Integer addtwo(int x, int y)  {
        return new Integer( x + y);
    }

    public static void main( String[] args)  {
        try {
            System.out.println("Starting server on port 8080");
            WebServer addTwoServer = new WebServer(8080);
            addTwoServer.addHandler("examples", new AddServer());
            addTwoServer.start();
            System.out.println("server running");
       }
        catch (Exception webServerStartError) {
            System.err.println( "JavaServer " + webServerStartError.toString());
         }
    }
}
```

Example - Client

```
import java.util.*;
import org.apache.xmlrpc.*;

public class XmlRpcExample {
    public static void main (String args[])  {
    try {
      XmlRpcClient xmlrpc =  new XmlRpcClientLite("http://127.0.0.1:8080/");
            Vector parameters = new Vector ();
      parameters.addElement (new Integer(5) );
      parameters.addElement (new Integer(3) );

      Integer sum = (Integer) xmlrpc.execute("examples.addtwo", parameters);

      System.out.println( sum.intValue() );
    } catch (java.net.MalformedURLException badAddress) {
      badAddress.printStackTrace( System.out);
    } catch (java.io.IOException connectionProblem) {
      connectionProblem.printStackTrace( System.out);
    } catch (Exception serverProblem) {
      serverProblem.printStackTrace( System.out);
    }
  }
}
```

XML-RPC Datatypes

array

base64

boolean

date/time

double

integer

string

struct (hash table)

Consequences

Benefits

Protocol = public methods

Handles the network communications

Handles generation/parsing of messages

Multiple language support

Platform independent

Simple

Drawbacks

Long messages

Limited support for objects

No state

Authentication not supported

JSON-RPC

Similar to XML-RPC

Uses JSON instead of XML

Supports bi-directional requests (peer-to-peer)

Uses either
- Sockets
- HTTP requests