3C03 Concurrency: Message Passing

Wolfgang Emmerich

Outline

- Motivation
- Synchronous Message Passing
- Modelling Synchronous Message Passing in FSP
- Selective Receive
- Asynchronous Message Passing
- Modelling Asynchronous Message Passing in FSP
- Rendevous in Java

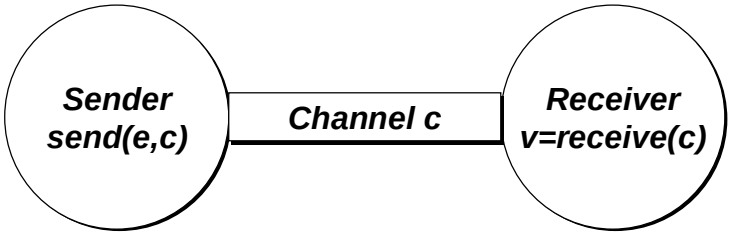
Absence of Shared Memory

- *In previous lectures interaction between threads via shared memory*
- *In java references to shared objects*
- *Usually encapsulated in Monitors*
- *In a distributed setting shared memory does not exist*
- *Communication is achieved via passing messages between concurrent/parallel threads*

Forms of Message Passing

- *Principal Operations*
  - *send*
  - *receive*
- *Synchronization*
  - *Synchronous*
  - *Asynchronous*
  - *Rendevous*
- *Multiplicity*
  - *one-one*
  - *many-one*
  - *many-many*
- *Anonymity*
  - *anonymous message passing*
  - *non-anonymous message passing*
- *Receipt of Messages*
  - *Unconditional*
  - *Selective*

Synchronous Message Passing



- *send(e,c): Send e to channel c. Sending process is blocked until channel received e*
- *v=receive(c): Receive a value into a local variable v from channel c. The calling process is blocked until a message is sent into channel*
- *No buffering*

Sync. Message Passing in Java

- *Encapsulate message passing abstractions in monitor* `Channel`*:*

```
class Channel extends Selectable {
  public synchronized void send (Object v)
         throws InterruptedException{...}
  public synchronized Object receive() {...}
}
```

Demo

Modelling Sync. Message Passing

```
range M=0..9
SENDER = SENDER[0],
SENDER[e:M]=(chan.send[e]->SENDER[(e+1)%10]).
RECEIVER = (chan.receive[v:M]->RECEIVER).
||SYNCMSG = (SENDER || RECEIVER)
           /{chan/chan.{send,receive}}.
```

- *To avoid re-labelling:*

```
range M=0..9
SENDER = SENDER[0],
SENDER[e:M]=(chan[e]->SENDER[(e+1)%10]).
RECEIVER = (chan[v:M]->RECEIVER).
||SYNCMSG = (SENDER || RECEIVER).
```


LTSA

Selective Receives

- *Receiving from more than one channel*
- *Stuck if we choose the wrong channel*
- *Selective receives (e.g. Occam or Ada):*

```
select when G1 and v1=receive(chan1) => S1;
    or when G2 and v2=receive(chan2) => S2;
    ...
    or when G3 and vn=receive(chann) => S3;
end;
```

