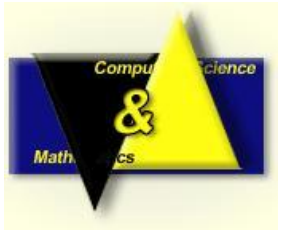
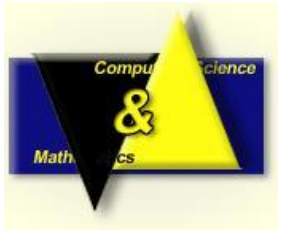


Discrete Event Systems in Software Architecture Design

Rein Smedinga
department of software engineering
University of Groningen



Introduction

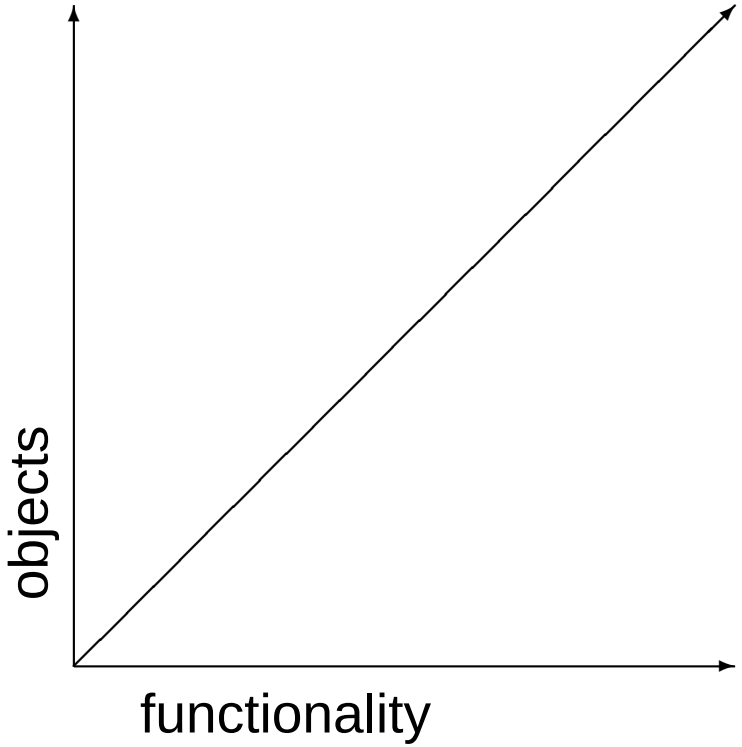
Software development still is:

- always too late
- always more expensive
- always incorrect
- always incomplete
- always . . .

Software development nowadays:

1. find requirements
2. create an architecture
3. make design decisions
4. implement
5. use / maintain / . . .

Introduction

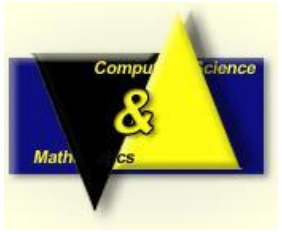


Development based on:

- procedurial, imperative
- object oriented
- trend: use both!

But: try to keep all aspects seperated as long as possible

e.g. aspect oriented programming



Where discrete events come in. . .

Our idea:

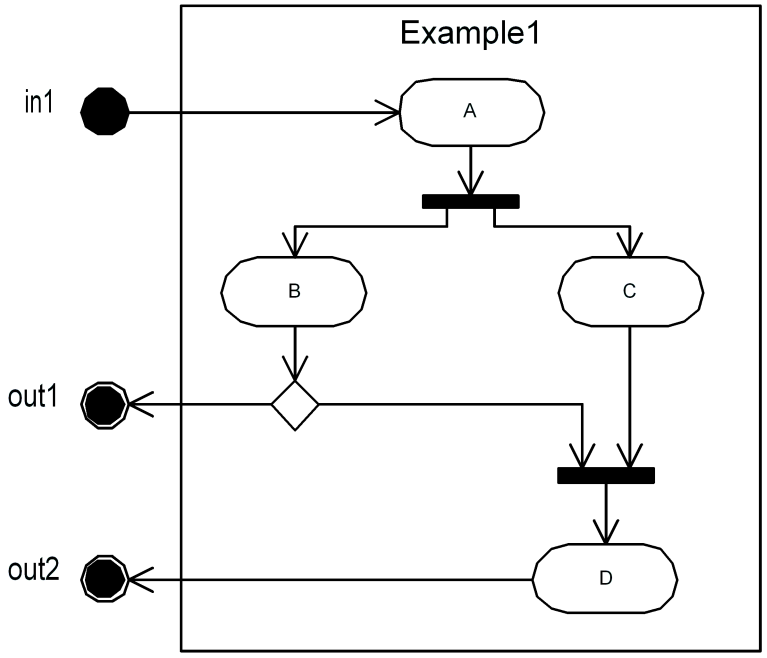
- combine functionality and object orientation

