Agile modeling – for INF5150

Version 091002

ICU 0-1

Oblig 2

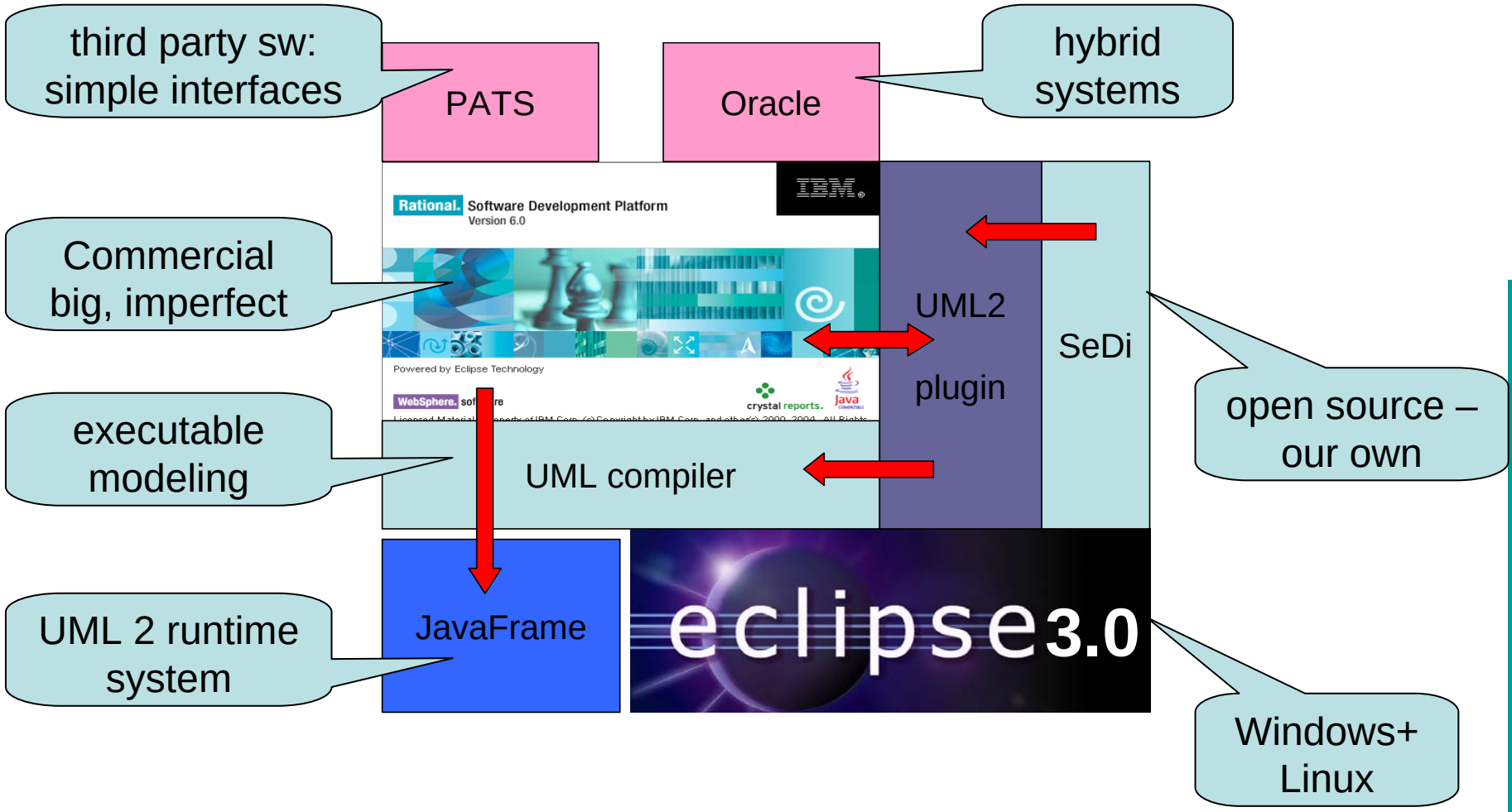
- Also Oblig2 is made individually
- This does not mean you need to work alone
  - but you need to acquire the competence
  - and the knowledge of absolutely all the details

ICU0 – your very first ”I see you” system

surveillance at your fingertips,

first we only observe ourselves

Tools for executable modeling in INF5150



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.