- *Note similarity to FSP guarded actions*

Modelling Selective Receives

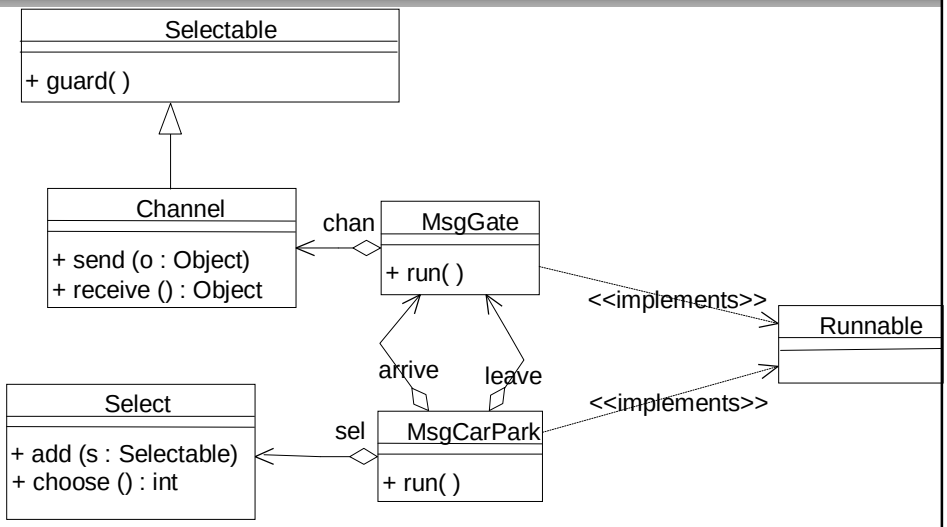
- ***Example: Car Park Control***

```
CONTROL(N=4) = SPACES[N],
SPACES[i:0..N]=(when(i>0)arrive->SPACES[i-1]
               |when(i<N)depart->SPACES[i+1]
               ).
ARRIVALS=(arrive->ARRIVALS).
DEPARTURES=(depart->DEPARTURES).
||CARPARK=(ARRIVALS||DEPARTURES||CONTROL(4)).
```

- ***How to implement `CONTROL` using Message Passing?***

Demo

Classes for Selective Receive



Implementing Selective Receives

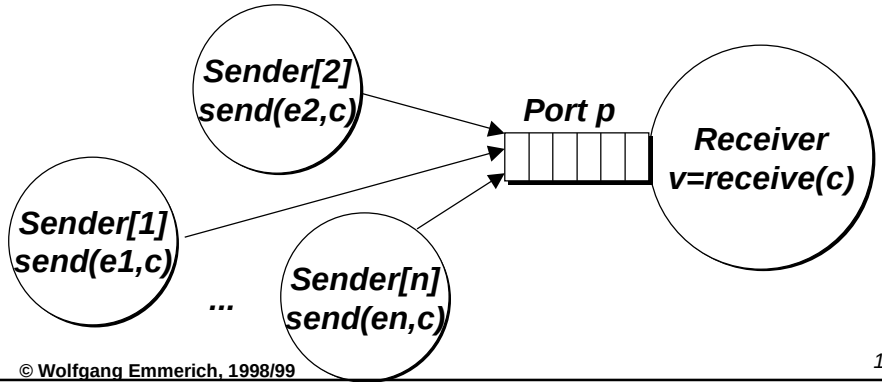
```
class MsgGate implements Runnable {
    private Channel chan;
    private Object signal = new Object();
    public MsgGate (Channel c) {chan=c;}
    public void run() {
        try {
          while(true)  {
            ThreadPanel.rotate(12);
            chan.send(signal);
            ThreadPanel.rotate(348);
          }
        } catch (InterruptedException e){}
    }
}
```


Implementing Selective Receives

```
class MsgCarPark implements Runnable {
  private Channel arrive,leave;
  private int spaces,N;
  public MsgCarPark(Channel a, Channel l, int capacity) {
    leave=l; arrive=a; N=spaces=capacity;
  }
  public void run() {
    try {
     Select sel = new Select();
     sel.add(leave); sel.add(arrive);
     while(true) {
      ThreadPanel.rotate(12);
      arrive.guard(spaces>0);
      leave.guard (spaces<N);
      switch (sel.choose()) {
      case 1:leave.receive();display(++spaces); break;
      case 2:arrive.receive();display(--spaces); break;
      }
     }
    } catch (InterruptedException e){}
  }
}
```

Asynchronous Message Passing

- *Send does not block*
- *Messages are queued at the receiver*
- *We refer to these queues as ports*
- *Communication can be many-to-one*



Async. Message Passing in Java

- *Two operations*
  - *send(e,p): send value e to port p. Calling process not blocked*
  - *v=receive(p): receive value into var v from port p. Calling process is blocked if no value queued to port.*
- *Implementation of Ports in Java:*

```
class Port extends Selectable{
Vector queue;
 public synchronized void send(Object v) {...}
 public synchronized Object receive()
             throws InterruptedException {...}
}
```