Object orientation – use fragments/components/. . .

Functionality – use events from/to/in these fragments

Notation: activity diagrams with swimlanes from UML

An example




```
fragment Example1 (in in1; out out1, out2)
begin in1 ; A
;       fork B
        ;    if condition then out1
             else s
             fi
        ||
        C ; s
        end
;       D ; out2
end
```

In formal notation:

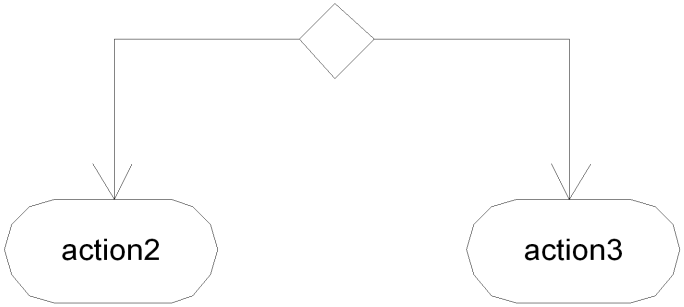
$$in1 \cdot A \cdot ((B \cdot (out1 \leftrightarrow s)) \| (C \cdot s)) \cdot D \cdot out2$$

Elements of this notation

Concatenation:


```
action1 ; action2
```

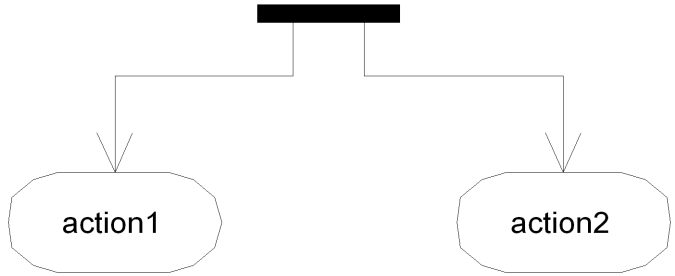
Choice:



if `B`
then `action2`
else `action3`
fi

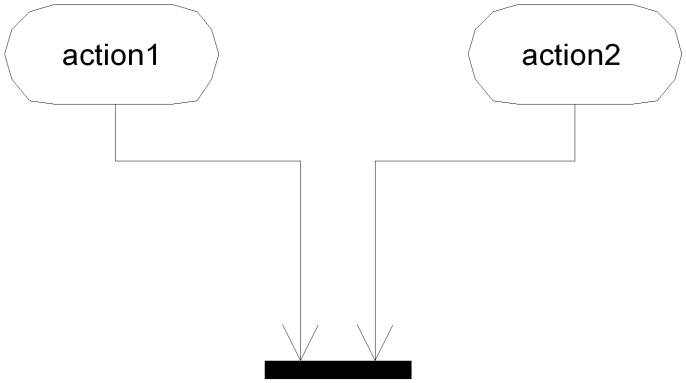
Elements of this notation

Fork:




```
fork
     action1
||
     action2
end
```

Sync:

