



C340 Concurrency: Concurrent Architectures: Filter Pipelines

Wolfgang Emmerich



Outline

- ***Motivation***
- ***Concurrent Prime Sieve of Eratosthenes***
- ***Modelling Prime Sieve in FSP***
- ***Buffer Tolerance***
- ***Abstraction from Filter Tasks***
- ***Architectural Property Analysis***
- ***Java Prime Sieve Implementation***
- ***Buffering***



Concurrent Architectures

- ***Software architectures identify software components and their interaction***
- ***In the context of this course architectures are process structures together with they way processes interact***
- ***Aim to ignore many of the details concerned with application***
- ***Study structures that can be used in many different situations and applications***

© Wolfgang Emmerich, 1999

3



Concurrent Architectures

- ***This is the first of three lectures each identifying a particular architectural style. Architectural styles are re-occurring patterns of components and connectors***
- ***We discuss***
 - *Filter pipelines*
 - *Supervisor workers*
 - *Announcer listener*
- ***Each of these commonly occur in concurrent and distributed systems.***

© Wolfgang Emmerich, 1999

4



Filter Pipelines

- **Filters receive input value stream and transform them into output value stream.**
- **We consider filters with one input and one output stream**
- **Filters are connected by pipelines**
 - *Redirect output of one filter to input of next*
 - *May buffer values to de-couple processes from each other*
- **Example (Unix):**
 - *cat c340.txt 1b11.txt d50.txt | sort | less*

© Wolfgang Emmerich, 1999

5



Example: Prime Sieve

- **Goal: compute primes between 2 and N**
- **Classic algorithm by Eratosthenes known as the Prime Sieve:**

```
for (i:2..N) sieve[i]:=i;
for (i:2..N)
  if (sieve[i]!=0) print(i);
  for (j:i..N)
    if (sieve[j]%i=0) sieve[j]:=0;
  end
end
```

© Wolfgang Emmerich, 1999

6

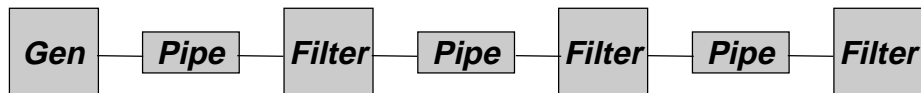


Prime Sieve FSP Model

■ Idea:

- **Generate a Stream of numbers 2..N**
- **Create one filter for each number between 2 and N that filters all the numbers that are multiples and only outputs the others**
- **Interconnect Filters by Pipes**

■ Leads to Filter Pipeline:



© Wolfgang Emmerich, 1999

7



Prime Sieve in FSP

```

const MAX=9
range NUM=2..MAX
set S={[NUM],eos}
PIPE=(put[x:S]->get[x]->PIPE).
GEN=GEN[2],
GEN[x:NUM]=(out.put[x]->if x<MAX then GEN[x+1]
                else (out.put.eos->end->GEN)).
FILTER=(in.get[p:NUM]->prime[p]->FILTER[p]
        |in.get.eos->ENDFILTER),
FILTER[p:NUM]=(in.get[x:NUM]->
                if x%p!=0 then (out.put[x]->FILTER[p])
                else FILTER[p]
                |in.get.eos->ENDFILTER),
ENDFILTER=(out.put.eos->end->FILTER).
|| PRIMES(N=4)=
  (gen:GEN | pipe[0..N-1]:PIPE | filter[0..N-1]:FILTER)
  /{pipe[0]/gen.out,
    pipe[i:0..N-1]/filter[i].in,
    pipe[i:1..N-1]/filter[i-1].out,
    end/{filter[0..N-1].end,gen.end}}
  @{filter[0..N-1].prime,end}.

