The API Socket (UDP)

TEchnologies Internet (TEI)

Olivier Liechti

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Foundations

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

W1

Introduction to Java IO

Byte and character streams, dealing with files, the decorator pattern, custom reader/writers classes, buffered IOs

W2

Socket API - TCP

Client and server programming, sockets and streams, multi-threaded servers

W3

Application-level Protocol

How to specify your own communication protocol? Implement the client and the server.

W4

Socket API - UDP

Client and server programming, broadcast/multicast, service discovery protocols

UDP

Application-Level Protocol with UDP

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

The protocol must allow **clients** to send **commands** to a **server**.

A command is defined by an **operation** and by **payload data** (on which the operation is to be executed). After executing an operation, the server will send a reply to the client. The reply will contain the result of the operation.

It should be possible for a client to **issue several commands in sequence**, without waiting for the first replies to arrive. This means that the protocol must provide a way to associate a reply with a prior request.

Last but not least, the protocol must support **automatic service discovery**. This means that a client should be able to automatically find servers on the local network (using multicast).

Specifications (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **The Discovery Phase**
  - The server should join the multicast group **239.255.14.46** and accept discovery datagrams on port **2446**.
  - At startup time, the client should create a discovery datagram, with the payload "**HELLO**" and send it to the above multicast address and port.
  - The client should then wait for one server to respond. The response will have the following structure: **IPADDRESS#PORT**. The client will use **IPADDRESS** and **PORT** for the request processing phase.

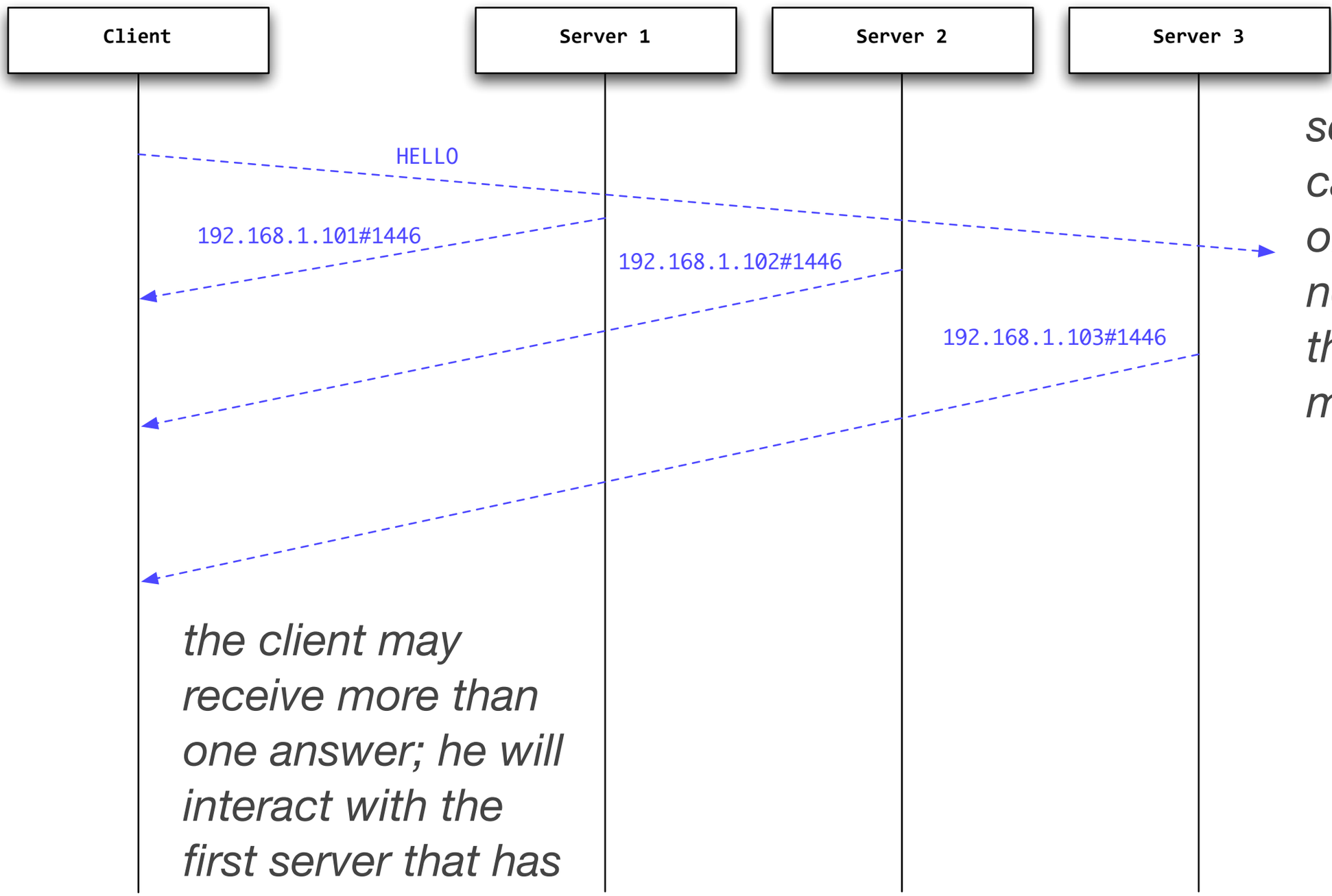
Specifications (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **The Request Processing Phase**
  - The client can send any number of requests to the server. Every command has the following structure: **COMMAND#P1=V1#P2=V2#REQUESTID**, where:
    - **COMMAND** is the command to execute,
    - **P*n*=V*n*** is an attribute-value pair (there can be several, the number depends on the command)
    - **REQUESTID** is a number that uniquely identifies the request and that will be provided in the corresponding reply.
  - The server sends replies with the following structure: **STATUSCODE#RESULT#REQUESTID**, where:
    - **STATUSCODE** indicates whether the command could be processed or not (and why)
    - **RESULT** contains the result of the specific command
    - **REQUESTID** contains the unique id of the request, at the origin of this reply