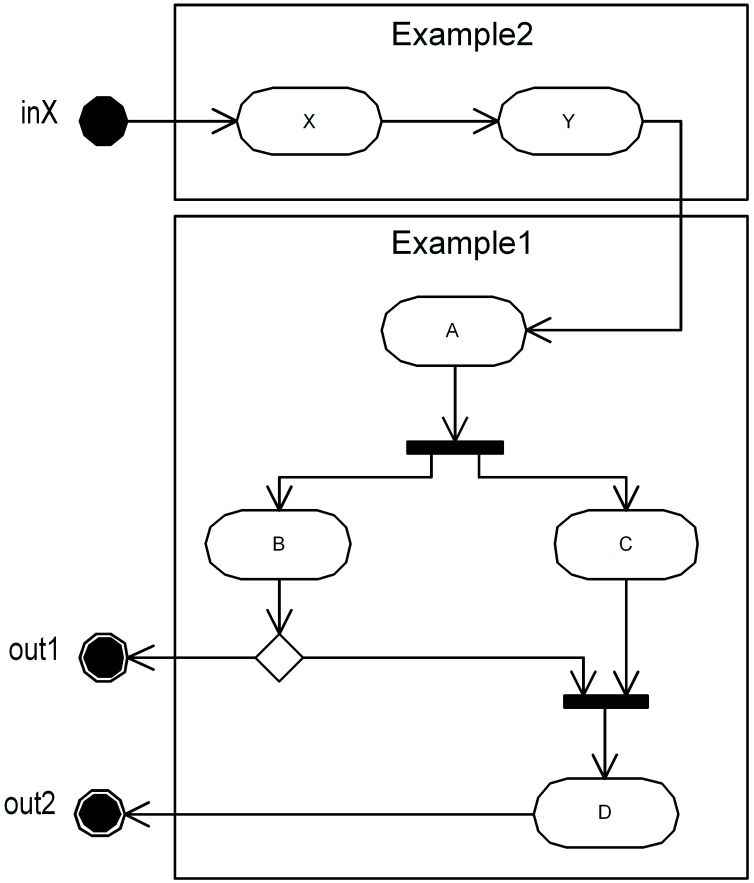



```
fork
     action1 ; s
||
     action2 ; s
end
```

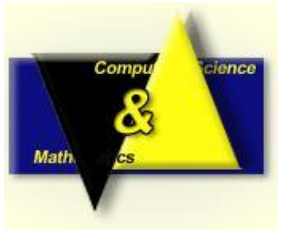
Where DES realy comes in. . .

Coupling fragments by using syncronization: couple **in**'s to **out**'s

Example:




```
fork
      Example1 (in in1; out out1, out2)
||   {with outY = in1}
      Example2 (in inX; out outY)
end
```



DES + SA ⇒ ...

What can we do with this?
What are the benefits?

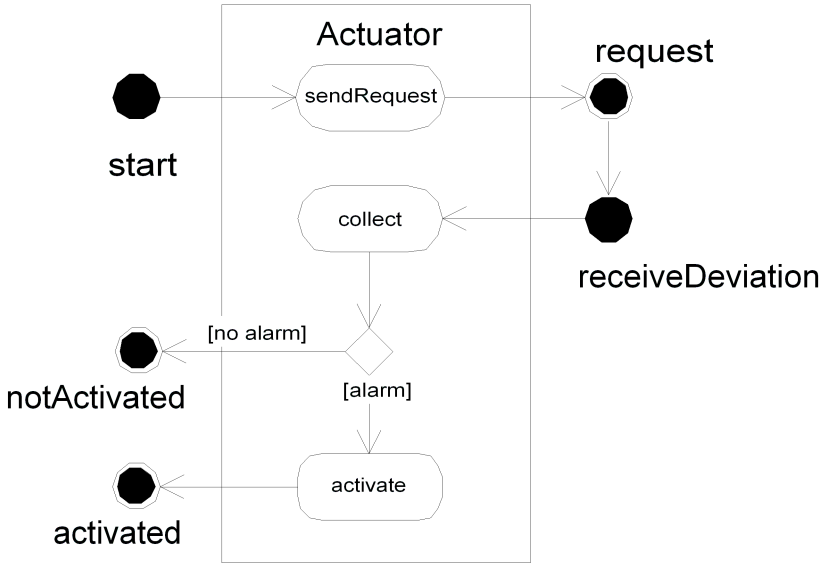
Answer:

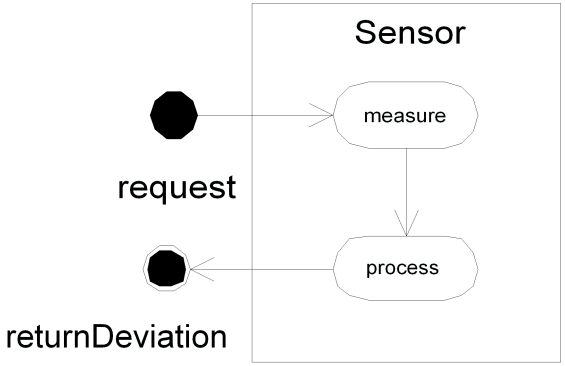
- we can develop individual fragments
- fragments can be combined
- One fragment can be changed without affecting others
- Or: changing one aspect does not influence others

So: DES + SA ⇒ independency of aspects

Also: we can do aggregation / inheritance / superimposition

Example: fire alarm system



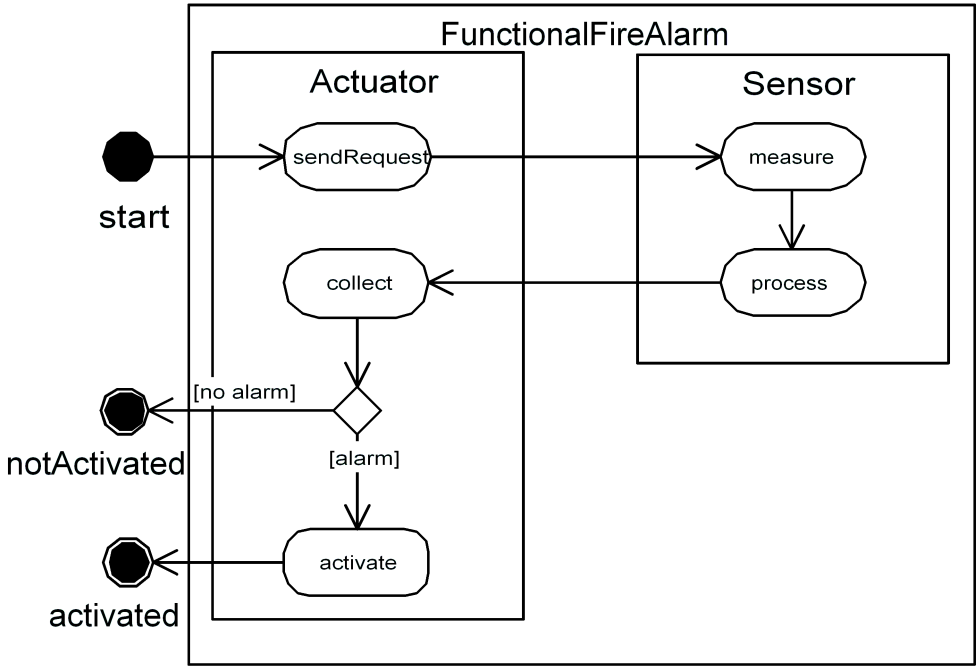


Can be coupled by :

```
Sensor.request = Actuator.request
Sensor.returnDeviation = Actuator.receiveDeviation
```

Example: fire alarm system

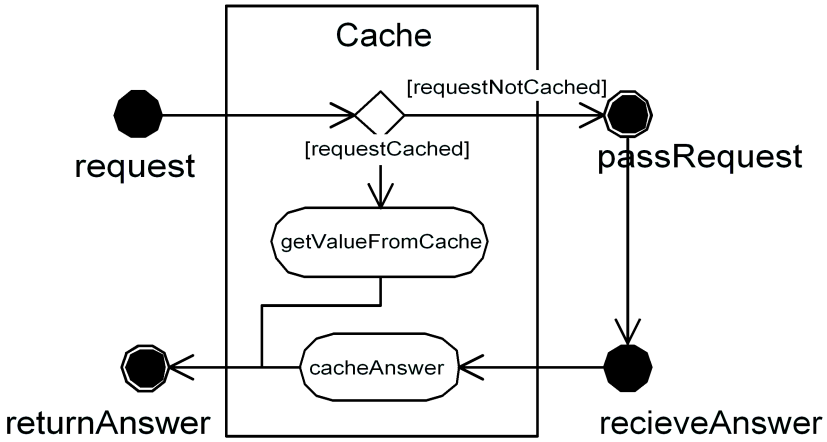
Result:



Coupled, but each fragment keept individually changable without effecting the other

Example: fire alarm with caching

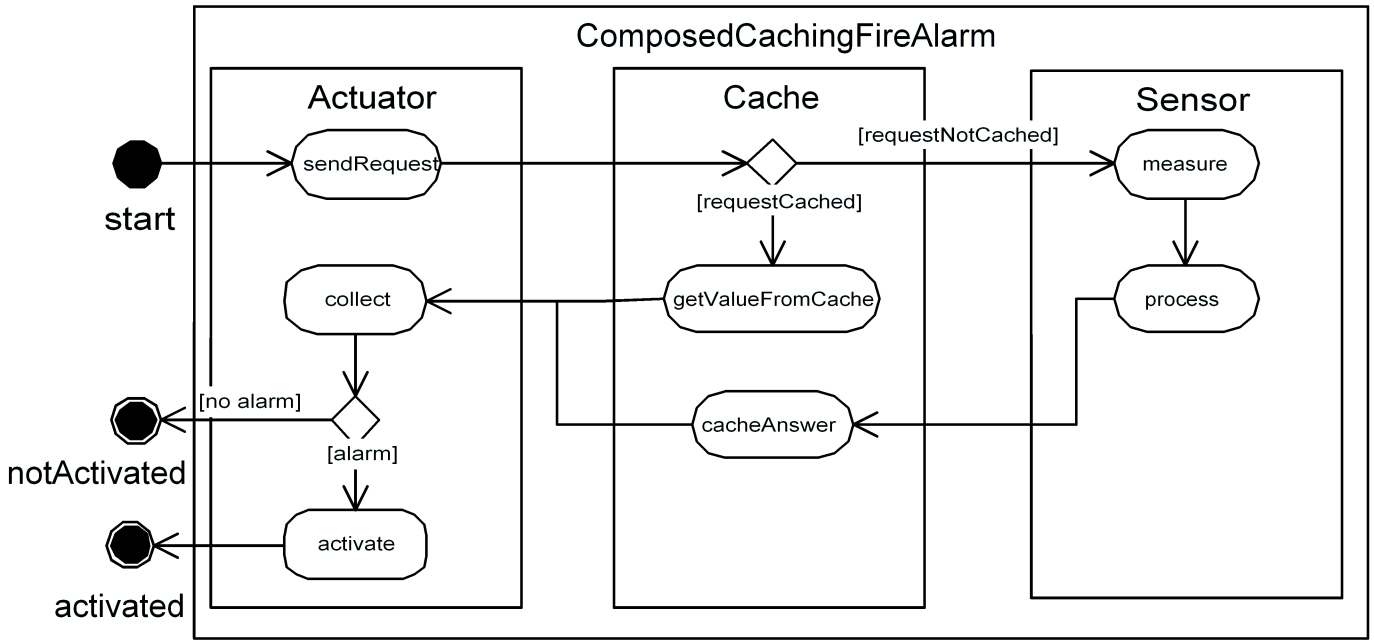
First the caching fragment:



Should be placed in between the Actuator and the Sensor.

Example: fire alarm with caching

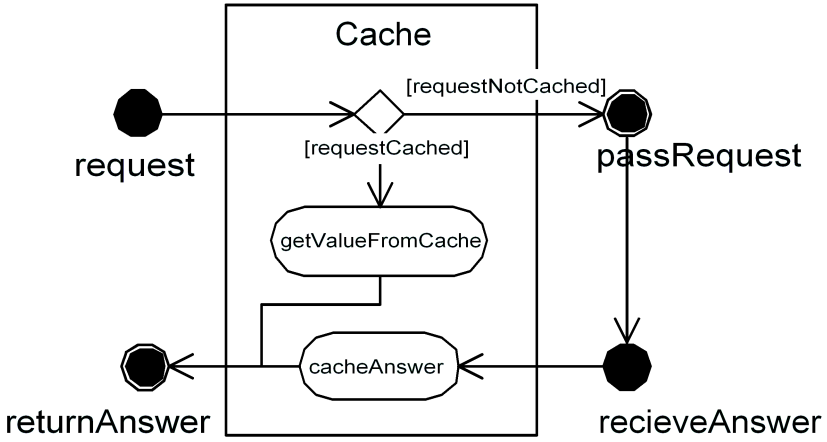
Result:

