



C340 Concurrency: Model-based Design

Wolfgang Emmerich



Outline

- ***Role of Modelling in System Development***
- ***Refining Models into Designs***
 - ***FSP Actions and Operations***
 - ***FSP Processes and Threads***
 - ***FSP Processes and Monitors***
 - ***FSP Safety Properties and Monitor Invariants***
- ***Single-Lane Bridge Revisited***



What are models?

- *Abstract representations of the real world*
- *Neglect details considered unnecessary*
- *Example: mathematical model of bridge*
- *Supports analysis to check properties of interest*

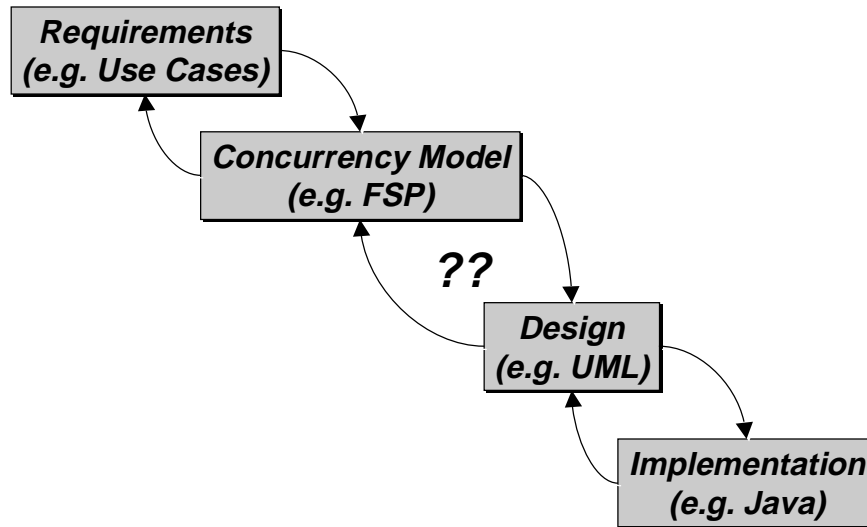


Why do we model?

- *Guide system design*
- *Improve system design in a cost-effective way*
- *Obtain formal basis for verifying correctness of the system*



Engineering Concurrent Programs



© Wolfgang Emmerich, 1998/99

5



Model-based Design

- **What is the relationship between concurrency model (e.g. in FSP) and design (e.g. in UML)?**
- **How can we derive the UML design for a concurrent program from an FSP model?**
- **How do we transfer properties that have been proven to hold in FSP into an UML design?**

© Wolfgang Emmerich, 1998/99

6



Single Lane Bridge FSP

```

CAR = (request->enter->exit->CAR).
NOPASS1=C[1], C[i:ID]=([i].enter->C[i%N+1]).
NOPASS2=C[1], C[i:ID]=([i].exit->C[i%N+1]).
|| CONVOY=( [ID]:CAR | NOPASS1 | NOPASS2 ).
|| CARS=(red:CONVOY | blue:CONVOY).
SAFEBRIDGE=SAFEBRIDGE[0][0],
SAFEBRIDGE[nr:T][nb:T]=
  (when (nb=0) red[ID].enter->BRIDGE[nr+1][nb]
   |when (nr=0) blue[ID].enter->BRIDGE[nr][nb+1]
   |when (nr>0) red[ID].exit -> BRIDGE[nr-1][nb]
   |when (nb>0) blue[ID].exit-> BRIDGE[nr][nb-1]).
FAIRBRIDGE=FAIRBRIDGE[0][0][0][0][True],
FAIRBRIDGE[nr:T][nb:T][wr:T][wb:T][bt:B]=
  (when (wr<N) red[ID].request->BRIDGE[nr][nb][wr+1][wb][bt]
   |when (wb<N) blue[ID].request->BRIDGE[nr][nb][wr][wb+1][bt]
   |when (nb==0&&(wb==0 | !bt))red[ID].enter->BRIDGE[nr+1][nb][wr-1][wb][bt]
   |when (nr==0&&(wr==0 | bt))blue[ID].enter->BRIDGE[nr][nb+1][wr][wb-1][bt]
   |when (nr>0) red[ID].exit -> BRIDGE[nr-1][nb][wr][wb][True]
   |when (nb>0) blue[ID].exit-> BRIDGE[nr][nb-1][wr][wb][False]).
|| SINGLELANEBRIDGE = (CARS || FAIRBRIDGE)>>{red[ID].exit,blue[ID].exit}.
property ONEWAY=(red[ID].enter -> RED[1] | blue[ID].enter ->BLUE[1],
  RED[i:ID]=(red[ID].enter->RED[i+1]
    |when (i==1)red[ID].exit->ONEWAY)
    |when (i>1) red[ID].exit->RED[i-1]),
  BLUE[i:ID]=(blue[ID].enter->BLUE[i+1]
    |when (i==1)blue[ID].exit->ONEWAY)
    |when (i>1) red[ID].exit->BLUE[i-1]).
|| TESTSAFETY = (CARS || SAFEBRIDGE).
© Wolfgang Emmerich, 1998/99

