



# More services – more processes

.... not quite the same thing!

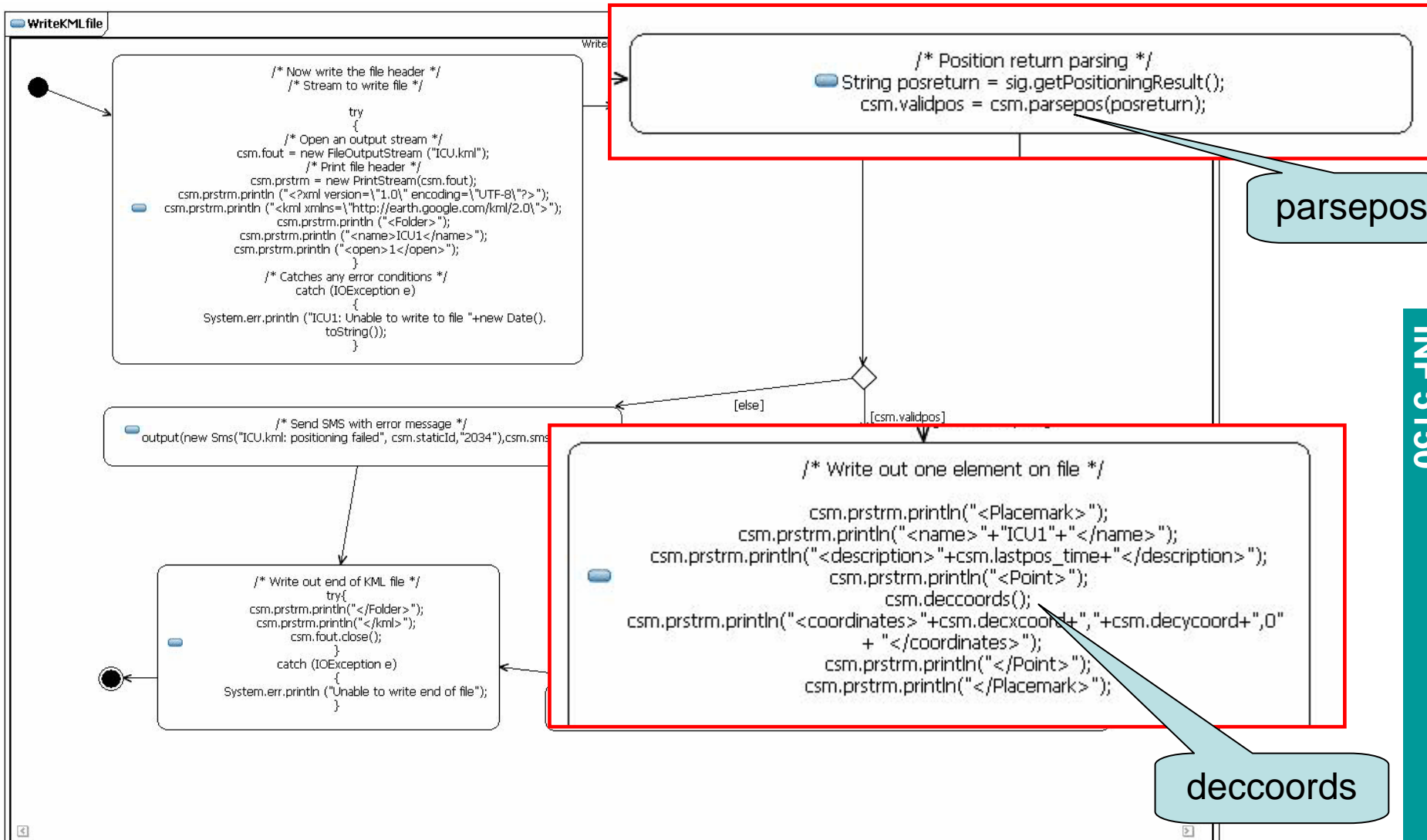
Version 071019



## About operations and methods

In order to keep the low-level java code away from the beautiful symbols of our UML models, we may want to separate some of the nitty, gritty details in out in chunks

# We will introduce operations/methods



# RSM coding rules for state machines (1)

- Trigger of transitions
  - Name of the transition
  - or Generate a SignalTrigger by rightclicking on transition
- Effect of transition
  - Name of effect
  - or Use one Action within an Activity diagram (forget flow lines etc.) created when doubleclicking the effect icon.
- Inside the effect
  - JavaFrame statements
  - or Branch by using Choice points
    - outgoing transitions from a choice point should have a guard (predicate condition for this piece of the transition)

## RSM coding rules for state machines (2)

- output (Signal, Port, csm)
  - sends a signal through the local port.
  - typically the signal is like "new S(parm1, parm2)"
  - typically the port is like "csm.toSomewhere"
  - "csm" is like a keyword meaning "current state machine"
- To read from the consumed signal, use "sig"
  - sig has been cast to the right type (normally)
  - Example: "sig.parm1" when sig is consumed as object of class S
- UML defer
  - to add a **deferrable trigger**, make sure the trigger to be deferred has a signaltrigger element in the state machine
  - right click the state > Properties > DeferrableTrigger and add the appropriate signaltrigger.
  - But you will not see the **defer** in the diagram – only in the model