Discovery Phase

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

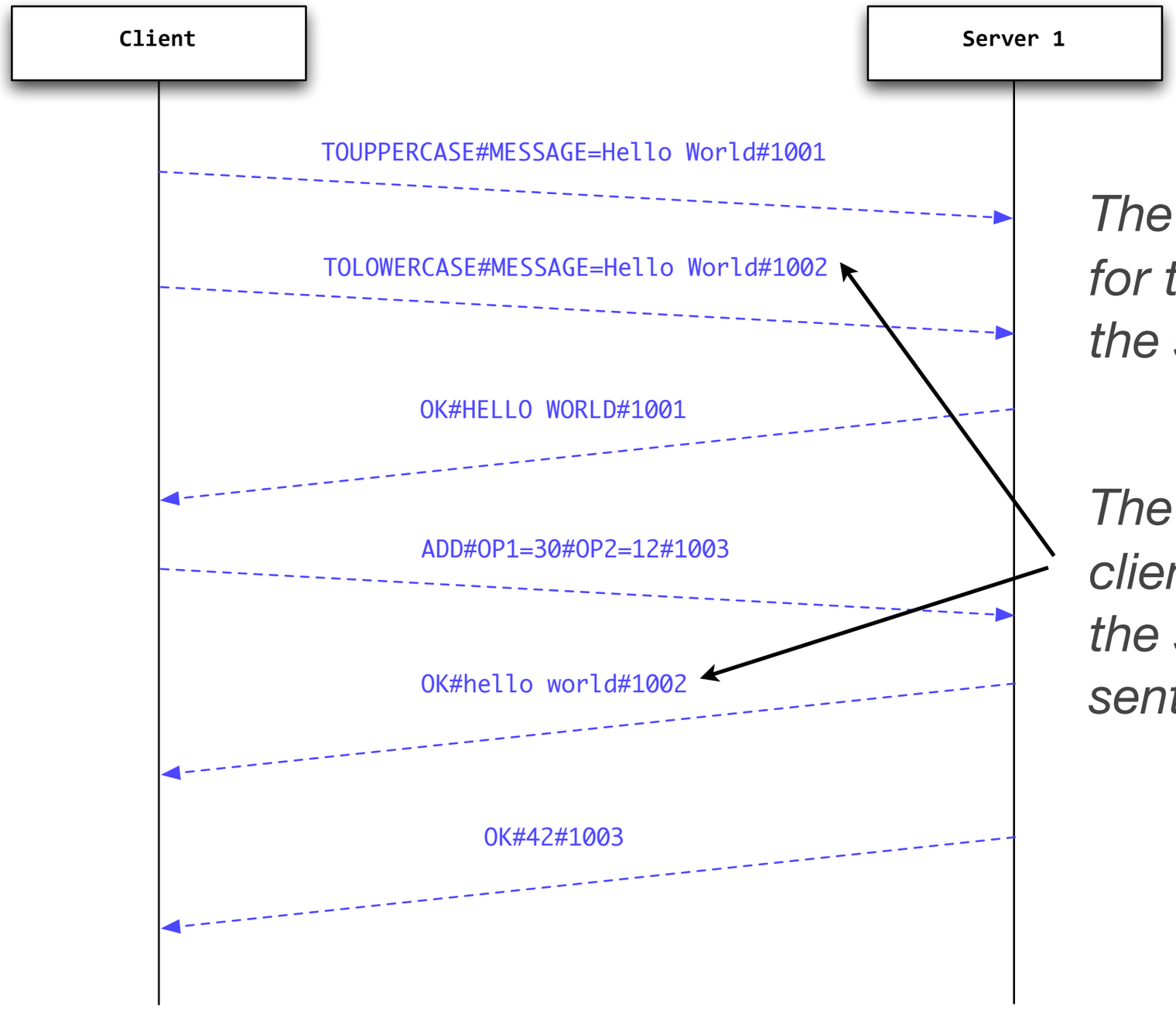


several servers can be running on the local network; all of them join the multicast group

the client may receive more than one answer; he will interact with the first server that has sent a response

Request-Reply Phase

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud



*The client **does not have to wait** for the first answer before sending the second request.*

*The **request ids** are used by the client to associate replies sent by the server to requests previously sent to the server.*

Discovery Phase - Server-side (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
public void run() {
    try {
        socket = new MulticastSocket(Protocol.DISCOVERY_PORT);
        InetAddress group = InetAddress.getByName(Protocol.DISCOVERY_MULTICAST_GROUP);
        socket.joinGroup(group);
        System.out.println("Server joined the multicast group " + group);
```

See next slide

```
        socket.leaveGroup(group);
        socket.close();
    } catch (IOException ex) {
        Logger.getLogger(DiscoveryProtocolWorker.class.getName()).log(Level.SEVERE,
null, ex);
    }
}
```

Discovery Phase - Server-side (2)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
while (shouldRun) {
    byte[] buf = new byte[Protocol.BUFFER_SIZE];
    DatagramPacket discoveryRequest = new DatagramPacket(buf, buf.length);
    socket.receive(discoveryRequest);
    System.out.println("Server received a DISCOVERY request.");
    System.out.println("-> " + new String(buf));
    String coordinates = InetAddress.getLocalHost().getHostAddress() +
      Protocol.SEPARATOR + Protocol.REQUEST_ACCEPT_PORT;
    DatagramPacket discoveryReply = new DatagramPacket(coordinates.getBytes(),
      coordinates.length());
    discoveryReply.setAddress(discoveryRequest.getAddress());
    discoveryReply.setPort(discoveryRequest.getPort());
    socket.send(discoveryReply);
    System.out.println("Server sent a DISCOVERY reply on port " +
      discoveryRequest.getPort());
    System.out.println("-> " + coordinates);
}
```

