



Agile modeling – for INF5150

Version 080926

ICU 0-1



Group formation for Oblig 2

- While Oblig 1 must be individually solved, Oblig 2 shall be achieved in a group of 3-5 persons
- Divide the group in
 - PhD students and those with a Master already
 - Those with INF2120
 - Those with special needs
 - The rest
- Everybody signs up their name on the blackboard in the appropriate column
- The lecturer will select the groups
 - and add those that are not present



ICU0 – your very first “I see you” system

surveillance at your fingertips,
first we only observe yourself



Agile modeling

- "agile"
 - = having a quick resourceful and adaptable character
- executable models!
- very stepwise approach
 - each step will have its specification and executable model
 - each step should be tested
- We shall use one example throughout the course
 - with many steps
 - intended to be mirrored by the project exercise model
- Every week a working program!



Manifesto for Agile Software Development

- We are uncovering better ways of developing software by doing it and helping others do it.
- Through this work we have come to value:
 - **Individuals and interactions** over processes and tools
 - **Working software** over comprehensive documentation
 - **Customer collaboration** over contract negotiation
 - **Responding to change** over following a plan
- That is, while there is value in the items on the right, we value the items on the left more.



Dialectic Software Development

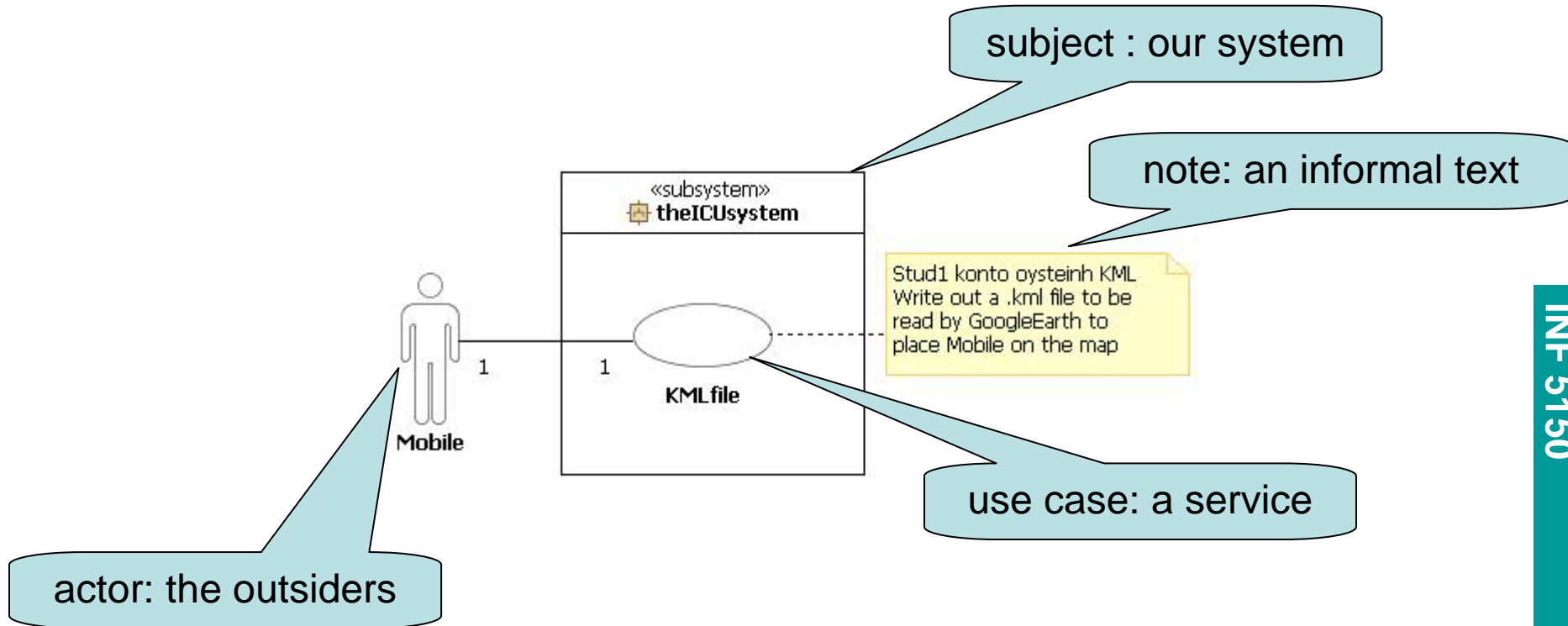
- Software Development is a process of learning
 - once you have totally understood the system you are building, it is done
- Learning is best achieved through conflict, not harmony
 - discussions reveal problematic points
 - silence hides critical errors
- By applying different perspectives to the system to be designed
 - inconsistencies may appear
 - and they must be harmonized
- Inconsistencies are not always errors!
 - difference of opinion
 - difference of understanding
 - misunderstanding each other
 - a result of partial knowledge
- Reliable systems are those that have already met challenges



Buzzzzz 1: Agility

- Join your project group – this is its first assignment!
- Give 3 reasons for why agile modeling/programming is a good approach
- Give 3 possible problems for an agile approach
- Give each pro and each con a short name

UML Use Cases – very very simple



Use cases in a separate package

The screenshot shows the Papyrus UML IDE interface. The main workspace displays a Use Case diagram for the package 'theICUSystem'. An actor named 'Mobile' is connected to a use case named 'KML file'. A note is attached to the 'KML file' use case with the text: 'Stud1 konto oystein KML Write out a .kml file to be read by GoogleEarth to place Mobile on the map'. The left sidebar shows the Navigator and Outline views, with the 'ICU' package structure visible. The right sidebar shows the Palette with various UML elements and relationships. The bottom status bar indicates the current diagram is a 'Use Case diagram of ICUusecases'.

UML Sequence Diagrams: a more precise way

The screenshot shows the Papyrus UML editor interface. The main diagram is a sequence diagram titled "sd KMLfile" with two lifelines: "mob[STAT-ID] : Mobile" and "icysystem : ICUsystem". The diagram contains several messages: a signal message "«signal» Sms('Stud1 konto oystein KML', 2034, STAT-ID)", a signal message "«signal» PosRequest", a signal message "«signal» PosResult", and a signal message "«signal» Sms('KML has been generated', STAT-ID, 2034)". A note "(Write out on KML file)" is attached to the diagram. The Properties window at the bottom shows the details for the selected "PosRequest" message.

General	Property	Value
Profile	eAnnotations	[]
Profile	interaction	KMLfile
Comments	nameExpression	
Appearance	namespace	KMLfile
Advanced	ownedComment	[]
Advanced	ownedElement	[]
Advanced	owner	KMLfile
Advanced	receiveEvent	Receive2
Advanced	sendEvent	Send2
Advanced	signature	PosRequest

Interaction

Sequence diagram

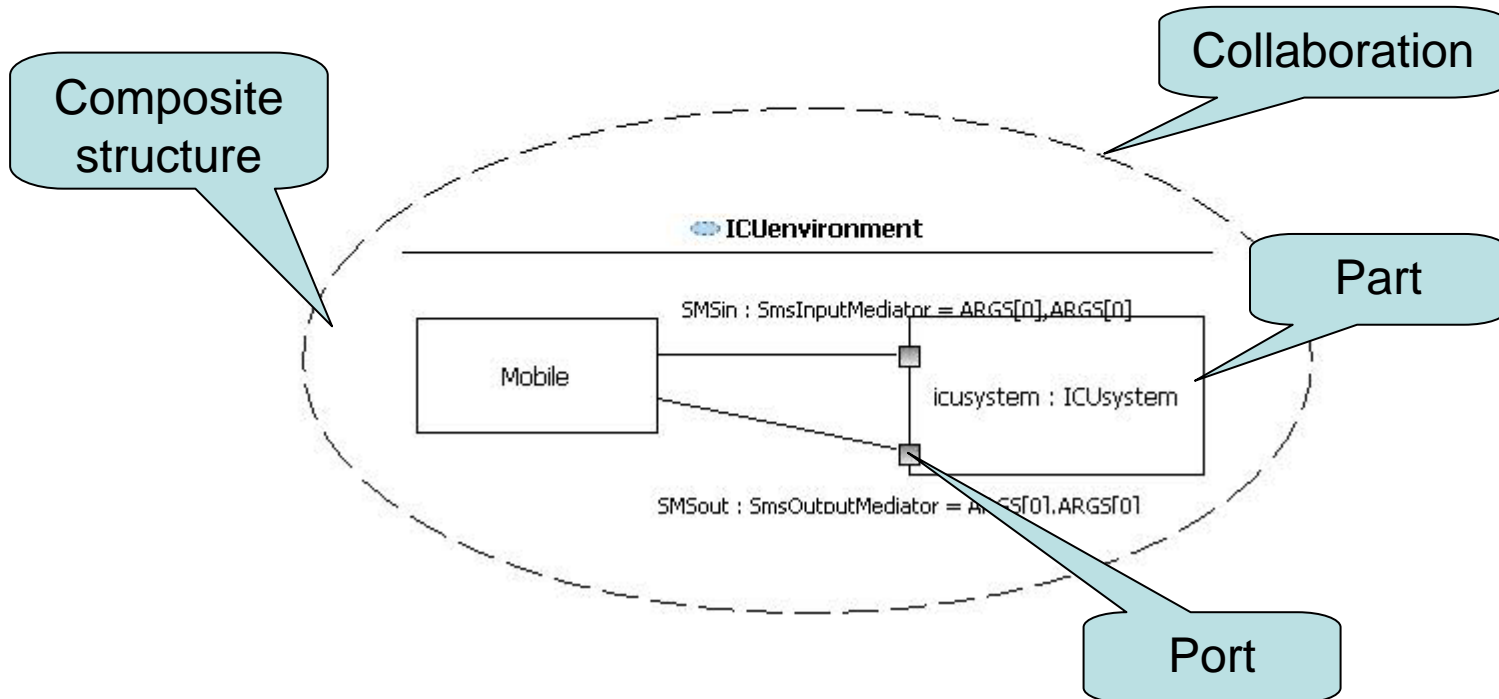
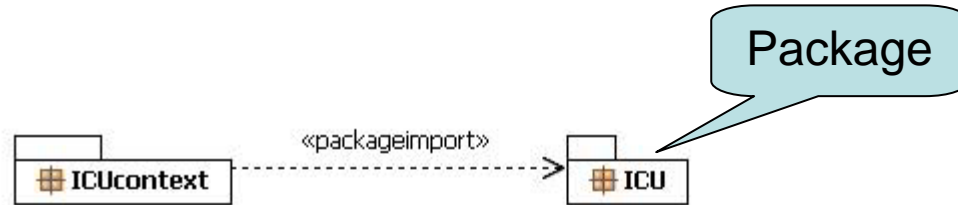
Lifeline