```



7



Deriving Thread Classes

- **Those FSP processes that represent active entities**
- **Candidates: Processes that are instantiated multiple times**
- **Example in Single Lane Bridge: Cars**

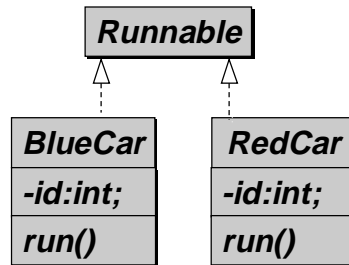
© Wolfgang Emmerich, 1998/99

8



Deriving Thread Classes

```
CAR = (request->enter->exit->CAR).
NOPASS1=C[1], C[i:ID]=([i].enter->C[i%N+1]).
NOPASS2=C[1], C[i:ID]=([i].exit->C[i%N+1]).
||CONVOY=([ID]:CAR ||NOPASS1 ||NOPASS2).
||CARS=(red:CONVOY ||blue:CONVOY).
```



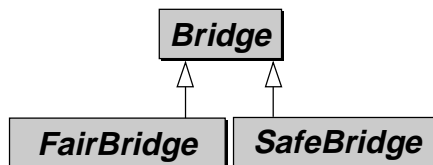
Deriving Monitors

- *Those FSP processes that are not active do not have to be separate threads*
- *If they represent shared resources are likely to become monitors*
- *Example in FSP: Bridge*



Deriving Monitors

FAIRBRIDGE[nr:T][nb:T][wr:T][wb:T][bt:B]= ...
SAFEBRIDGE[nr:T][nb:T]= ...



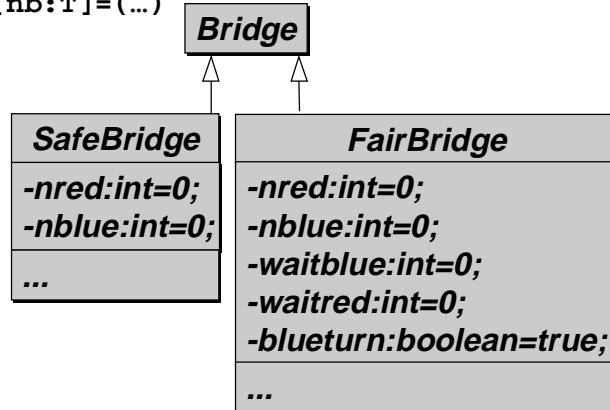
Deriving Monitor Variables

- *FSP Processes that represent monitors often maintain a local state in terms of index variables*
- *These index variables also have to be used monitors*
- *They have to become instance variables of the monitor*
- *Example: Instance Variables of FairBridge and SafeBridge*



Deriving Monitor Variables

```
FAIRBRIDGE=FAIRBRIDGE[0][0][0][0][True],  
FAIRBRIDGE[nr:T][nb:T][wr:T][wb:T][bt:B]=(...).  
SAFEBRIDGE=SAFEBRIDGE[0][0],  
SAFEBRIDGE[nr:T][nb:T]=(...)
```



© Wolfgang Emmerich, 1998/99

13



Deriving Monitor Operations

- *FSP processes that became monitors engage in actions*
- *These actions are candidates to become operations in monitor classes*
- *For implementation purposes it may be appropriate to subsume operations or split these operations*