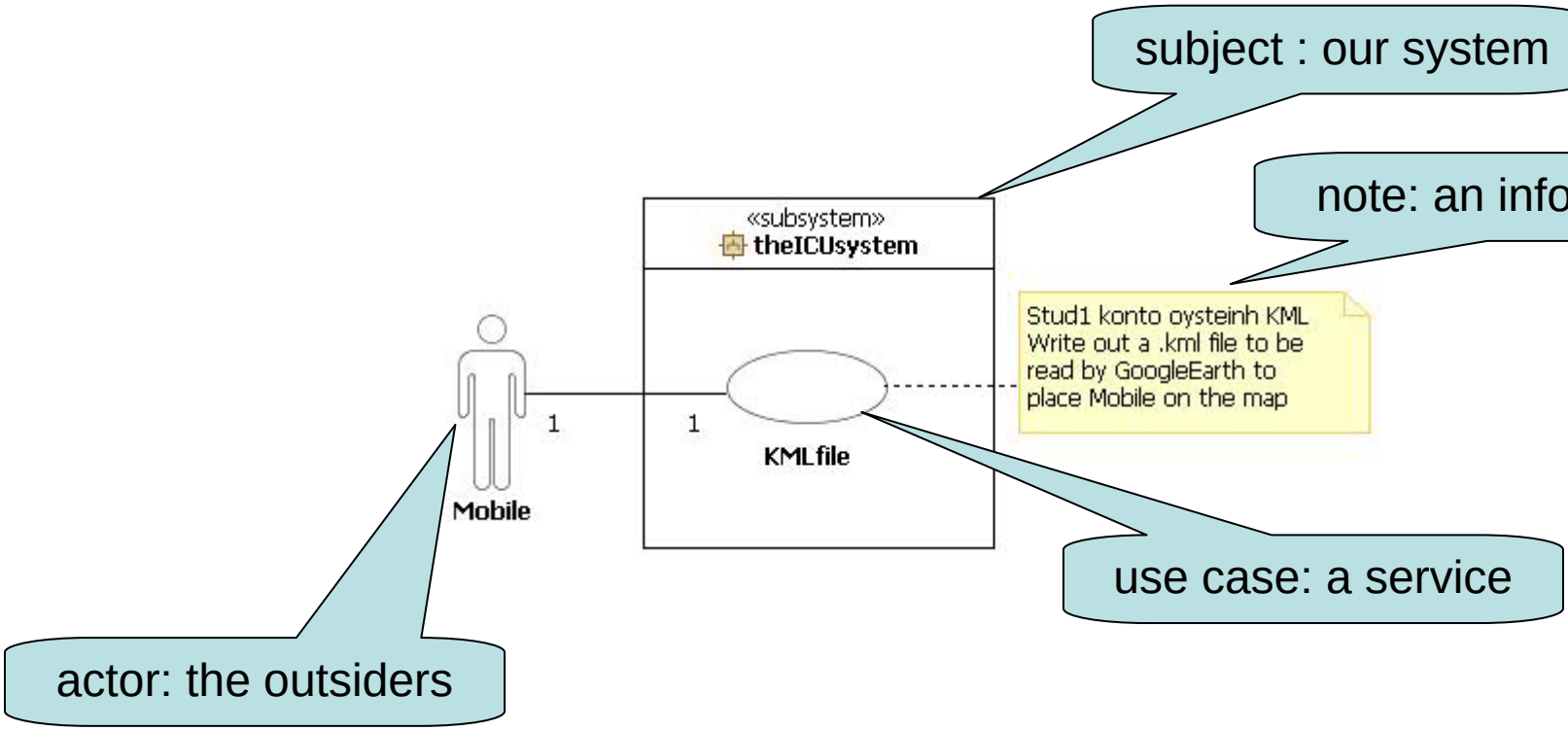
Dialectic Software Development

- 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

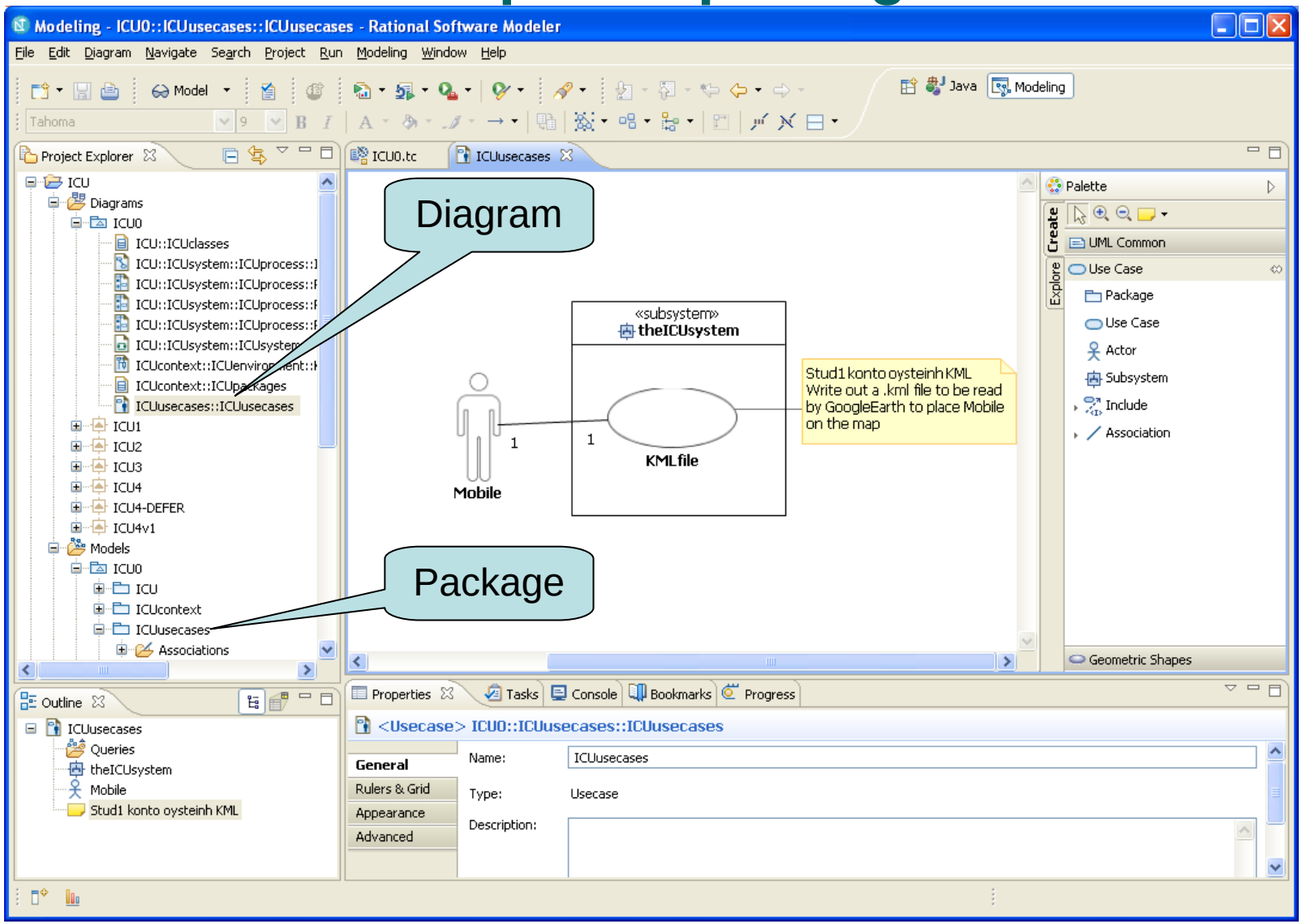
Buzzzzz 1: Agility

- 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

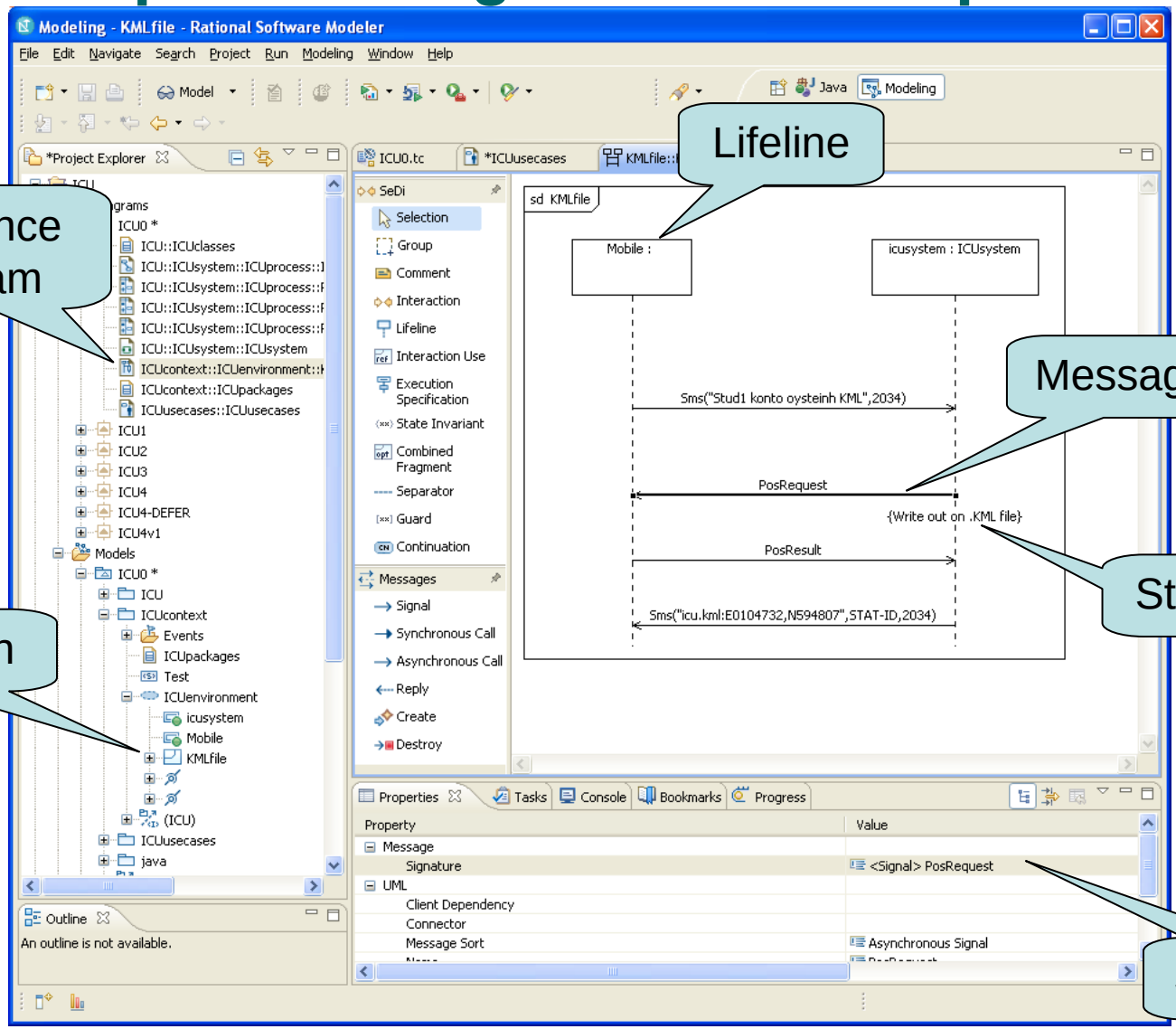