# UML distinguish between operation and method

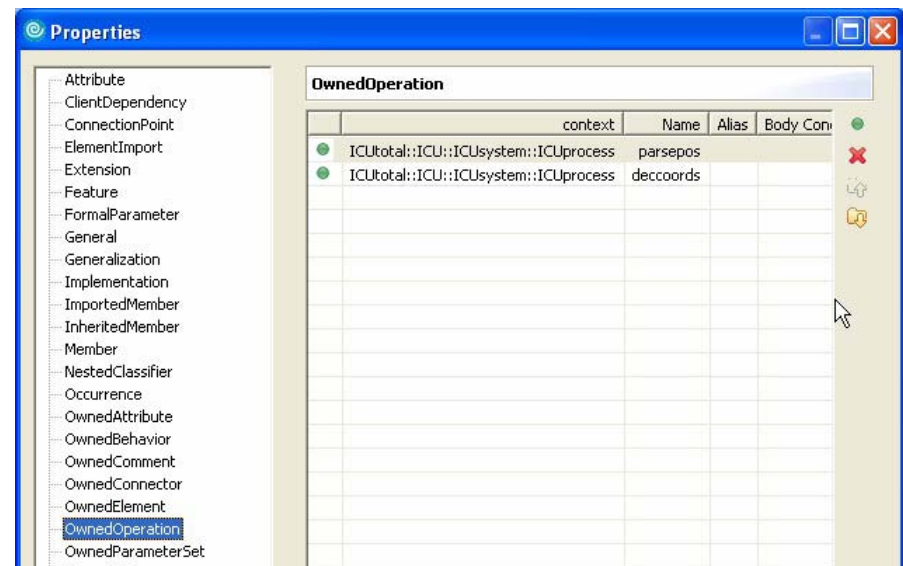
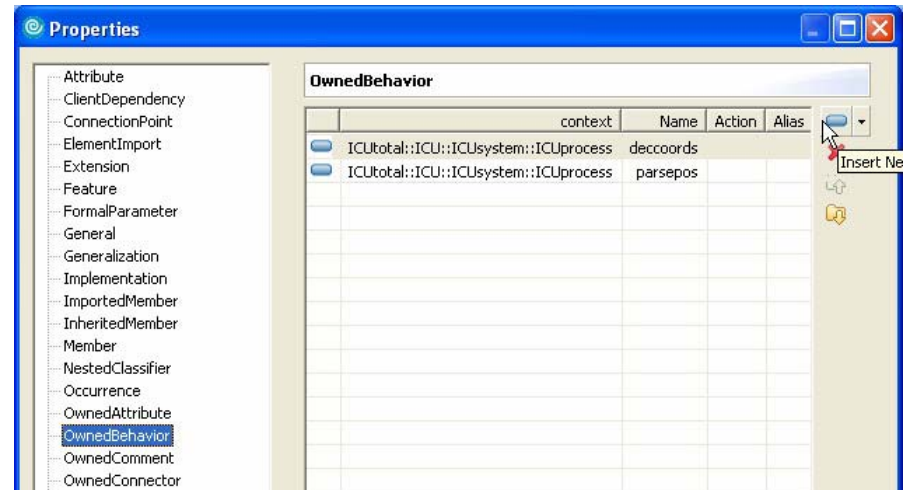
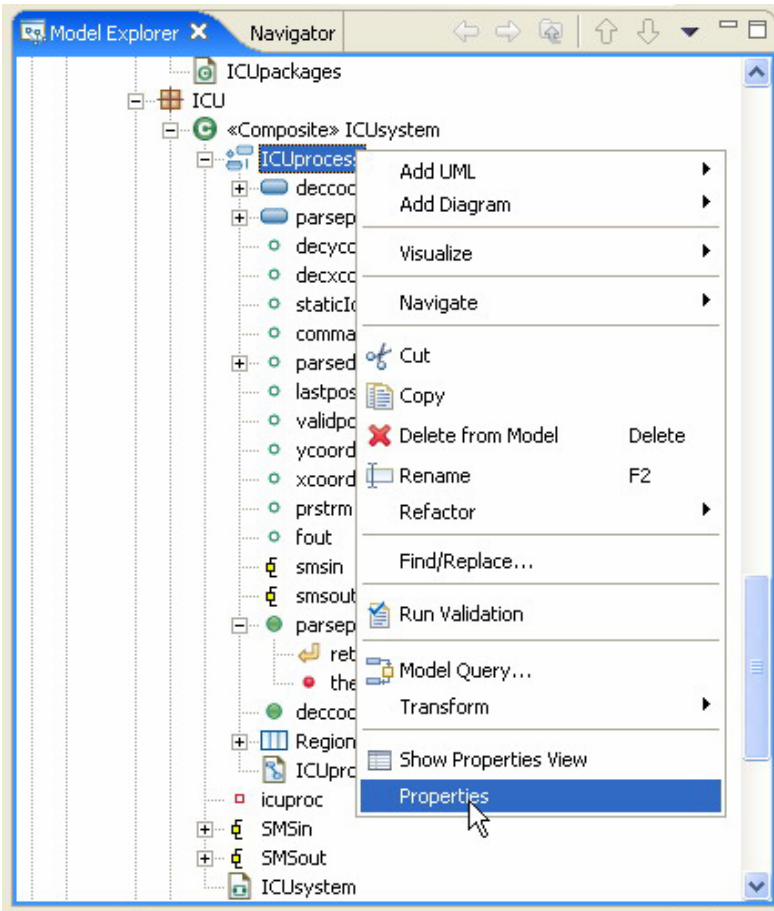
The screenshot shows the UML Model Explorer and Properties windows. The Model Explorer displays a package hierarchy: ICUpackages > ICU > «Composite» ICUsystem > ICUprocess. Under ICUprocess, several elements are listed, including 'parsepos' (a method) and 'parsepos()' (an operation). The Properties window is open to the 'Method' tab, showing a table with the following data:

context	Name	Action	Alias	At	...
ICUtotal::ICU::ICUsystem::ICUprocess	parsepos				

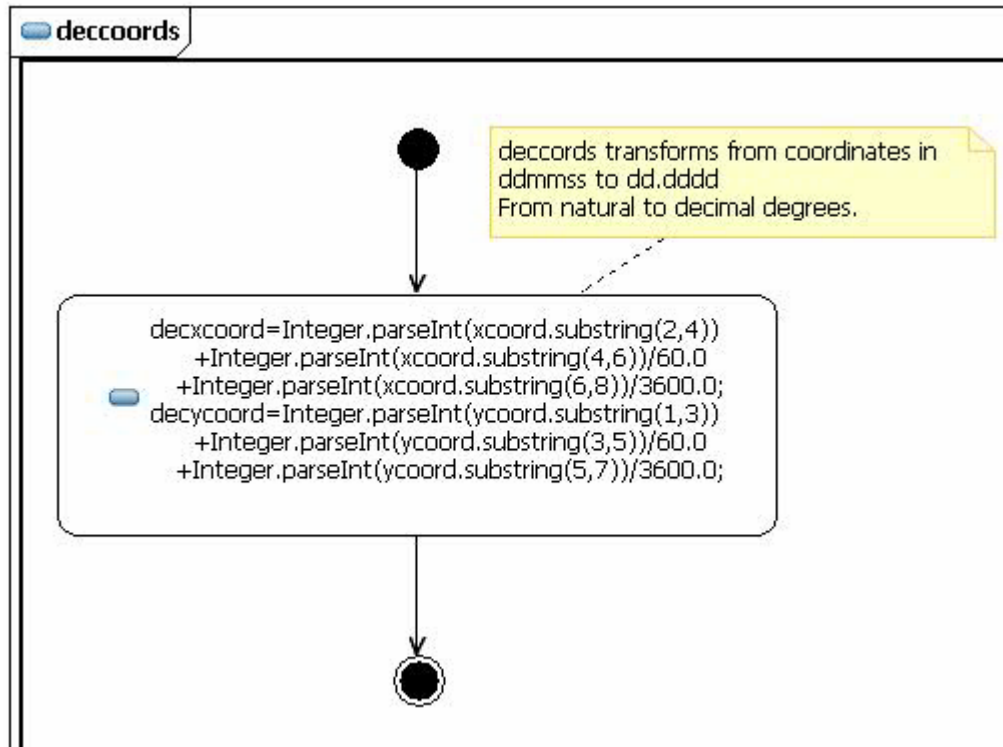
Two callouts highlight the elements in the Model Explorer:

- A callout pointing to the 'parsepos' element (a blue cylinder icon) says "parsepos – the method".
- A callout pointing to the 'parsepos()' element (a green circle icon) says "parsepos – the operation".

