TASKS AND THREADS

CONCURRENCY IN ADA AND JAVA

Jürgen F H Winkler
Institute of Computer Science
Friedrich Schiller University, Jena

Munich, 2001.Mar.28
Overview

Terminology

Actor in Ada
Actor in Java

Process in Ada
Process in Java

Access Coordination in Ada
Access Coordination in Java

Timeout in Ada
Timeout in Java

Summary
Clearly distinguish between

- static entities in the program (text) and

- dynamic entities during program execution

<table>
<thead>
<tr>
<th></th>
<th>static</th>
<th>dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>task (object)</td>
<td>process</td>
</tr>
<tr>
<td>Java</td>
<td>Thread* - object</td>
<td>process</td>
</tr>
<tr>
<td>neutral</td>
<td>actor</td>
<td>process</td>
</tr>
</tbody>
</table>
Actor in Ada: Task-Object

Task Type MyTaskType Spec

statically
entry similar to procedure

Task Body MyTaskType

Entry Bodies are embedded in the statement sequence of the task body

=> wide variety of behavior possible
**ACTOR IN JAVA: THREAD*-OBJECT**

class Thread

```java
class Thread
{
    public void start()
    {
    }

    public void run()
    {
    }

    other()
    {
    }
}
```

two special methods

```java
public void start()
{
}

public void run()
{
}
```

and other methods

```java
class MyThreadType
{
    public void run()
    {
        myBody
    }

    myMethod()
    {
        ...
    }
}
```

run() is the essential method

is usually reimplemented in a derived class

run() corresponds to the task body in Ada
PROCESS IN ADA: a single execution of some task type object

statically declared

\[
\begin{align*}
T1 & : \text{MyTaskType} \\
T2 & : \text{YourTaskType} \\
\text{BEGIN} & \quad \text{-- start } T1, T2 \\
\ldots &
\end{align*}
\]

dynamic object

\[
\begin{align*}
\text{TRef1 := new MyTaskType;} \\
\quad \text{-- create and start} \\
\ldots &
\end{align*}
\]

direct interaction between processes via entry call / rendezvous
BEGIN

T1.E1( ... )

E1 for T0

E1 for T2

T1.E1( ... )
**PROCESS IN JAVA: EXECUTION OF A THREAD^OBJECT**

```java
MyThreadType T1 = new MyThreadType(); // creation
YourThreadType T2 = new YourThreadType();
...
T1.start(); // start and execute T1.run() concurrently
T2.start();
```

NO direct interaction between processes
i.e. between their
run( ) methods
If T2 executes

    T1.run();

the method T1.run() is just executed by T2
i.e. concurrently to the execution initiated
by T1.start
ACCESS COORDINATION IN ADA: PROTECTED TYPE / OBJECT

- T1
  - PO.E
- T2
  - PO.E
- PO
  - Entry E
  - Function F
  - Procedure P

- exclusive RW-access + barrier (coord via state)
- concurrent R-access
- exclusive RW-access
**ACCESS COORDINATION IN ADA: STACK**

PROTECTED TYPE StackTy IS

ENTRY Push(Val: IN Integer);
ENTRY Pop(Val: OUT Integer);

PRIVATE

MaxNumValues : constant Integer := 10;
StackData : ARRAY(1..MaxNumValues);
TOS: Integer Range 0..MaxNumValues := 0;

END StackTy;

PROTECTED BODY StackTy IS

FUNCTION IsEmpty Return Boolean IS Return TOS=0; END;
FUNCTION IsFull Return Boolean IS Return TOS = MaxNumValues;

ENTRY Push(Val: IN Integer) WHEN Not IsFull IS

TOS := TOS+1;
StackData(TOS) := Val;

END Push;

ENTRY Pop(Val: OUT Integer) WHEN Not IsEmpty IS

Val := StackData(TOS);
TOS := TOS-1;

END Pop;