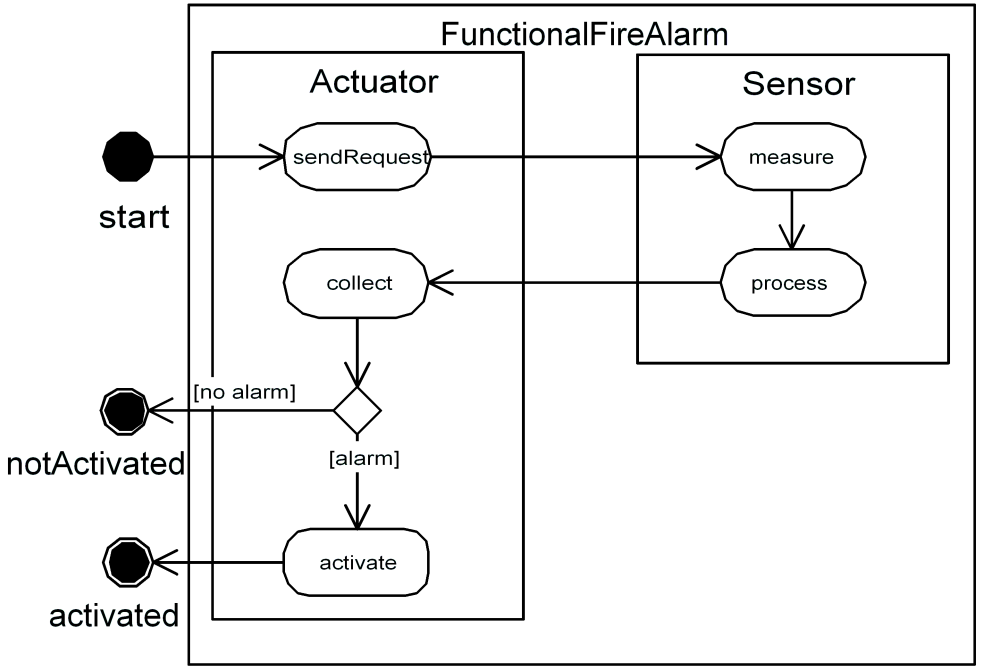


But. . . we can do it directly from the Functional Fire Alarm as well:
We use superimposition then.

Example: cached fire alarm system

Superimpose the cache-fragment in the functional fragment, i.e.