UML Use Cases – very very simple



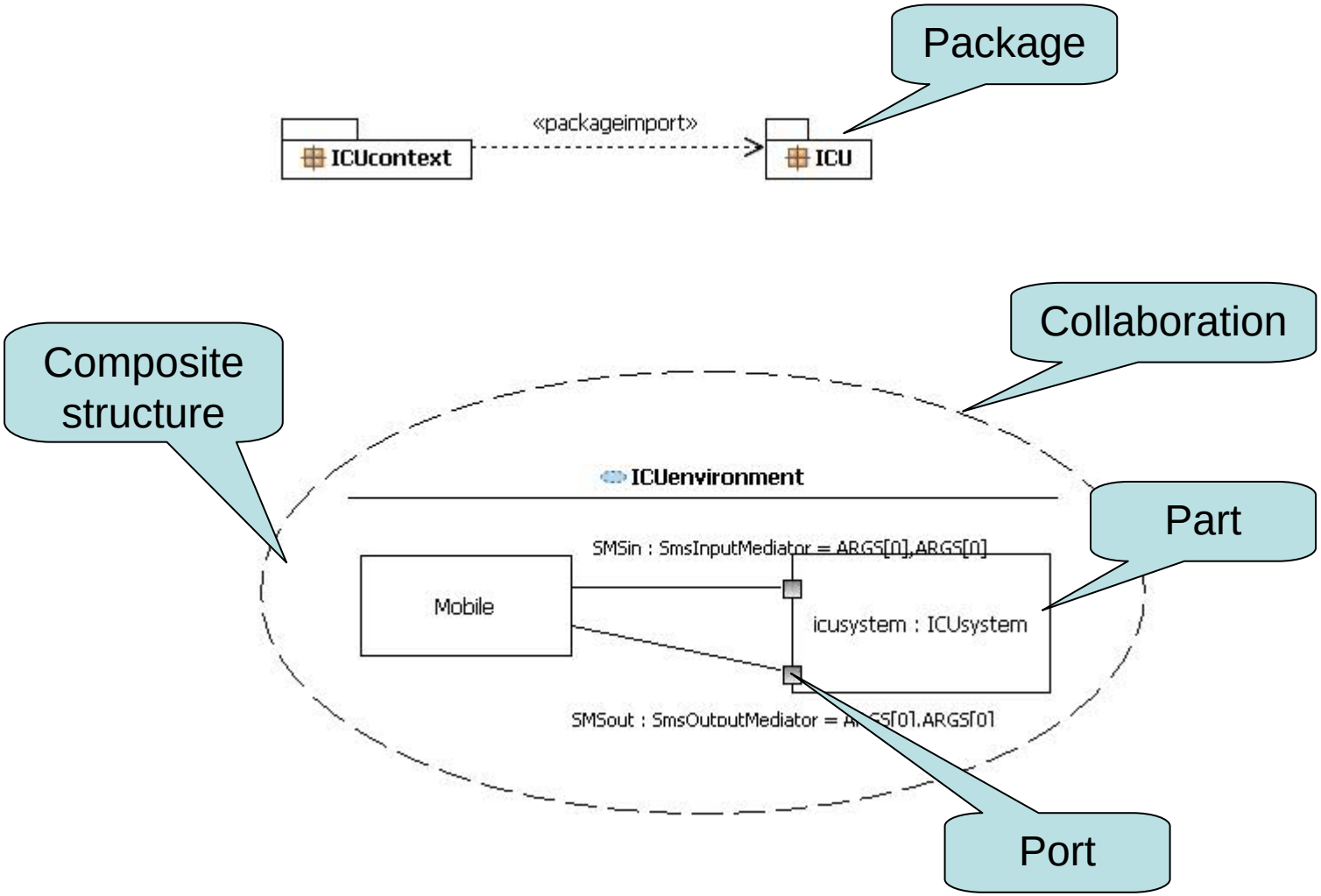
Use cases in a separate package



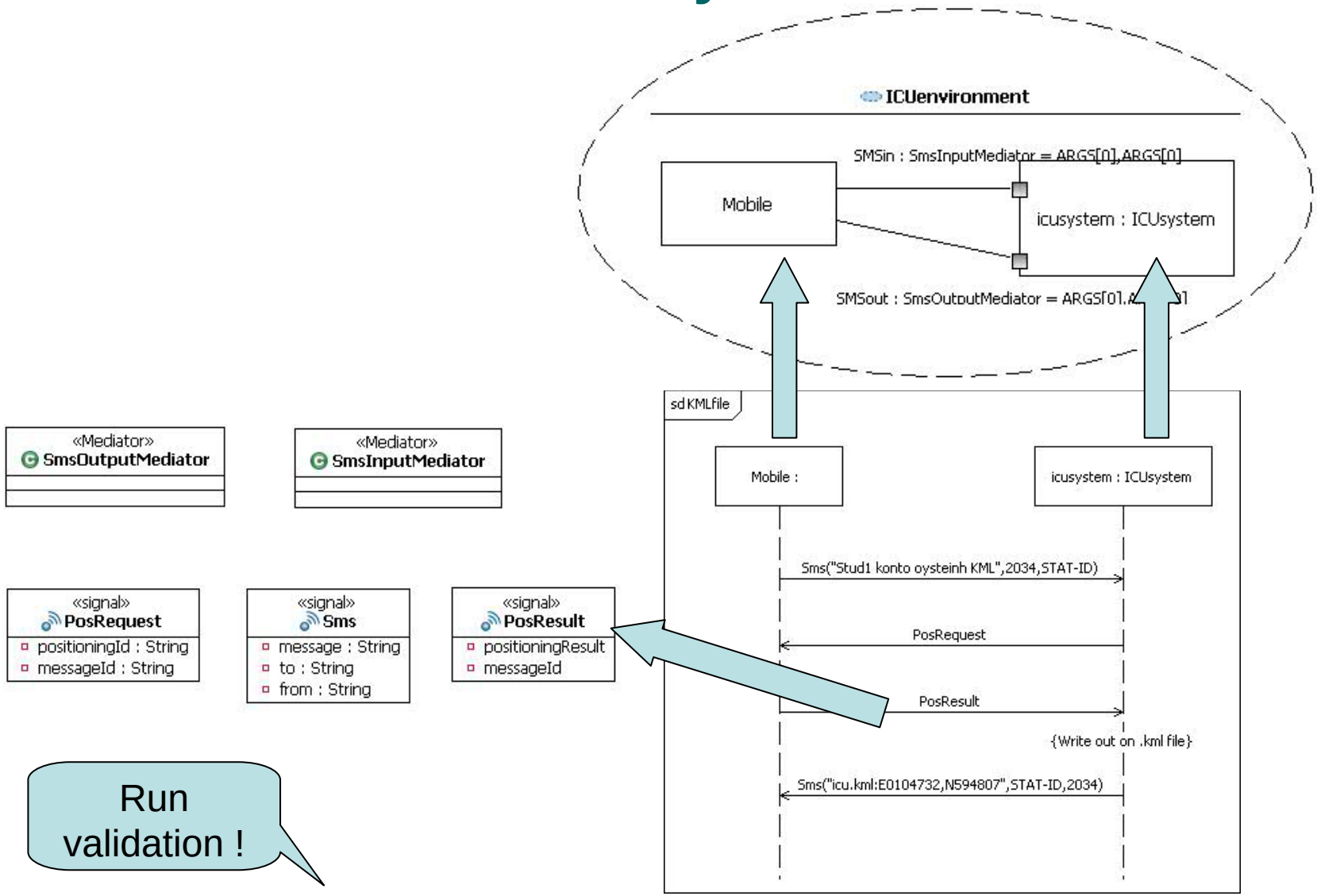
UML Sequence Diagrams: a more precise way



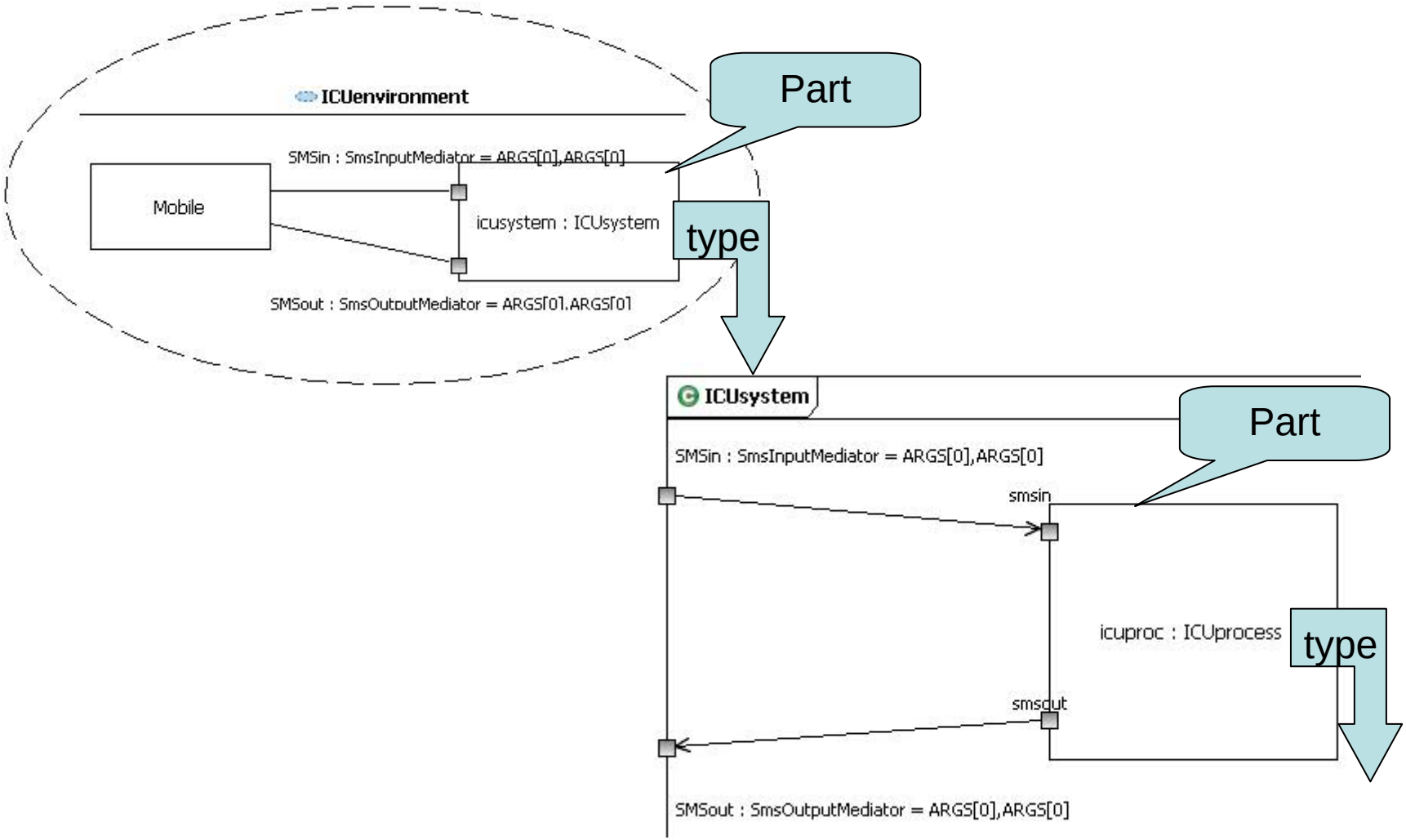
Packages, Collaboration, Composite Structure