# How to make operations and activities



# The method definition is an action diagram







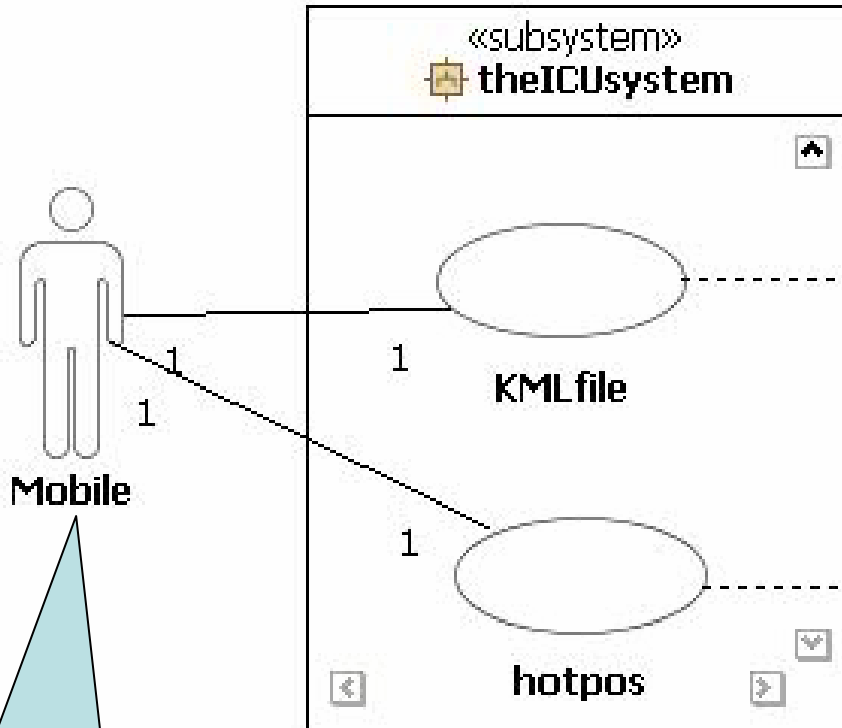
## More than one service

We introduce our second service

*hotpos*

where it may be useful to apply methods  
ie. define concepts only one place

# Adding a new service

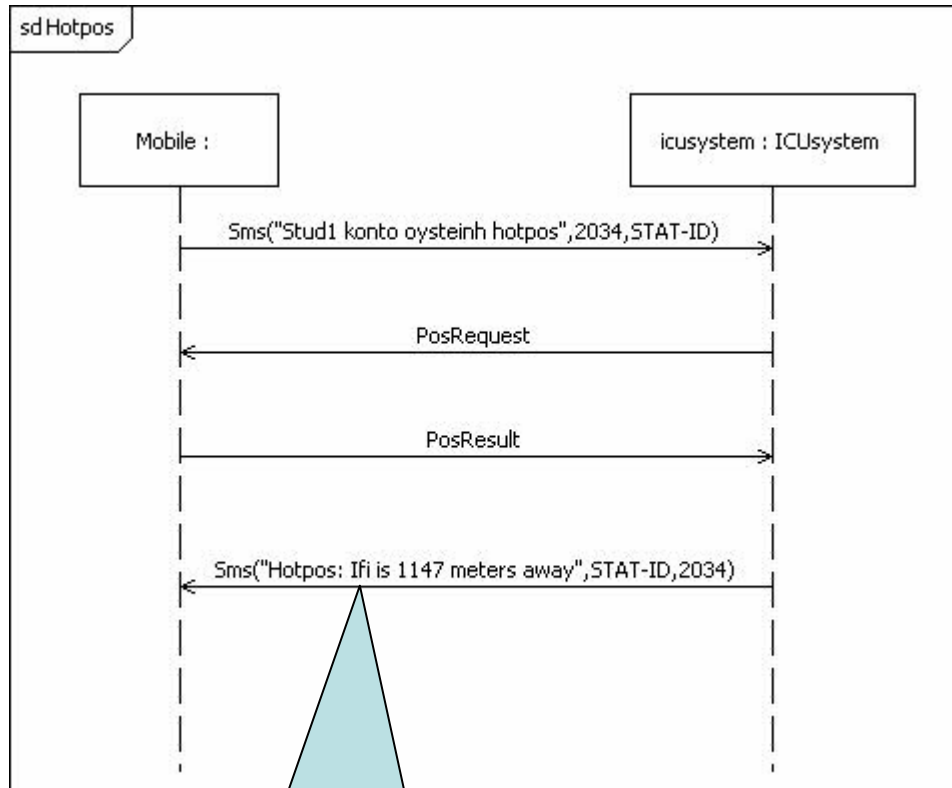


Stud1 konto oystein h KML  
Write out a .kml file to be read by GoogleEarth to place Mobile on the map

Stud1 konto oystein h hotpos  
Send back an SMS with info on where the user is relative to some hotspots.

