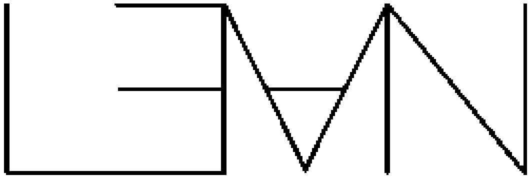
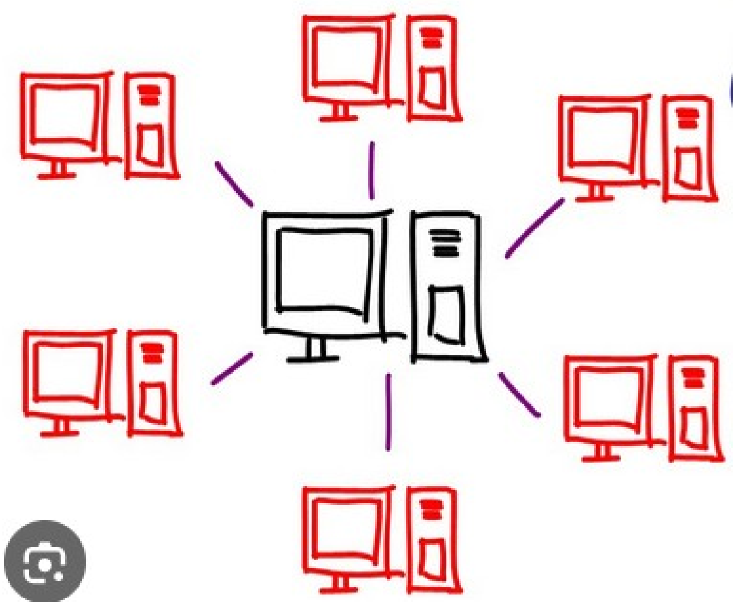
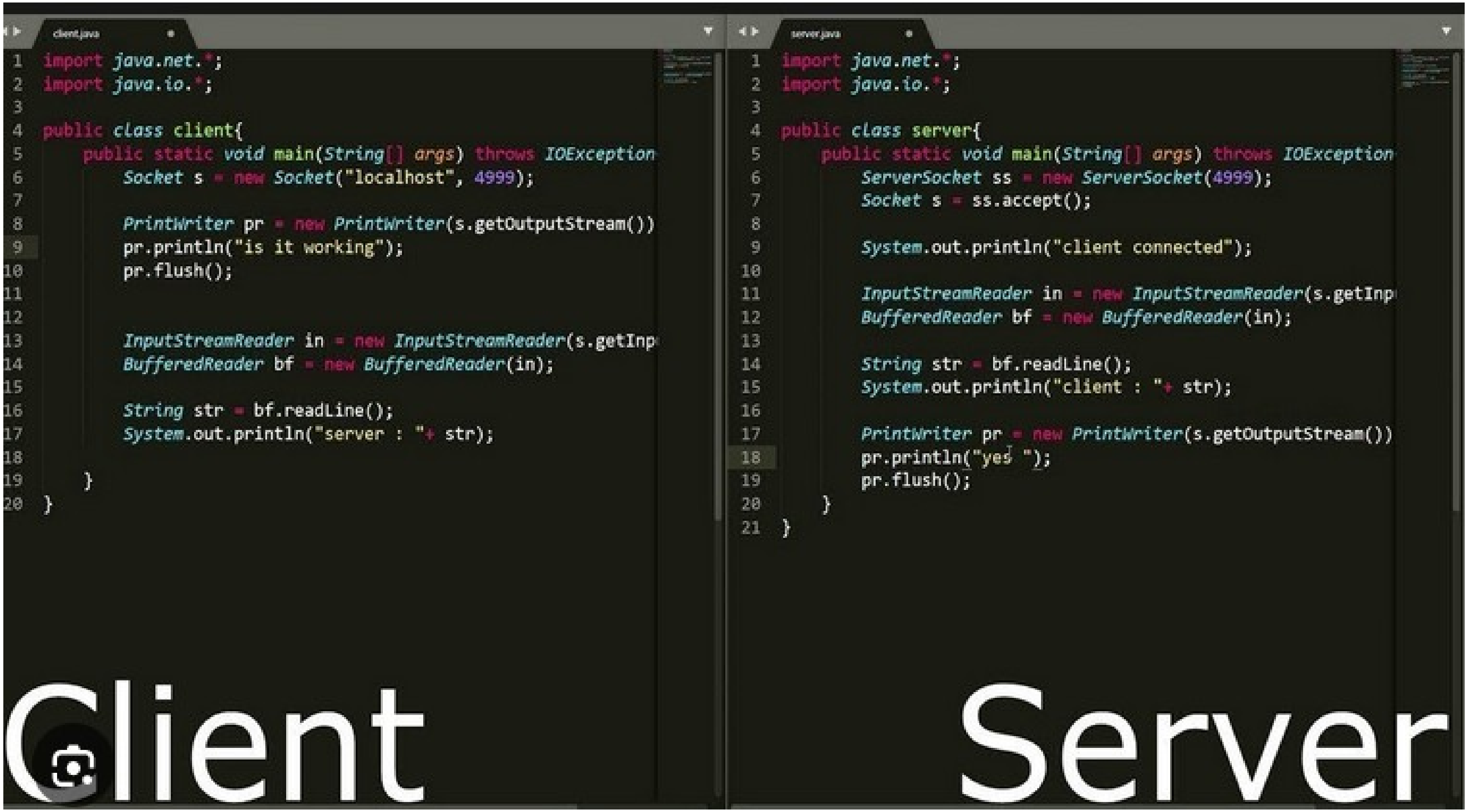
Choregraphies and Session Types

