AARHUS UNIVERSITET

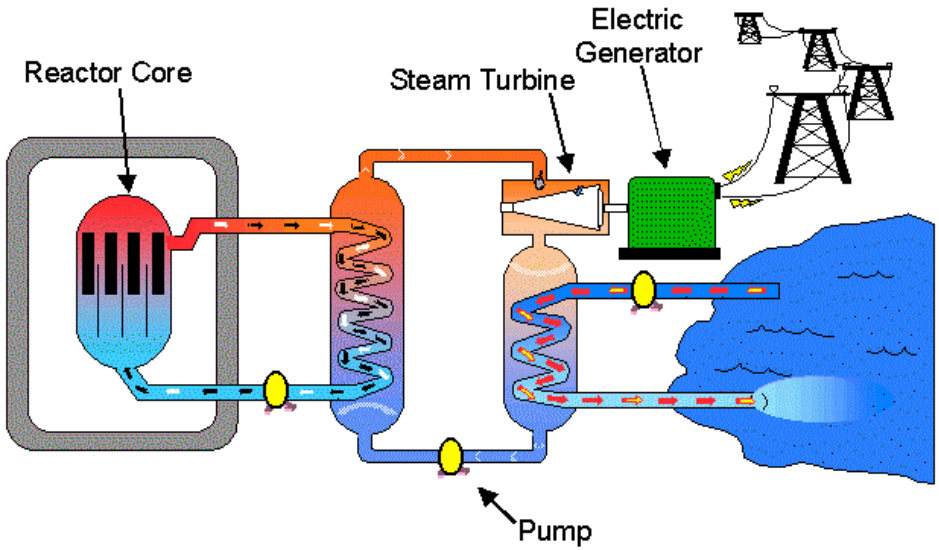
Software Architecture in Practice

Case: TS-05

Software for nuclear reactor core temperature process control

Architectural requirements are tough

– BOOM ☹



Requirements

The reactor core is controlled by graphite rods that control the amount of free neutrons in the core – and thereby the fission process

If the number of free neutrons gets too low, the fission rate increases exponentially – which is rather – Tjernobyl-like... ☹

We can also control flow of cooling water in the core

Architecture overview – in words...

Our architecture is classic 'closed-loop process control'

- loop {
  - *sense* environment, *calculate* response, *actuate*
- }

Two types of nodes

- The **temperature sensor node** is a small computing node:
  - temperature sensor hardware
  - on-board Java Virtual Machine (J2SE)
  - TCP/IP allowing Java RMI
- The **temperature monitor node** is PC type
  - monitoring, processing, control actuators
  - combine readings from many sensors

Architecture overview – in words...

For efficient control

- sensors more than 100 places in core
- 10 readings per second at least for each sensor

TS-05 in our course

We will use the case to

- do architectural documentation using views
  - to avoid both documentation *and* designing !
- describe architectural requirements
  - using the quality attribute scenario technique
- scratch in the surface of architectural design
  - using tactics
- exemplify architectural prototyping

TS-05 in our course

Pedagogical reasons

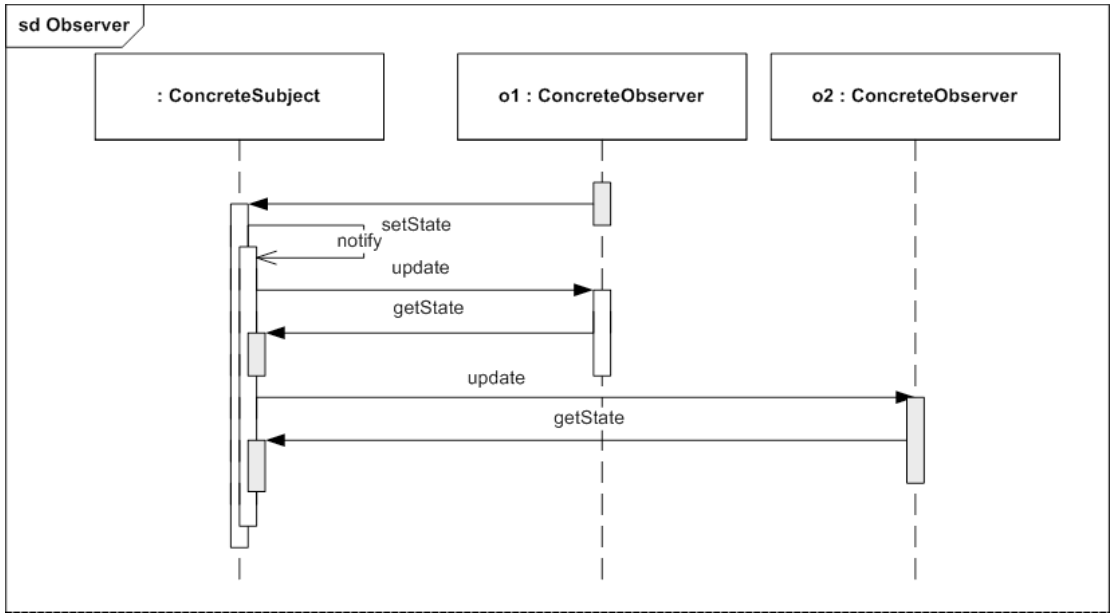
- Architecture is *not* something detached from the implementation
  - Often architectural significant decisions are deeply coupled to detailed implementation decisions; sometimes to just a few lines of code
- Small system: 150 lines of Java code
  - 8 Java source files
    - 3 interfaces (Sensor, Event, Listener)
    - 3 implementing classes
    - 2 main programs (Sensor, Monitor)
    - 3 packages (ts05, ts05.sensor, ts05.monitor)
  - Build and deployment: Ant script
    - 190 lines XML
    - Can be considered Black Box...
- No design involved !