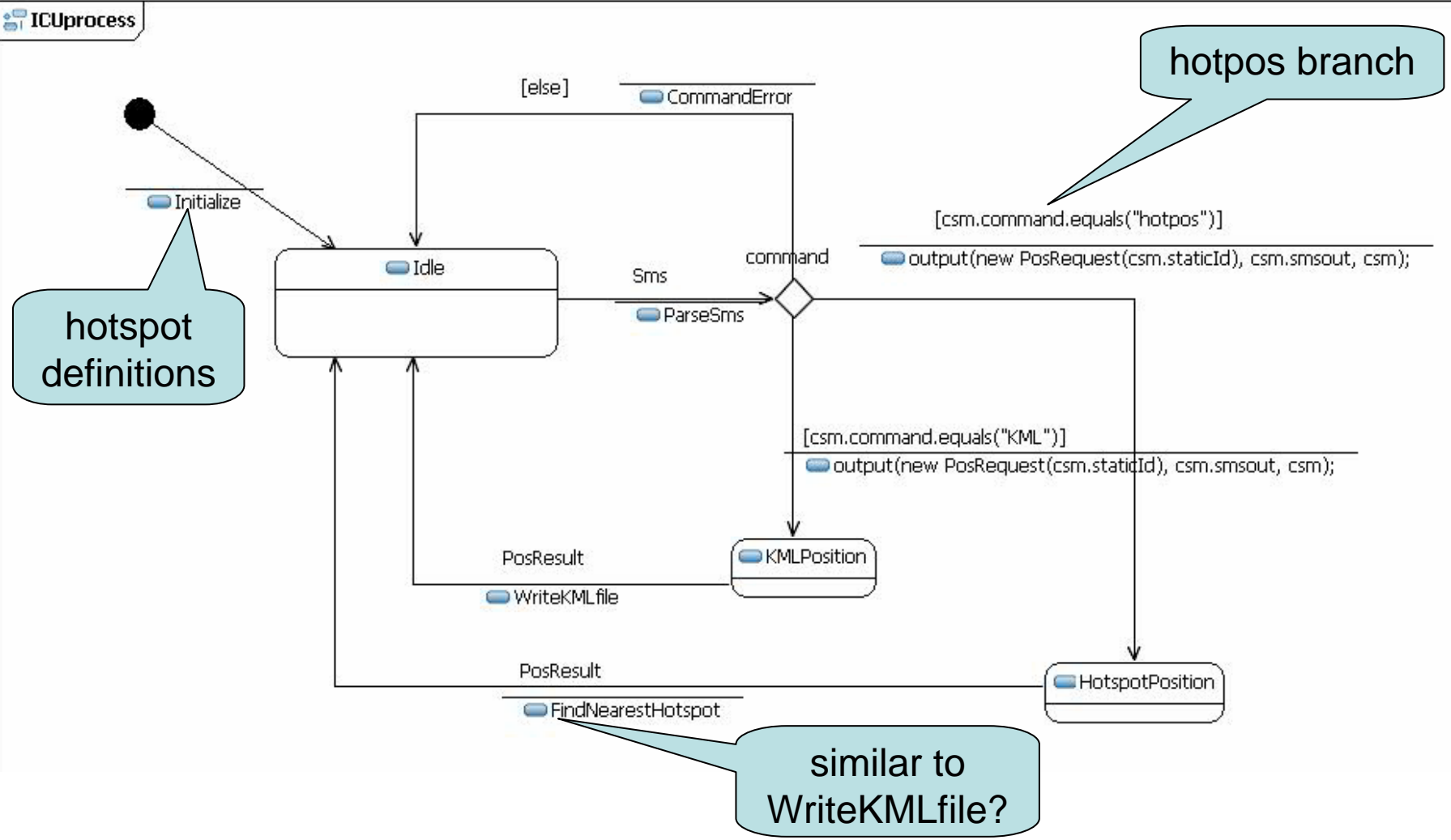
only one user at a time

# Hotpos described by a sequence diagram



need to know where Ifi is

# The modified ICUprocess

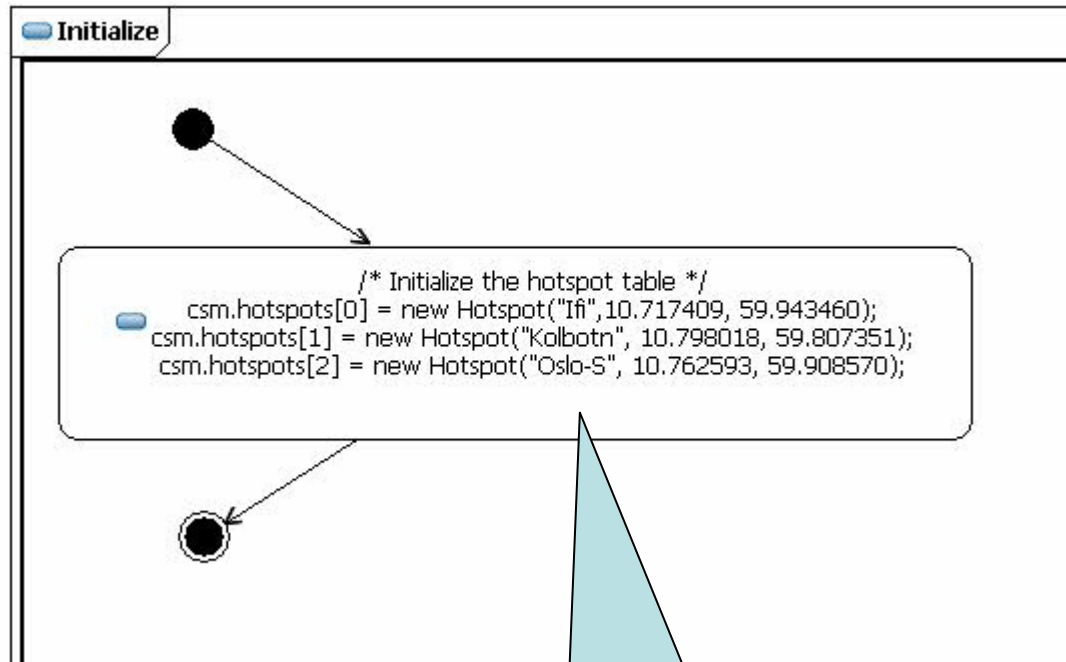




## Buzz 1: Why limiting to one user?

- Make up pairs with one person just beside you
- Discuss for 3 minutes why we have restricted the system to consider only one user at the time

# Hardcoding the hotspots



Feel free to add your own hotspots. Remember to change the size of the array

# FindNearestHotspot

a little robustness,  
but it does not  
cover no return

FindNearestHotspot

FindNearestHotspot

```

/* Parse the position return */
String posreturn = sig.getPositioningResult();
csm.validpos = csm.parsepos(posreturn);
    
```



```

/* Send SMS with error message */
output(new Sms("Hotpos: positioning failed", csm.staticId, "2034"), csm.smsout, csm);
    
```

reusing the  
operation/method

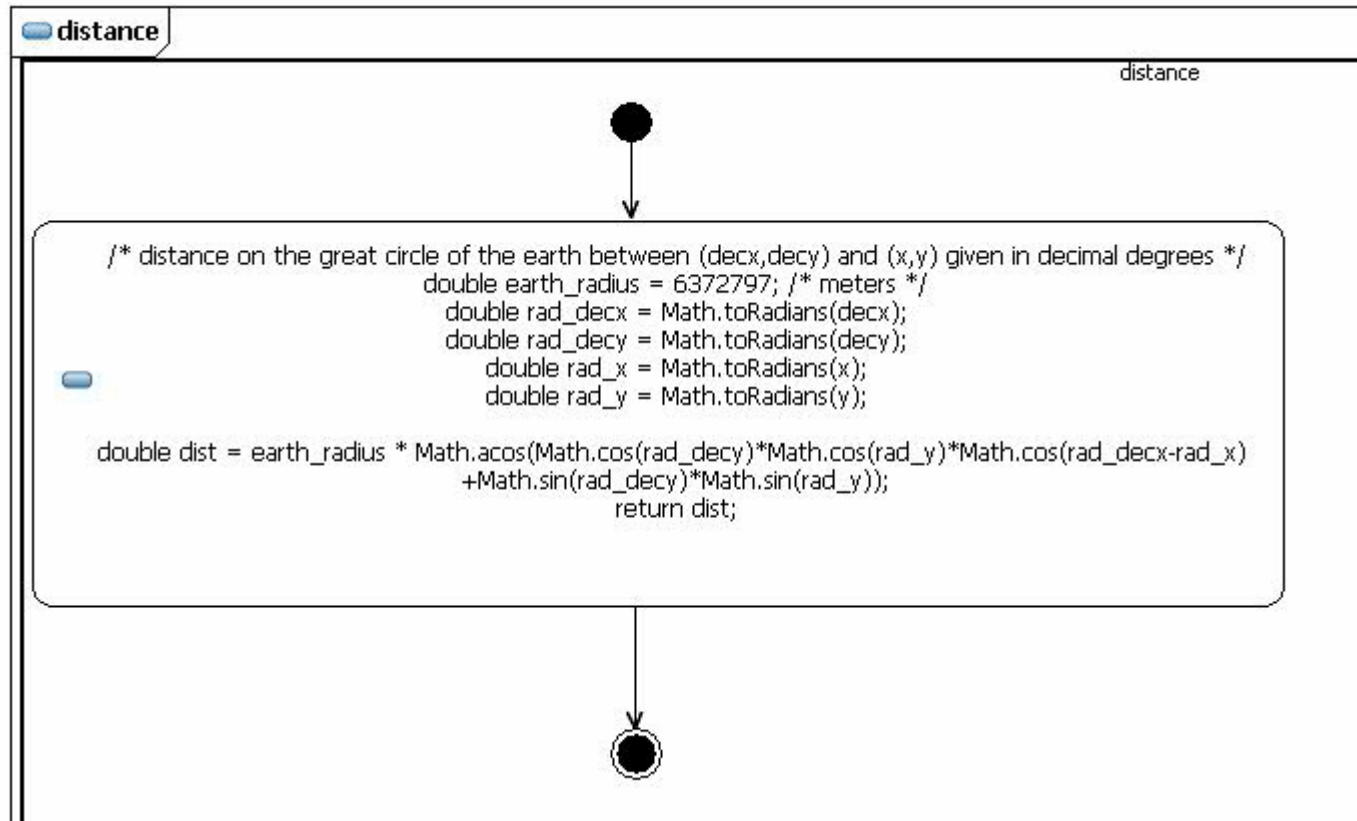
```

/* convert to decimal degrees */
csm.deccoords();
/* Find nearest hotspot */
int no_hotsp = csm.hotspots.length;
double smallestdist= 1000000;
double curdist;
int index_smallest=-1;
for (int h = 0; h<no_hotsp; h++) {
    curdist = csm.hotspots[h].distance(csm.decxcoord, csm.decycoord);
    if (curdist < smallestdist) {smallestdist = curdist; index_smallest=h;};
};
/* here we have found the smallest distance */
/* send the result back as an SMS */
output(new Sms("Hotpos: "+csm.hotspots[index_smallest].hotname+" is "+(int)smallestdist+ " meters away", csm.staticId, "2034"), csm.smsout, csm);
    
```

Hotspot.distance()