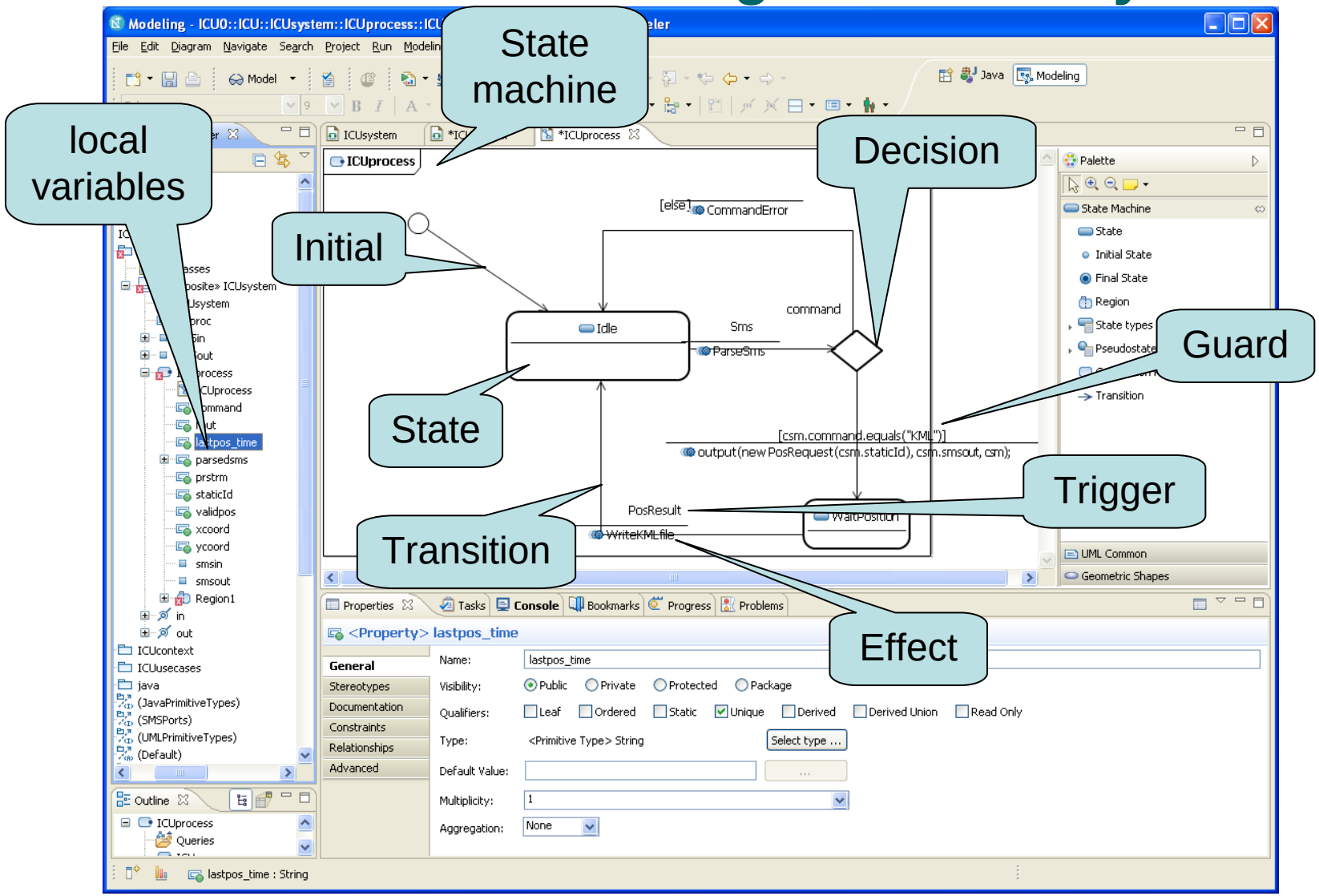
Model-time Consistency!



Structure hierarchy



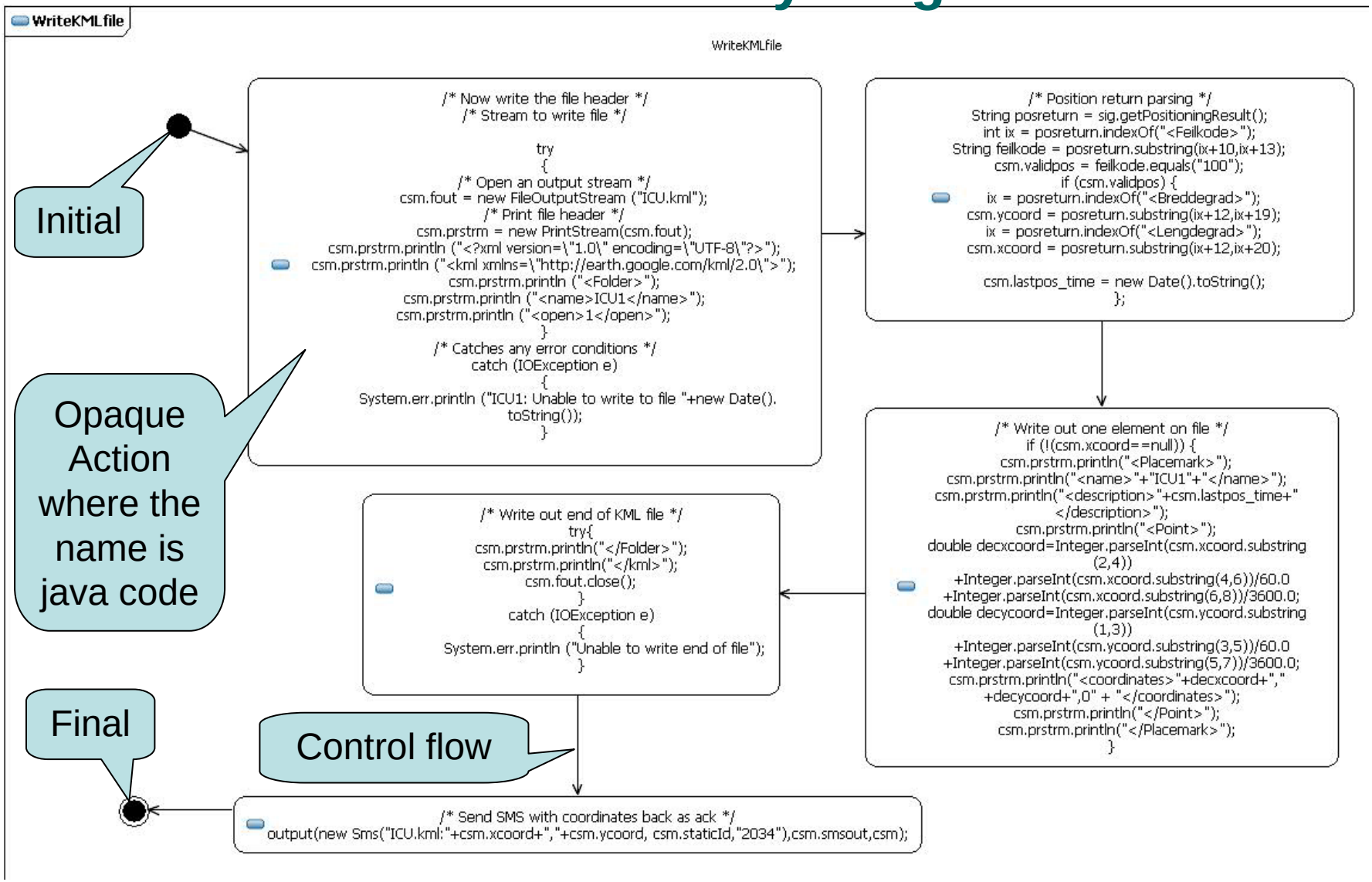
A State Machine defining the whole system



JavaFrame action language

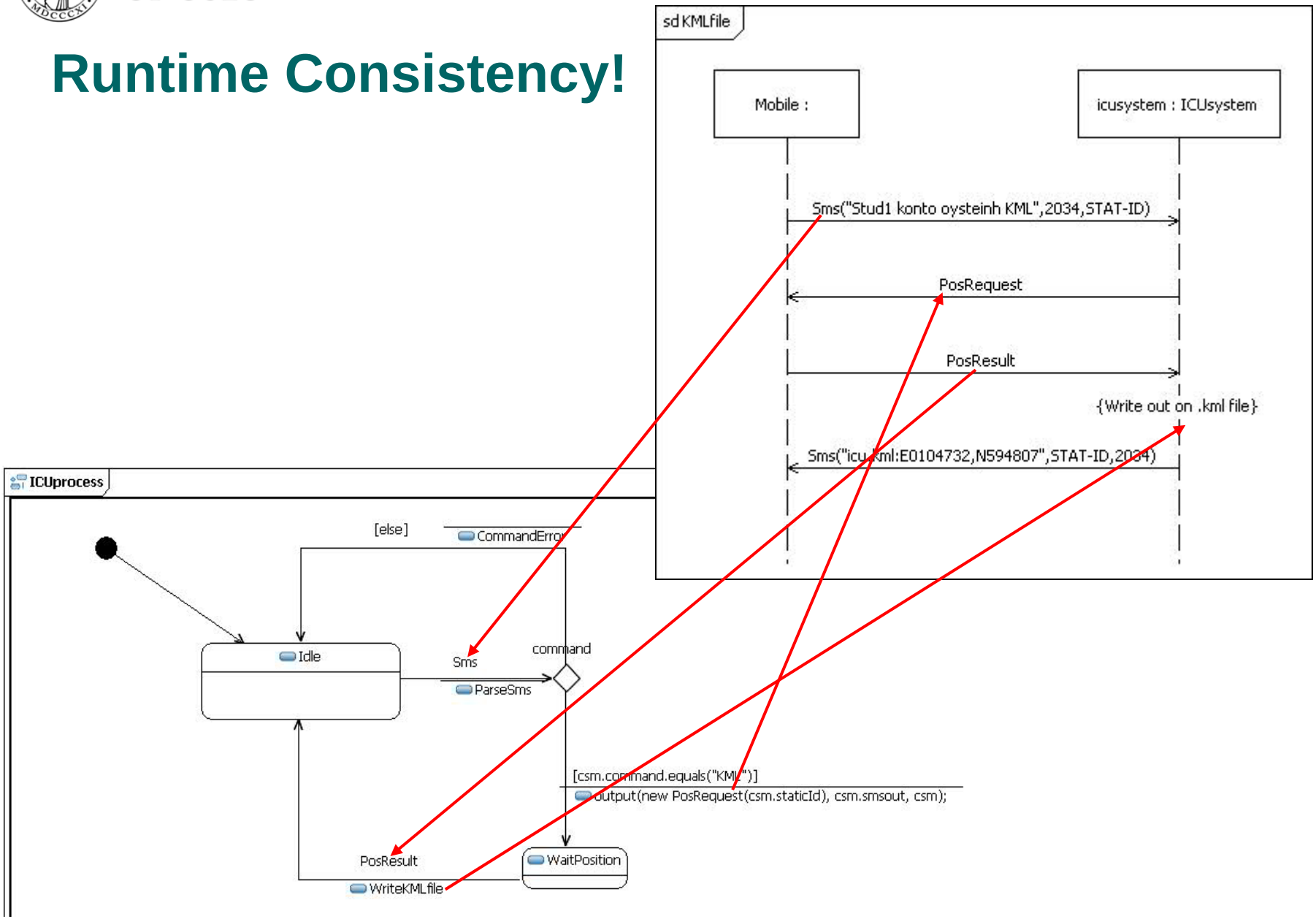
- In principle all java can be used
  - but we try only to use simple constructs
  - we prefer to use Activity constructs for loops/choices etc.
- *output (Signal, Port, csm)*
  - sends a signal through a local port.
  - typically the signal is like ”*new S(parm1, parm2)*”
  - typically the port is like ”*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