Message

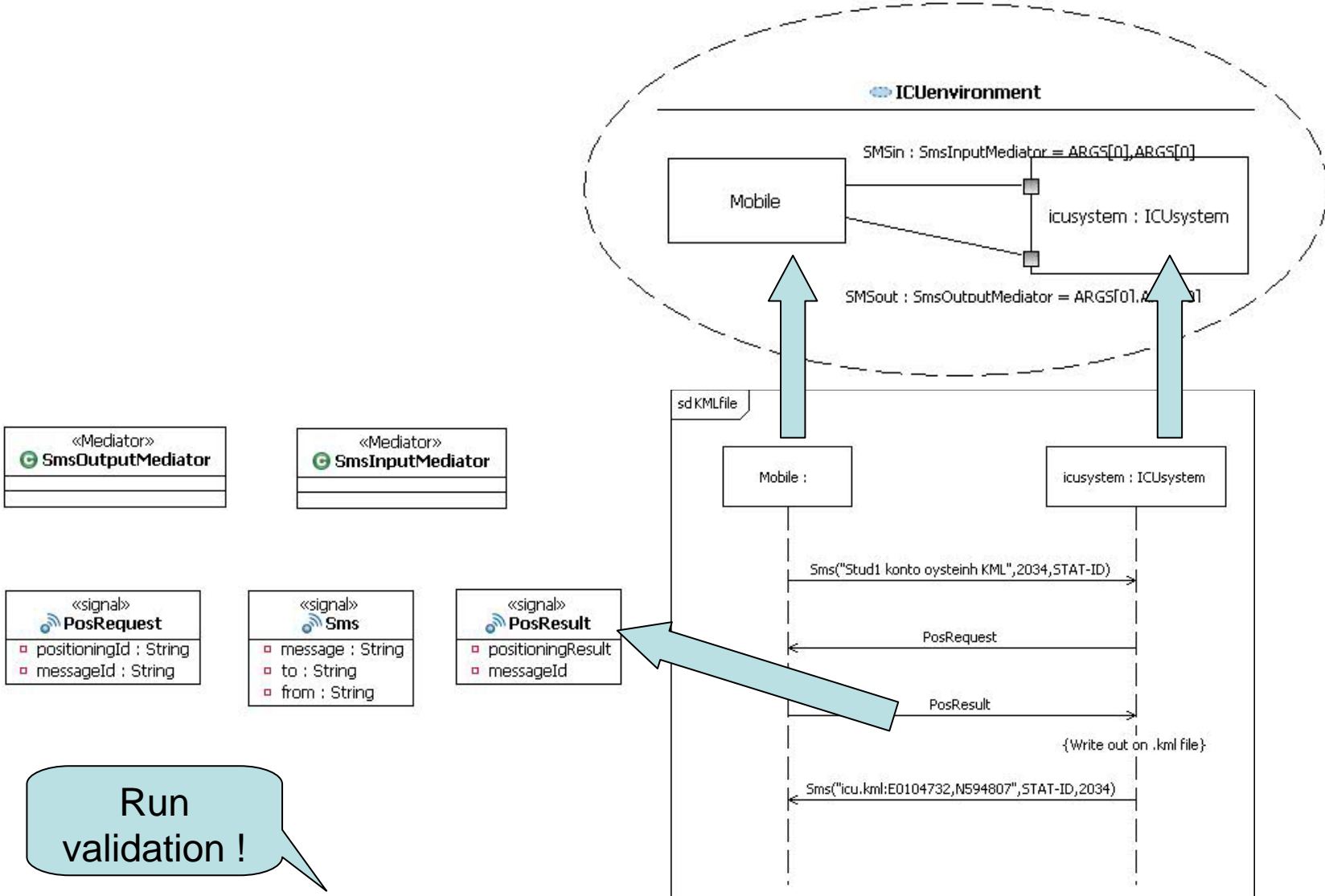
State inv.

Signature

Packages, Collaboration, Composite Structure

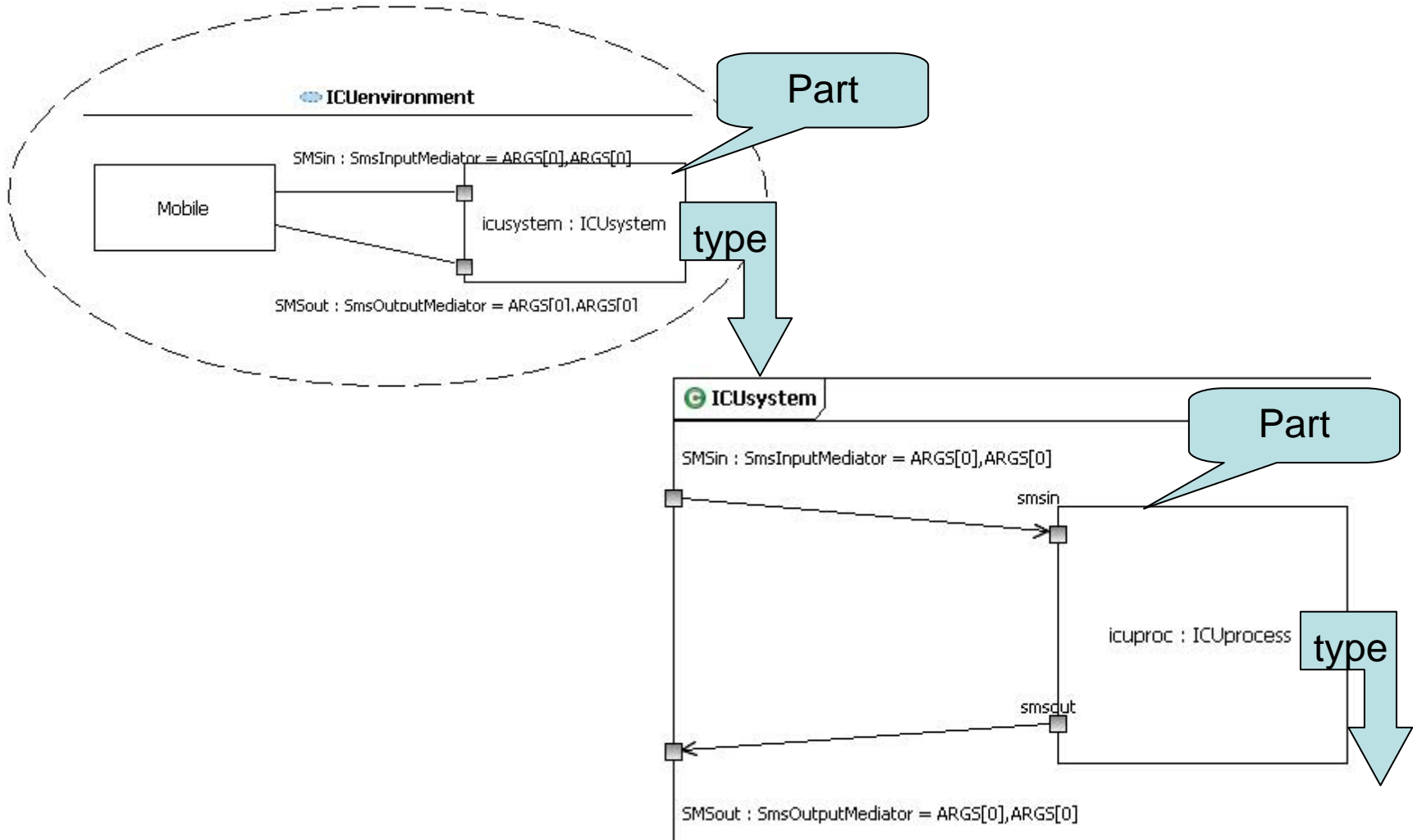


Model-time Consistency!