© Wolfgang Emmerich, 1998/99

14

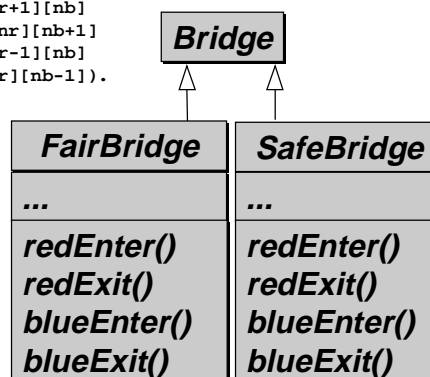


Deriving Monitor Operations

```

FAIRBRIDGE[nr:T][nb:T][wr:T][wb:T][bt:B]=
  (when(wr<N) red[ID].request->BRIDGE[nr][nb][wr+1][wb][bt]
  |when(wb<N) blue[ID].request->BRIDGE[nr][nb][wr][wb+1][bt]
  |when(nb==0&&(wb==0||!bt))red[ID].enter->BRIDGE[nr+1][nb][wr-1][wb][bt]
  |when(nr==0&&(wr==0||!bt)) blue[ID].enter->BRIDGE[nr][nb+1][wr][wb-1][bt]
  |when(nr>0) red[ID].exit -> BRIDGE[nr-1][nb][wr][wb][True]
  |when(nb>0) blue[ID].exit-> BRIDGE[nr][nb-1][wr][wb][False]).
SAFEBRIDGE[nr:T][nb:T]=
  (when (nb==0) red[ID].enter->BRIDGE[nr+1][nb]
  |when (nr==0) blue[ID].enter->BRIDGE[nr][nb+1]
  |when (nr>0) red[ID].exit -> BRIDGE[nr-1][nb]
  |when (nb>0) blue[ID].exit-> BRIDGE[nr][nb-1]).

```



© Wolfgang Emmerich, 1998/99

15



Deriving Monitor Conditions

- **Guards of those FSP processes that have become monitors determine conditions for synchronization**
- **Can be directly derived by negation**
- **Negation due to opposite perspectives:**
 - **Guard: When can action happen?**
 - **Monitor: How long do we have to wait before action can happen?**

© Wolfgang Emmerich, 1998/99

16



Deriving Monitor Conditions

■ **FSP:**

```
FAIRBRIDGE[nr:T][nb:T][wr:T][wb:T][bt:B]=  
...  
|when(nb==0&&(wb==0 || !bt))red[ID].enter->...
```

■ **Java:**

```
class FairBridge {  
synchronized void redEnter() throws InterruptedException{  
    ++waitred;  
    while (nblue>0 || (waitblue>0 && blueturn)) wait();  
    --waitred;  
    ++nred;  
}  
...  
}
```



Deriving Monitor Invariants

- ***Safety properties that affect FSP processes that have become monitors have to hold all the time***
- ***Safety properties become monitor invariants***



Deriving Monitor Invariants

■ **FSP:** property `ONEWAY=(red[ID].enter -> RED[1] | blue[ID].enter -> BLUE[1], RED[i:ID]=(red[ID].enter->RED[i+1] | when (i==1)red[ID].exit->ONEWAY | when (i>1) red[ID].exit->RED[i-1]), BLUE[i:ID]=(blue[ID].enter->BLUE[i+1] | when (i==1)blue[ID].exit->ONEWAY | when (i>1) red[ID].exit->BLUE[i-1])).`

■ **Java:**

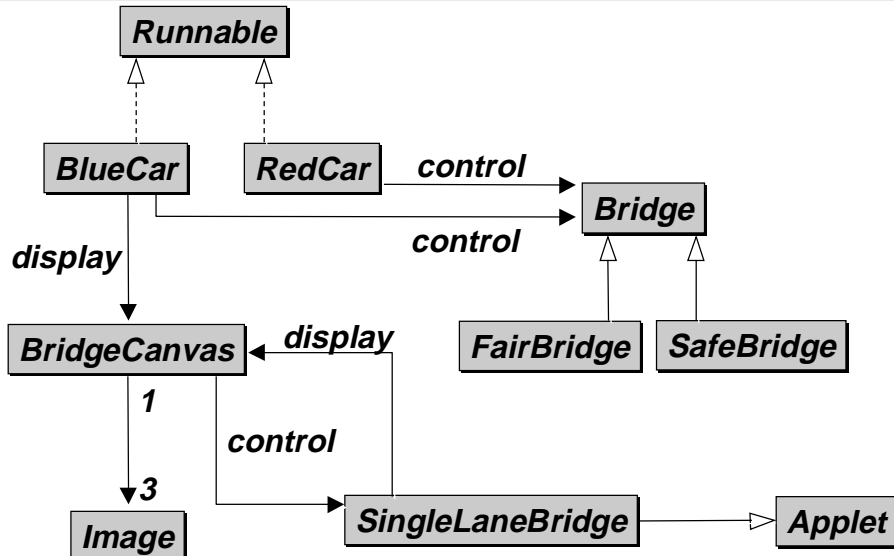
```
class SafeBridge extends Bridge {
  synchronized void redEnter() throws InterruptedException
  { while (nblue>0) wait(); ++nred; }
  synchronized void redExit()
  { --nred; if (nred==0) notifyAll(); }
  synchronized void blueEnter() throws InterruptedException
  { while (nred>0) wait(); ++nblue; }
  synchronized void blueExit()
  { --nblue; if (nblue==0) notifyAll(); }
}
```

© Wolfgang Emmerich, 1998/99

19



UML Overview of Single Lane Bridge



© Wolfgang Emmerich, 1998/99

20



Summary

- *Role of Modelling in System Development*
- *Refining Models into Designs*
 - *FSP Actions and Operations*
 - *FSP Processes and Threads*
 - *FSP Processes and Monitors*
 - *FSP Safety Properties and Monitor Invariants*