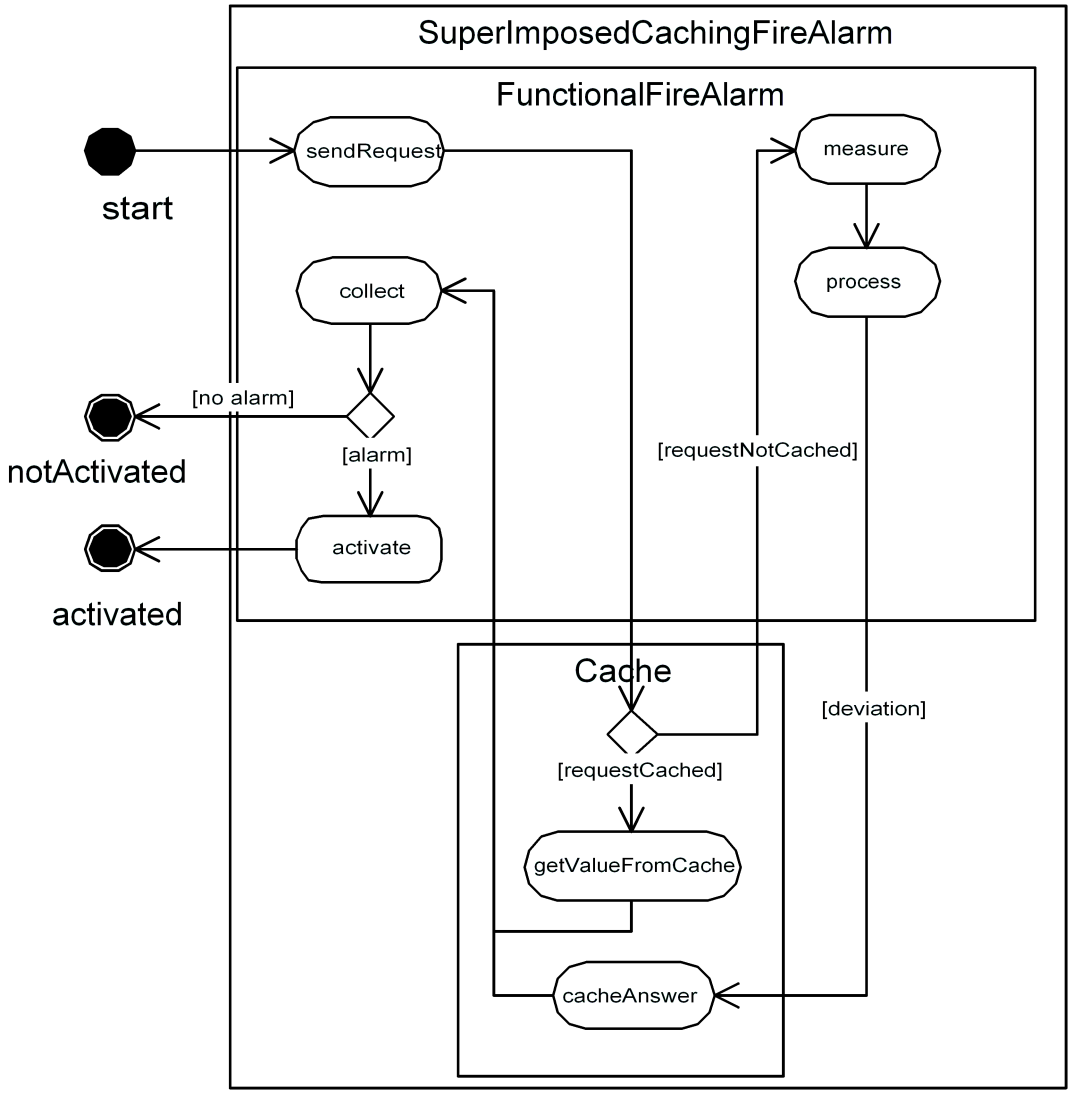




By connecting:

```
Cache.request = FunctionalFireAlarm.sendRequest.after
Cache.passrequest = FunctionalFireAlarm.measure.before
Cache.receiveAnswer = FunctionalFireAlarm.process.after
Cache.returnAnswer = FunctionalFireAlarm.collect.before
```

Example: cached fire alarm system



Formal aspects of superimposition

Fragment A:

$$a \cdot b \cdot c$$

We suppose enough internal events, e.q.

$$a \cdot f_1 \cdot f_2 \cdot \cdots \cdot f_n \cdot b \cdot g_1 \cdot \cdots \cdot g_m \cdot c$$

Fragment B:

$$d \cdot e$$

Coupling by:

$$B.e = A.b.\textbf{before}$$

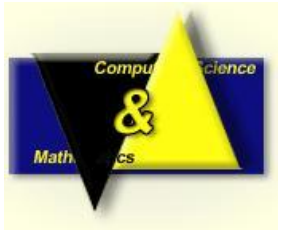
Thus:

$$B.e = A.b.\textbf{before} = A.f_n$$

Gives (with $f_n = e$):

$$\begin{aligned} & (a \cdot e \cdot b \cdot c) \| (d \cdot e) \\ = \quad & (a \| d) \cdot e \cdot b \cdot c \\ = \quad & (a \| d) \cdot b \cdot c \end{aligned}$$

(we do not show internal events)



Conclusions and further research

- DES seems useful in software architecture
- Method should be “tested” in practical situations
- . . . and need to be extended to next phases in development, e.g., detailed design, implementation, linking, run-time

Thank you . . . any questions?