Names are used to uniquely identify entities and refer to locations. An important issue is name resolution.

Names

A **name** is a string of characters that is used to refer to an entity (e.g. host, printer, file).

Entities have access points to invoke operations on them ⇒ **address** is the name of the access point.

An identifier is a name which uniquely identifies an entity.

Name space

Names in distributed systems are organized into a name space.

Name spaces are organized hierarchically.

Representation as a labeled directed graph.

Path along graph edges specifies the entity name, e.g. documents/projects/lecture2003/concept.txt; absolute vs relative path names.

**Name resolution**: a name lookup returns the identifier or the address of an entity, e.g. **LDAP Name Service**.
Example from the client perspective

```java
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            echoSocket = new Socket("www.in.tum.de", 7); // create Socket
            // create Writer, Reader
            out = new PrintWriter(echoSocket.getOutputStream(), true);
            in = new BufferedReader(
                new InputStreamReader(echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("unknown host in.tum.de");
            System.exit(1);
        } catch (IOException e) {
        }
        // read streams
        BufferedReader stdin = new BufferedReader(
            new InputStreamReader(System.in));
        String user_input;
        while ((user_input = stdin.readLine()) != null) {
            out.println(user_input);
            System.out.println("echo: " + in.readLine());
        }
        // close streams and sockets
        out.close();
        in.close();
        stdin.close();
        echoSocket.close();
    }
}
```

Example

System.err.println("unknown host in.tum.de");
System.exit(1);
}
```

Bidirectional communication

Usage of the request-answer scheme for massage exchange.

**Sockets**

Communication between sender and receiver is influenced by the following situations:
- loss of request messages,
- loss of answer messages,
- sender crashes and is restarted,
- receiver crashes and is restarted.

**Different types of call semantics**

Example

```java
void bind(SocketAddress bindpoint) { Binds the socket to a local address.
}
void close() { Closes this socket.
}
void connect(SocketAddress endpoint) { Connects this socket to the server.
}
void connect(SocketAddress endpoint, int timeout) { Connects this socket to the server with a specified timeout value.
```
Any communication between a sender and a receiver is subject to communication failures. Therefore, we distinguish between different call semantics.

**at-least-once semantics**

**exactly-once semantics**

**last semantics**

Under a last semantics, the requested service operation is processed once or several times, however, only the last processing produces a result and, potentially, some side-effects.

**at-most-once semantics**

Under an at-most-once semantics, the requested service operation is processed once or not at all.

Example for providing at-most-once semantics:

After timeout at the sending site the request is not retransmitted.

The request is transmitted in the context of a transaction.