Master Thesis until April 2024 by me :)





First Google search result for client/server program

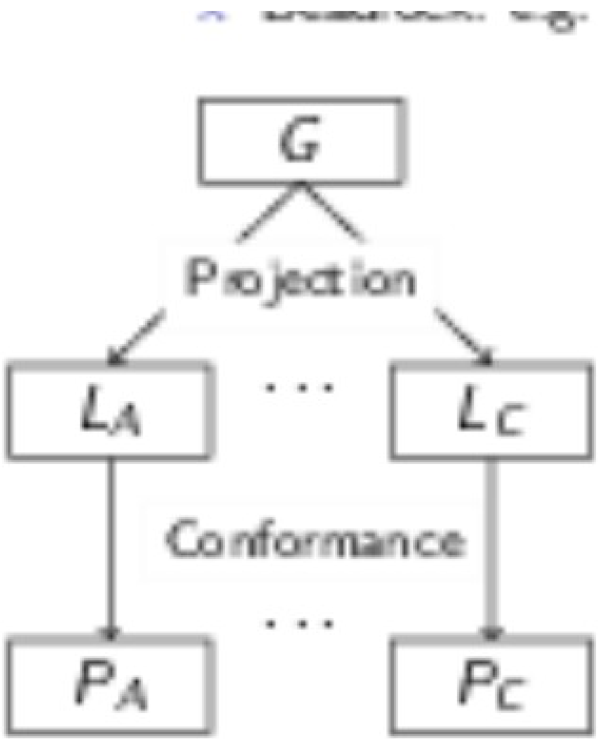


```java
import java.net.*;
import java.io.*;

public class client{
    public static void main(String[] args) throws IOException
        Socket s = new Socket("localhost", 4999);

        PrintWriter pr = new PrintWriter(s.getOutputStream())
        pr.println("is it working");
        pr.flush();


        InputStreamReader in = new InputStreamReader(s.getInp
        BufferedReader bf = new BufferedReader(in);

        String str = bf.readLine();
        System.out.println("server : "+ str);

    }
}
```

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

public class server{
    public static void main(String[] args) throws IOException
        ServerSocket ss = new ServerSocket(4999);
        Socket s = ss.accept();