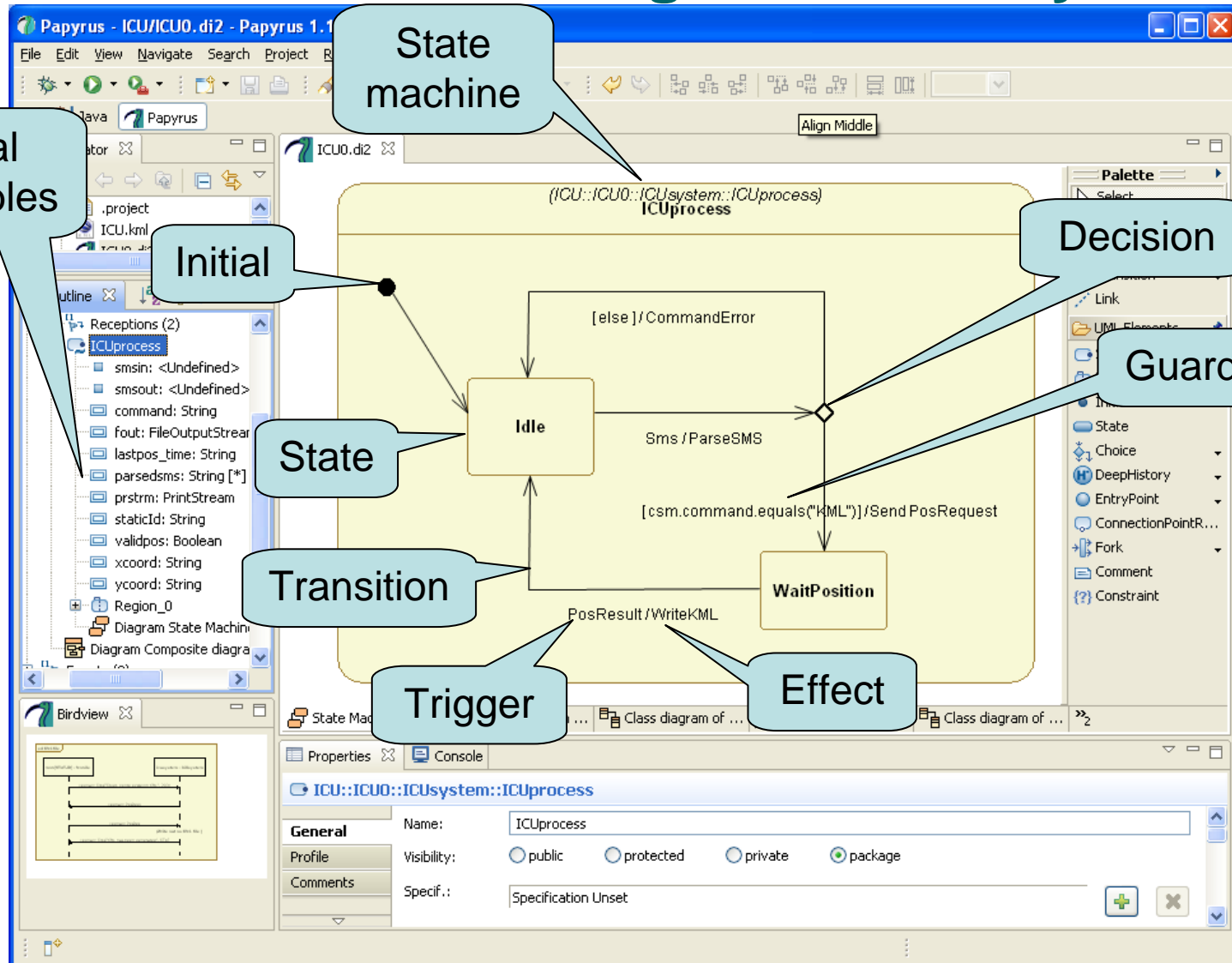


Run validation !

Structure hierarchy



A State Machine defining the whole system



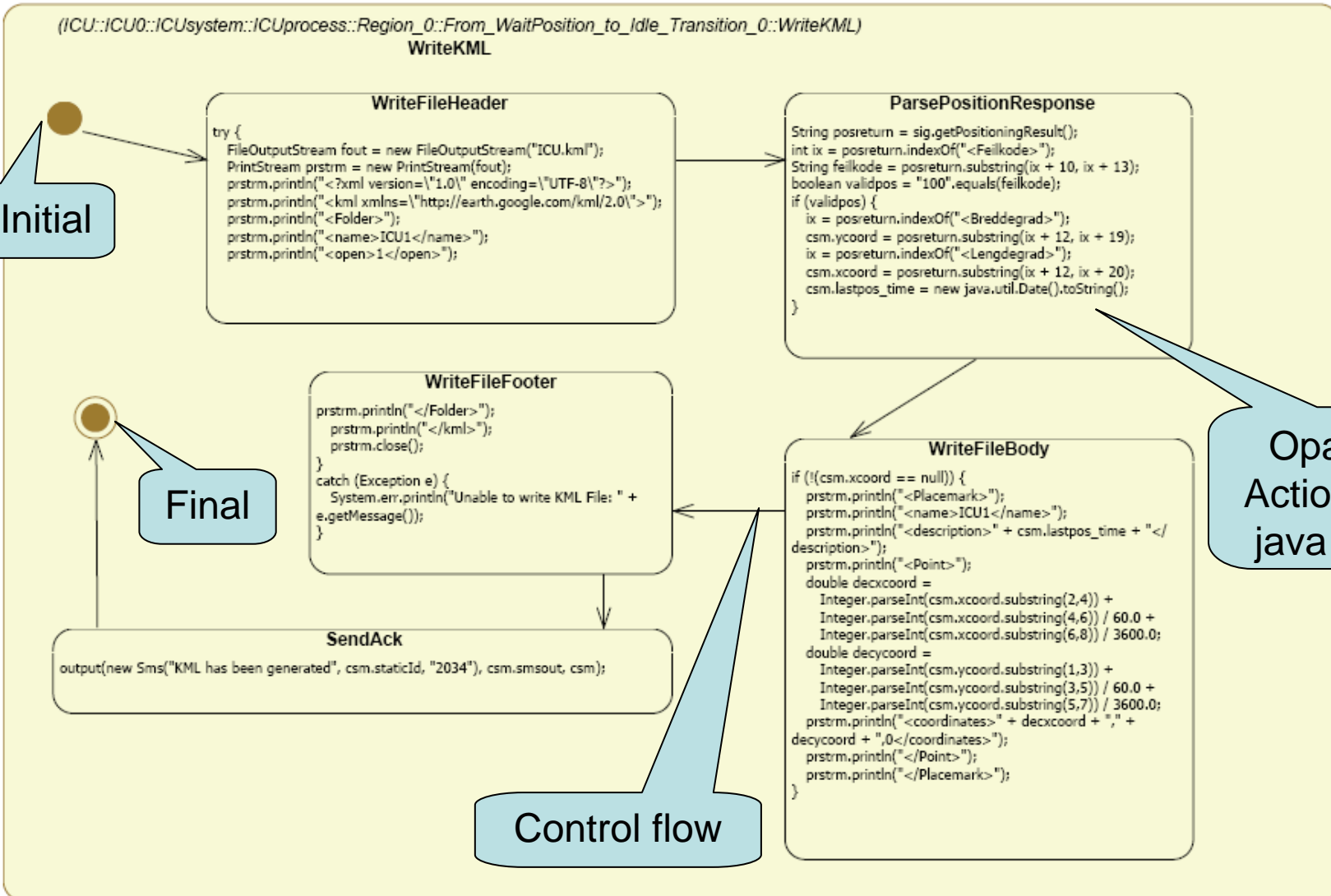
Papyrus coding for state machines

- Trigger of transitions
 - Trigger relates to a visible Signal
 - Create a Reception in the enclosing class
 - Associate the Reception to the desired Signal
 - In the transition, create a trigger (+) to be a SignalEvent
 - Choose the appropriate Reception from the menu in the dialogue box
- Defer
 - Deferrable Triggers are defined in the State where they are deferred
- Effect of transition
 - In the transition add an effect (+)
 - In the simplest cases this can be just an OpaqueBehavior
 - Otherwise you may use an Activity
 - and an Activity Diagram will be automatically created