Discovery Phase - Client-side (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
            /*
             * This first thing we need to do is to send a discovery
message on the multicast group
             */

            InetAddress group = InetAddress.getByName
(Protocol.DISCOVERY_MULTICAST_GROUP);
            DatagramPacket packet;
            String message = Protocol.DISCOVERY_HELLO_MSG;
            packet = new DatagramPacket(message.getBytes(), message.length
(), group, Protocol.DISCOVERY_PORT);
            MulticastSocket socket = new MulticastSocket();
            System.out.println("Client sent discovery packet");
            System.out.println("-> " + message);
            socket.send(packet);
```

Discovery Phase - Client-side (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
            /*
             * We have sent the discovery message, now let's wait until
the (first) server comes back with his
             * coordinates.
             */

            byte[] data = new byte[Protocol.BUFFER_SIZE];
            DatagramPacket disoveryResponse = new DatagramPacket(data,
data.length);
            socket.receive(disoveryResponse);
            System.out.println("Client received discovery response packet
of " + disoveryResponse.getLength() + " bytes");
            System.out.println("data: " + new String(data));
            byte[] payload = new byte[disoveryResponse.getLength()];
            System.arraycopy(data, 0, payload, 0, payload.length);
            String[] tokens = new String(payload).split(Character.toString
(Protocol.SEPARATOR));
            String hostname = tokens[0];
            int port = Integer.parseInt(tokens[1]);
            return new ServerCoordinates(hostname, port);
```

Request-Reply Phase - Client-side (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
    public void run() {
        while (true) {
            try {
                byte[] buffer = new byte[Protocol.BUFFER_SIZE];
                DatagramPacket packet = new DatagramPacket(buffer,
buffer.length);
                socket.receive(packet);
                System.out.println("Received reply: " + packet.getLength
());
                Reply reply = new Reply();
                reply.unmarshal(new String(buffer, 0, packet.getLength
()));
                callbackListener.onReplyAvailable(reply);
            } catch (IOException ex) {
                Logger.getLogger(ReplyAcceptWorker.class.getName()).log
(Level.SEVERE, null, ex);
            }
        }
    }
```

This is the main loop, where we process server replies as they become available.

Request-Reply Phase - Client-side (2)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
    public void submitRequest(Request request) {
        try {
            byte[] data = request.marshal().getBytes();
            DatagramPacket packet = new DatagramPacket(data, data.length);
            packet.setAddress(InetAddress.getByName("localhost"));
            packet.setPort(Protocol.REQUEST_ACCEPT_PORT);
            socket.send(packet);
        } catch (IOException ex) {
            Logger.getLogger(ReplyAcceptWorker.class.getName()).log
(Level.SEVERE, null, ex);
        }
    }
```

This is the method to send a request to the server.