INF 5150

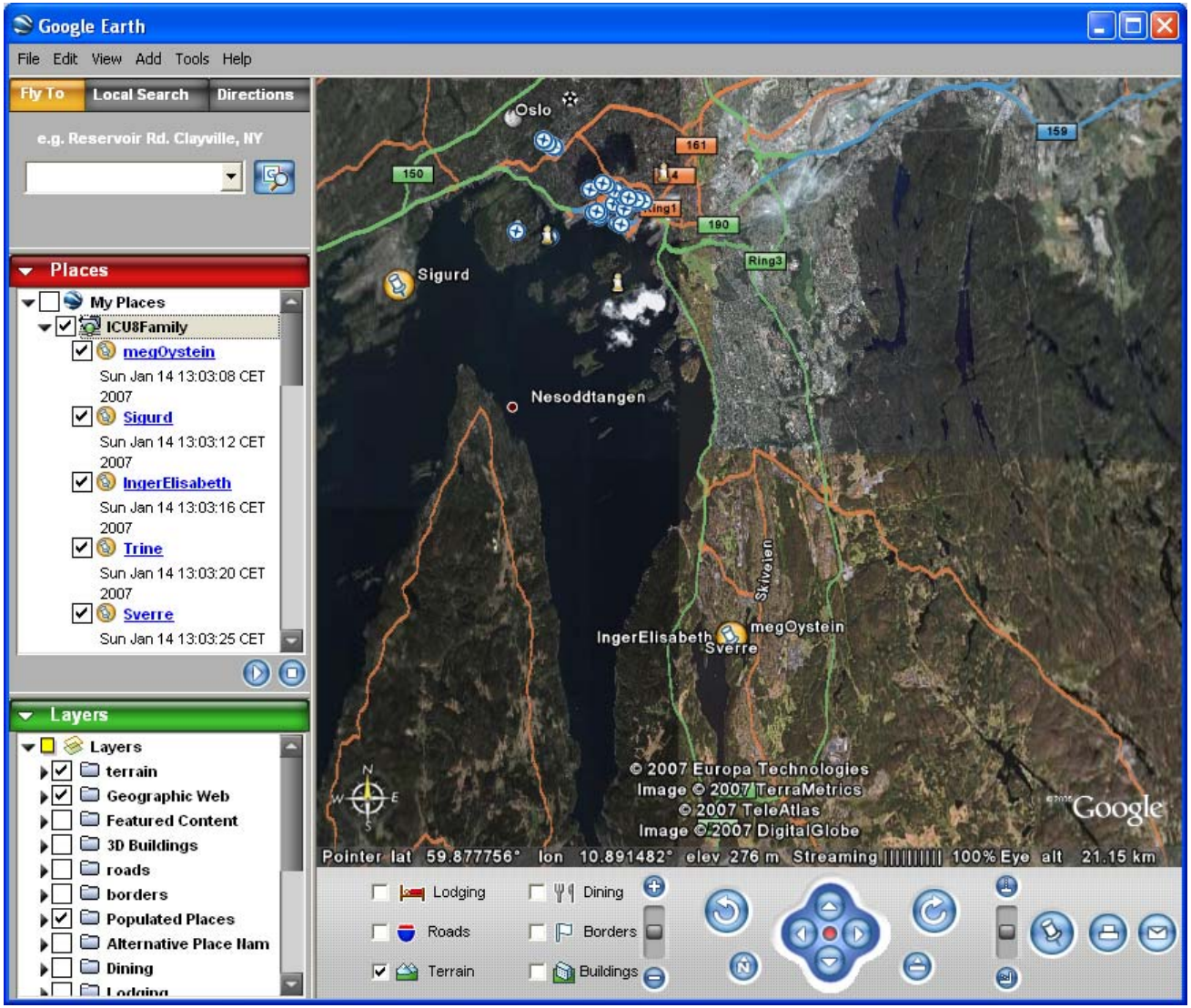
Runtime Consistency!



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

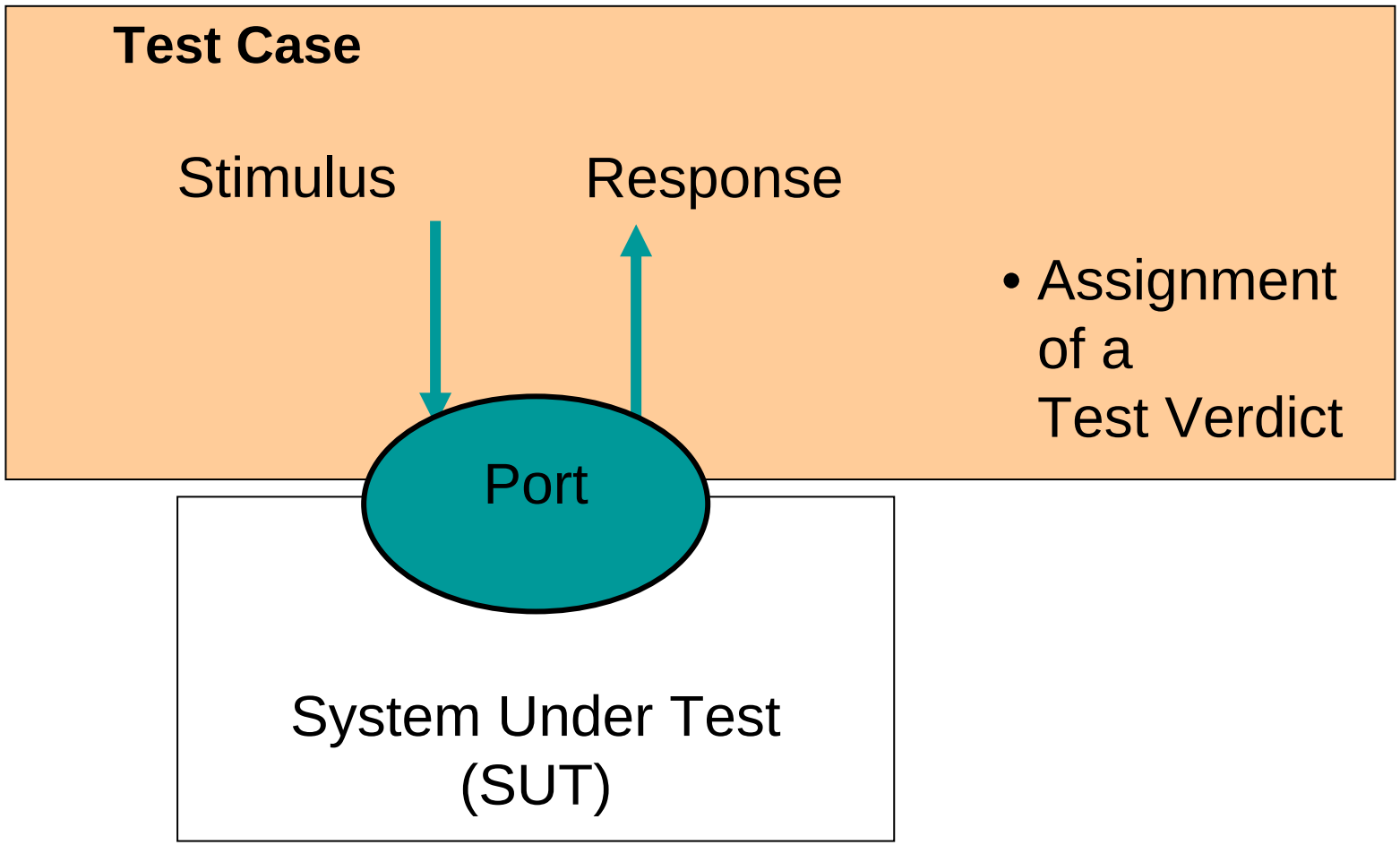
Testing is …

- 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.