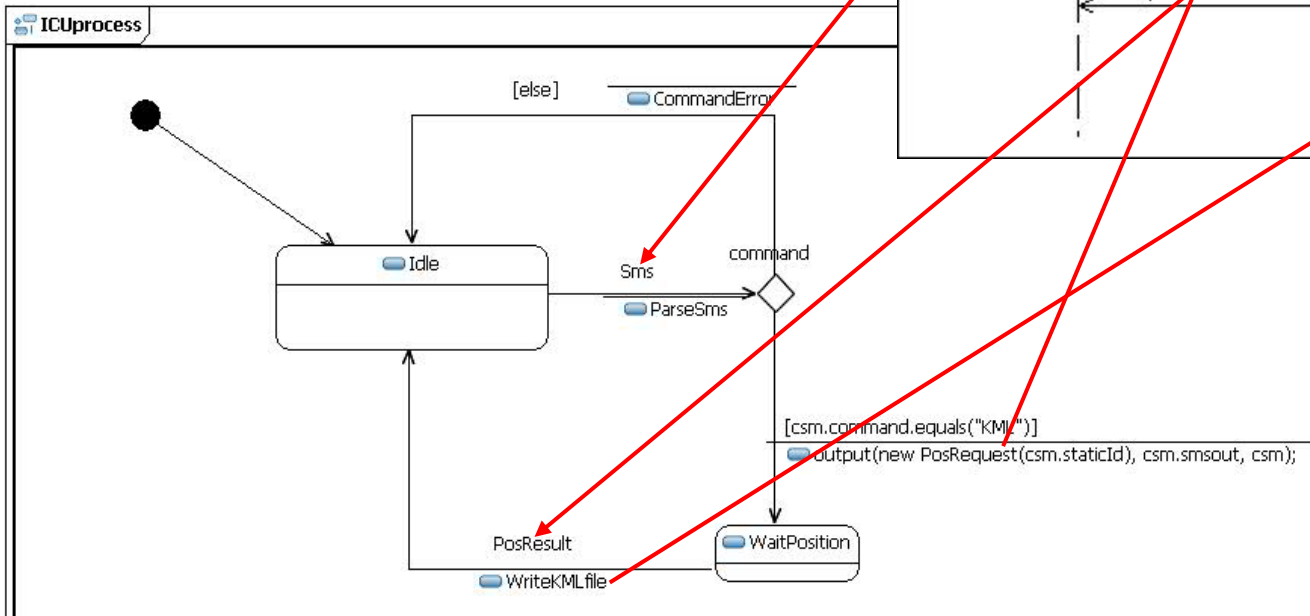
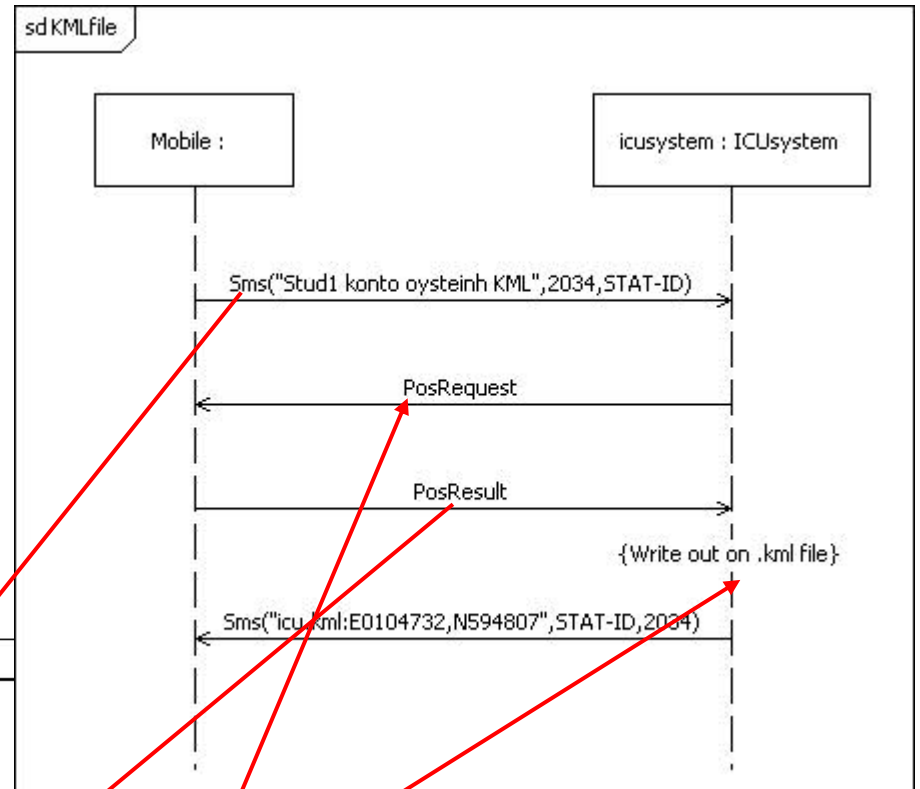
JavaFrame action language

- In principle all java can be used
 - but we try only to use simple constructs
 - we prefer to use Activity constructs for loops/choices etc.
- *output* (*Signal, Port, csm*)
 - sends a signal through a 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 most recent 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

Transition Effect – Activity Diagram



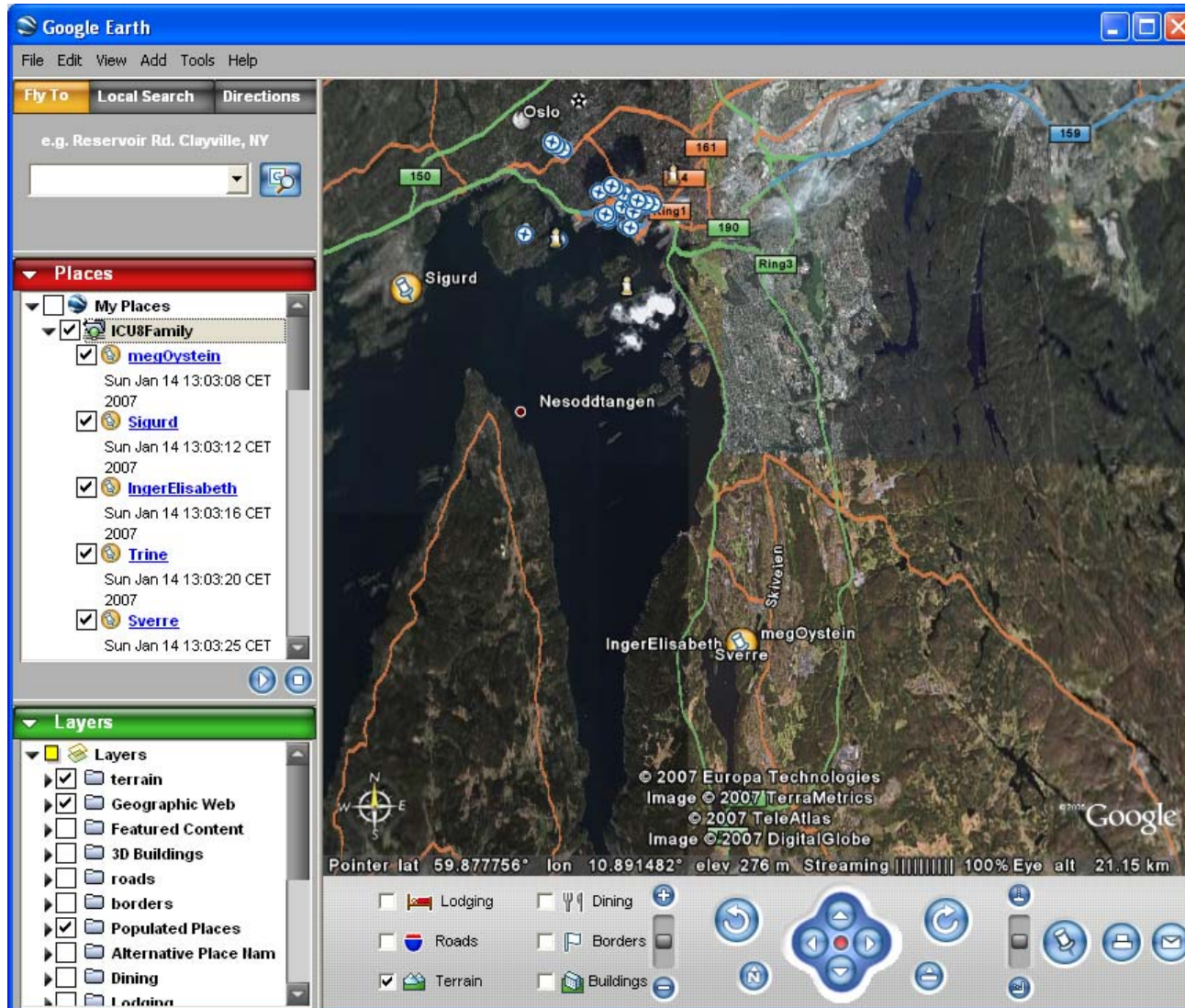
Runtime Consistency!



Buzzzzz 2: Refinement

- Assume that the semantics of the state machine are the traces that it potentially may produce (given all reasonable input from a Mobile) as positive traces and all other traces as negative.
- Is the state machine *ICUprocess* a refinement of the interaction *KMLfile*?
- Is the opposite refinement true? (that *KMLfile* is a refinement of *ICUprocess*)

KML: using GoogleEarth to place mobiles





Testing ICU0

by using the UML Testing Profile
with foils also from
Prof. Dr. Ina Schieferdecker

The Problem

■ Software

- Increases in complexity, concurrency, and dynamics
- Quality is key
 - Functionality
 - Performance
 - Scalability
 - Reliability
 - Usability
 - Efficiency
 - Maintainability
 - ...