Demo

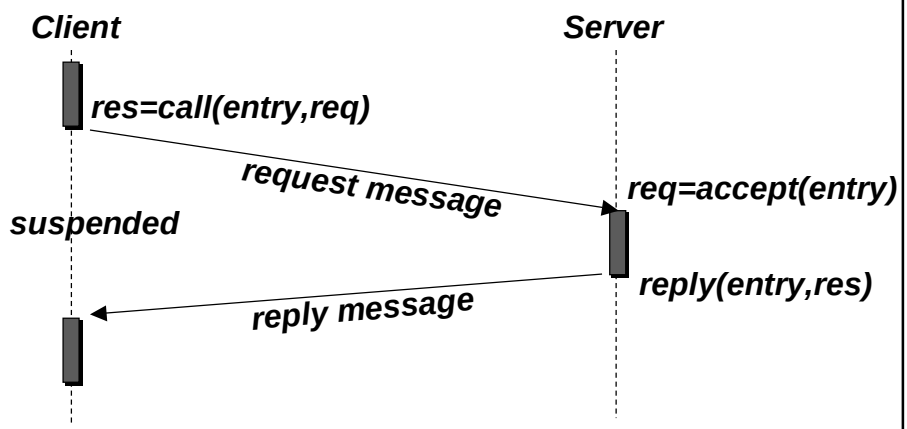
Modelling Async. Message Passing

```
range M = 0..4
set S = {[M],[M][M]}
PORT = (send[x:M]->PORT[x]),
PORT[h:M] = (send[x:M] ->PORT[x][h]
            |receive[h]->PORT),
PORT[t:S][h:M] = (send[x:M] ->PORT[x][t][h]
                  |receive[h]->PORT[t]).
ASENDER=ASENDER[0],
ASENDER[e:M]=(port.send[e]->ASENDER[(e+1)%4]).
ARECEIVER=(port.receive[v:M]->ARECEIVER).
||ASYNCM=(s[1..2]:ASENDER||port:PORT||ARECEIVER
  /{s[1..2].port.send/port.send}.
```

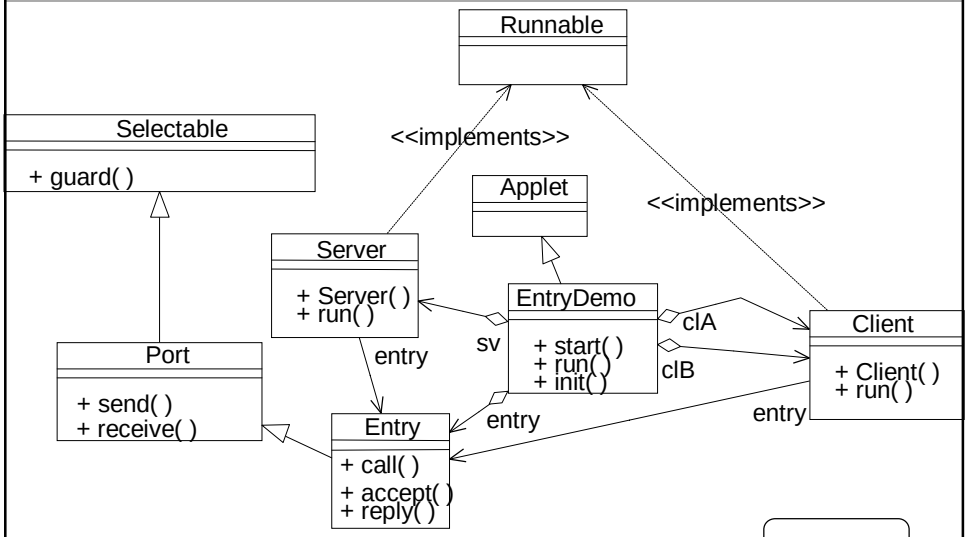

LTSA

Rendevous Message Passing

- *Request-Reply Protocol to support client-server interaction*



Rendevous in Java



Demo

Summary

- ***Synchronous Message Passing***
- ***Modelling Synchronous Message Passing in FSP***
- ***Selective Receive***
- ***Asynchronous Message Passing***
- ***Modelling Asynchronous Message Passing in FSP***
- ***Rendevous***