# Hotspot.distance()



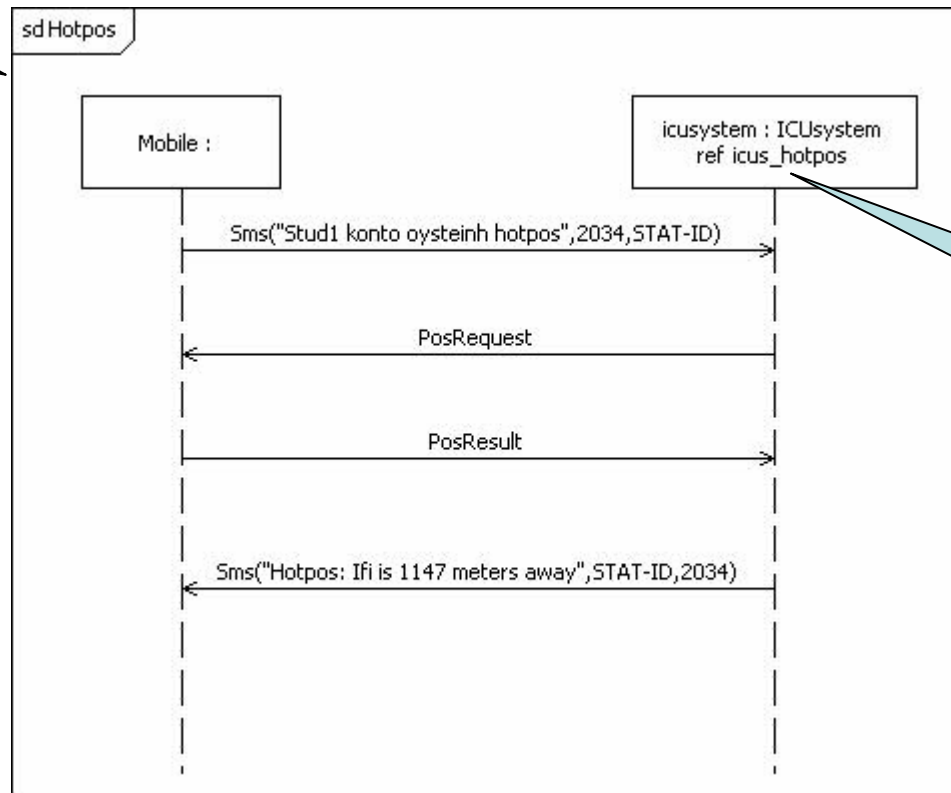


## Separation of concerns

- We want to separate different concerns of the ICU system through using separate state machines that communicate
- The architecture of the ICUSystem will evolve
- One process controls
  - the handling of SMSes
  - and the production of the KML file
- One process controls the handling of the data
  - which are still going to be hardcoded (for now)
- These processes communicate with signals that we define ourselves

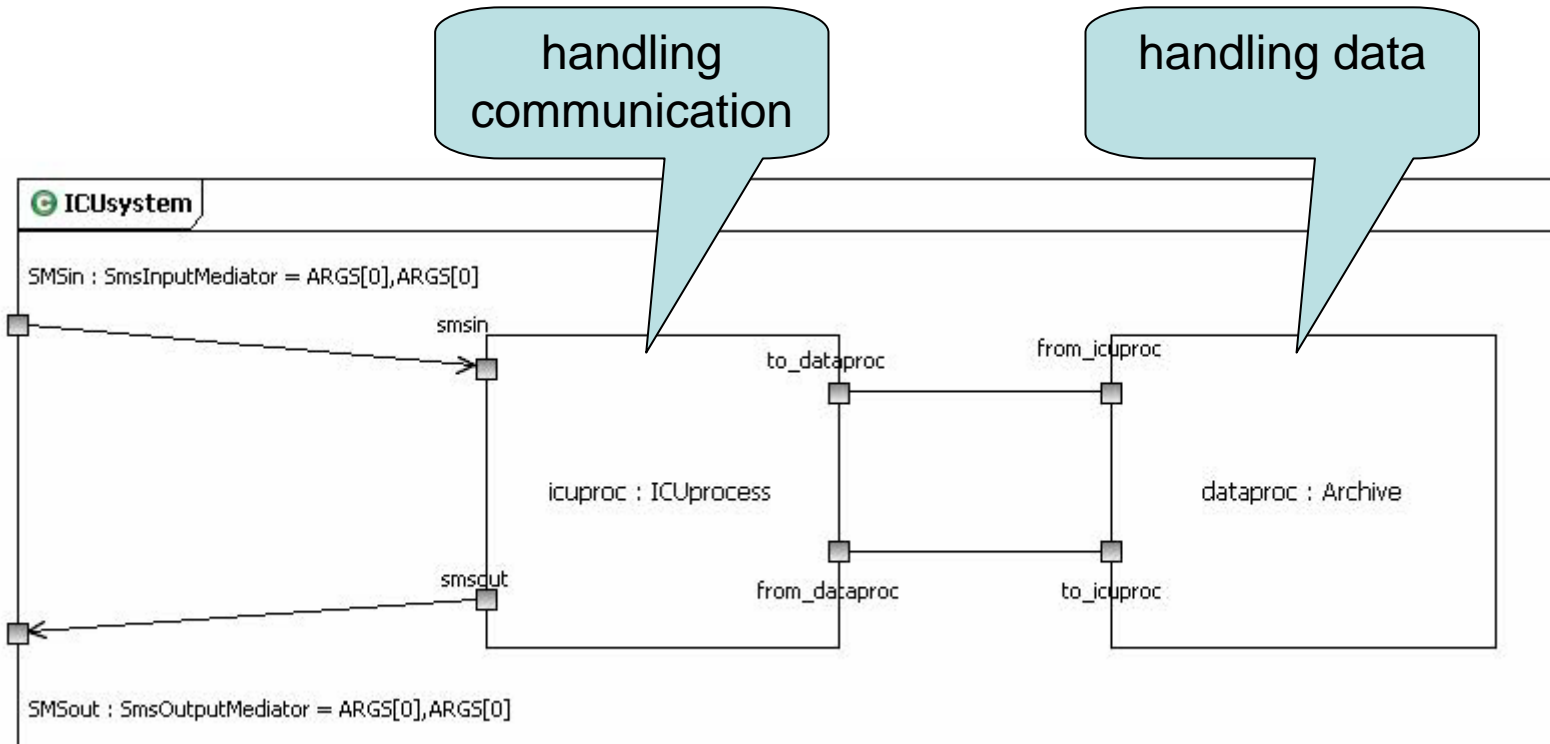
# Hotpos service – as seen from the context

very similar  
to ICU2!



what is this?