```

© Wolfgang Emmerich, 1999

LTSA

8



Abstraction from Application Details

- *Above Prime Sieve Model has just one buffer slot*
- *Explosion in state space occurs if we attempt to model bigger buffer space in pipes*
- *From an architectural point of view it is not important that integers are passed as buffer elements*
- *We can abstract from this application detail*

© Wolfgang Emmerich, 1999

9



General Filter Pipeline

```
|| AGEN=GEN/{out.put/out.put[ NUM]}.
|| AFILTER=FILTER/{out.put/out.put[ NUM],
                    in.get/in.get[ NUM],
                    prime/prime[ NUM]}.
|| APIPE=PIPE/{put/put[ NUM],get/get[ NUM]}.

|| PRIMES(N=4)=(gen:AGEN | pipe[0..N-1]:APIPE |
                filter[0..N-1]:AFILTER)
                /{pipe[0]/gen.out,
                 pipe[i:0..N-1]/filter[i].in,
                 pipe[i:1..N-1]/filter[i-1].out,
                 end/{filter[0..N-1].end,gen.end}
                }.
```

© Wolfgang Emmerich, 1999

LTSA

10



Buffered Pipelines

```
||MPIPE(B=4)=
  if B==1 then APIPE
  else (APIPE/{mid/get}||MPIPE(B-1)/{mid/put})
  @{put,get}.

||PRIMES(N=4)=(gen:AGEN||pipe[0..N-1]:MPIPE||
  filter[0..N-1]:AFILTER)
  /{pipe[0]/gen.out,
  pipe[i:0..N-1]/filter[i].in,
  pipe[i:1..N-1]/filter[i-1].out,
  end/{filter[0..N-1].end,gen.end}
  }.
```

© Wolfgang Emmerich, 1999

LTSA

11



Architectural Property Analysis

- **Refer to properties for abstract model**
- **Concerned with structure and interaction**
- **Not with detailed operations**
- **General properties**
 - **Absence of deadlocks**
 - **Eventual termination**
 - **Ordering of results: Filters should produce results in the order in which they appear**

© Wolfgang Emmerich, 1999

12



Architectural Properties in FSP

■ Absence of deadlocks:

As usual

■ Termination of the system:

```
progress END = {end}
```

■ Production of results in proper order:

```
property
```

```
PRIMEP(N=4)=PRIMEP[0],
```

```
PRIMEP[i:0..N]=
```

```
  (when(i<N) filter[i].prime->PRIMEP[i+1]  
   |end->PRIMEP).
```

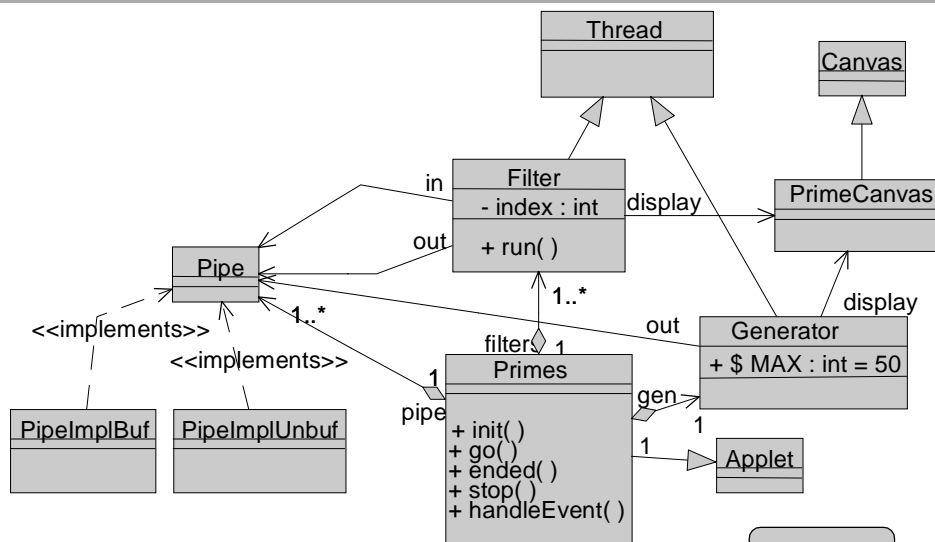
```
|| ORDER_CHECK=(PRIMES | PRIMEP).
```

© Wolfgang Emmerich, 1999

13



Java Prime Sieve Implementation



© Wolfgang Emmerich, 1999

Demo

14



Summary

- *Concurrent Software Architectures?*
- *Filter Pipelines*
- *Modelling Filters & Pipelines in FSP*
- *Abstraction from Filter Tasks*
- *Impact of Buffering*
- *Architectural Property Analysis*
- *Java Prime Sieve Implementation*
- *Buffering*
- *Next: Supervisor-Worker Architectures*