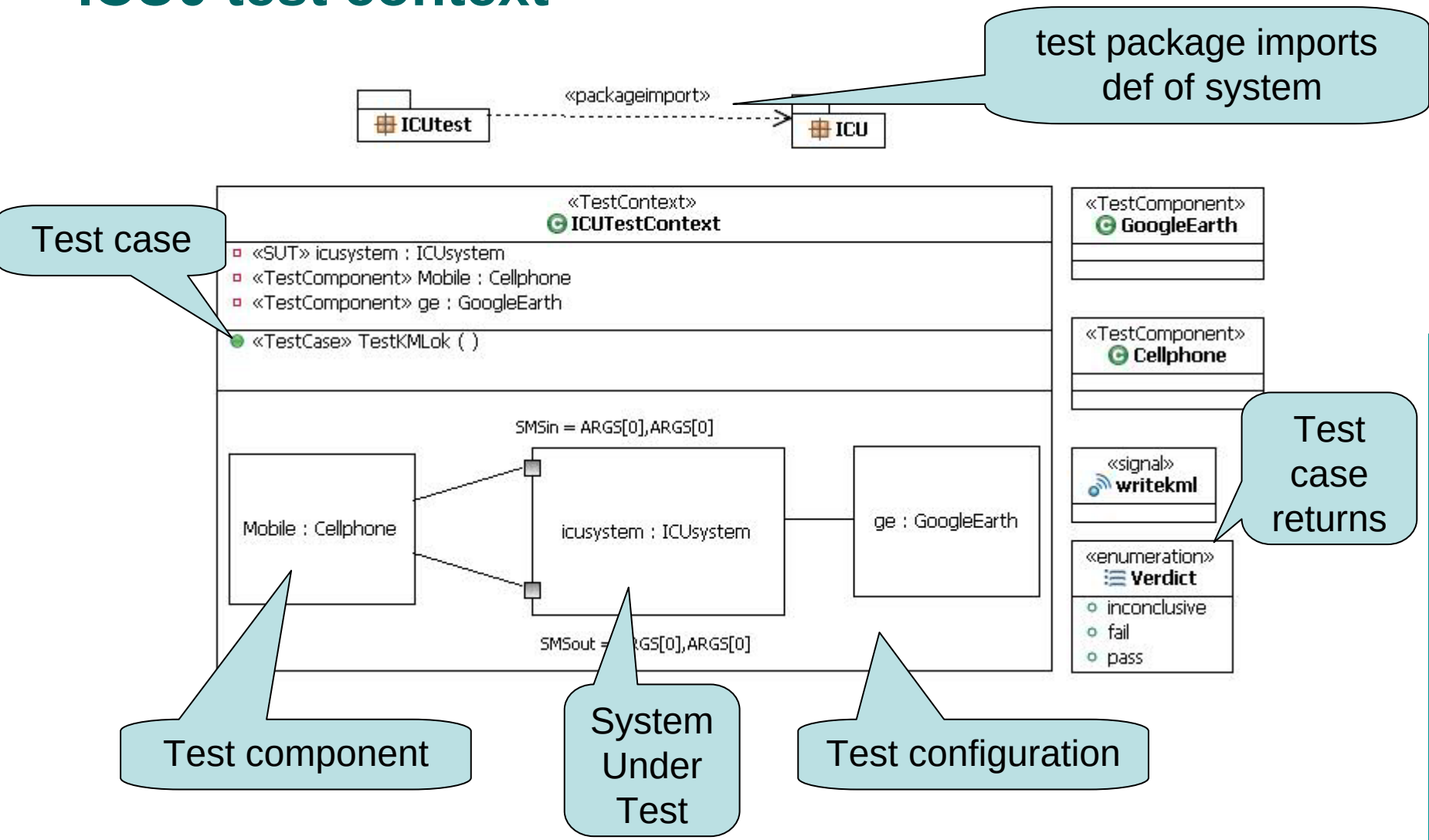
Goals of the UML Testing Profile

- 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

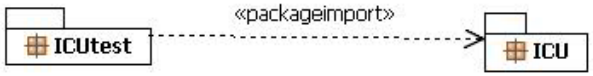
Test Concepts: Black-Box Testing

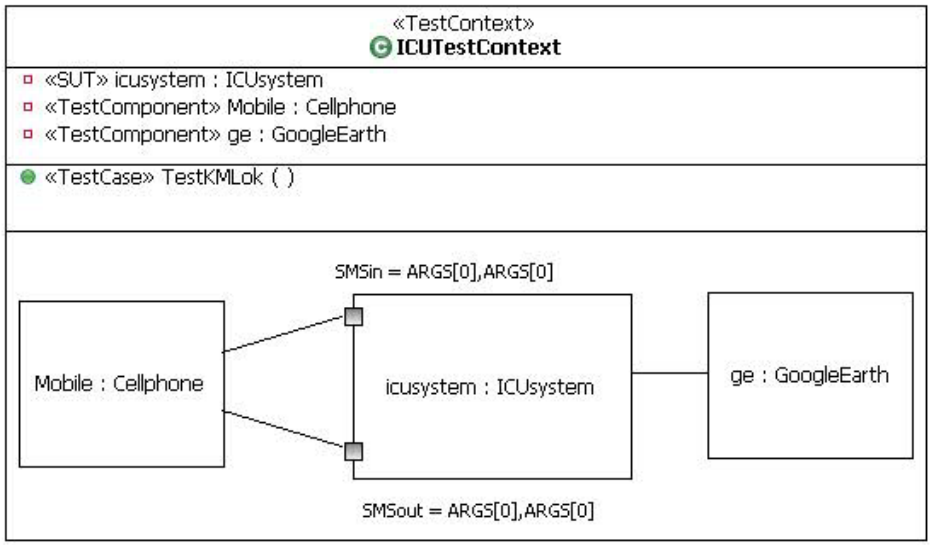


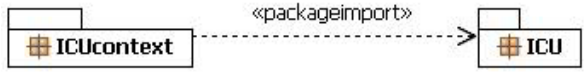
ICU0 test context

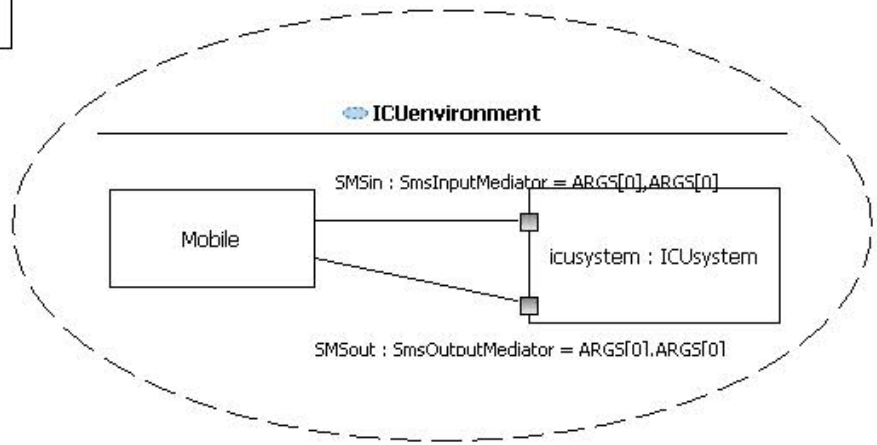


Test context and system context are similar

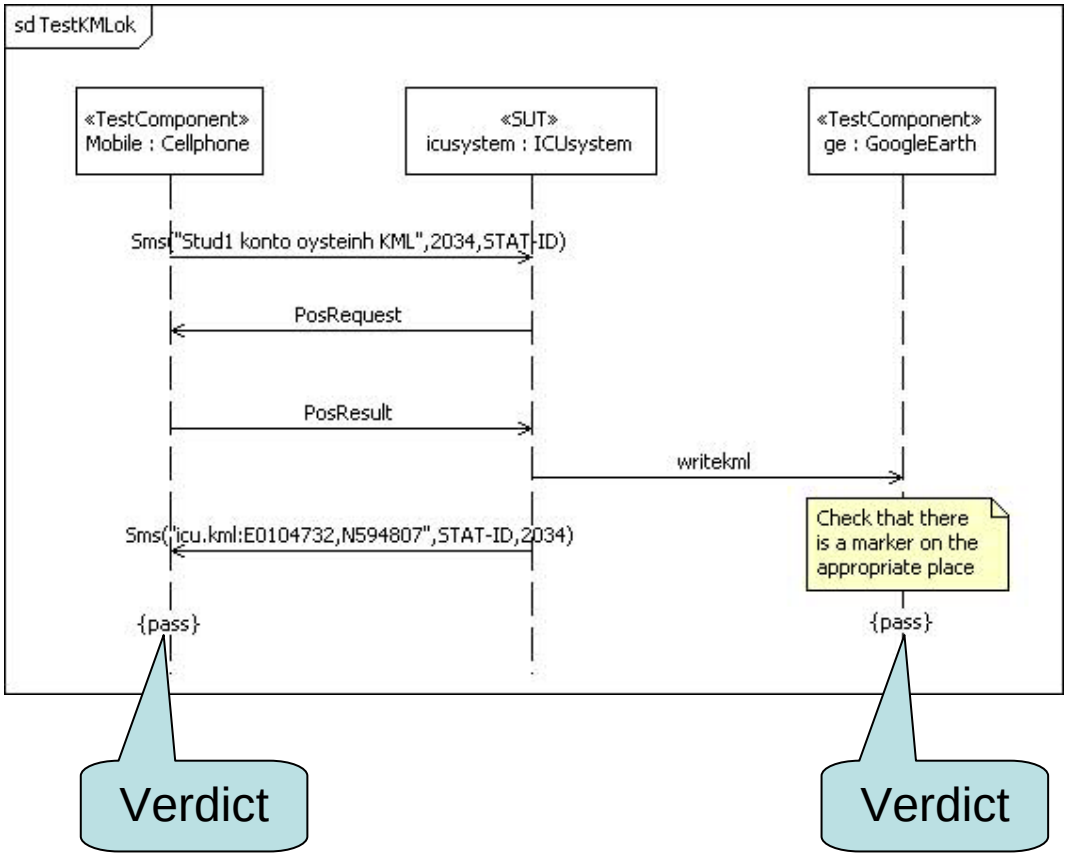


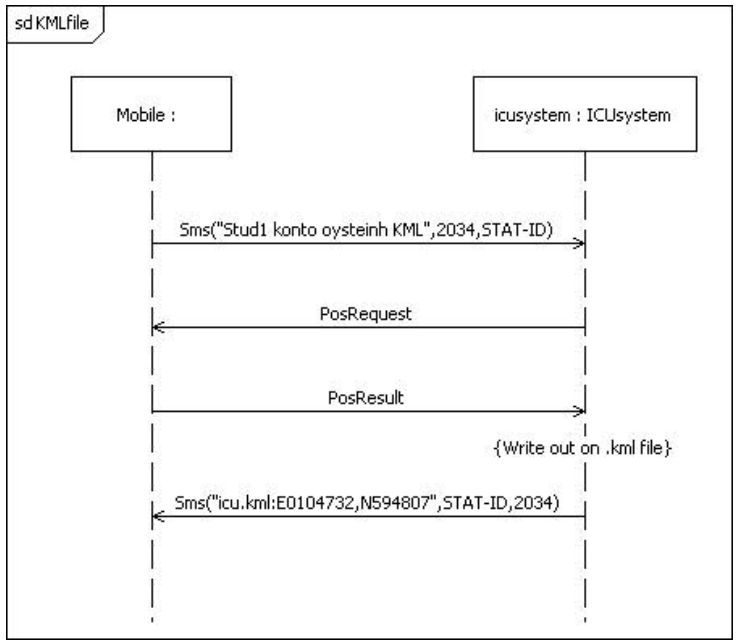






Test behavior and context behavior are similar