# Inside the ICUsystem

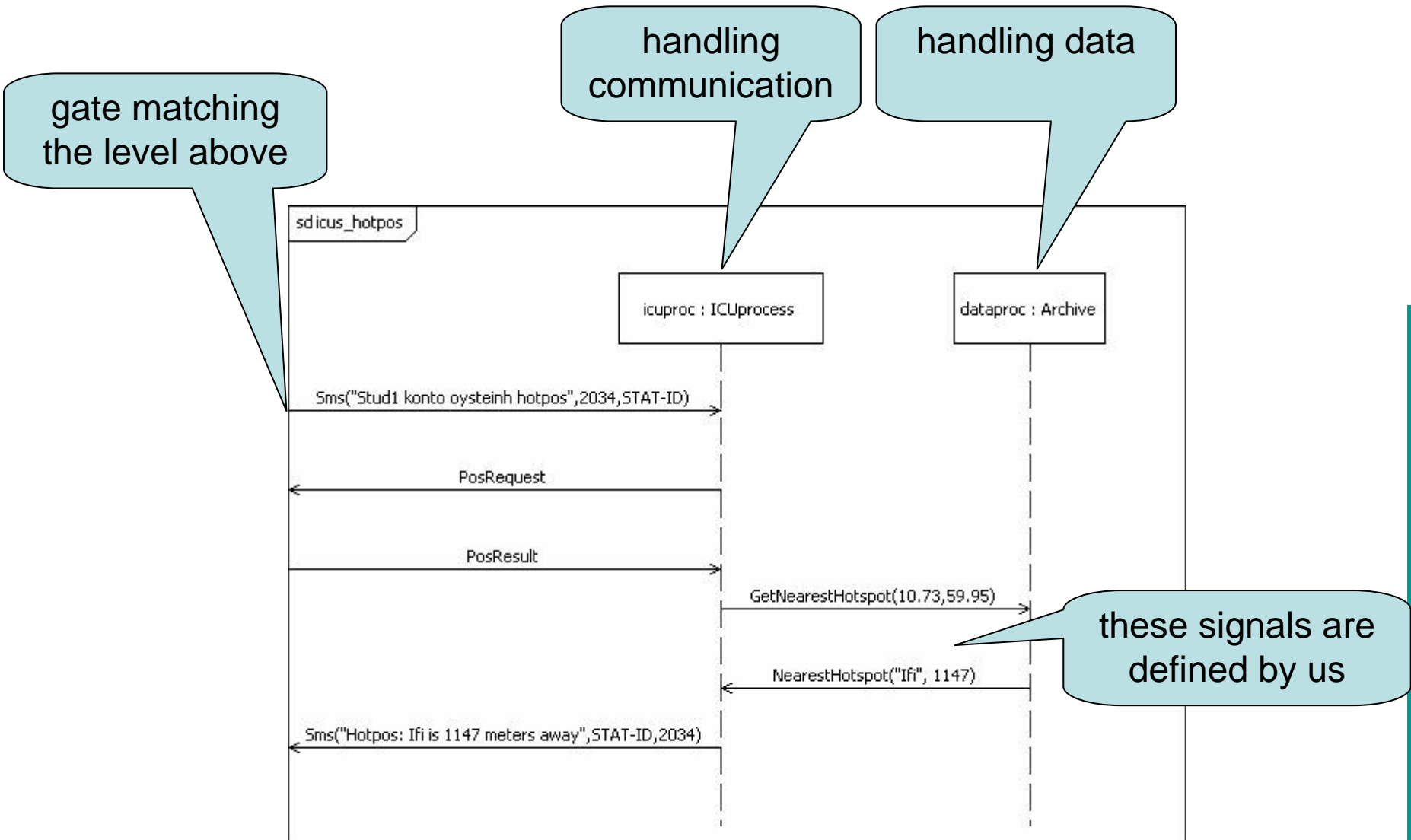


# Decomposing the ICUsystem

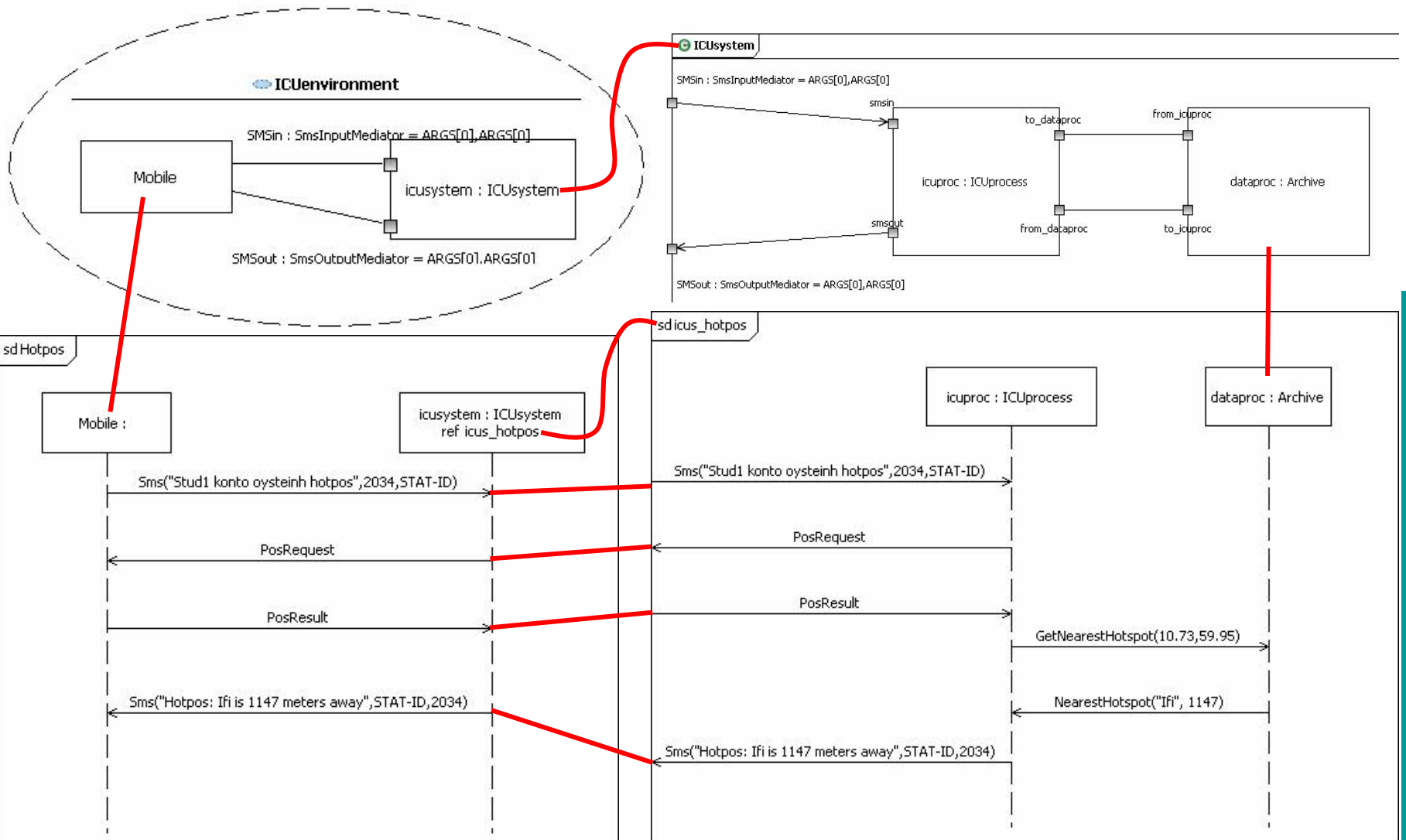
The screenshot shows a modeling tool interface with a project navigator on the left and a sequence diagram in the center. The project navigator lists a hierarchy of packages, including ICUcontext, ICUenvironment, ICU, ICUpackages, ICU, and ICUclasses. A red arrow points from the 'sd Hotpos' package in the navigator to the sequence diagram. Another red arrow points from the 'icuproc' package to the 'icuproc' lifeline in the diagram. A callout box with the text 'double click' points to the 'icuproc' lifeline. The sequence diagram shows a lifeline for 'Mobile :' and a lifeline for 'icuproc : icuproc'. The diagram contains the following messages:

- Mobile : sends Sms("Stud1 konto oystein hotpos",2034,STAT-ID) to icuproc
- icuproc returns PosRequest to Mobile
- icuproc sends PosResult to Mobile
- Mobile sends Sms("Hotpos: Ifi is 1147 meters away",STAT-ID,2034) to icuproc