        System.out.println("client connected");

        InputStreamReader in = new InputStreamReader(s.getInp
        BufferedReader bf = new BufferedReader(in);

        String str = bf.readLine();
        System.out.println("client : "+ str);

        PrintWriter pr = new PrintWriter(s.getOutputStream())
        pr.println("yes ");
        pr.flush();
    }
}
```

Choregraphies

- Intuition of a „play“ or a dance choreo
- A global description of computation and Communication
- Combines send and receive in one operation
- Think of MPI with MPI_Sendrecv()



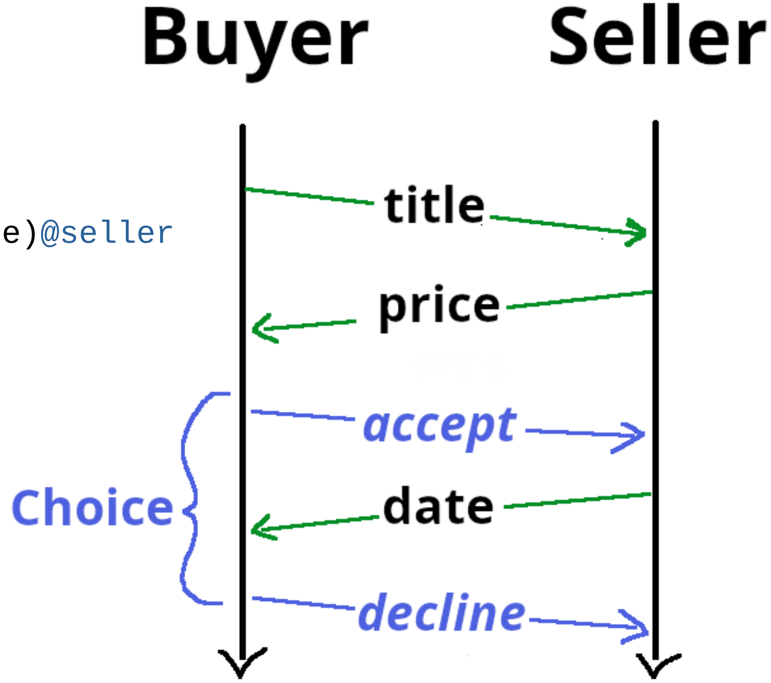
Session Types

- Describes the „communication protocol“ of parties as a type from a local view
- Easier verification for deadlock-freedom safety and liveness
- Traditional binary session types can be verified easily (duality)
- Extensions to Multiparty Session Types exist

Example

Choreography:

```
requested_title@seller <= title@buyer
price@buyer <= price_of(requested_title)@seller
if (price < budget)@buyer
  delivery_date@buyer <= delivery_date_of(requested_title)@seller
  delivery_date@seller
else
  0@buyer
```



Global Type string representation:

```
buyer  --> seller: Nat.
seller --> buyer : Nat.
buyer--> (seller):
{
  accept:
    seller --> buyer: Nat.
    end
[]decline:
    end
}
```

Enpoint Projection

- Projecting a Global Type (or program) to a session Type (or local program)

Session Type

```
?(buyer, Nat).
!(buyer, Nat).
choice@buyer{
  !(buyer, Nat).
  end
[]
  end
}
```

Lean executable Program

```
def client2: IO Unit := do
  let sock ← Socket.mk .inet .stream
  let local_addr: Socket.SockAddr4 := .v4 (.mk 127 0 0 1) 4599
  sock.connect local_addr
  let requested_title ← sock.recvNat
  sock.sendNat (price_of requested_title)
  Let branch ← sock.recvBool
  if (branch == true) then
     sock.sendNat (delivery_date_of requested_title)
  else
     ()
```

Further Goals

- Extend sendable Types
- Generate working Lean executable
- Implement a few „real world examples“, like cryptographic protocols
- Show that projecting a choreography to a location results in the same (or aquivalent) session type than projecting the corresponding Global Type to the location for my limited DSL