END;
ACCESS COORDINATION IN ADA: NOCREDITACCOUNT

PROTECTED TYPE NoCreditAccountTy IS

PROCEDURE Deposit(Amount: IN Positive);
ENTRY WithDraw(Amount: IN Positive);
PRIVATE

Balance : Natural := 0;
END NoCreditAccountTy;

PROTECTED BODY NoCreditAccountTy IS

PROCEDURE Deposit(Amount: IN Positive) IS

BEGIN Balance := Balance + Amount;
END Deposit;

ENTRY WithDraw(Amount: IN Positive)

WHEN Amount <= Balance IS

BEGIN Balance := Balance – Amount;
END WithDraw;
END NoCreditAccountTy;

Program is ILLEGAL!
Visibility in barrier: all globals outside of entry espec. parameters NOT visible
Solution: use REQUEUE and additional internal entry rather cumbersome

```ada
PROTECTED TYPE NoCreditAccountTy IS
   PROCEDURE Deposit(Amount: IN Positive);
   ENTRY WithDraw(Amount: IN Positive);
PRIVATE
   Balance : Natural := 0;
   WithDrawAmount : Positive;
   WithDrawOpen: Boolean := True;
   ENTRY InternalWithDraw(Amount: IN Positive);
END NoCreditAccountTy;

PROTECTED BODY NoCreditAccountTy IS

   PROCEDURE Deposit(Amount: IN Positive) IS
      BEGIN    Balance := Balance + Amount;
      END Deposit;

   ENTRY WithDraw(Amount: IN Positive) WHEN WithDrawOpen IS
      BEGIN IF Amount <= Balance
         THEN  Balance := Balance – Amount;
         ELSE   WithDrawAmount := Amount;
            WithDrawOpen := False;
            Requeue InternalWithDraw;
            -- parameter transmission implicit
         END;
      END;
   END WithDraw;

   ENTRY InternalWithDraw(Amount: IN Positive) IS
      WHEN WithDrawAmount <= Balance
      BEGIN  Balance := Balance – Amount;
      END InternalWithDraw;

END NoCreditAccountTy;
```


**ACCESS COORDINATION IN JAVA:** *ANY OBJECT CAN BE LOCKED*

Diagram:

- **T1**
  - `Oxy.M`
- **T2**
  - `Oxy.SM`
- **T3**
  - `Oxy.SM`

- Method `M` is `synchronized`.
- `Oxy` (any object)
- Nonexclusive RW-access
- Exclusive RW-access
only access to synchronized methods is coordinated
i.e. NOT a property of type / object

- T1
  - Oxy.M

- T2
  - Oxy.SM
  - wait for exclusive RW-access

- T3
  - Oxy.SM
  - wait for exclusive RW-access

- Method M
  - synchronized Method SM
  - nonexclusive RW-access
  - exclusive RW-access

- Oxy (any object)
  - T1
  - T3
State-based coordination

instead of entry barriers an event mechanism

wait / notify / notifyAll (only ONE “event”)

synchronized Method SM1

\[
\begin{align*}
&\ldots \\
&\text{if state is unsuitable} \\
&\text{then} \\
&\quad this.wait( )
\end{align*}
\]

synchronized Method SM2

\[
\begin{align*}
&\ldots \\
&\text{if state is now suitable} \\
&\text{then} \\
&\quad this.notifyAll( )
\end{align*}
\]

wait for notify / notifyAll

wait( )

notify( )

notifyAll( )

wait for exclusive RW-access

call S-method

RS

WS
Characteristic properties

- for each object there is one RS and one WS

- acquire exclusive access to object: *one* process from RS is *arbitrarily* chosen

- notify( ): *one* process from WS is *arbitrarily* chosen and transferred from WS to RS

- notifyAll( ): transfer all processes from WS to RS

\[\Rightarrow\] since there is only *one* WS and since *notify( ) chooses arbitrarily*

*notifyAll( )* must be used quite often

\[\Rightarrow\] processes are deblocked which cannot really continue

they just make another RS-execute-WS round-trip
Access Coordination in Java: Stack