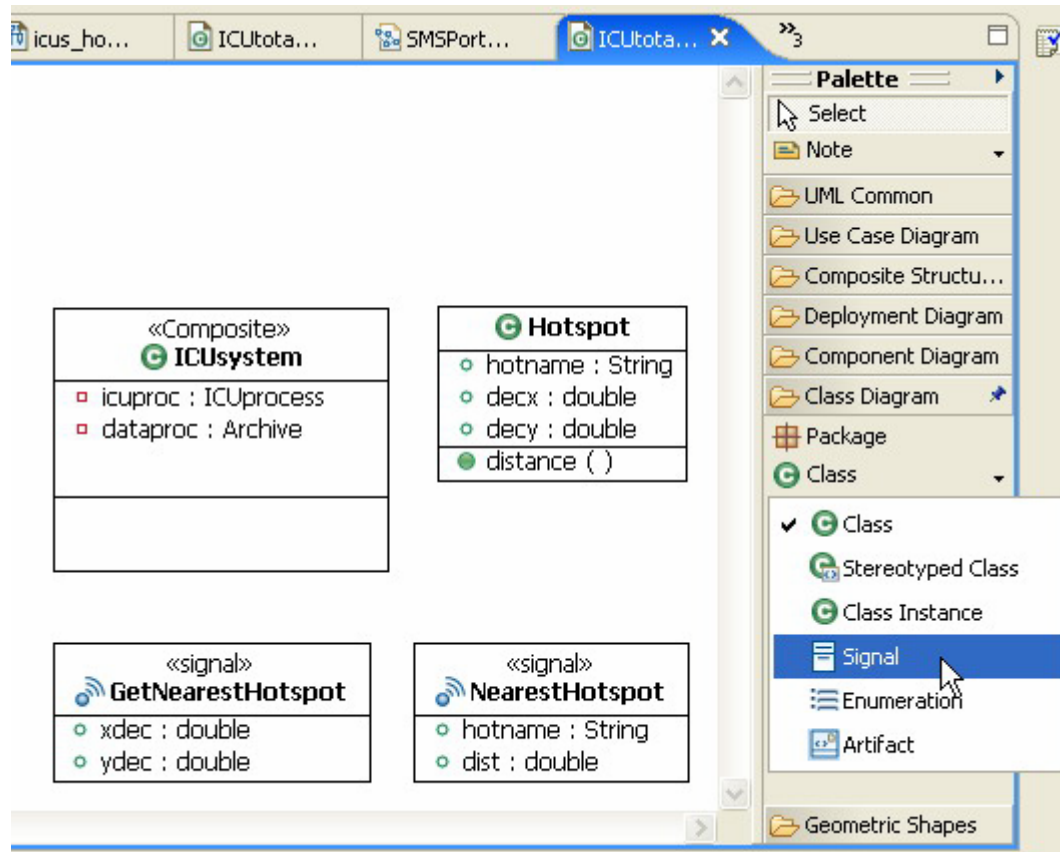
# The behavior inside ICUsystem



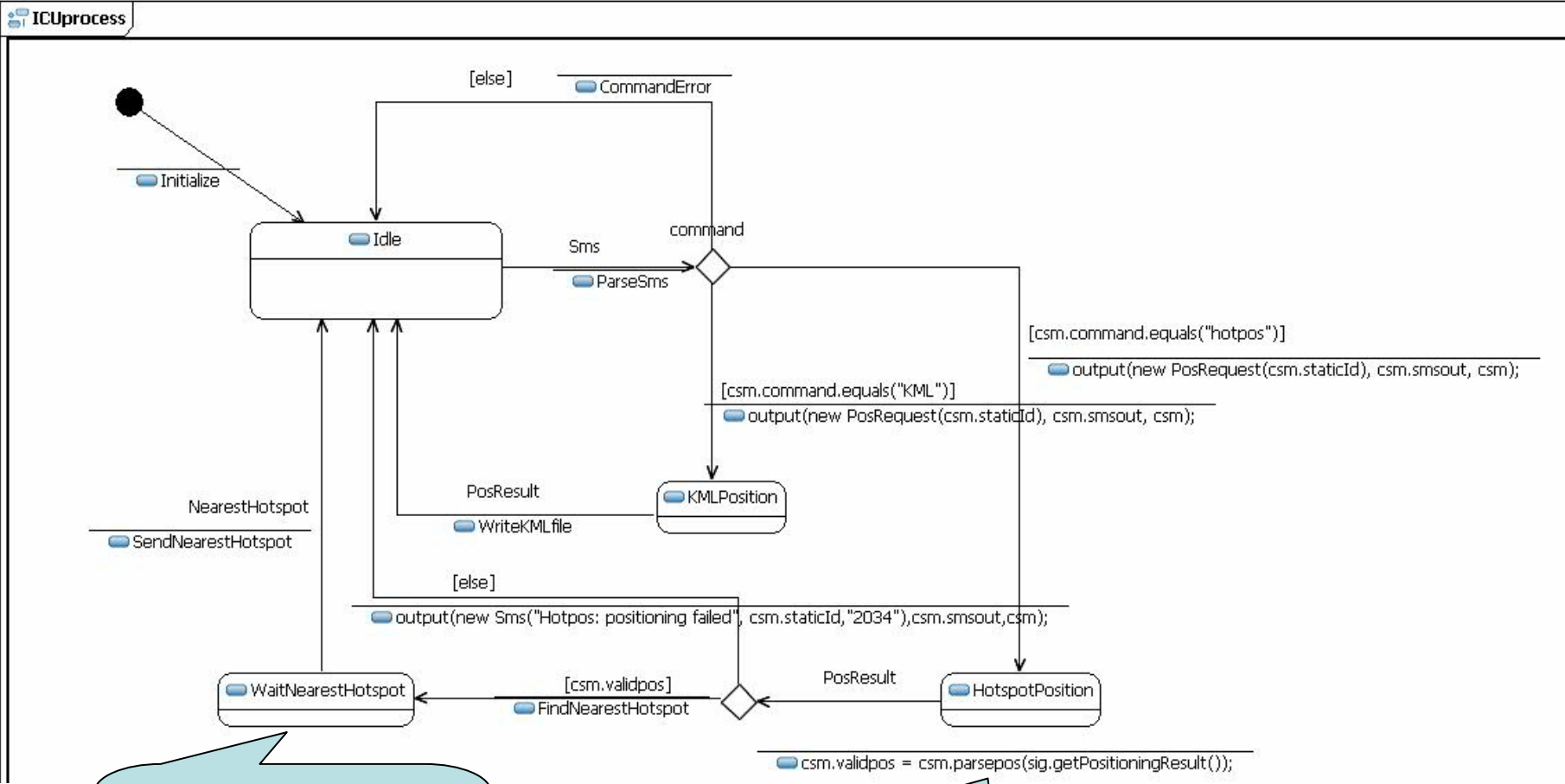
# The essence of decomposition



# The classes and signals



# ICUprocess revisited (when intro Archive)

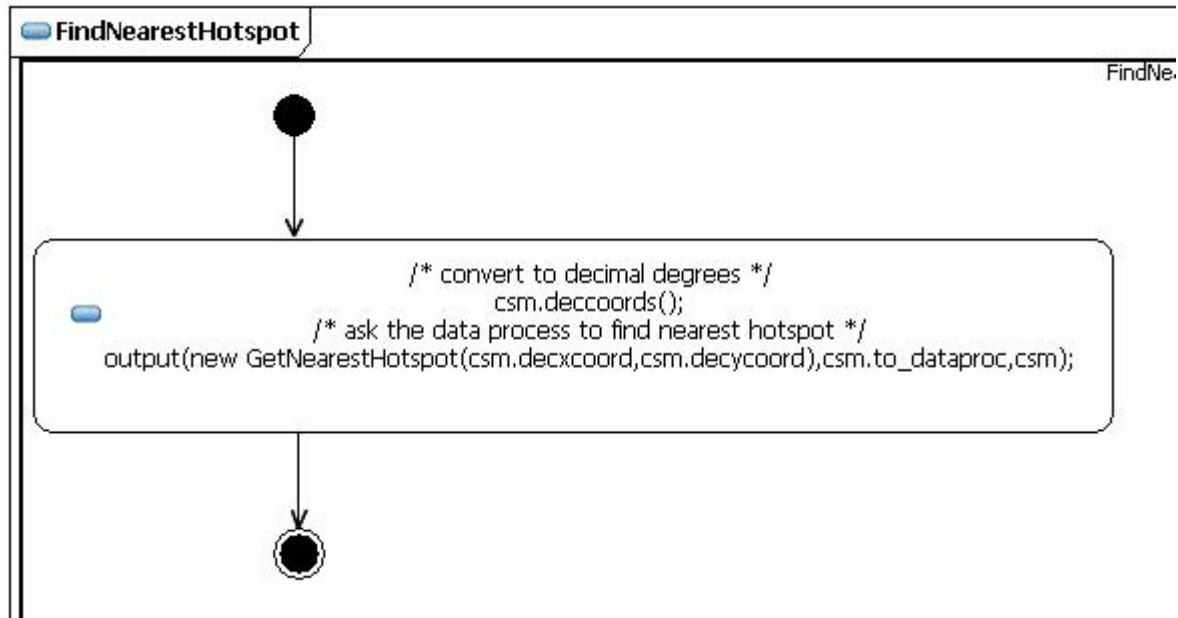


Here the data process does the calculations

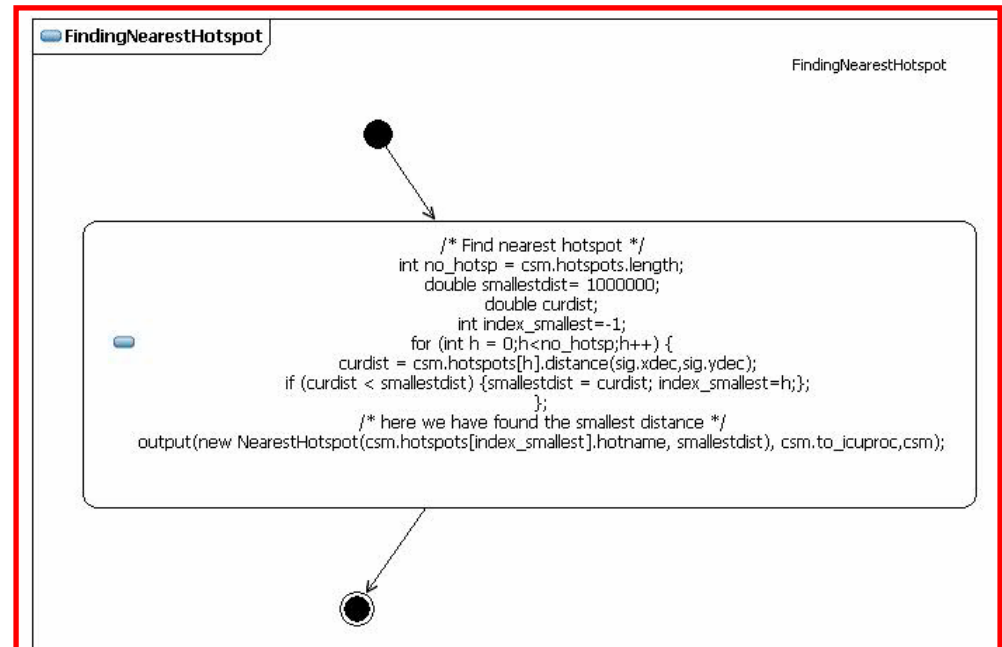
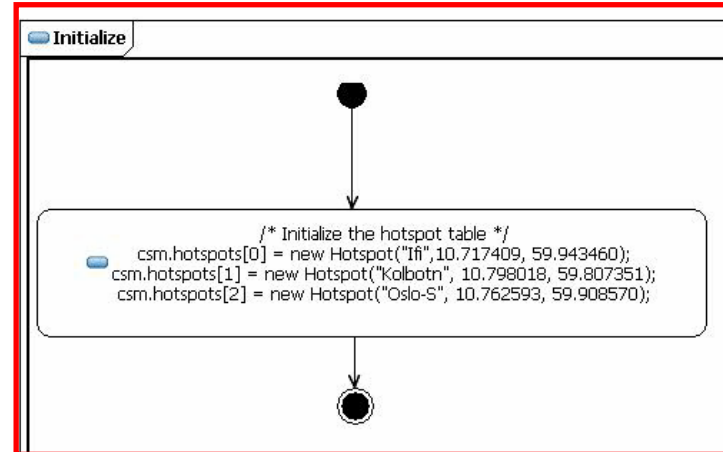
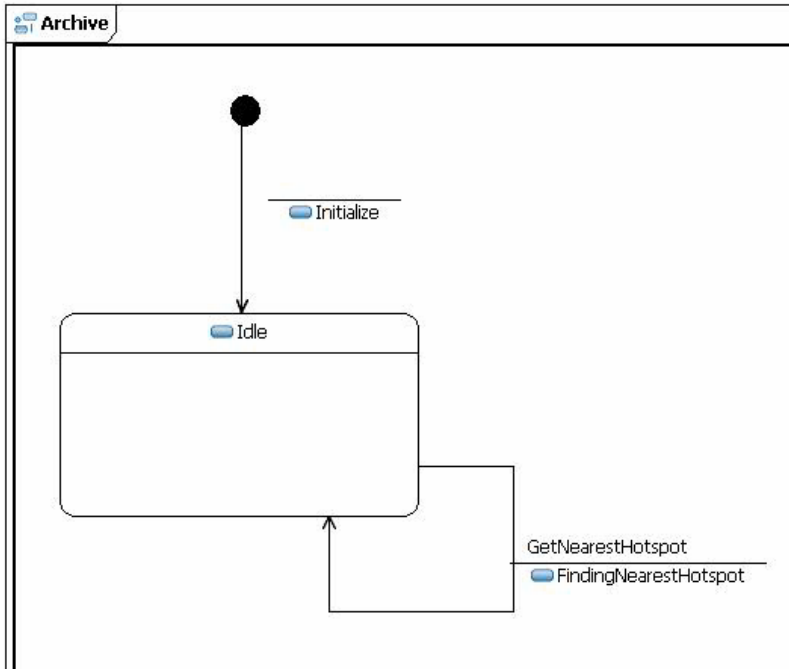
FindNearestHotspot has been split up



# FindNearestHotspot has become pure sending



# Archive – the data process





## Buzz 2: Why the Archive process?

- Pair up with another student
- Discuss 3 minutes what benefits there are with introducing the Archive process



## Why the separate data process?

- Isolate the work on the (semi-)persistent data
  - we shall later show how the handling of data can change without changing its interfaces
- Provide a simple critical region
  - this will be clearer later when we interface to a database system that works concurrently with our system
- The Archive process and the ICUprocess can be designed by different persons

# How to make the protocol with the Archive?

- Signals close to the application
  - this is what we have chosen
  - we want to branch on signals rather than on data
- Signals close to data
  - such as e.g. SQL
  - most important information will be in the parameters and branching will be on decision-nodes
- Do not worry about many signal types!