➤ Testing is

- Means to obtain objective quality metrics about systems in their target environment
- Central means to relate requirements and specification to the real system

Testing Today

- **Is**
 - Important
 - Means to obtain approval
 - Time critical
- **But often**
 - Rarely practiced
 - Unsystematic
 - Performed by hand
 - Error-prone
 - Considered being destructive
 - *Uncool*
*„If you are a bad programmer
you might be a tester“*
- **Conjecture:**
There is a lack of appropriate test methods and techniques



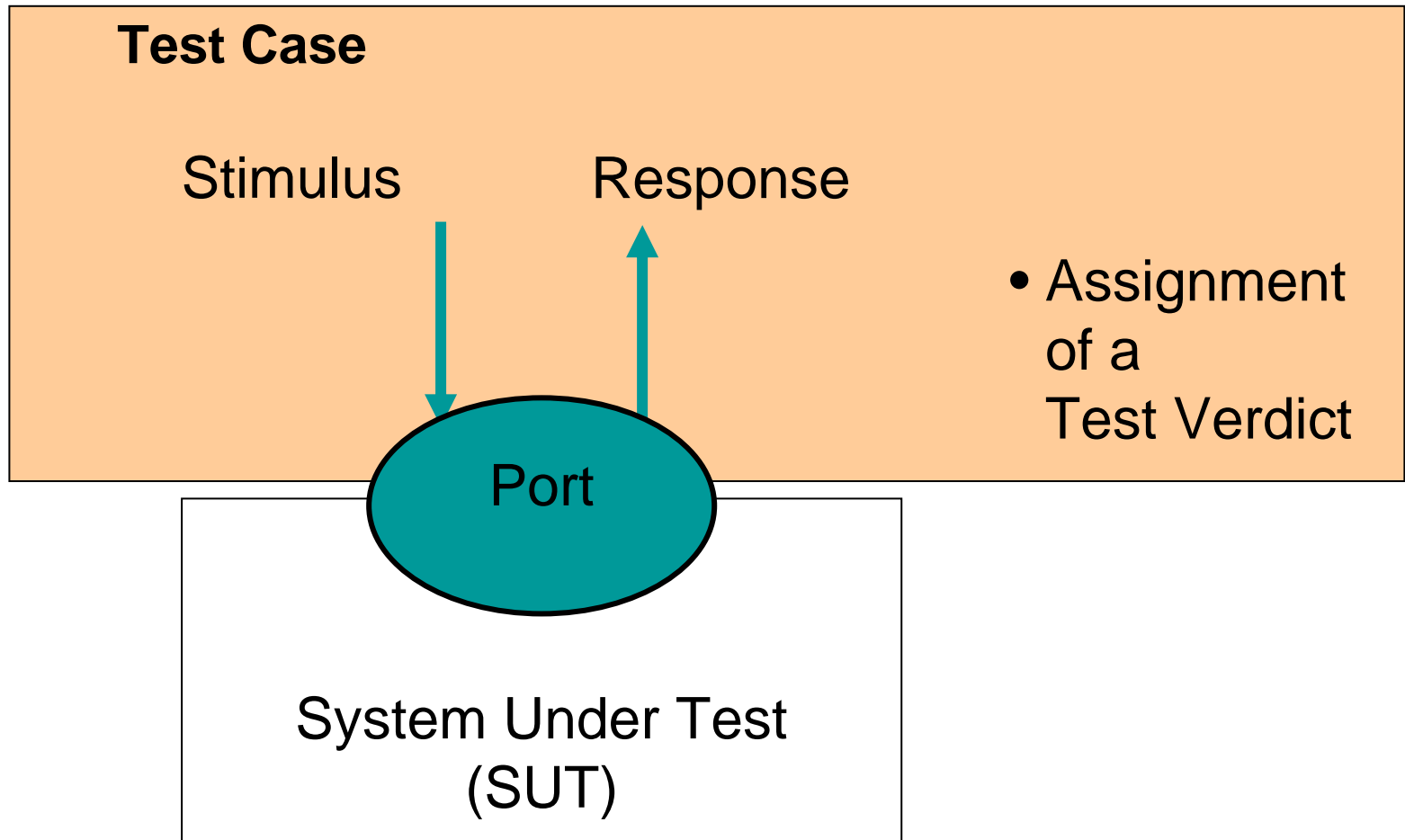
Testing is ...

- A **technical process**
- Performed by **experimenting** with a system
- In a **controlled** environment following a **specified** procedure
- With the intent of **observing** one or more **characteristics** of the system
- By demonstrating the **deviation** of the system's **actual** status from the **required** status/specification.

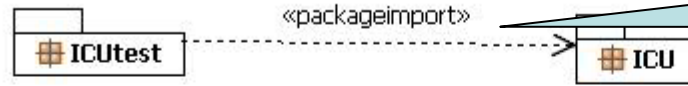
Goals of the UML Testing Profile

- Definition of a testing profile to capture all information that would be needed by different test processes
 - To allow **black-box testing** (i.e. at UML interfaces) of computational models in UML
- A testing profile based upon UML 2.0
 - That enables the **test definition and test generation** based on **structural** (static) and **behavioral** (dynamic) **aspects** of UML models, and
 - That is capable of **inter-operation with existing test technologies** for black-box testing
- Define
 - Test purposes for computational UML models, which should be related to relevant system interfaces
 - Test components, test configurations and test system interfaces
 - Test cases in an implementation independent manner

Test Concepts: Black-Box Testing

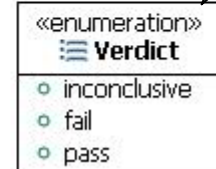
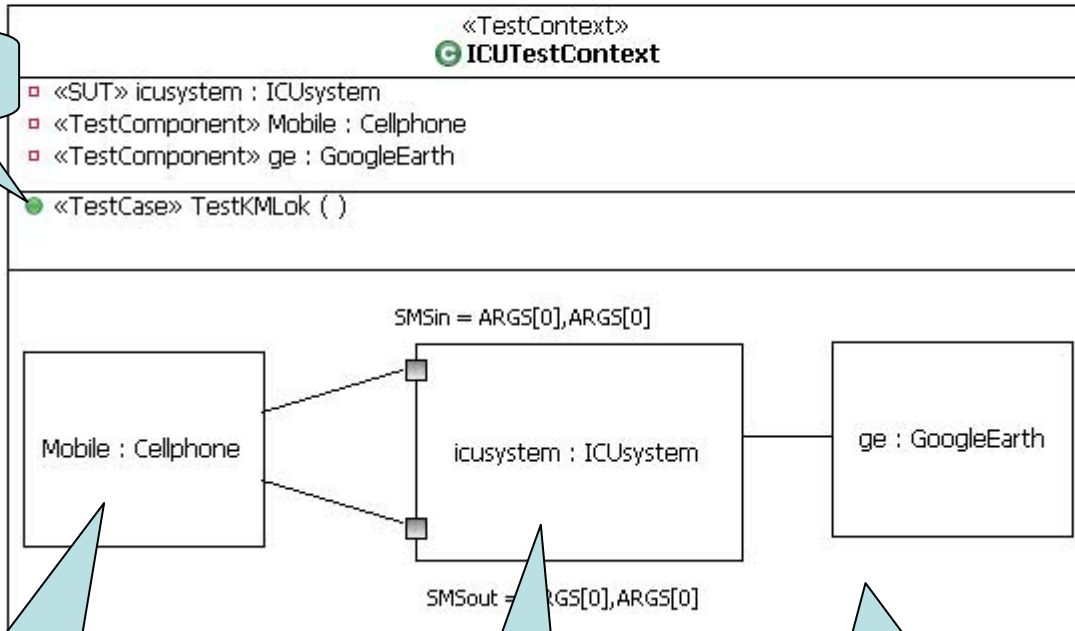


ICU0 test context



test package imports def of system

Test case



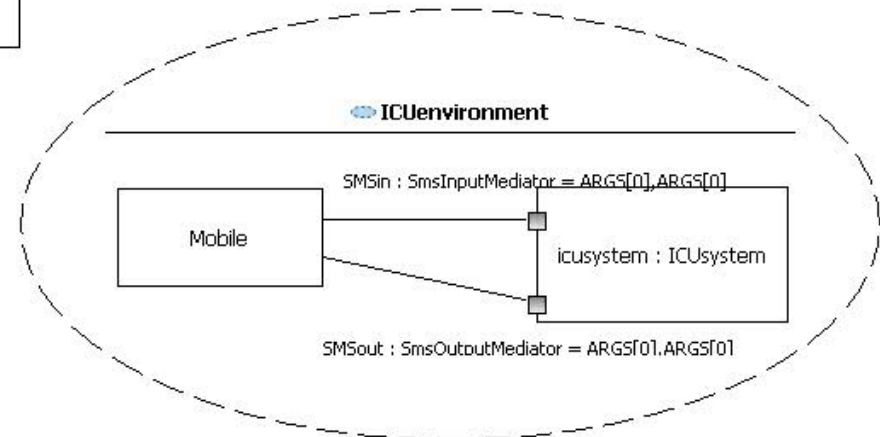
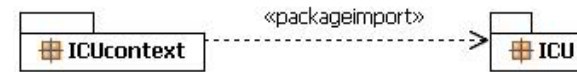
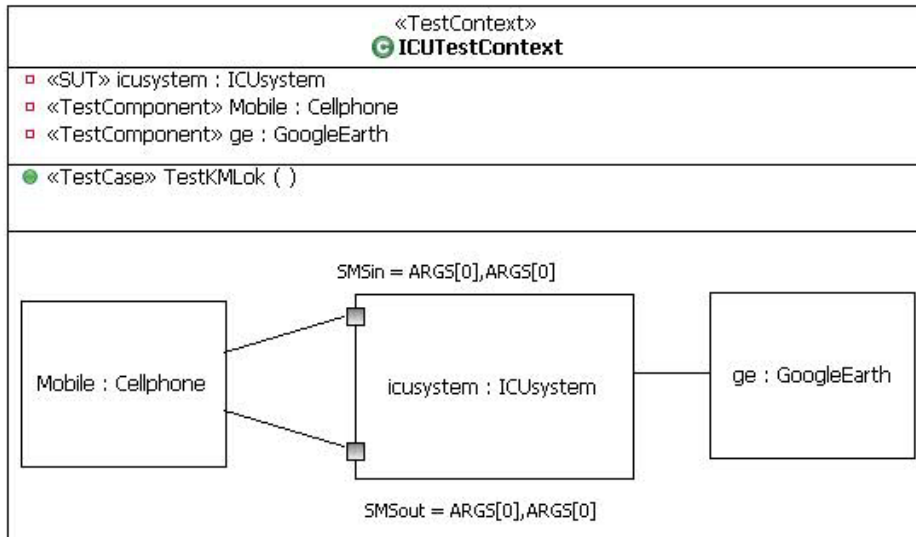
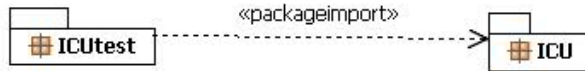
Test case returns