// A stack that has a 3 item limit

class Stack {
    static final int STACK_SIZE = 3;
    private int[] stack_store = new int[STACK_SIZE];
    private int stack_ptr = 0;

    // push item onto stack
    // If stack is full, wait until it has room
    synchronized public void push(int item) {
        while (stack_ptr >= STACK_SIZE) {
            try { wait(); }
            catch { InterruptedException e) { /* ignore */ }
        }
        if (stack_ptr == 0)
            notify(); // pop was awaiting stack to fill
        stack_store[stack_ptr++] = item;
    }

    // pop item off top of stack
    // If stack is empty, wait until it has item
    synchronized public int pop() {
        while (stack_ptr == 0) {
            try { wait(); }
            catch { InterruptedException e) { /* ignore */ }
        }
        if (stack_ptr >= STACK_SIZE)
            notify(); // push was awaiting stack to drain
        return (stack_store[--stack_ptr]);
    }
}


(has some flaws)
**TIME-OUT IN ADA**: CALLER SIDE

```
T1

select
    TO.E1(...);
or
delay 45.0;
end select;
```

**Characteristic properties**

- `wait` is expressed on the caller side
- can be applied to any entry (call)
- (time-out wrt BEGIN of rendezvous)

- `wait` also possible on the callee side

---

Time-out in Ada

1

F-0800-time-out-ada  J.Winkler  2001.02
**TIME-OUT IN JAVA : CALLER SIDE**

T1

Oxy.M1(\ldots, 45.0); 

Characteristic properties

seems to be very simple, but it isn’t

method M1 has to be modified in a rather complicated way

Assume: method M1 depends on suitable state
Solution

use wait (long timeout) in method M1

Problem: same WS as for other wait( )-calls is used

```java
public synchronized void M1(..., long timeout) {
    if state is not suitable
        then wait(timeout);
            if state is not suitable
                then return / throw
            end if
        end if
    end if
    perform action
    if state is suitable for waiting processes
        then notify( ) / notifyAll( )
    end if
}
```

Problem

process may continue due to some unrelated notify / notifyAll

=> could give up TOO EARLY
Solution

use wait (long timeout) in method M1 and
CHECK REPEATEDLY WHETHER timeout HAS ELAPSED

```java
public synchronized void M1(..., long timeout) {
    if state is not suitable
        then EndTime = CurrentTime+timeout;
            while true
                wait(EndTime-CurrentTime);
                if state is suitable
                    then break
                else
                    if CurrentTime >= EndTime
                        then return / throw
                    end if
                end if
        end while
    end if
    perform action
    if state is suitable for waiting processes
        then notify( ) / notifyAll( )
    end if
}
```

- Adapted from Doug Lea: “Concurrent Progr. in Java”
- still not an adequate solution
- each relevant operation has to be modified like that
- main problem: no connection between wait and state condition
### SUMMARY

<table>
<thead>
<tr>
<th>Structure of Actor</th>
<th>Ada</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Interaction</td>
<td>rendevous</td>
<td>-.</td>
</tr>
<tr>
<td>Access Coordination</td>
<td>protected object</td>
<td>synchronized method</td>
</tr>
<tr>
<td></td>
<td>very flexible problem: visib. in barrier</td>
<td>only excl. RW-access problem: primitive event mech.</td>
</tr>
<tr>
<td>Time-out</td>
<td>easy</td>
<td>quite cumbersome</td>
</tr>
<tr>
<td>Active Server</td>
<td>task with entries</td>
<td>very difficult</td>
</tr>
<tr>
<td>Remote Server</td>
<td>-.</td>
<td>RMI</td>
</tr>
</tbody>
</table>
REFERENCES AND FURTHER READING


CL 98  Chan, Patrick; Lee, Rosanna: The Java Class Libraries. Addison-Wesley, Reading etc., 1996.  0-201-63498-9


Lea 97  Lea, Doug: Concurrent Programming in Java. Addison-Wesley, Reading etc., 1997.  0-201-69581-2