INF 5150

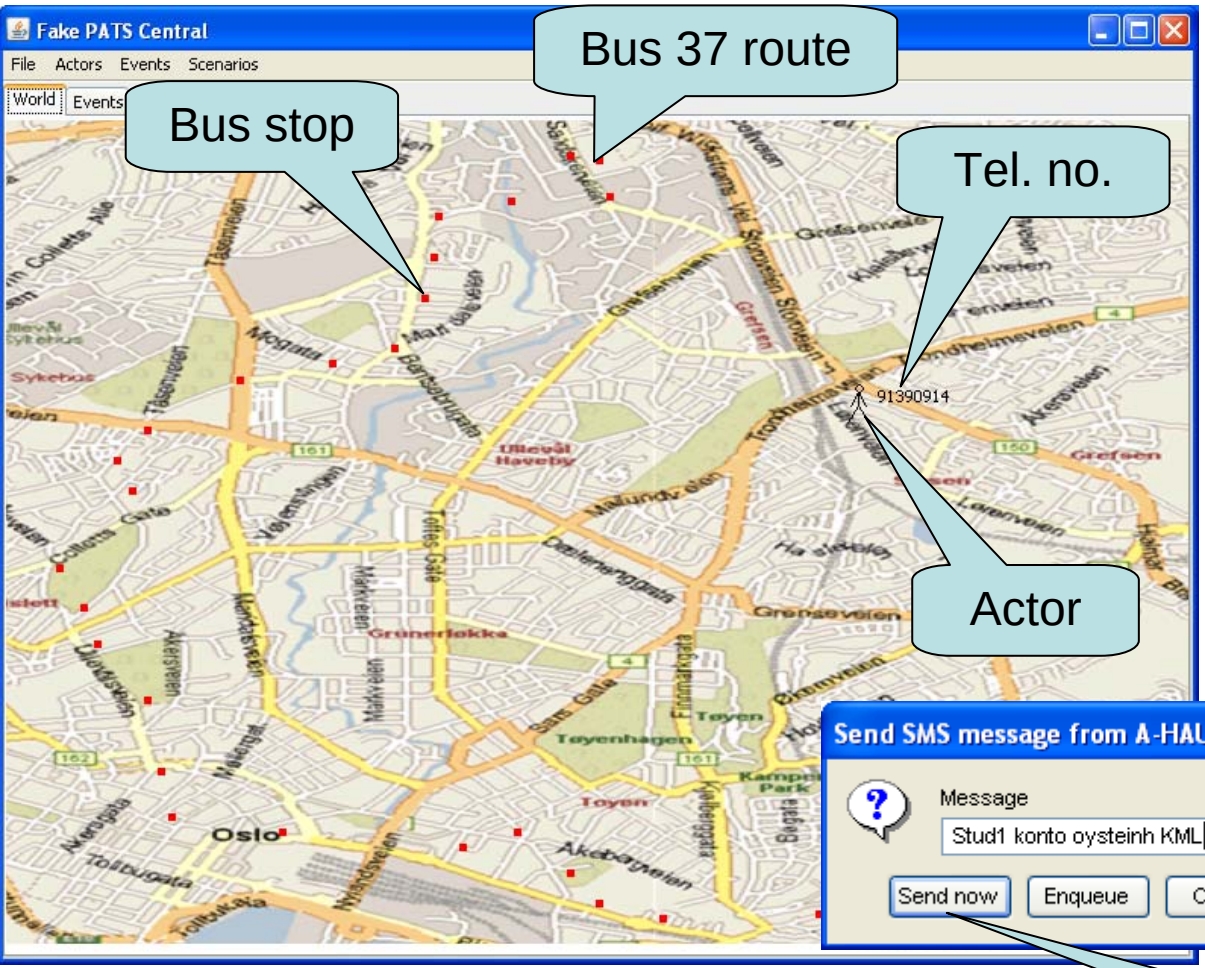
Buzzzz 3: Why both context behavior and tests?

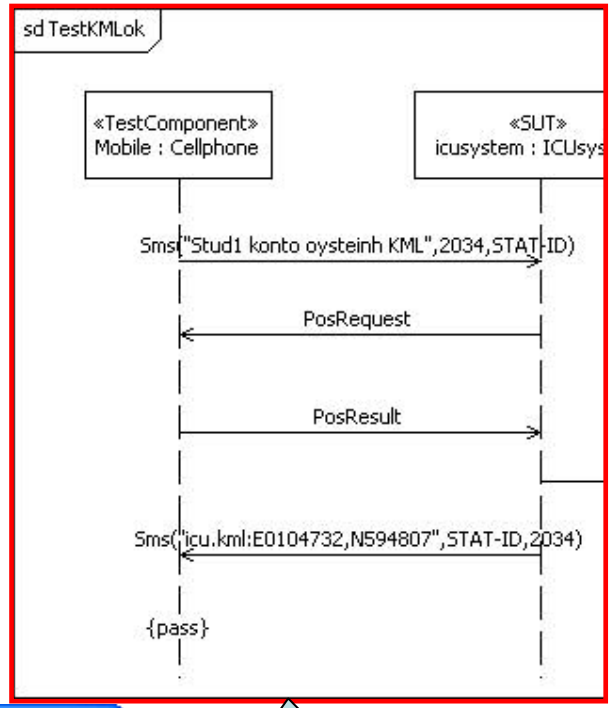
- Why do we need tests when we have context behavior
  - We do not always only want *pass* verdicts
    - we could also use the **neg** fragments in Sequence Diagrams
  - We may want more tests than context behaviors
- Tests should be explicit
  - Identify the SUT and the Test components
    - this distinction 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.jar is also a stand-alone program!