Test component

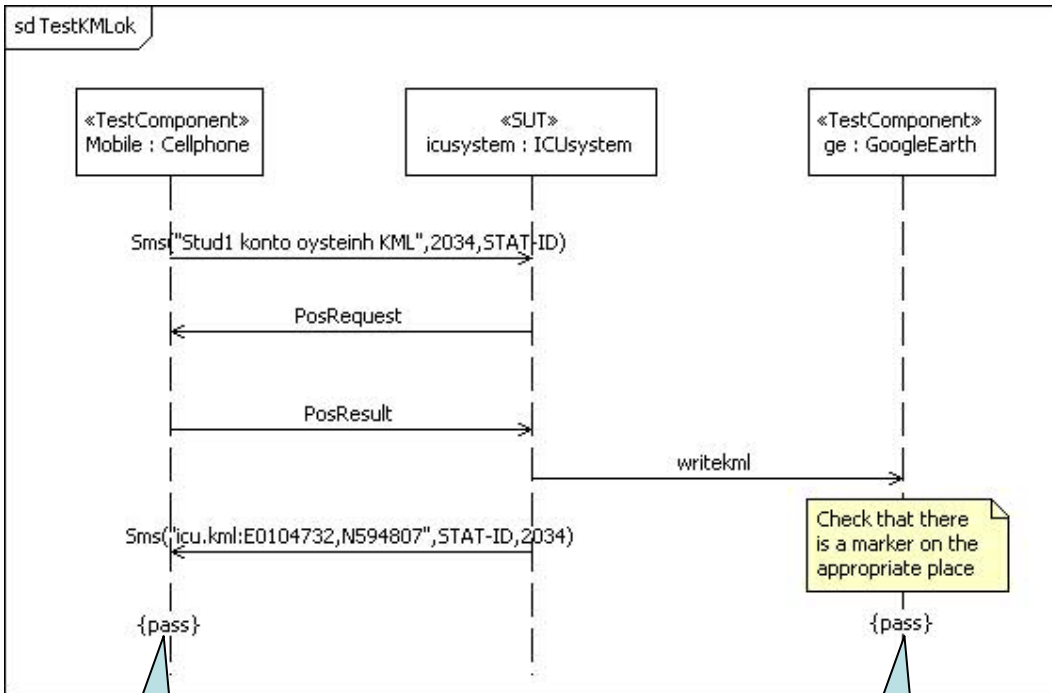
System Under Test

Test configuration

Test context and system context are similar

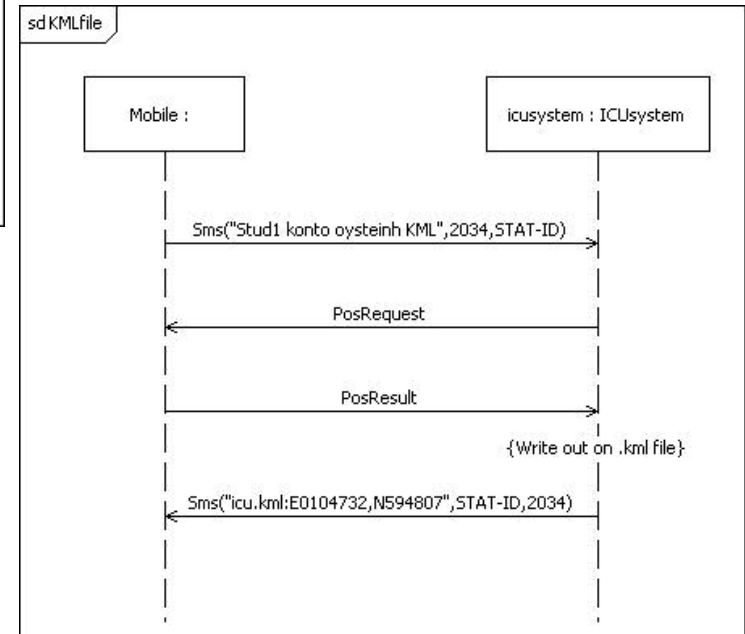


Test behavior and context behavior are similar



Verdict

Verdict





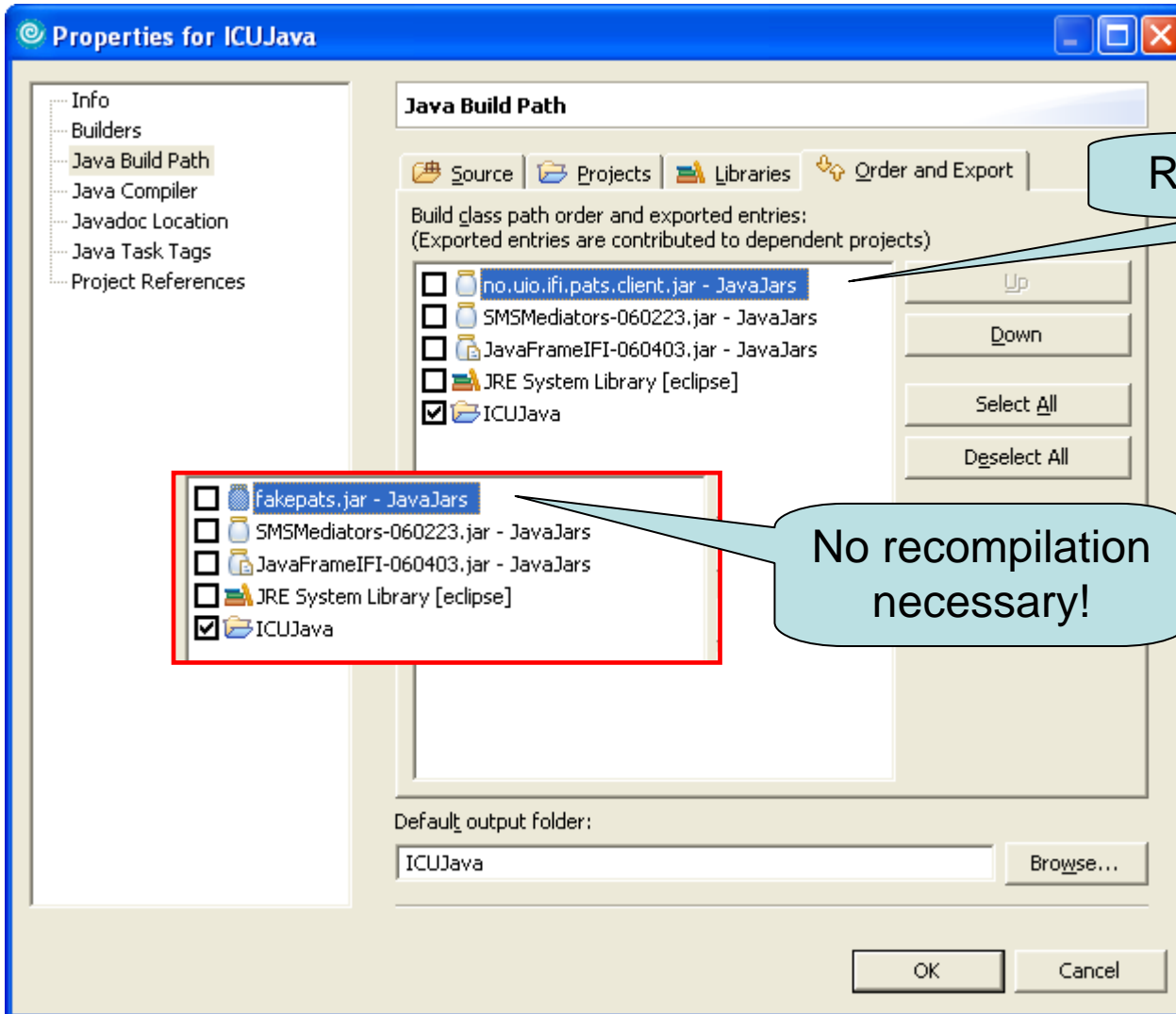
Buzzzz 3: Why both context behavior and tests?