Observer architecture

The TS-05 software system is modelled over the **observer pattern** in the **push model** variant.

– The same model used in Swing GUI components.

Pull variant
Observer protocol



Observer architecture

AARHUS UNIVERSITET

The push model dictate that *observers* get the actual data change as part of the *update* call.

– That is – no need for the ‘getData’ call !

Subject (information holder) = sensor

Observer (information processor) = monitor

Architectural prototyping

Risk management

- How to transfer temperature readings
- How is the Java RMI setup working?

Answer: *Architectural prototype*

- learning technology, experimenting architecture
- measurements, architectural evaluation
- skeleton system to “grow” final system

TS-05 Sensor interface

```
public interface TemperatureSensor extends Remote {
  /** register a listener to receive temperature data.
      @param tl the temperature listener instance to broadcast to.
  */
  public void addTemperatureListener( TemperatureListener tl )
    throws RemoteException;
}
```

A Sensor

```
public class TemperatureSensorImpl extends UnicastRemoteObject
  implements TemperatureSensor, Runnable {

  private List listenerList;

  public TemperatureSensorImpl() throws RemoteException {
    super();
    listenerList = new Vector();
  }
  private double temperature = 80.0;

  public void run() {
    while ( true ) {
      // wait 0.1 sec
      try {
        Thread.sleep(100);
      } catch ( InterruptedException e ) {}
      // make a new temperature reading
      temperature += Math.random()-0.49;
      notifyAllListeners();
    }
  }
  public synchronized void addTemperatureListener( TemperatureListener tl )
    throws RemoteException {
    listenerList.add( tl );
  }
  private int notifications=0;
  private void notifyAllListeners() {
    TemperatureEvent te = new TemperatureEventImpl(temperature);
    Iterator i = listenerList.iterator();
    while ( i.hasNext() ) {
      TemperatureListener tl = (TemperatureListener) i.next();
      try {
        tl.temperatureSampled( te );
      } catch ( RemoteException exc ) {
      }
    }
    System.out.println( "Broadcasted temperature "+temperature );
  }
}
```

TS-05 listener interface

```
public interface TemperatureListener extends Remote {
  /** this method is invoked every time a temperature sensor
      broadcasts a temperature
      @param t_event an object encapsulating the temperature value
      sampled.
  */
  public void temperatureSampled( TemperatureEvent t_event )
    throws RemoteException;
}
```

TS-05 : Code view

TS-05 temperature information

```
public interface TemperatureEvent extends java.io.Serializable {
  /** return the temperature reading that this event represents.
      @return the temperature reading measured in Celcius
  */
  public double getReading();
}
```

TS-05 – Server side

```
public class TemperatureSensorServer { … }
```

Responsibilities:

- Java RMI setup
  - Install security manager
  - Contact Name server and bind names
- Attach to concrete sensor instance (simulated!)

TS-05: Sensor server code

AARHUS UNIVERSITET

```
public class TemperatureSensorServer {

  public static void main(String[] args) {
    // define where the rmi registry is located...
    String registry_host = "//localhost/";
    System.out.println( "Using registry at "+registry_host );
    try {
      System.out.println( "Initializing  ..." );
      if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
      }
      System.out.println( "SecurityManager installed..." );

      // create a temperature sensor object representing the TS-05
      TemperatureSensorImpl
        tss = new TemperatureSensorImpl();

      String name = registry_host+"section1";
      Naming.rebind(name, tss);
      System.out.println( "Sensor object has been bound to name: "+name );

      // make tss run in its own thead...
      Thread t = new Thread( tss );
      t.start();
      System.out.println( "Sensors are up and running..." );
    } catch (Exception e) {
      System.err.println( e );
      System.exit(1);
    }
  }
}
```

TS-05: Code view

Monitor side

```
public class TemperatureMonitor {
```

Responsibility:

- Instantiate TemperatureListener instance
- Java RMI setup
- Demonstration !

```
public class TemperatureMonitor {

  /** instantiate the monitor */
  public static void main(String[] args) {

    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new RMISecurityManager());
    }

    String registry_host = "//localhost/";

    System.out.println( "Using registry at "+registry_host );

    try {
      // get the temperature sensor object reference from the
      // RMI object request broker...
        String name = registry_host+"section1";
        System.out.println( "Looking up object reference: "+name );
        TemperatureSensor ts = (TemperatureSensor) Naming.lookup(name);

        System.out.println( "Located sensor object..." );

        // Create a local temperature listener object
        // (observer pattern 'observer'-role)
        // and register it at the temperature sensor.
        // The object refers to the counter object
        TemperatureListenerImpl tl = new TemperatureListenerImpl();
        System.out.println( "Created listener..." );
        ts.addTemperatureListener(tl);
        System.out.println( "Added listener object to sensor" );

        // wait for callbacks
        System.out.println( "Finished; awaiting callbacks..." );

        // tricks-of-the-trade way of waiting on incoming calls...
        java.lang.Object sync = new java.lang.Object();
        synchronized (sync) {
          sync.wait();
        }
```

TS-05: Code view

Output:

– Not *that* exiting, but...

We have “first light”