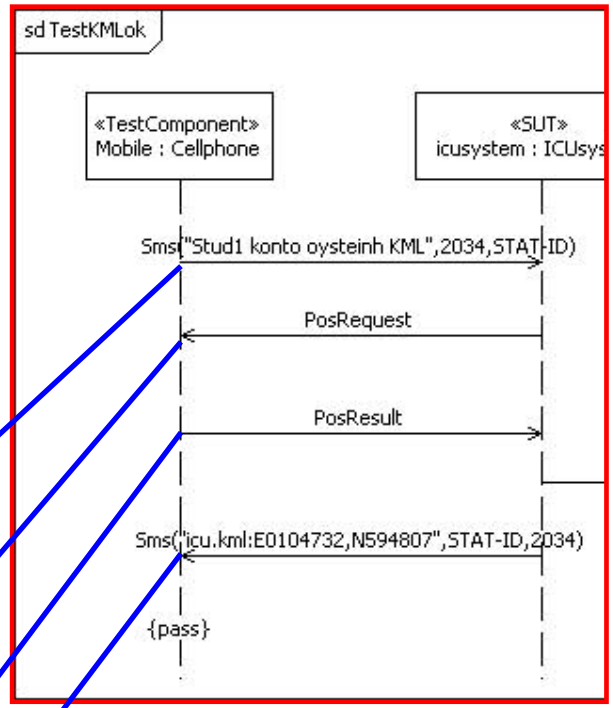


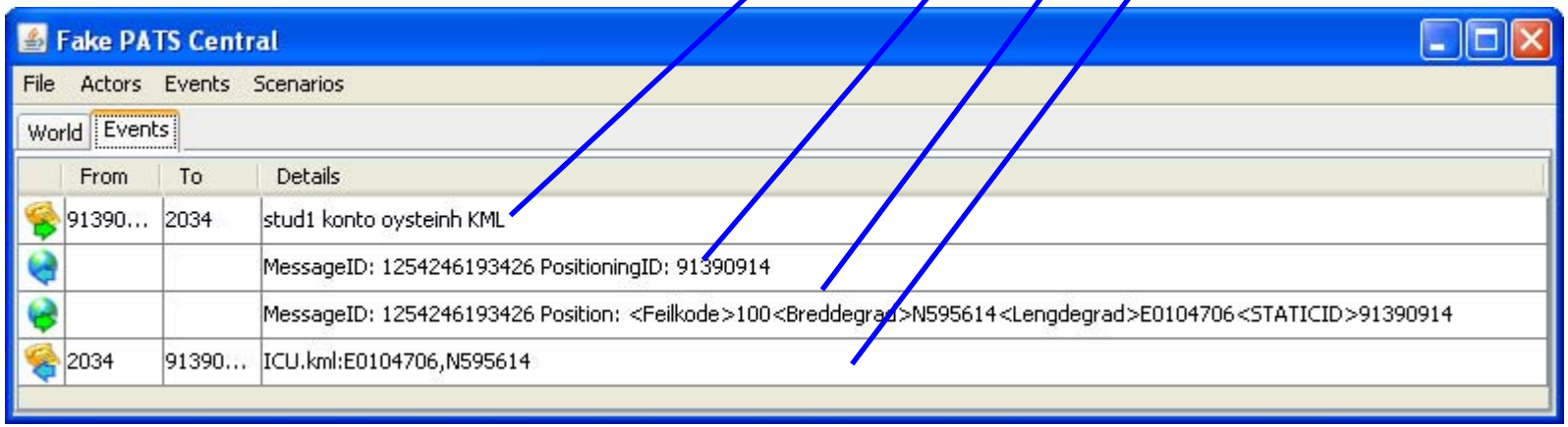
Start fakepats, then application

Send SMS from actor

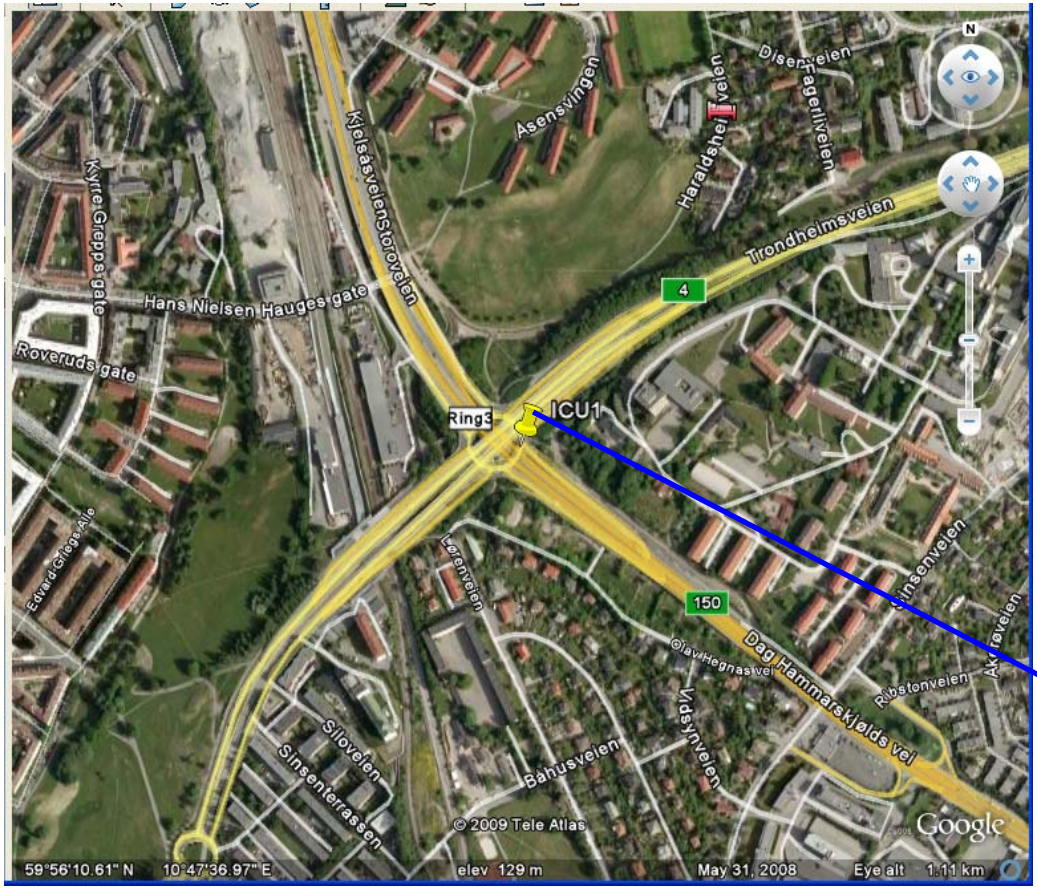
INF 5150

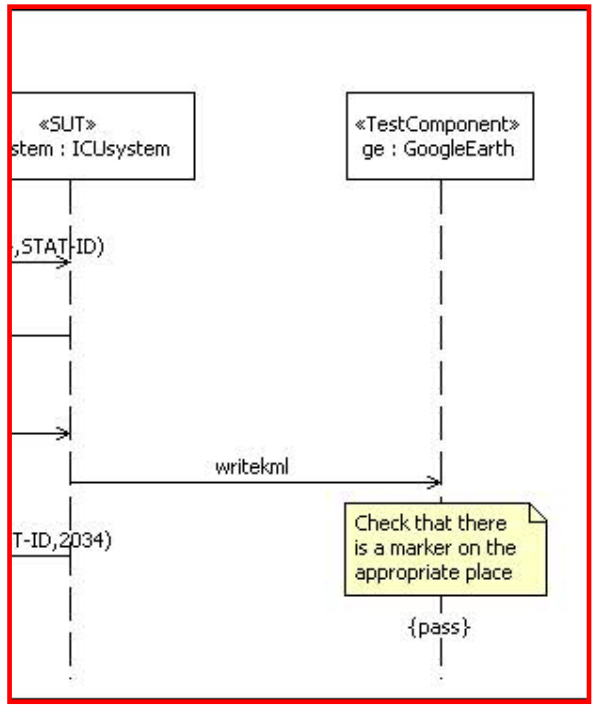
The verdict of the fake mobile

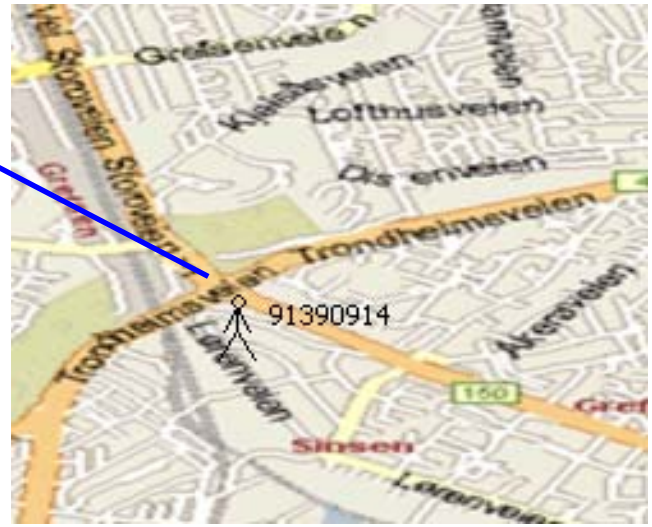




Verdict of GoogleEarth





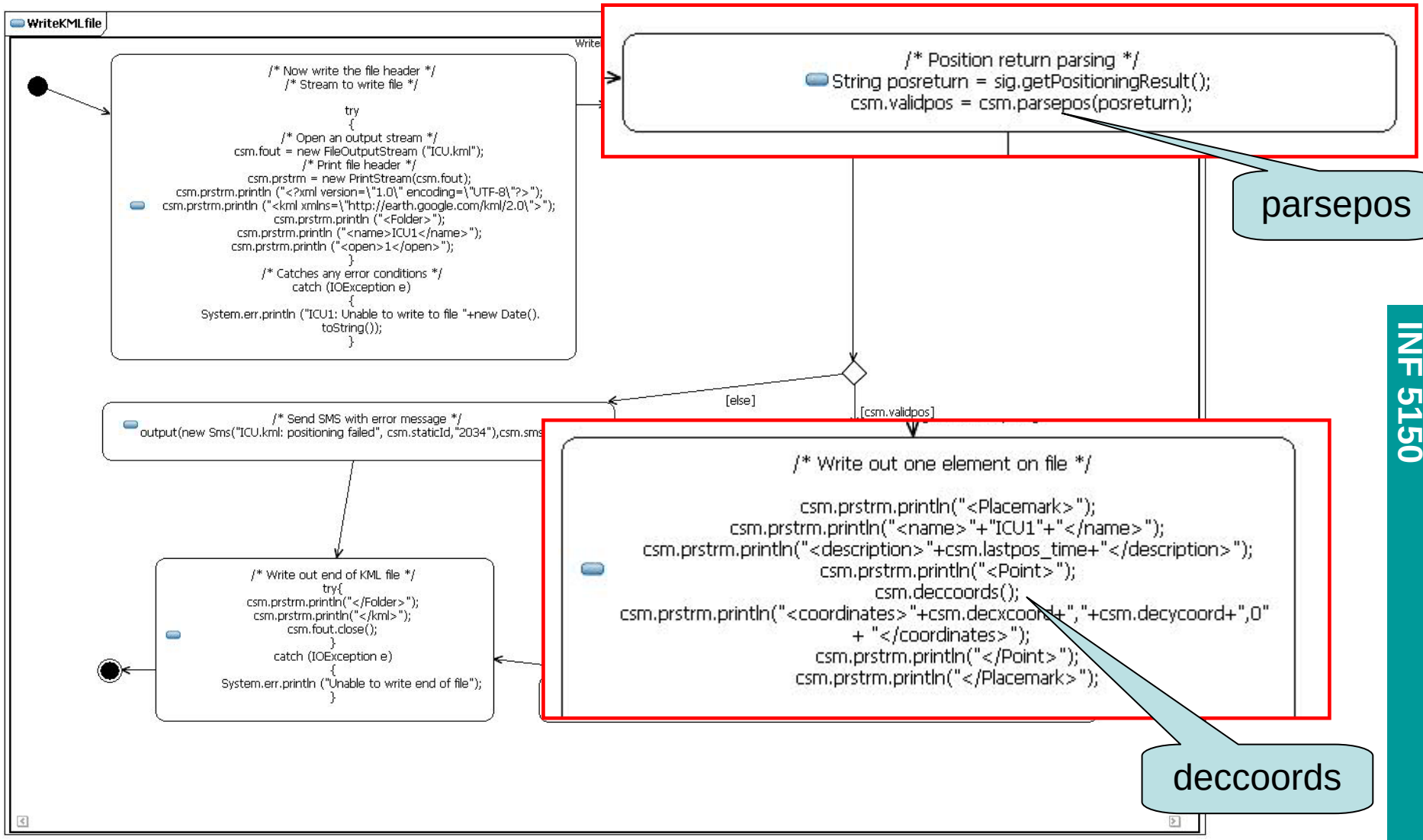


INF 5150

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



INF 5150

UML distinguish between operation and method