- Why do we need tests when we have context behavior
 - We do not always only want *pass* verdicts
 - we could also use the **neg** fragments in Sequence Diagrams
 - We may want more tests than context behaviors
- Tests should be explicit
 - Identify the SUT and the Test components
 - this distinction is not done in the context behavior sequence diagrams
 - Clearly specify the verdicts
 - context behaviors usually specify potential positive behaviors only

How to execute the tests

- Generated test components
 - we could specify the behavior of the test components
 - then compile and run the total test management system
 - and have the tool verify the test cases by comparison
- Manual execution on real environment
 - you operate the mobile phone, and observe the resulting SMSes
 - you observe also the GoogleEarth results
 - Disadvantage: slow procedure since you need to physically move
 - Advantage: it is the real thing
- Manual execution on simulated environment
 - FakePATS made by Frank Davidsen
 - Advantage: quicker turn-around, easier manipulation, cheaper

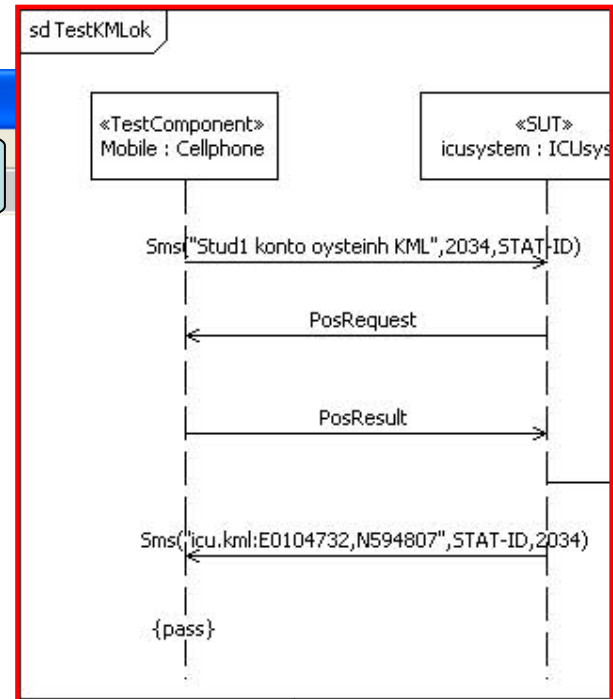
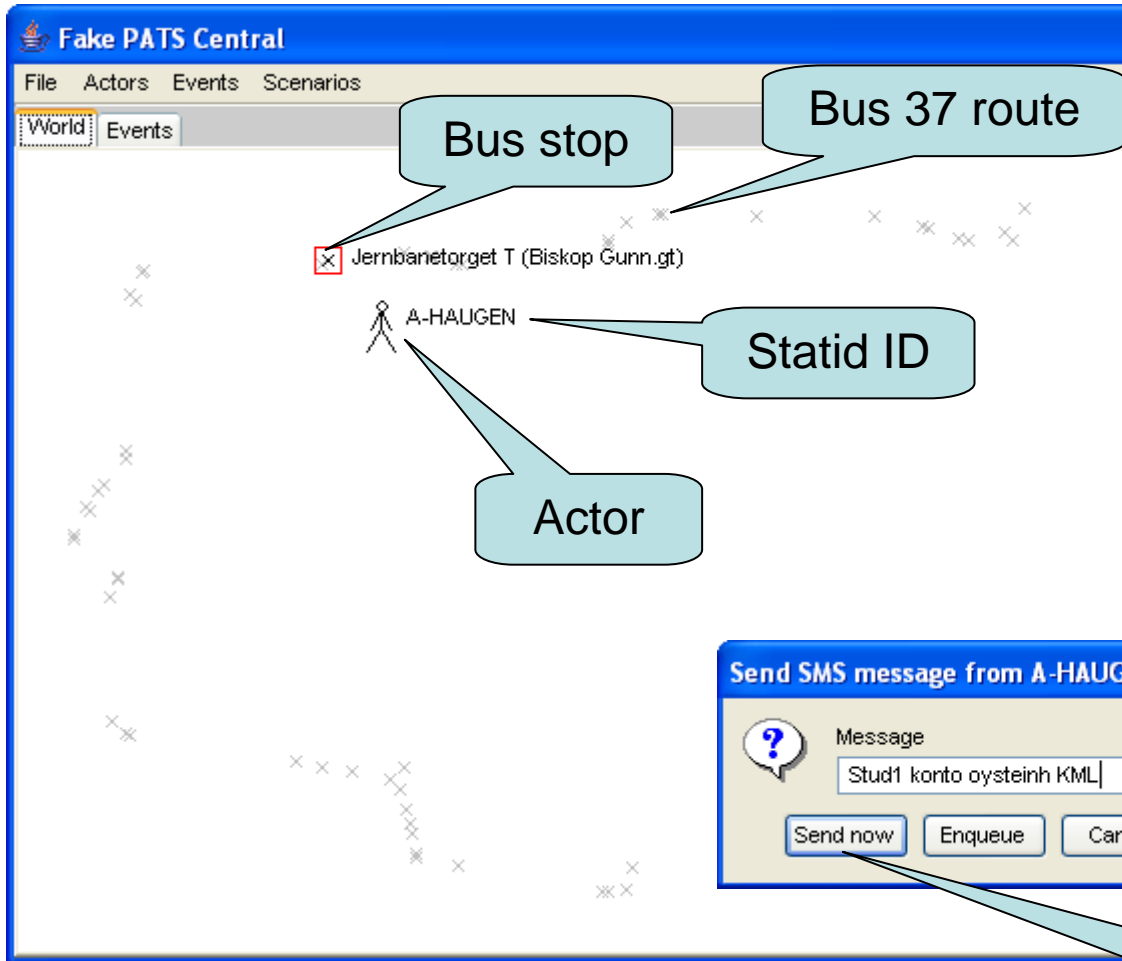
FakePATS instead of low level PATS-software



Replace this with FakePATS

No recompilation
necessary!

fakepats.jar is also a stand-alone program!

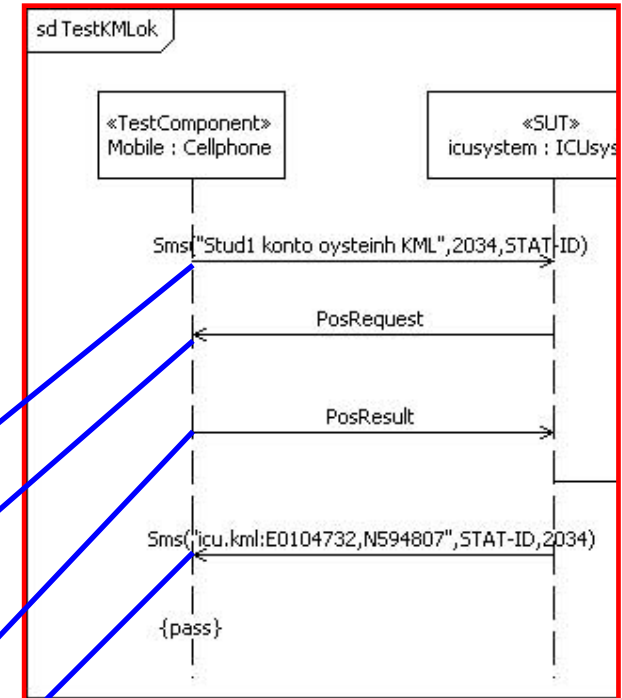


The dialog box has a title bar 'Send SMS message from A-HAUGEN' with a close button. It contains a question mark icon and a 'Message' label. Below the label is a text input field containing the text 'Stud1 konto oystein KML'. At the bottom of the dialog are three buttons: 'Send now', 'Enqueue', and 'Cancel'.