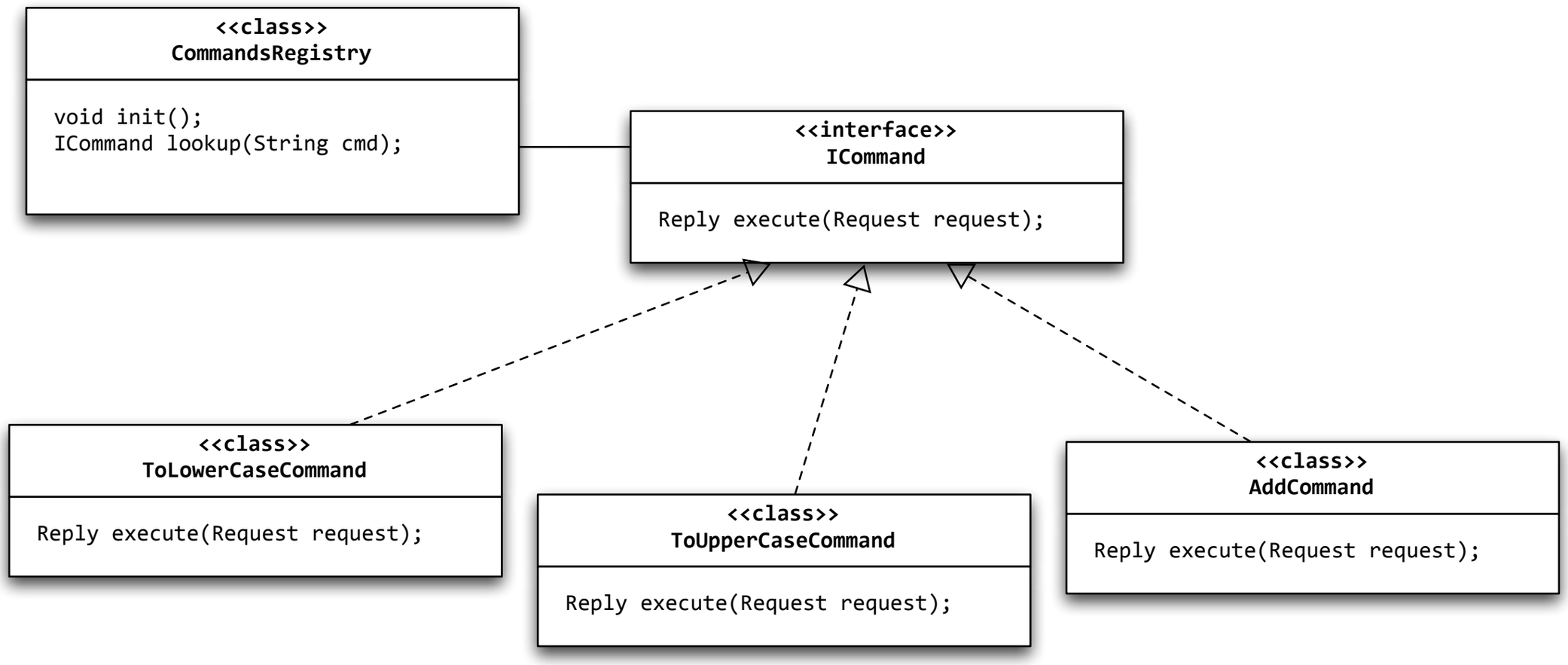
Request-Reply Phase - Server-side (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
public void run() {
     try {
          DatagramSocket socket = new DatagramSocket(Protocol.REQUEST_ACCEPT_PORT);

          while (true) {
               byte[] buffer = new byte[Protocol.BUFFER_SIZE];
               DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
               socket.receive(packet);
               System.out.println("Request packet " + packet.getLength());
               byte[] payload = new byte[packet.getLength()];
               System.arraycopy(buffer, 0, payload, 0, packet.getLength());
               Request request = new Request();
               request.unmarshal(new String(payload));
               System.out.println("Request received " + request);

               ICommand command = commandsRegistry.lookup(request.getCommand());
               Reply reply = command.execute(request);

               byte[] data = reply.marshal().getBytes();
               DatagramPacket replyPacket = new DatagramPacket(data, data.length);
               replyPacket.setAddress(packet.getAddress());
               replyPacket.setPort(packet.getPort());
               socket.send(replyPacket);
          }
     } catch (IOException ex) {
          Logger.getLogger(RequestAcceptWorker.class.getName()).log(Level.SEVERE, null, ex);
     }
}
```

Commands Registry

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud




```
public class ToLowerCaseCommand implements ICommand {
    @Override
    public Reply execute(Request request) {
        String message = request.getParameterValue("message");
        Reply reply = new Reply(request, "OK", message.toUpperCase());
        return reply;
    }
}
```