Start fakepats, then application

Send SMS from actor

The verdict of the fake mobile



INF 5150

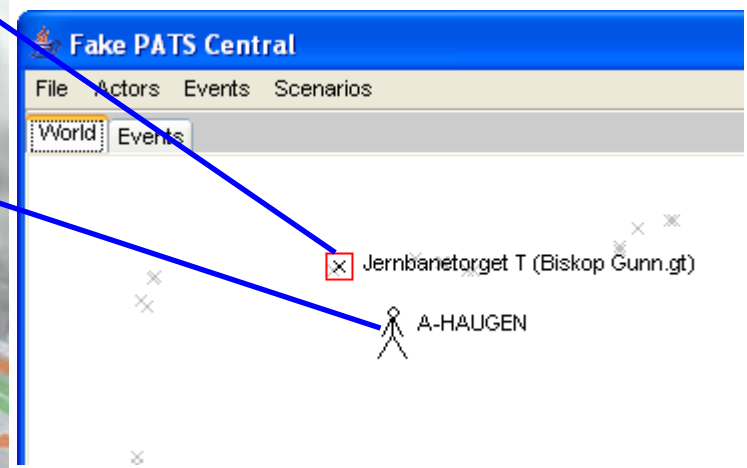
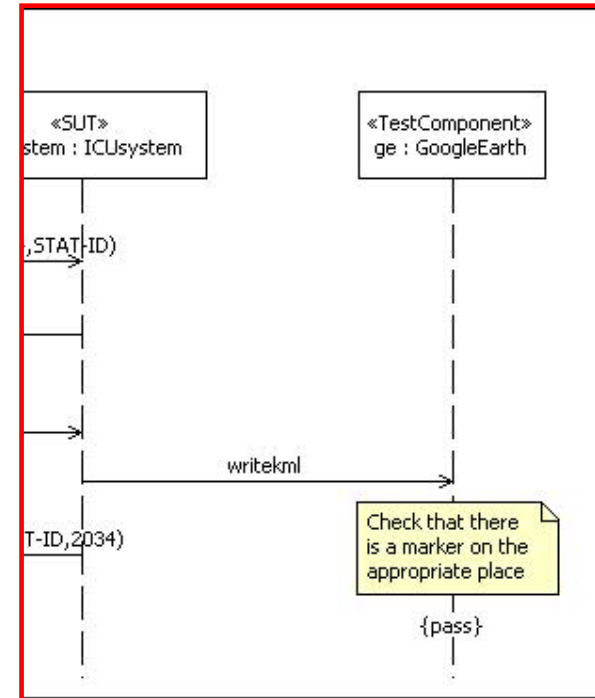
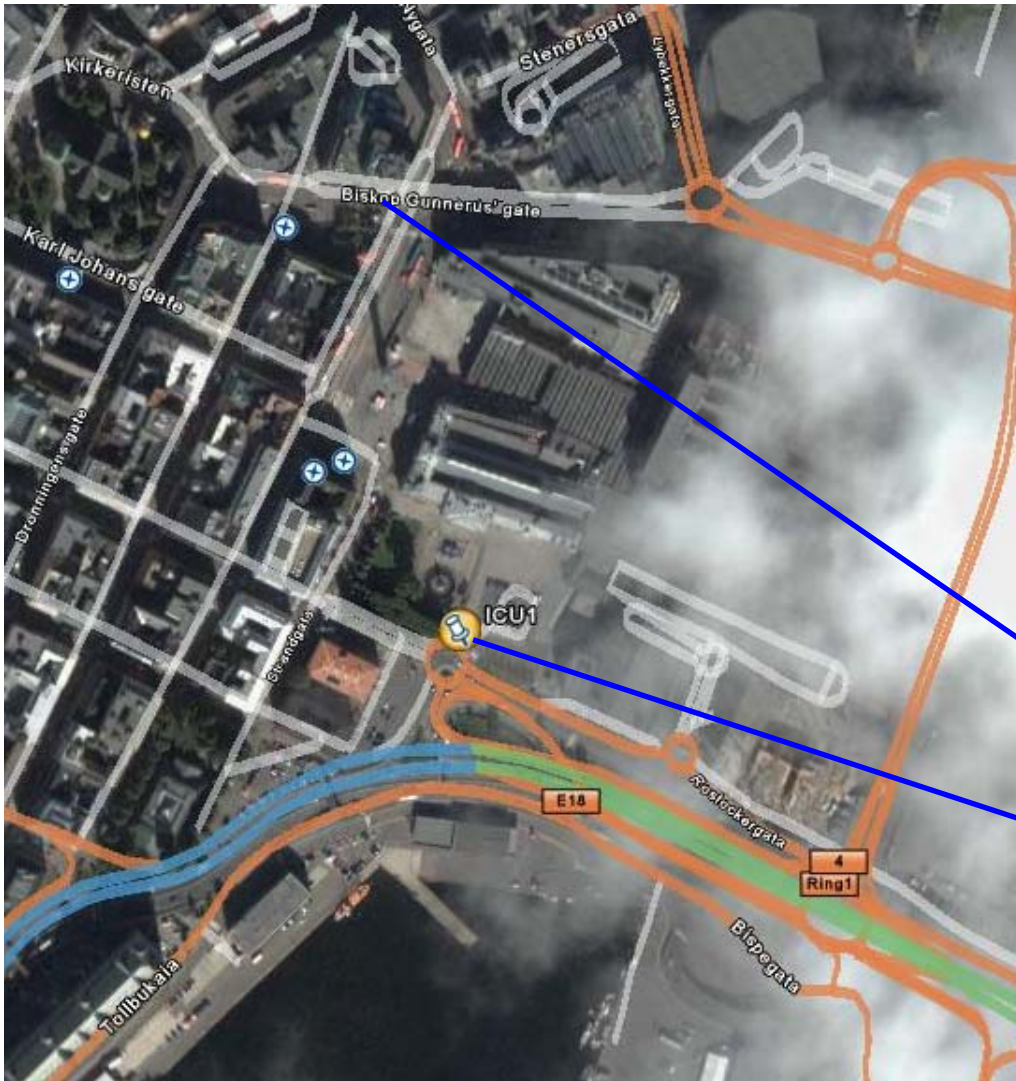
Fake PATS Central

File Actors Events Scenarios

World Events

From	To	Details
A-HAU...	2034	Stud1 konto oystein KML
		MessageID: 1170020023206 PositioningID: A-HAUGEN
		MessageID: 1170020023206 Position: <Feilkode>100<Breddegrad>N595458<Lengdegrad>E0104525<STATICID>A-HAUGEN
2034	A-HAUGEN	ICU.kml:E0104525,N595458

Verdict of GoogleEarth

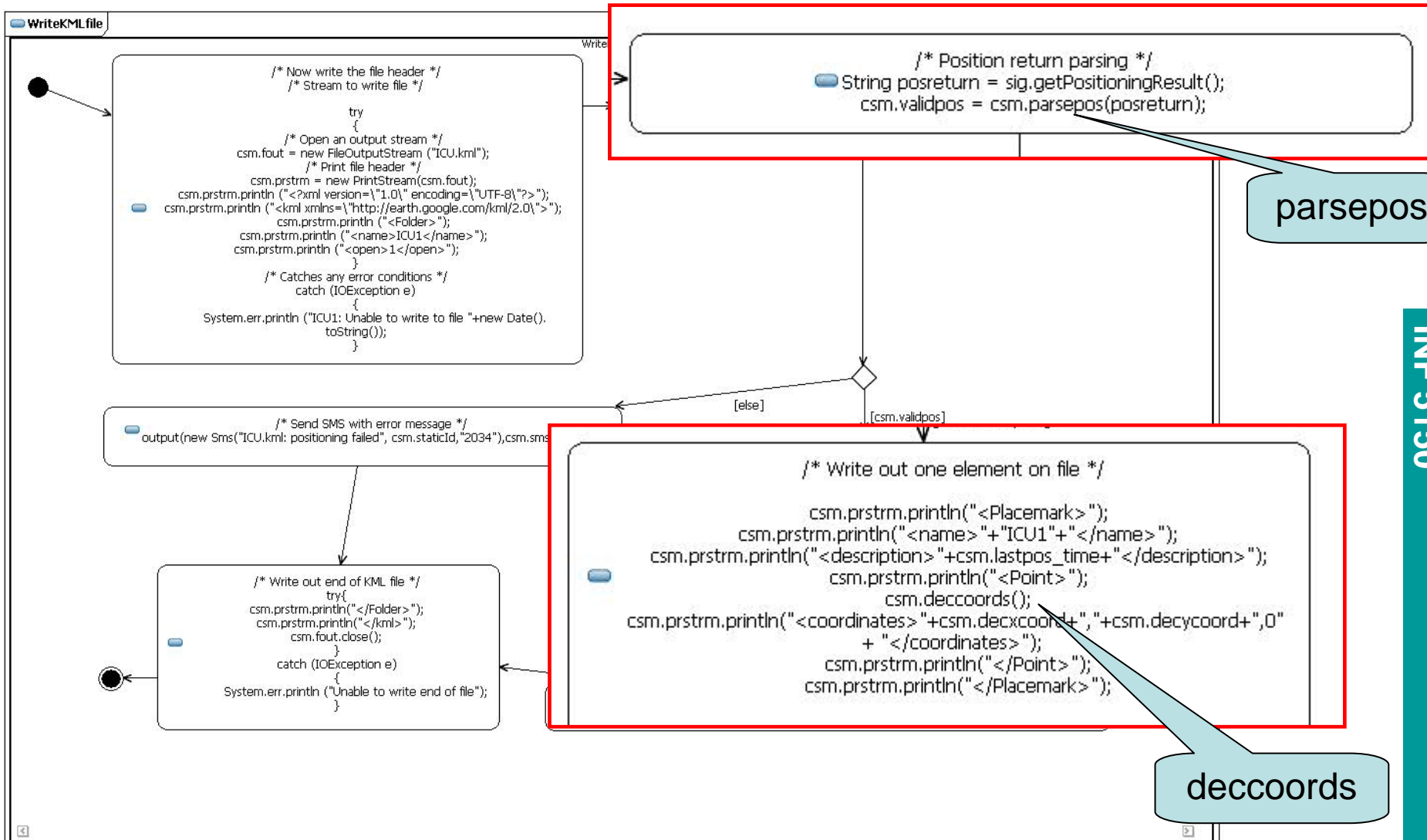




About operations and methods

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



UML distinguish between operation and method

The screenshot shows the Papyrus IDE interface. The main window displays a UML diagram with a callout box pointing to a `ParsePositionResponse` object, containing the code: `String posreturn = sig.getPositioningResult(); csm.validpos = csm.parsepos(posreturn);`. A callout box labeled "parsepos – the operation" points to this object. The Properties window at the bottom shows the details for `ICU::ICU1::ICUSystem::ICUProcess::parseposBehavior`, including its name, visibility (public), and signature: `ICU::ICU1::ICUSystem::ICUProcess::parsepos(String) : Boolean`. The Bodies section contains the implementation code for the method. A callout box labeled "parsepos – the method" points to the Properties window.

```
String posreturn = sig.getPositioningResult();
csm.validpos = csm.parsepos(posreturn);
```

parsepos – the operation

parsepos – the method

ICU::ICU1::ICUSystem::ICUProcess::parseposBehavior

General Name: parseposBehavior

Profile Visibility: public protected private package

Comments Specif.: ICU::ICU1::ICUSystem::ICUProcess::parsepos(String) : Boolean

Bodies:

```
int ix = posString.indexOf("<Feilkode>");
String feilkode = posString.substring(ix + 10, ix + 13);
boolean validpos = "100".equals(feilkode);
if (validpos) {
    ix = posString.indexOf("<Breddegrad>");
    ycoord = posString.substring(ix + 12, ix + 19);
    ix = posString.indexOf("<Lengdegrad>");
    xcoord = posString.substring(ix + 12, ix + 20);
    lastpos_time = new Date().toString();
}
return validpos;
```

Parameters: