UNIVERSITY
OF OSLO

More Testing with U2TP
... and more on routing
... and a few other things

Version 091113

ICU 6-9

INF 5150

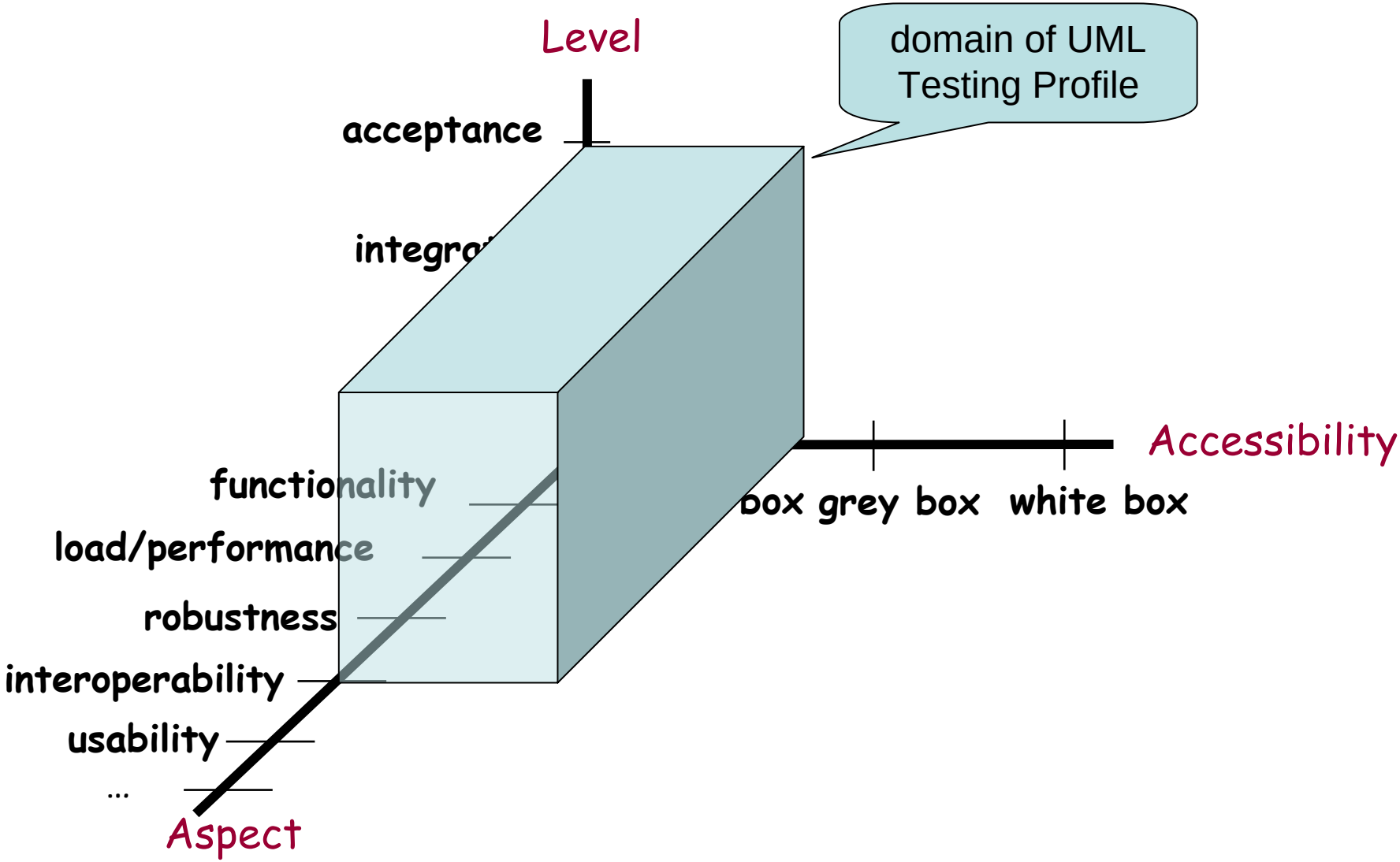
Testing is …

- 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.

Buzz 1: Why Model-driven Testing?

- Spend 2 minutes with one person beside you
- List reasons in favor and against model-driven testing
  - write your reasons down on a piece of paper
  - we shall come back to your reasons somewhat later
- This will probably reveal your prejudices of what model-driven testing is and can be used for

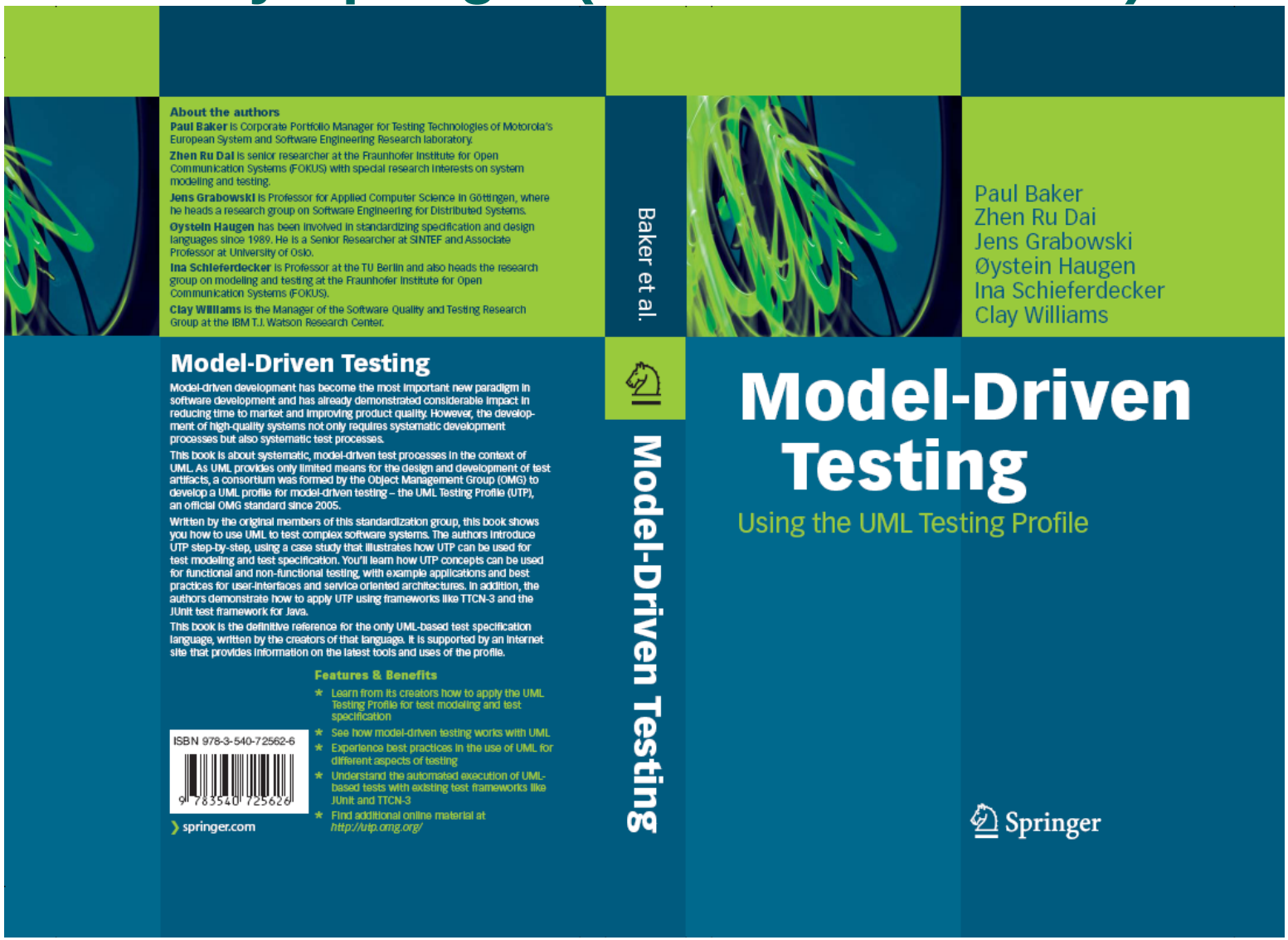
Types of Testing



UML Testing Profile

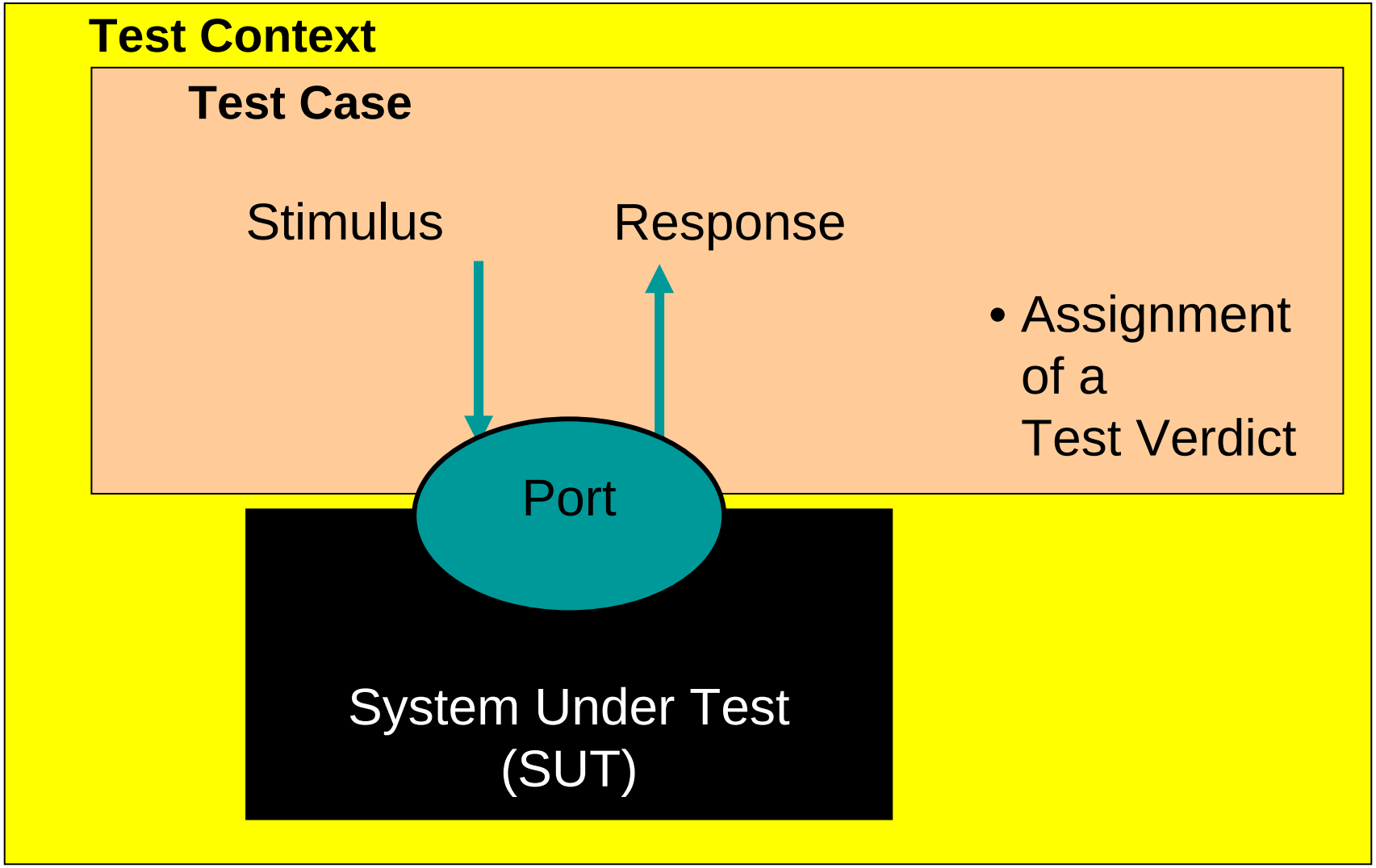
- To allow black-box testing (i.e. at UML interfaces) of computational models in UML
- A testing profile based upon UML 2.0
  - That enables the test definition and test generation based on structural (static) and behavioral (dynamic) aspects of UML models, and
  - That is capable of inter-operation with existing test technologies for black-box testing
- Standardized profile recommended by OMG
  - http://www.omg.org/cgi-bin/doc?formal/05-07-07

A book by Springer (978-3-540-72562-6)

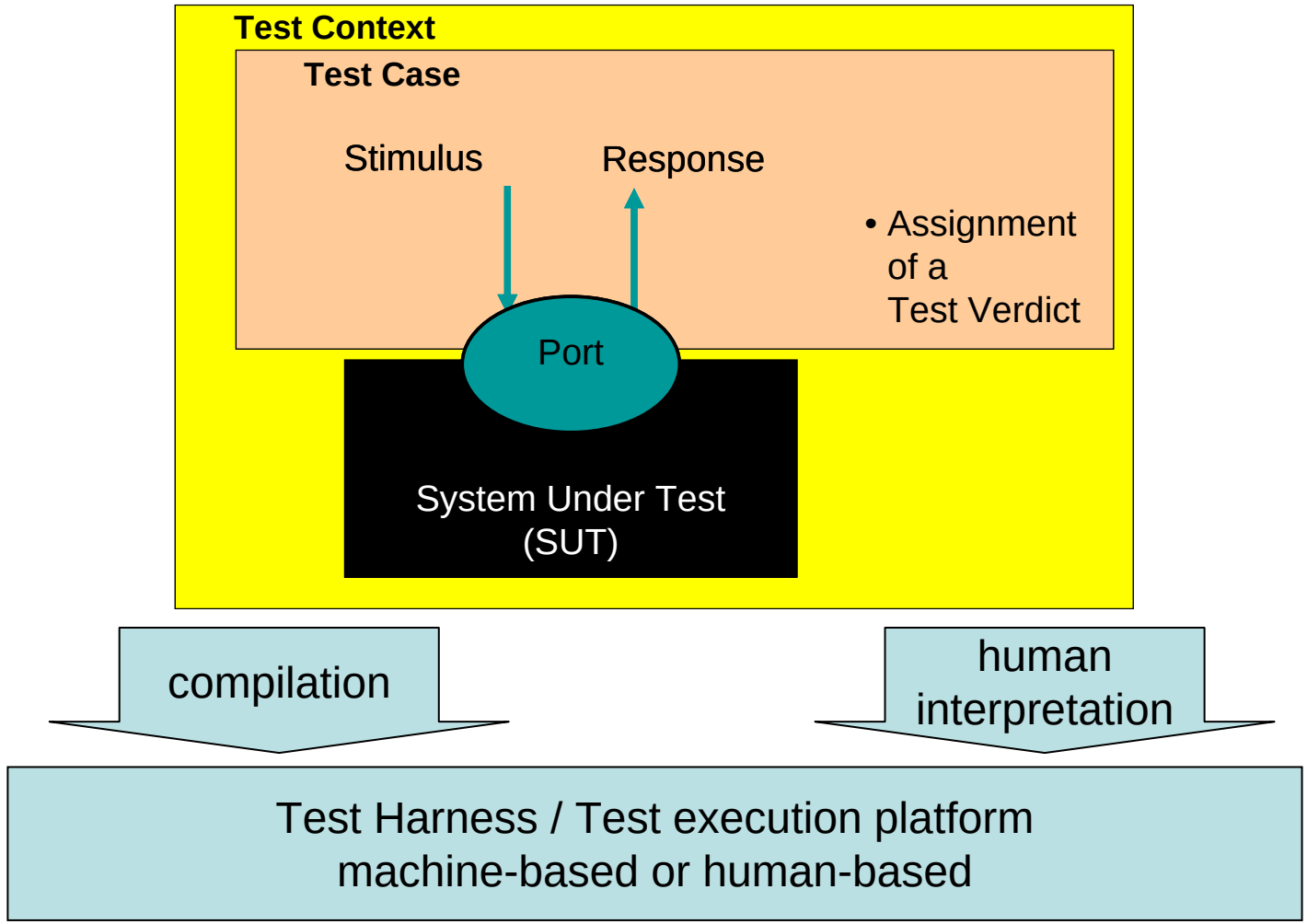


INF 5150

Test Concepts: Black-Box Testing



Test Execution



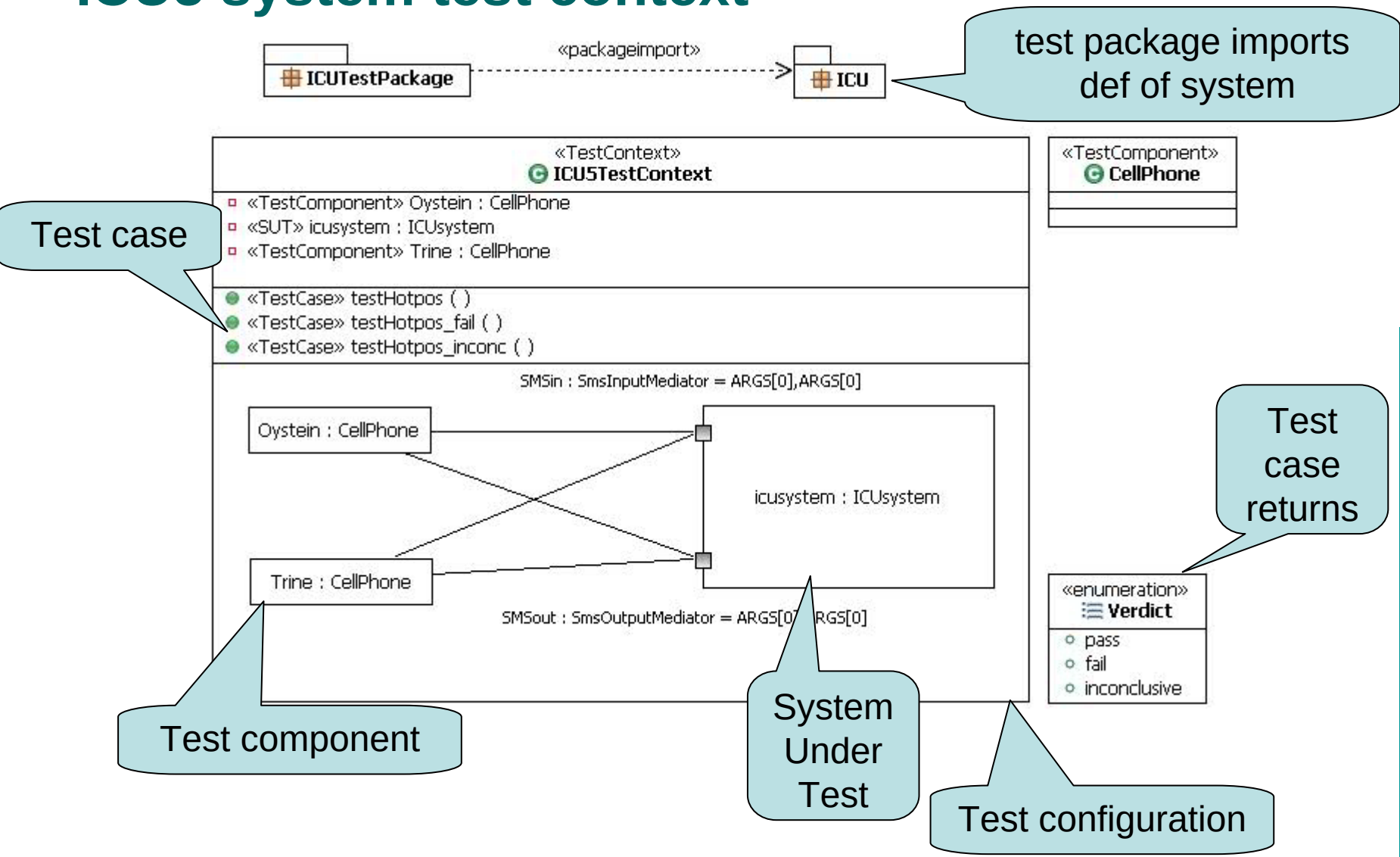
Unit Level Testing

- For unit level testing, the SUT is the smallest unit, e.g. a class or an operation, which is to be tested.

System and Acceptance Level Testing

- The goal of system level testing is to verify that the system under test conforms to its requirements.
- Acceptance testing is basically an extension of system testing in which requirements and the test cases associated with validating them are developed by or with the customer(s) and are considered to be contractual descriptions of the required system behavior.

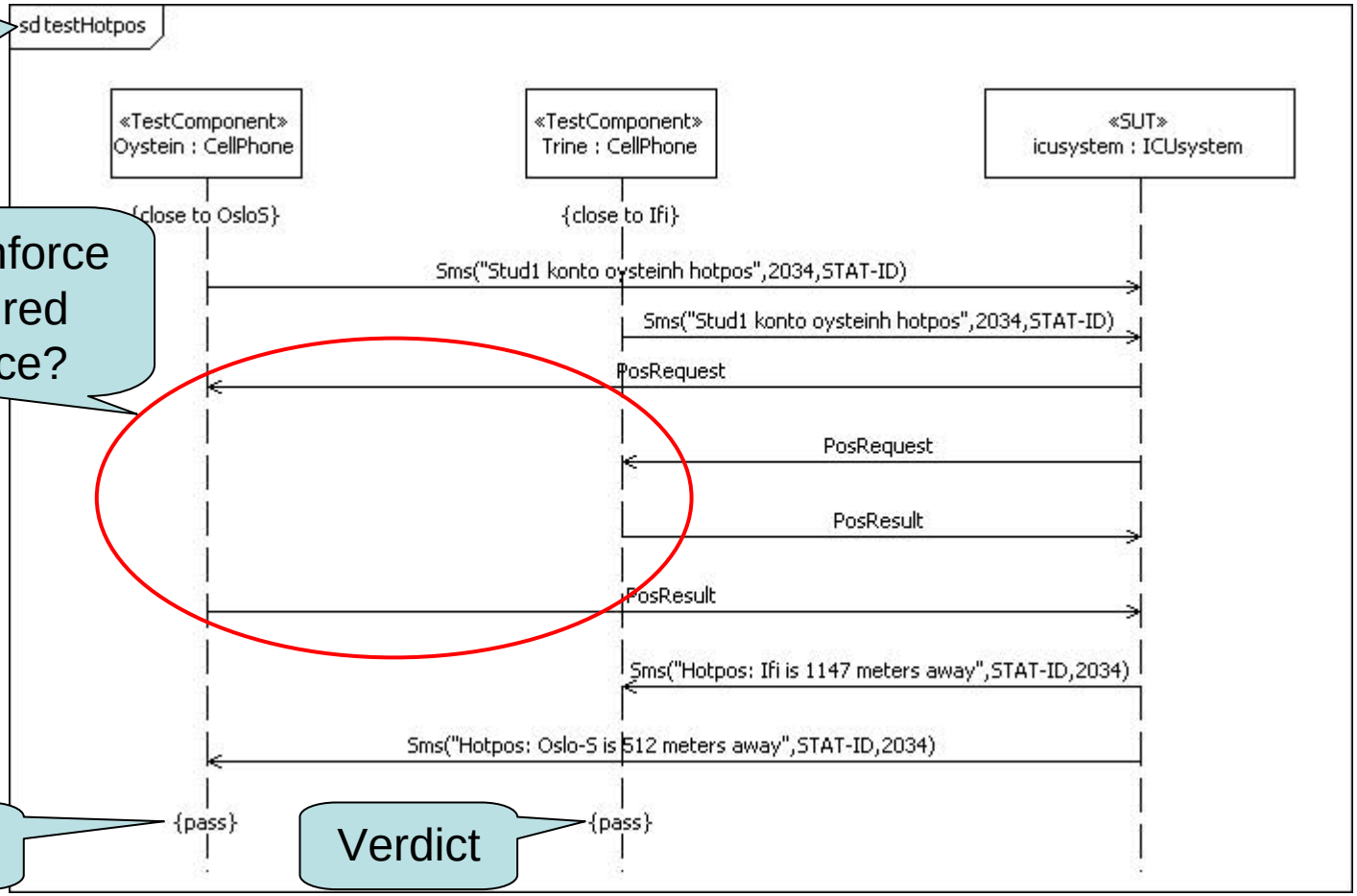
ICU5 system test context



Test control problems

Verdict?

How to enforce the desired sequence?

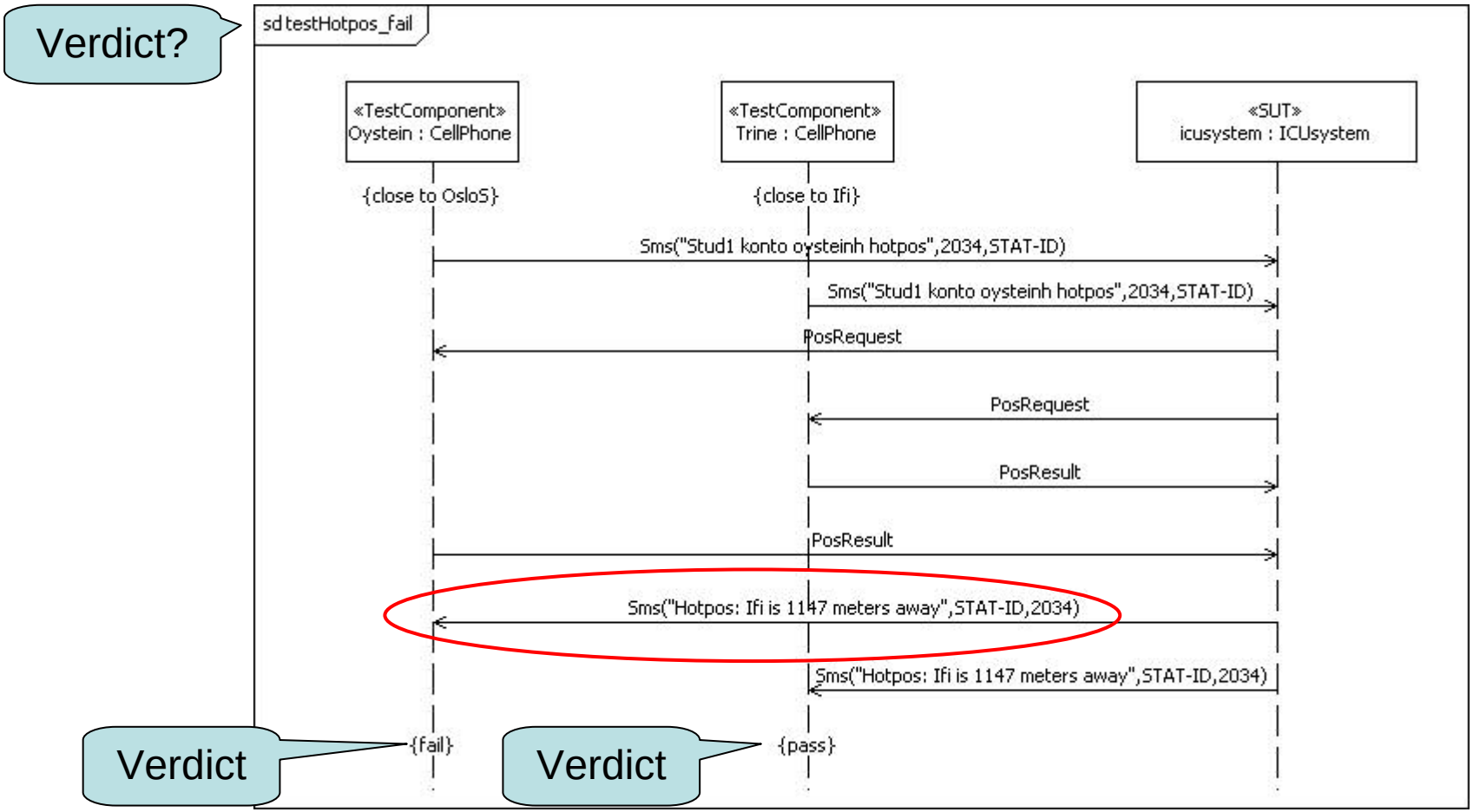


Verdict

Verdict

INF 5150

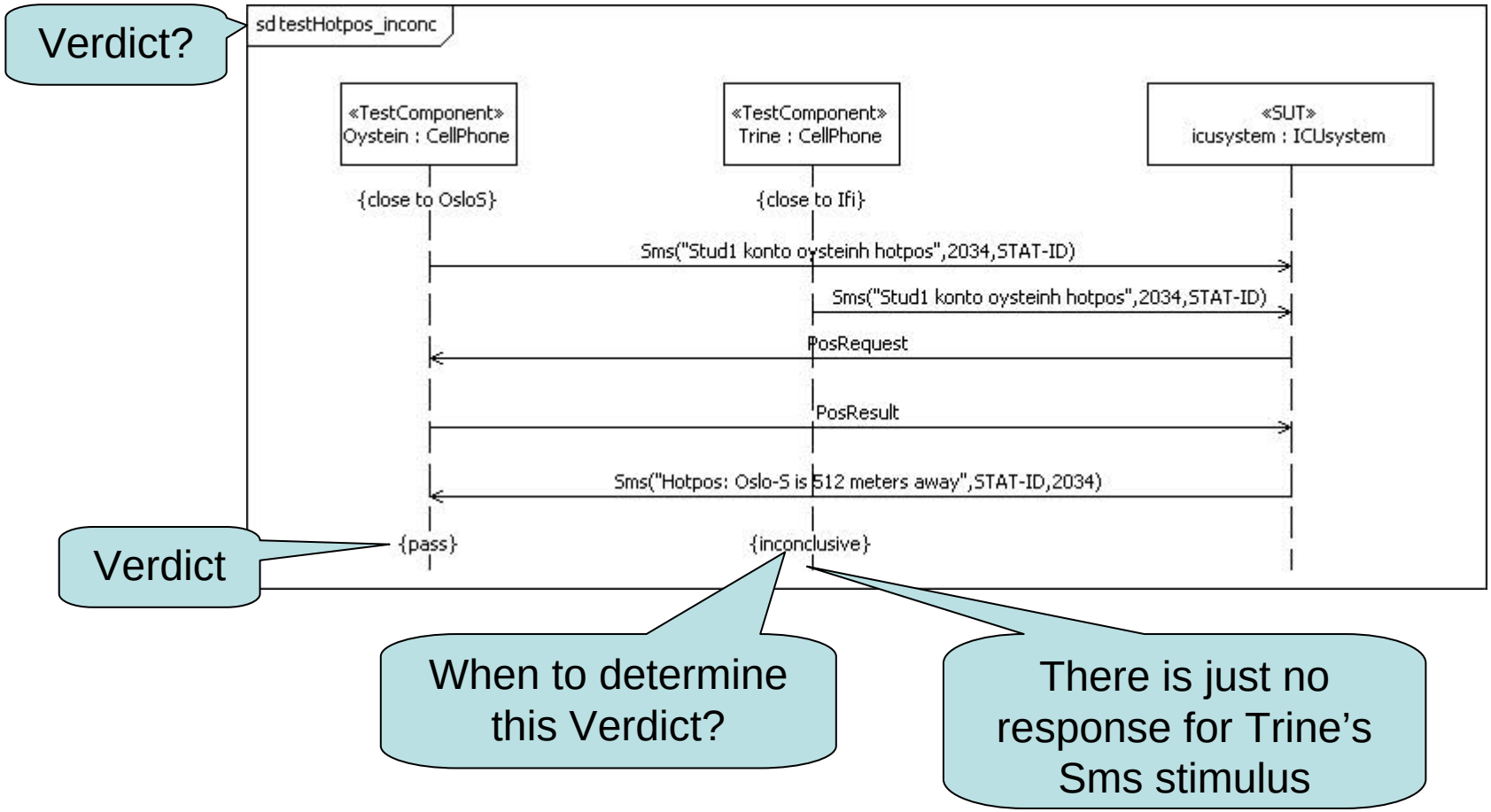
Arbitration (1)



Arbitration (2)

- One *fail* is normally enough to force a *fail* test case
  - *pass, inconclusive, fail* is ordered such that the least value is the final resulting verdict
- Arbitration can also be defined by the user
  - possibly one single *fail* should not suffice to force a full failure

Arbitration (3)



Test strategy – acceptance test

- validate that the functionality of the system is correct with respect to the requirements
- validate the non-functional (extra-functional) properties
  - Performance testing
    - to see if a system can meet its expected response times under typical workloads.
  - Load testing
    - to determine whether the system suffers from resource problems (memory leaks, buffer problems) or otherwise undergoes degradation
  - Stress testing
    - to determine how gracefully it may recover from extreme situations
  - Reliability
    - the probability of correct operation of a piece of software for a specified amount of time.
- validate that the supported software distribution and deployment configurations are correctly supported

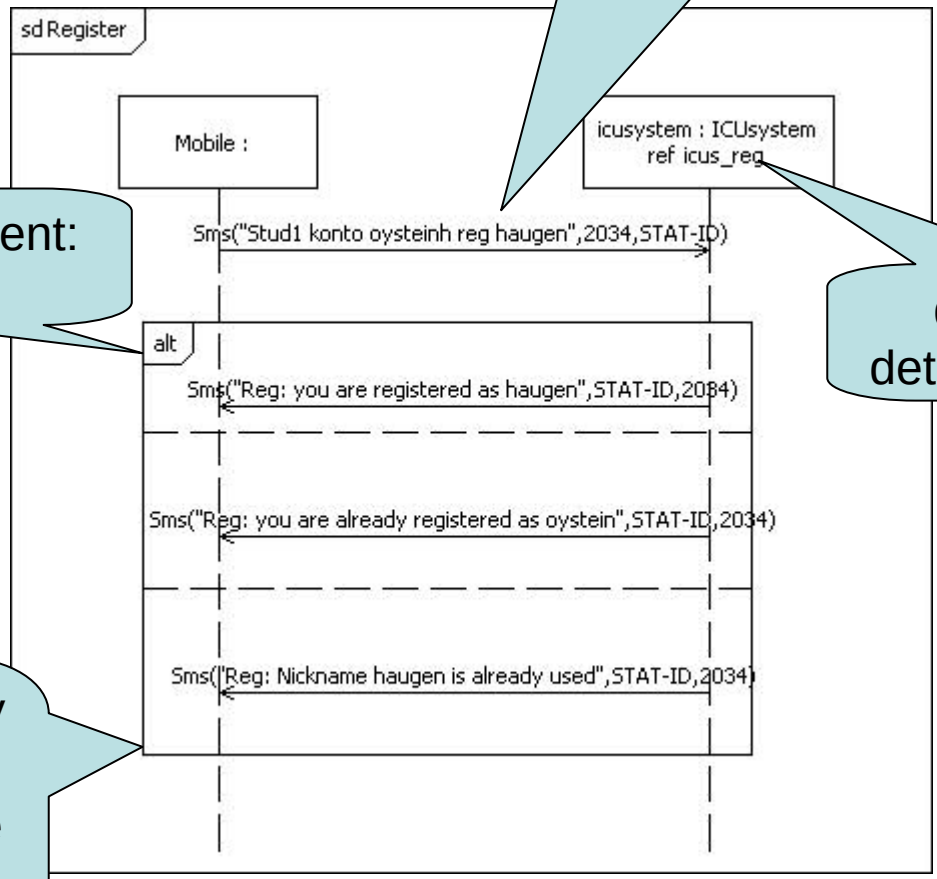
Buzz 2: Testing strategy – advanced tests (Buzz 5 min)

- Challenges
  - How to test systems that have non-deterministic behavior?
    - due to concurrency
  - How to test time requirements
    - test probes changes the timing of the system
  - How to simulate extreme load situations
    - 1 million SMS-messages within a minute

Dynamic Data (still transient)
Signal hierarchies
Combined Fragments of Sequence Diagram

Register users

Associate the nickname ”haugen” with the static id ”STAT-ID”

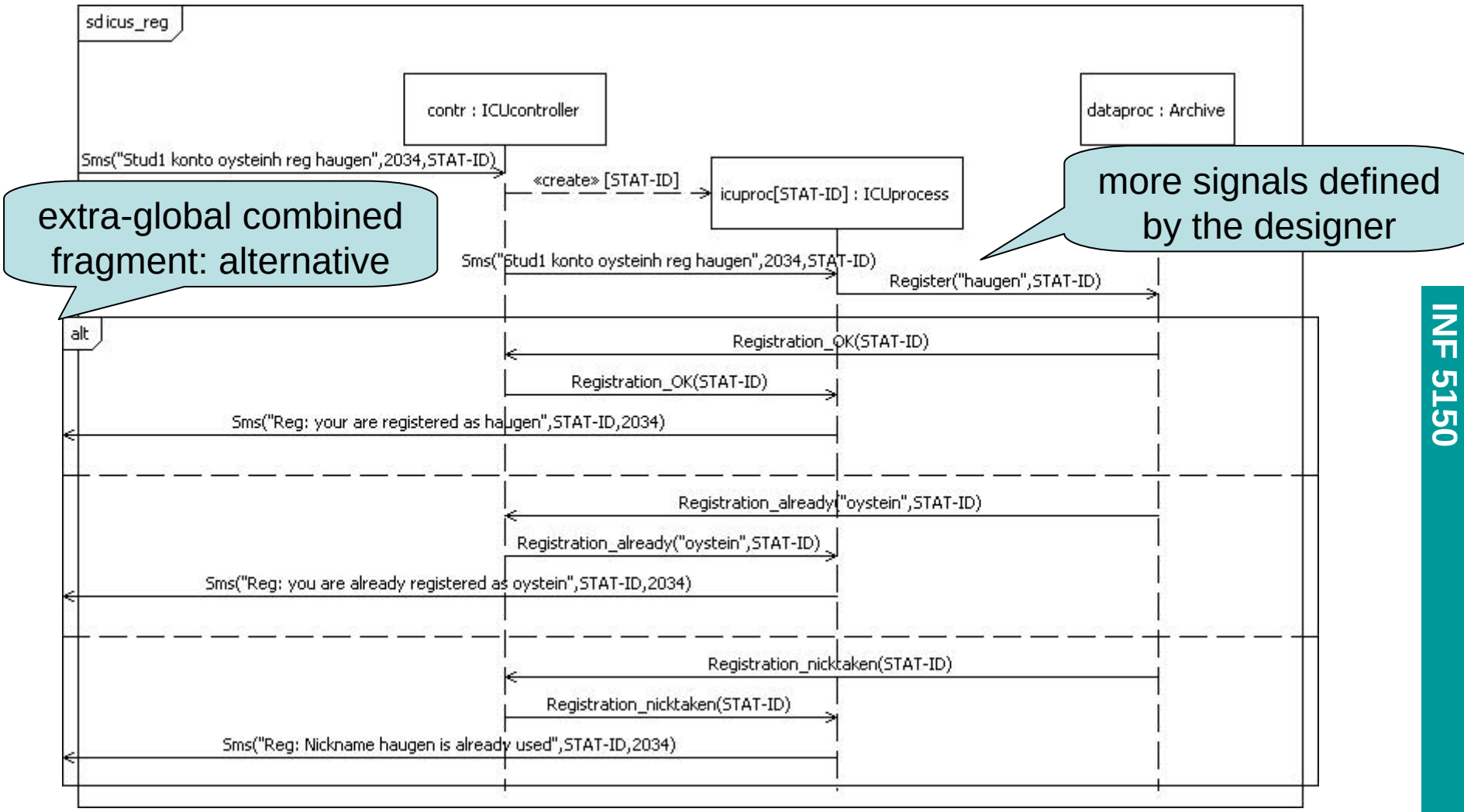


combined fragment: alternative

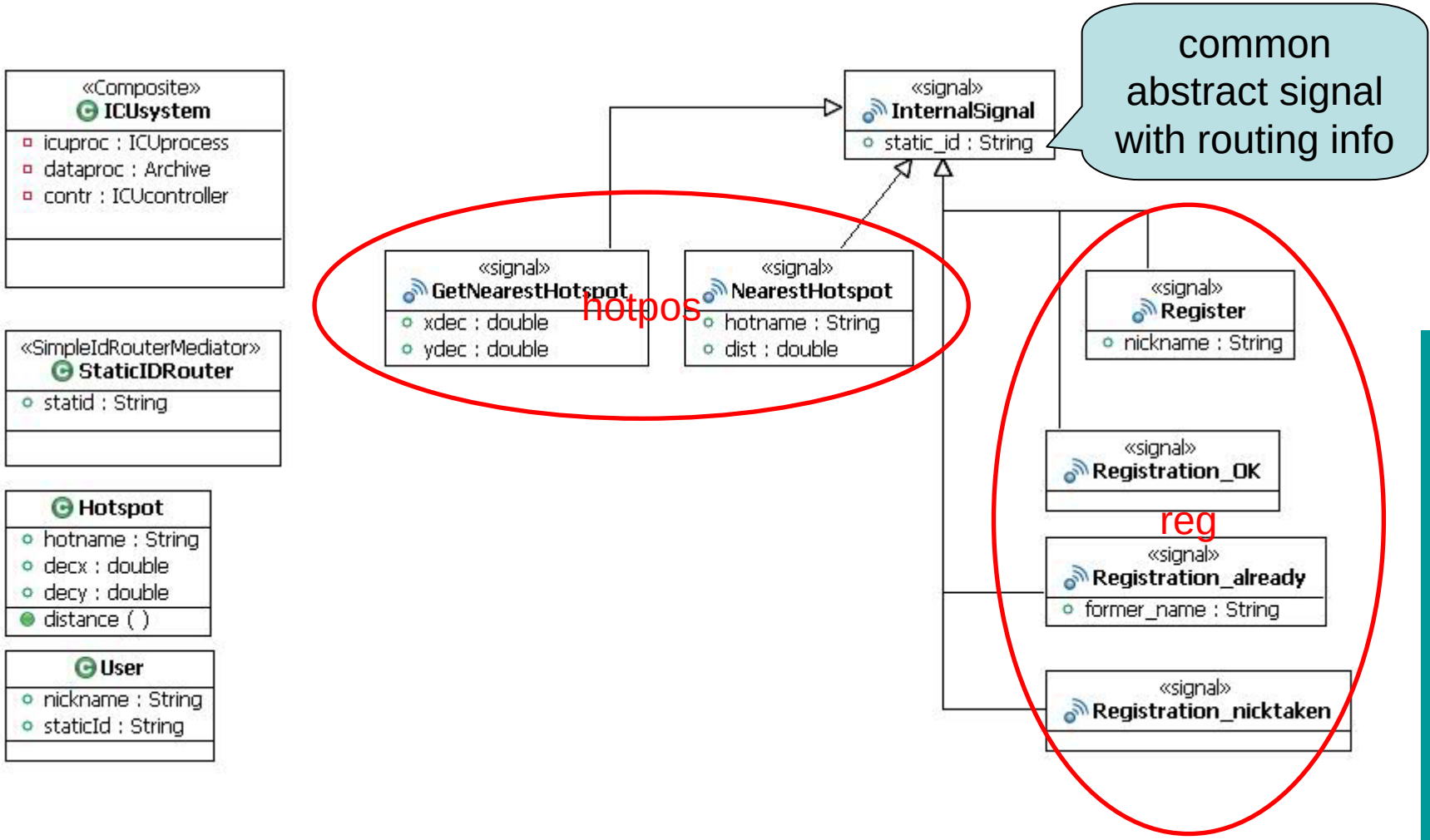
decompose for detailed specification

even though we may describe alternative execution traces, we are not obliged to describe them all!

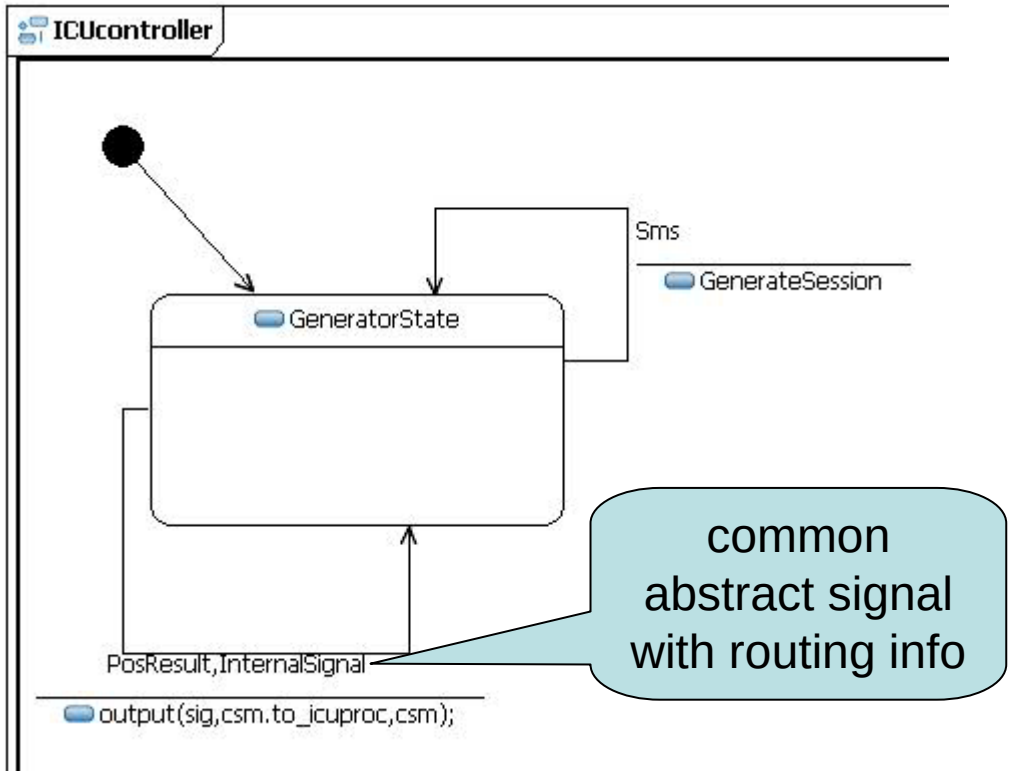
Register users - decomposition



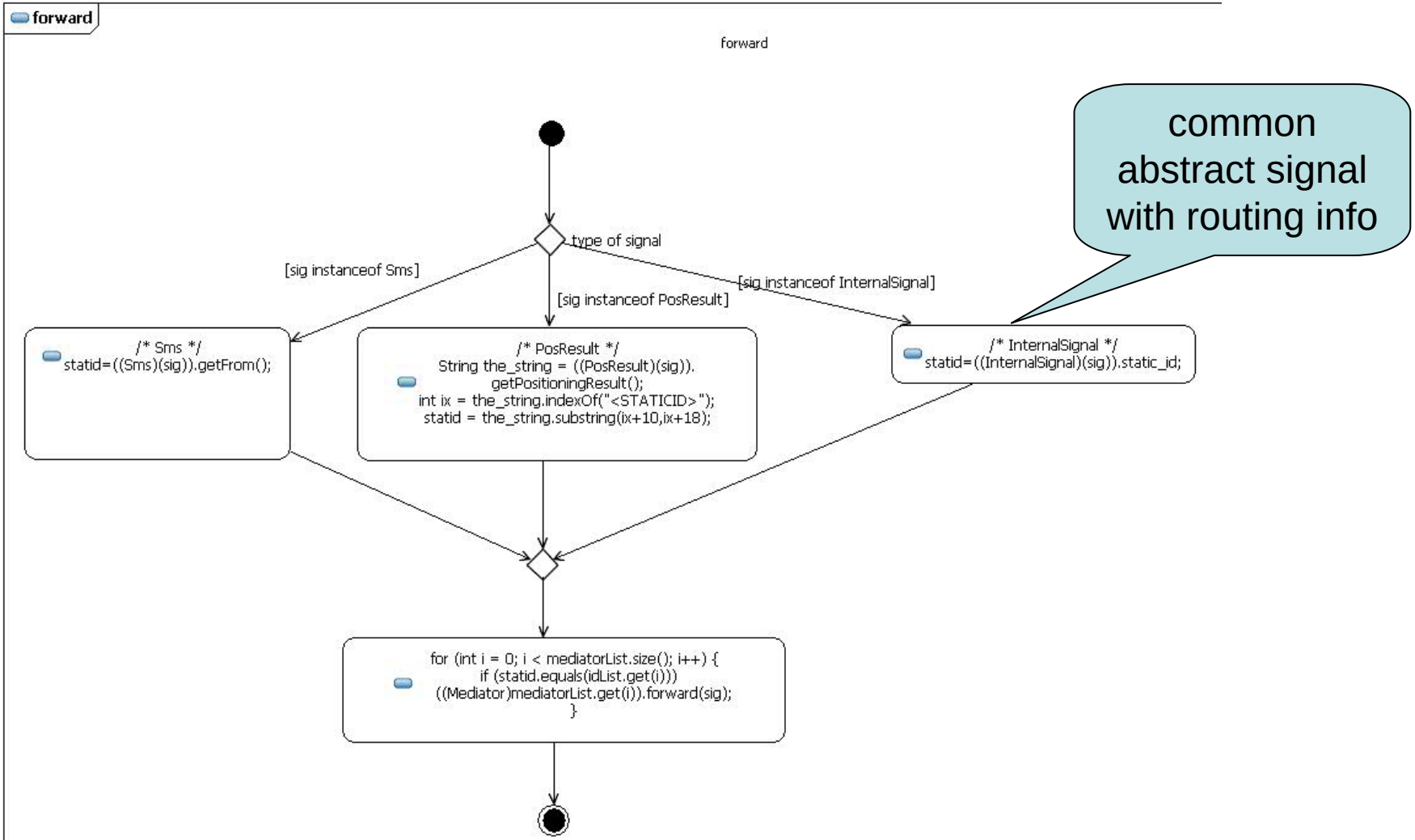
Signal hierarchy – normal object orientation



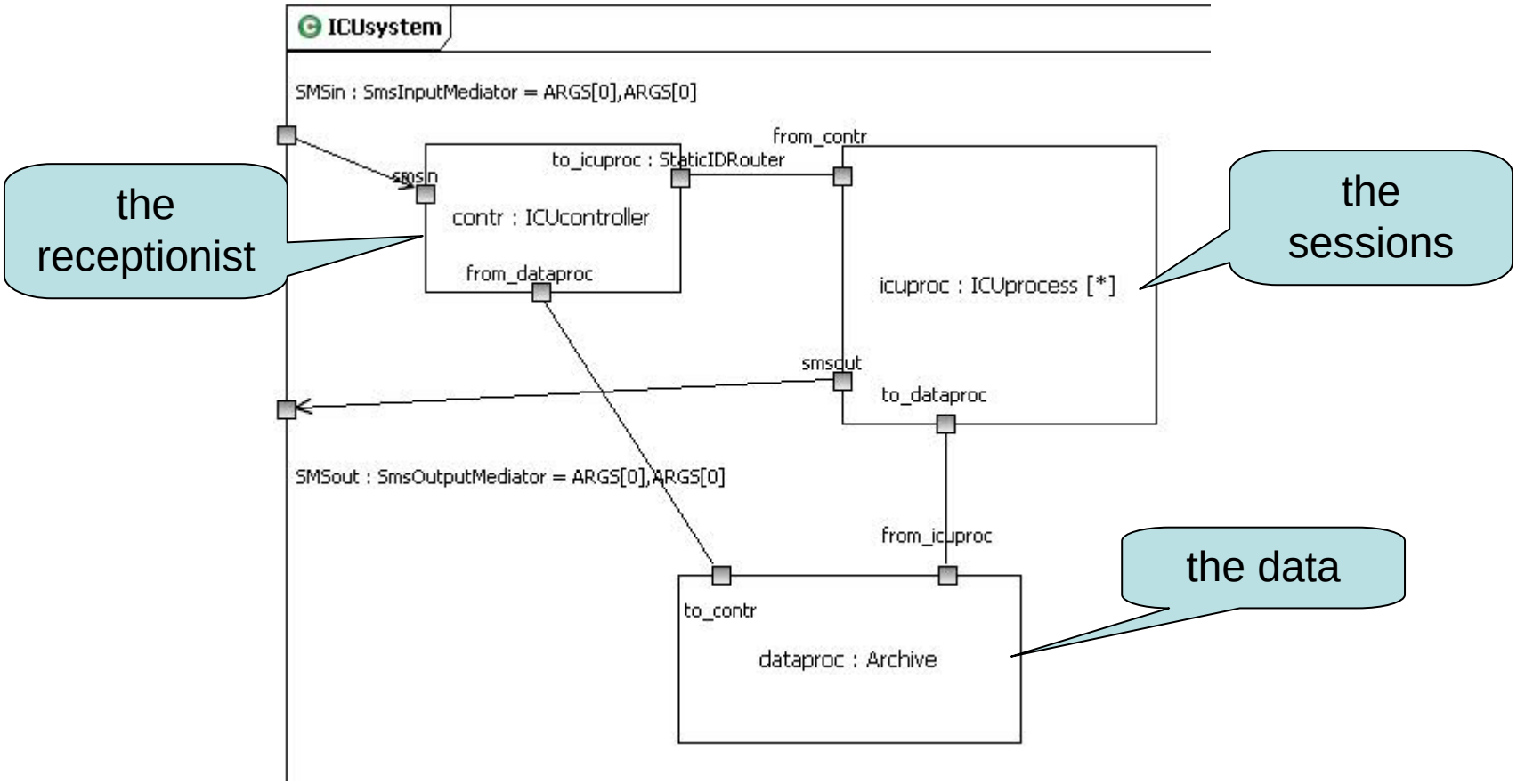
Effect on routing (1)



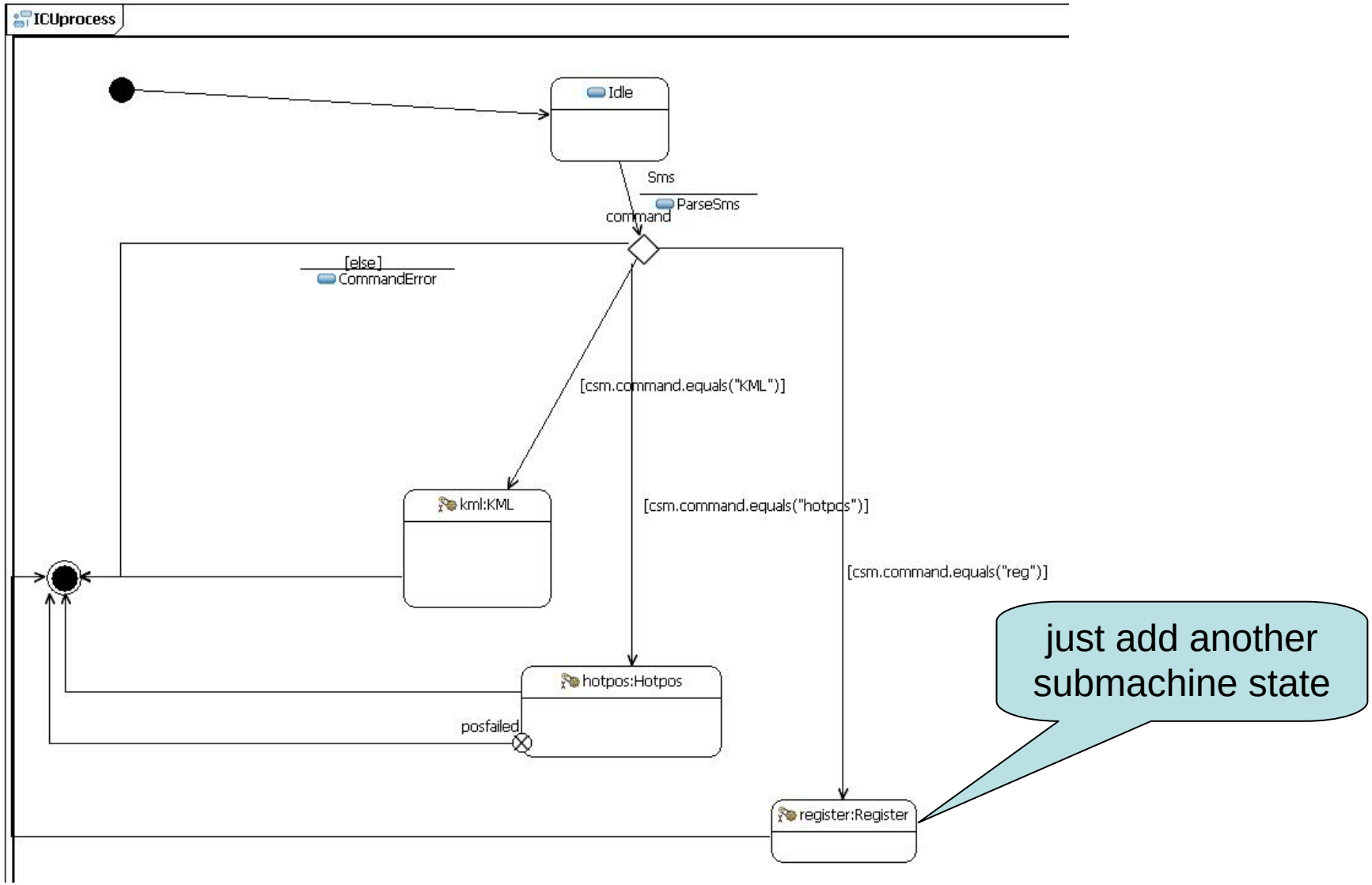
Effect on routing (2)



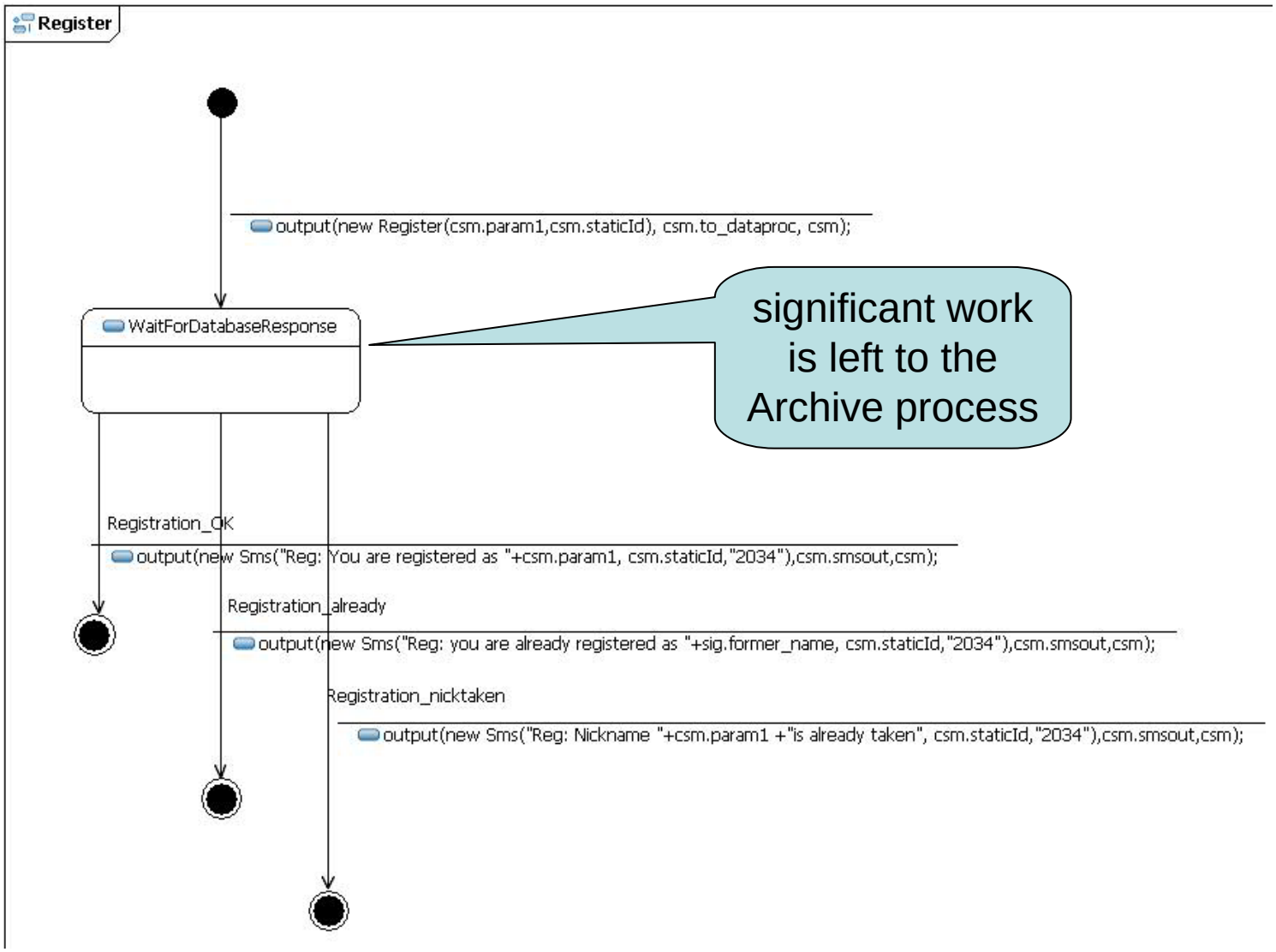
Repeating our simple composite structure



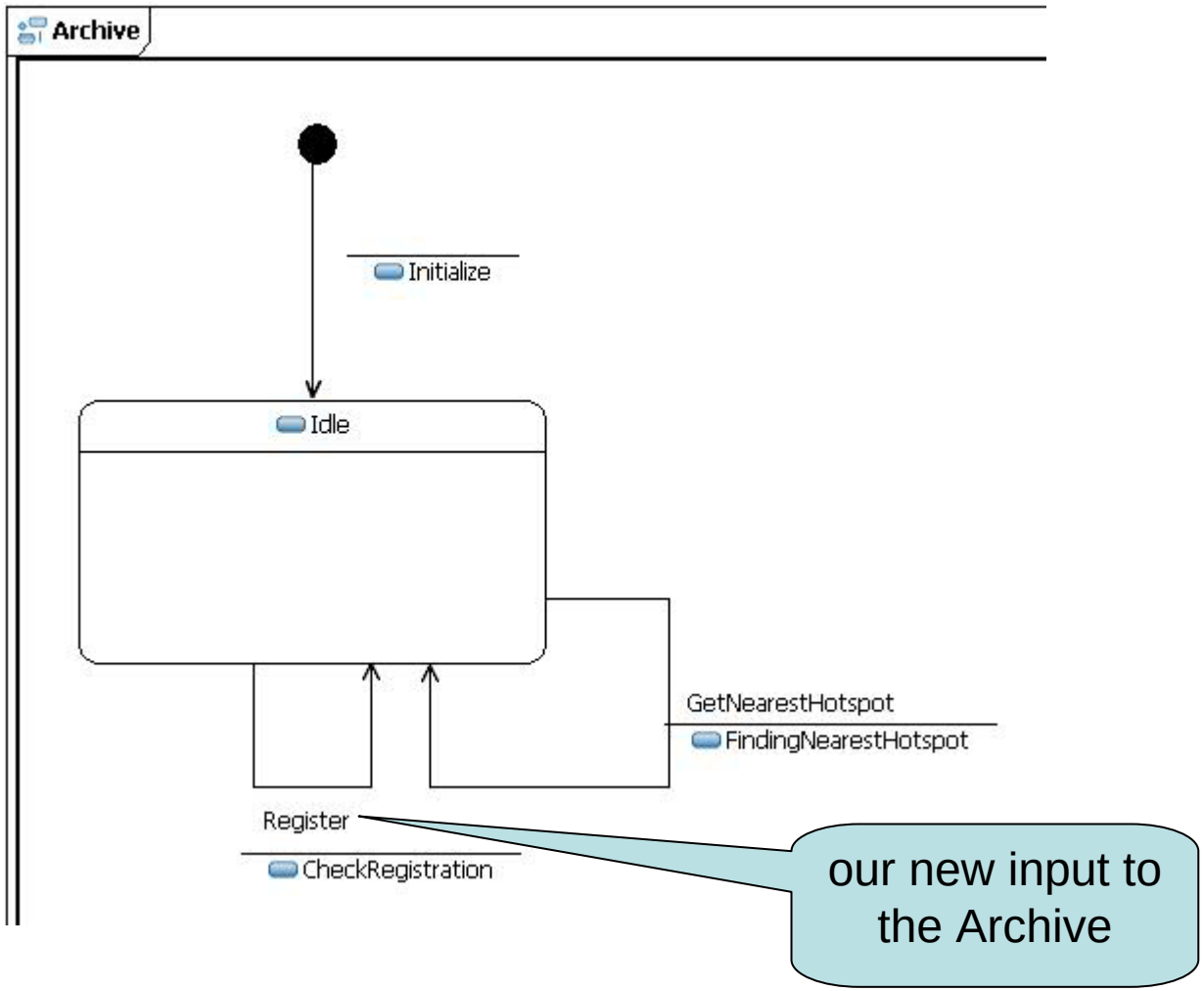
Adding a new service is no sweat!



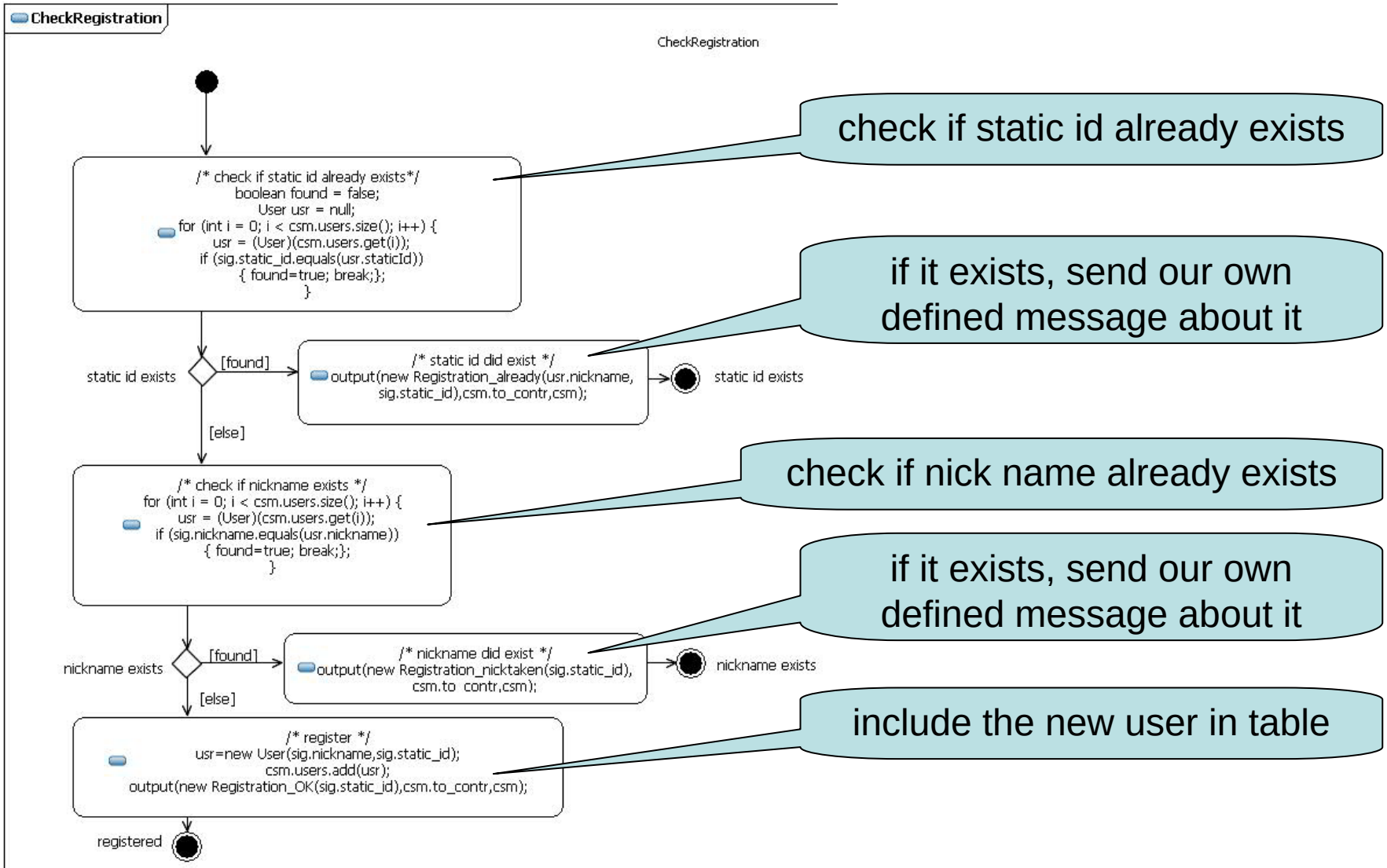
The new state machine



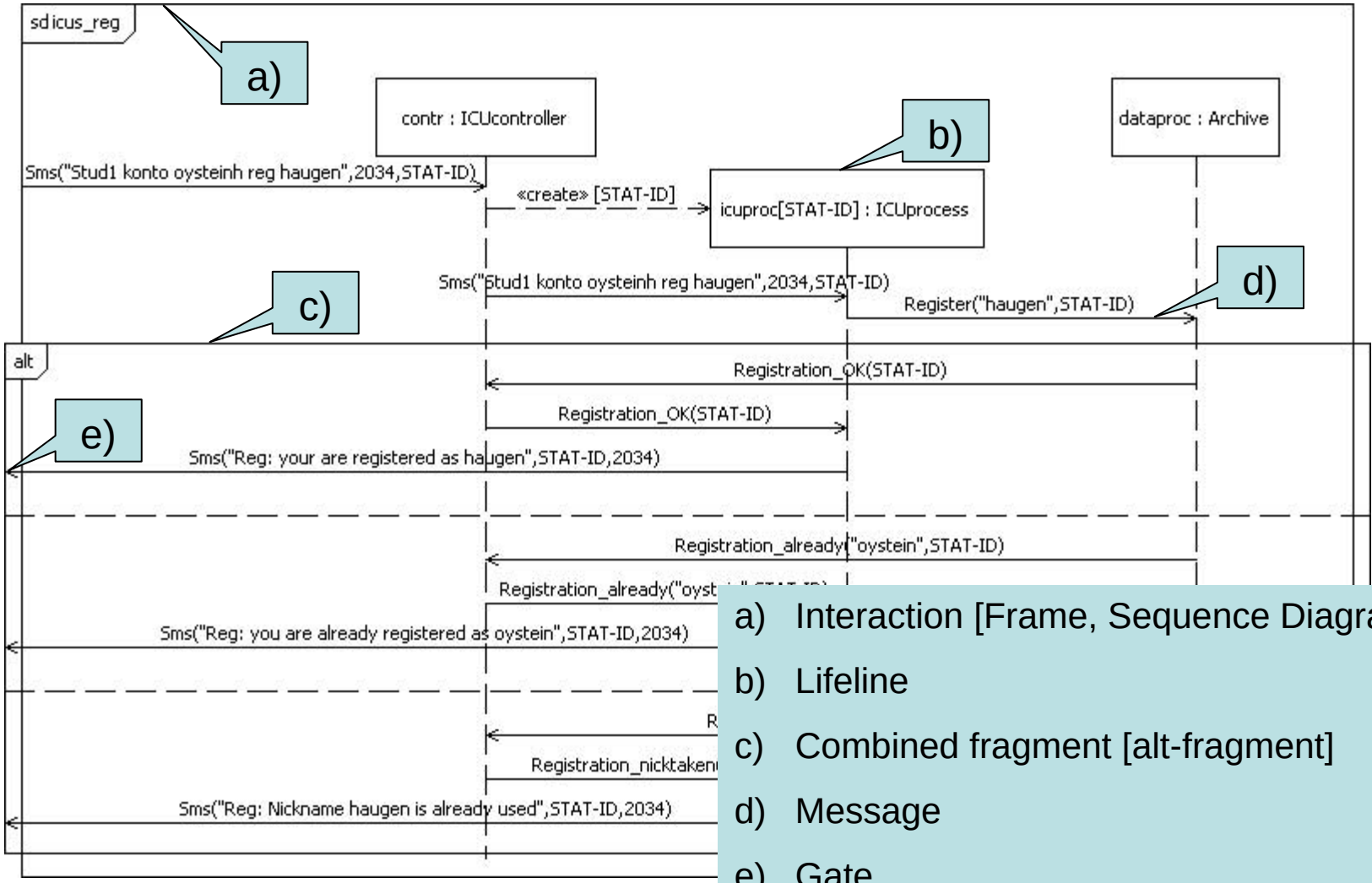
The Archive process enhanced



Check registration transition



Write down the names of these UML concepts



a) Interaction [Frame, Sequence Diagram]

b) Lifeline

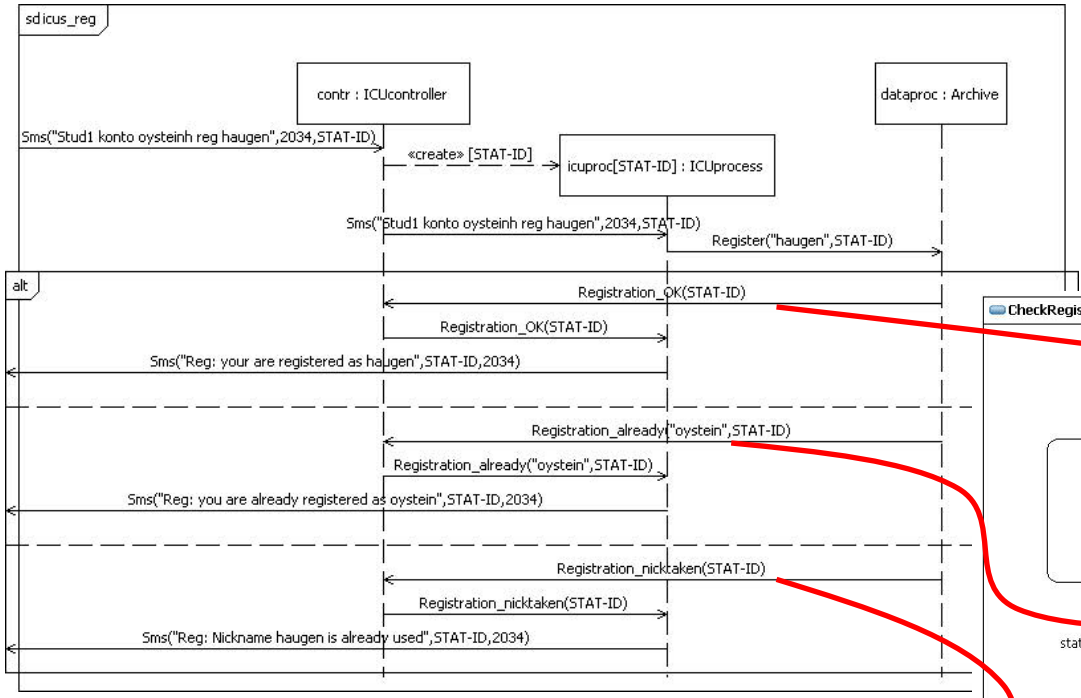
c) Combined fragment [alt-fragment]

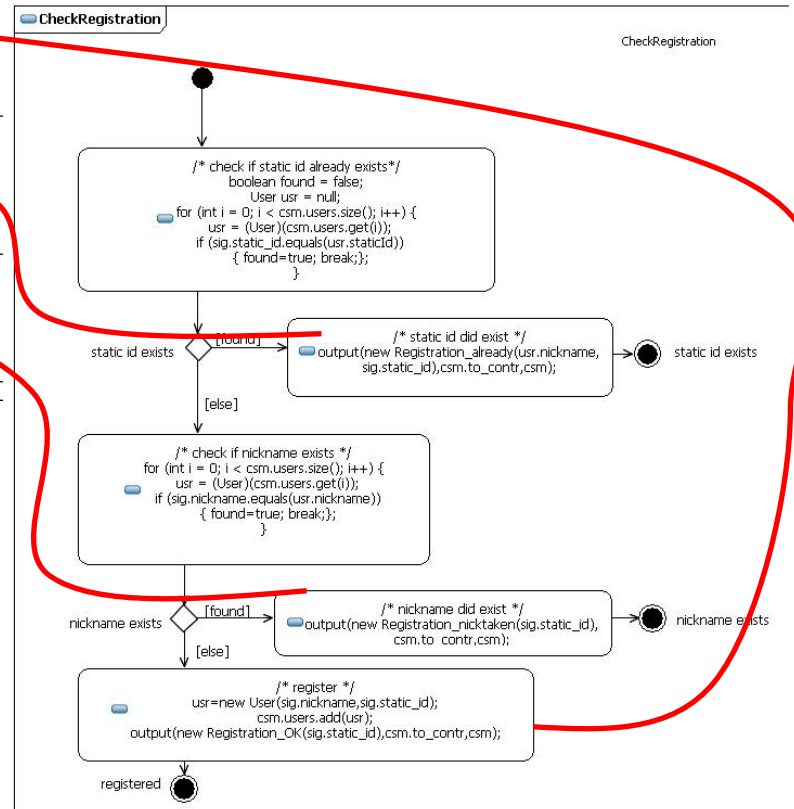
d) Message

e) Gate

INF 5150

Check consistency with spec!





INF 5150

Summary of adding the *reg* service

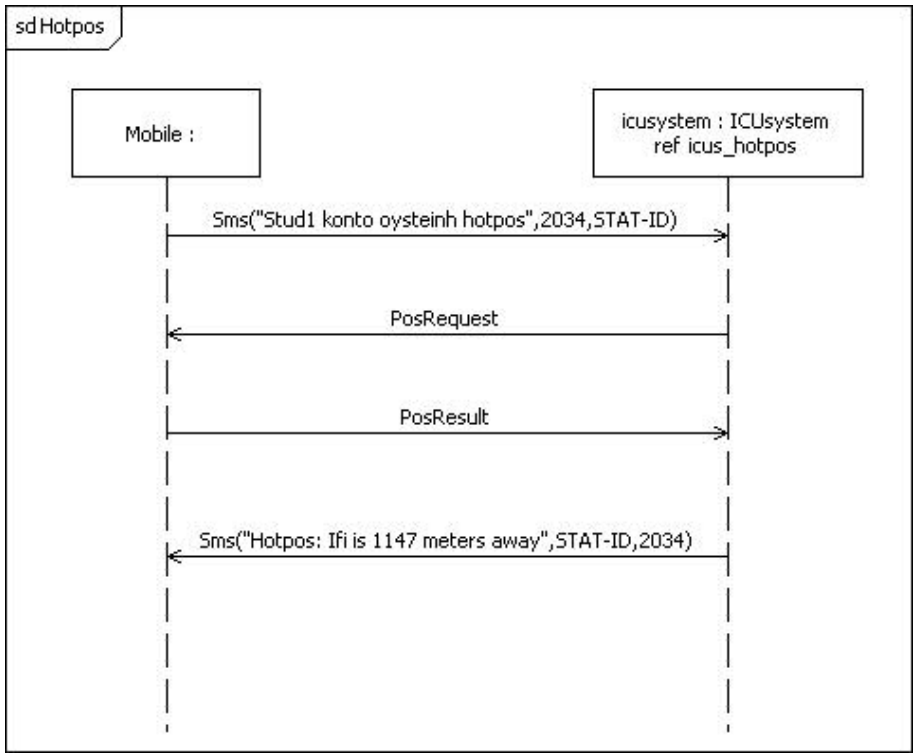
- Specify what the new service is supposed to do
  - use case in prose
  - sequence diagrams on context and detailed levels
- Define necessary internal signals
  - make sure routing will be performed properly
- Define a new submachine state in the session process
- Define the corresponding state machine
  - this may involve the data process
    - add new transitions with new data operations
- Notice that the old system is hardly changed!
  - there are only additions
- Check the consistency between specification and design

On Routing
- and a few other lessons

Lessons to be learned now

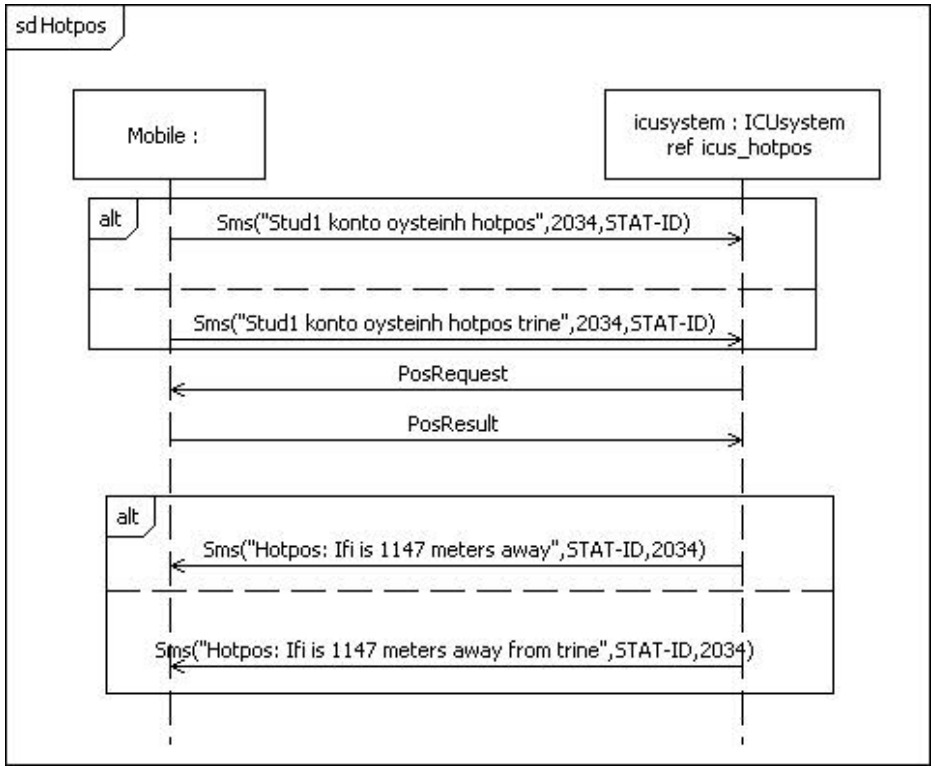
- Small changes to the user's specification may result in rather far reaching effects on the software
- 3rd party software interface can be quite important
- Routing can be done several ways
- Agile modeling normally involves re-engineering
  - that sometimes may become rather fundamental

Hotpos – as of ICU6



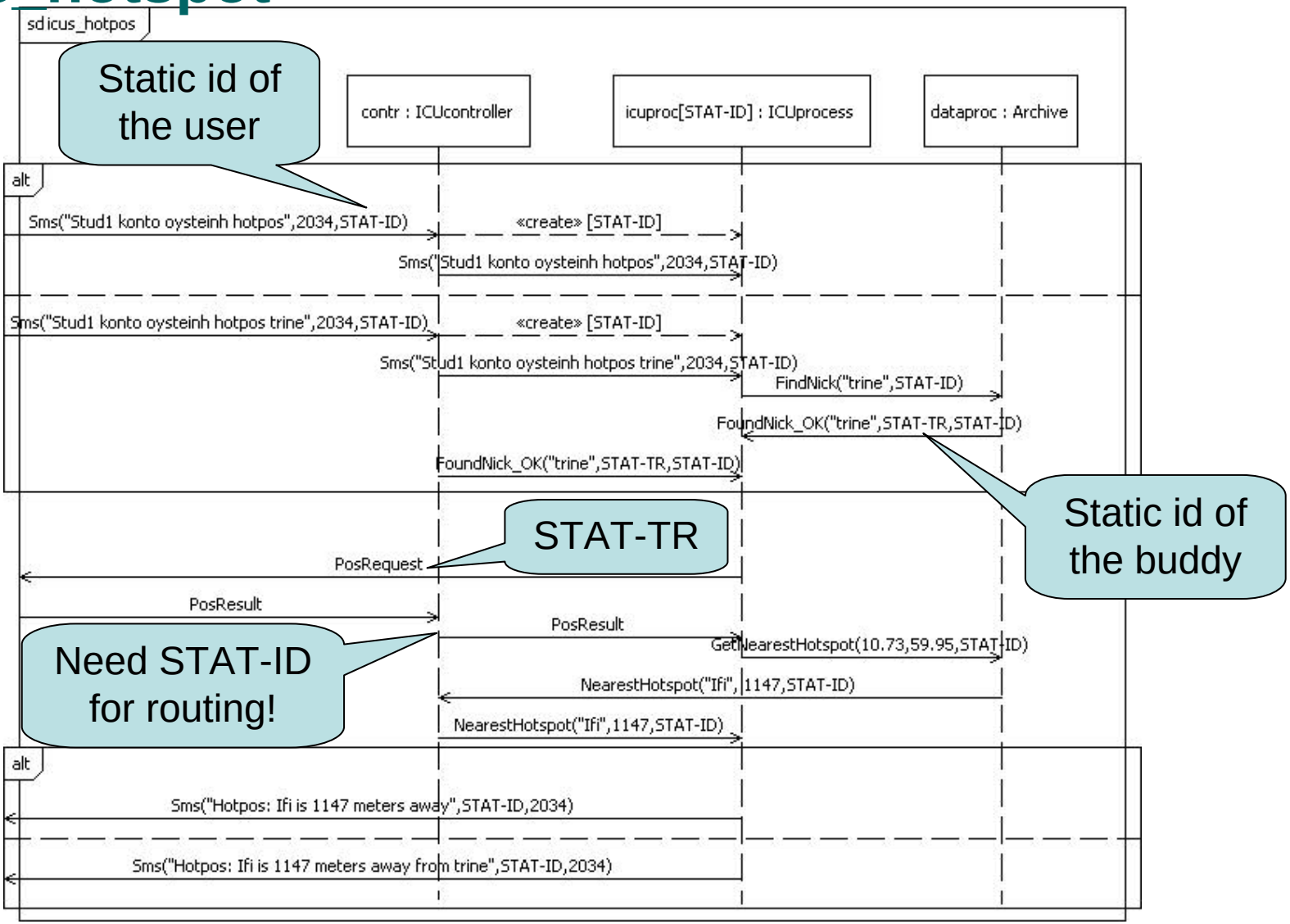
- Only ask *hotpos*
- Returning the position of the user relative to the hotspots

Hotpos – as of ICU7 – a minor change?

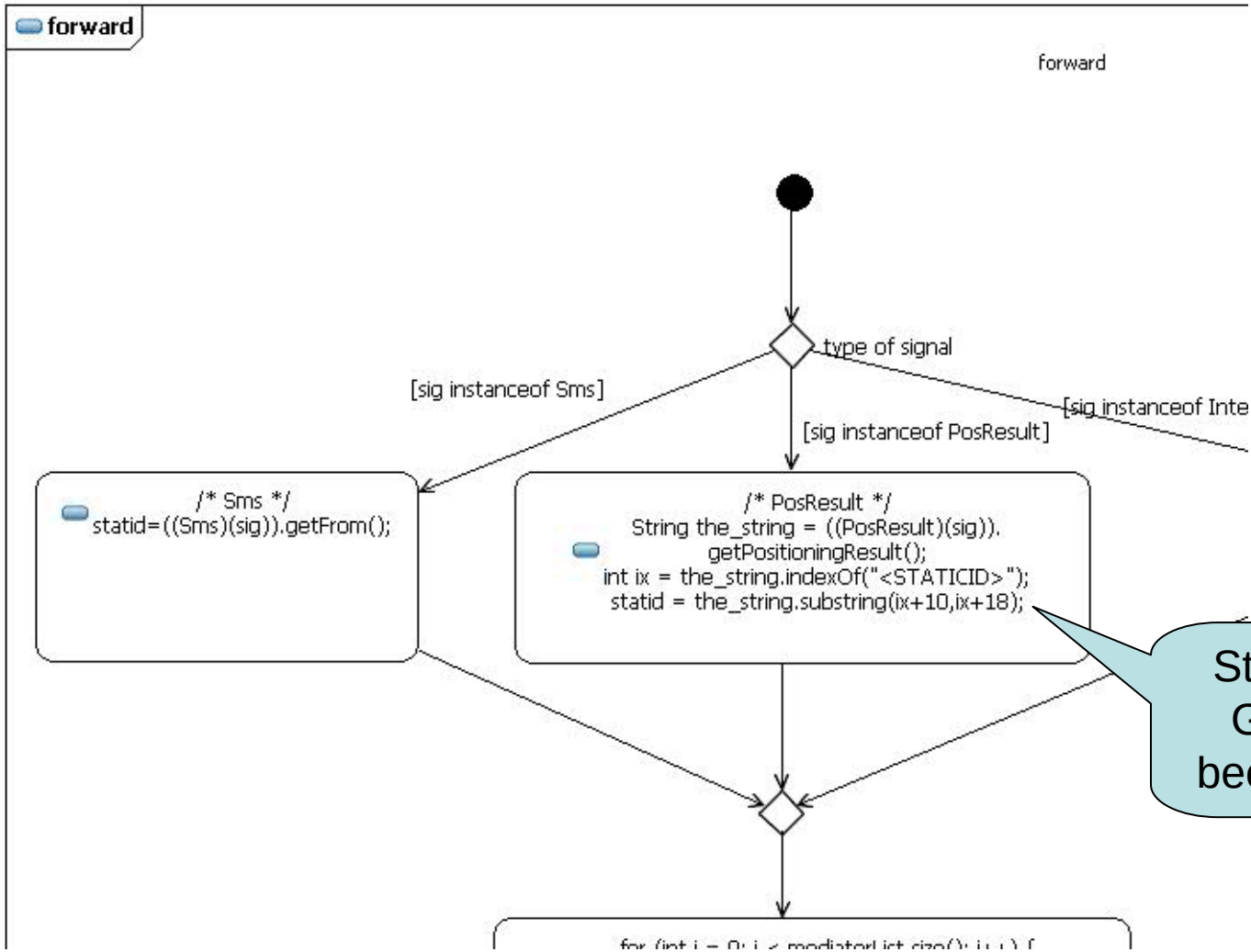


- Only hotpos is as before
- *hotpos nickname*
  should give the position of the person registered with the nickname relative to the users hotspots
- What could possibly be the problem?
- Let us look at the decomposition!

icus_hotspot



Routing PosResult in ICU6



The problem

- We want PosResult to be routed according to STAT-ID
  - STAT-ID is the static id of the user which identifies the session
- PosResult returns an XML-string which includes the static id of the positioned GSM
  - which is no longer identical to the user!
- PosResult is a message not defined by you, but in principle by a third party
  - which means you cannot change the interface!

- We need to take a closer look at the SMSMediator interface!

PosResult – the javadoc (1/2)

All Classes
PosRequest
PosResult
Sms
SmsInputMedia
SmsOutputMed

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

smsmediators

Class PosResult

```
java.lang.Object
  └ se.ericsson.eto.norarc.javaframe.Message
      └ smsmediators.PosResult
```

All Implemented Interfaces:
java.lang.Cloneable

public class **PosResult**
extends se.ericsson.eto.norarc.javaframe.Message

Description: Object representing a Positioning result

Field Summary

Fields inherited from class se.ericsson.eto. **sage**

nextMessage

Constructor Summary

PosResult(java.lang.String positioningResult, java.lang.String messageId)
Constructor for positioning requests

we have used this

could we benefit from this?

INF 5150

PosResult – the javadoc (2/2)

All Classes
PosRequest
PosResult
Sms
SmsInputMedia
SmsOutputMed

Constructor Detail

PosResult

```
public PosResult(java.lang.String positioningResult,
                 java.lang.String messageId)
```

Constructor for positioning requests

faulty documentation

Parameters:

`positioningResult` - to the phone/person you want to locate
`messageId` - a unique identifier for the message

what does this mean?

Method Detail

getMessageId

```
public java.lang.String getMessageId()
```

Returns:
Returns the messageId.

---

setMessageId

```
public void setMessageId(java.lang.String messageId)
```

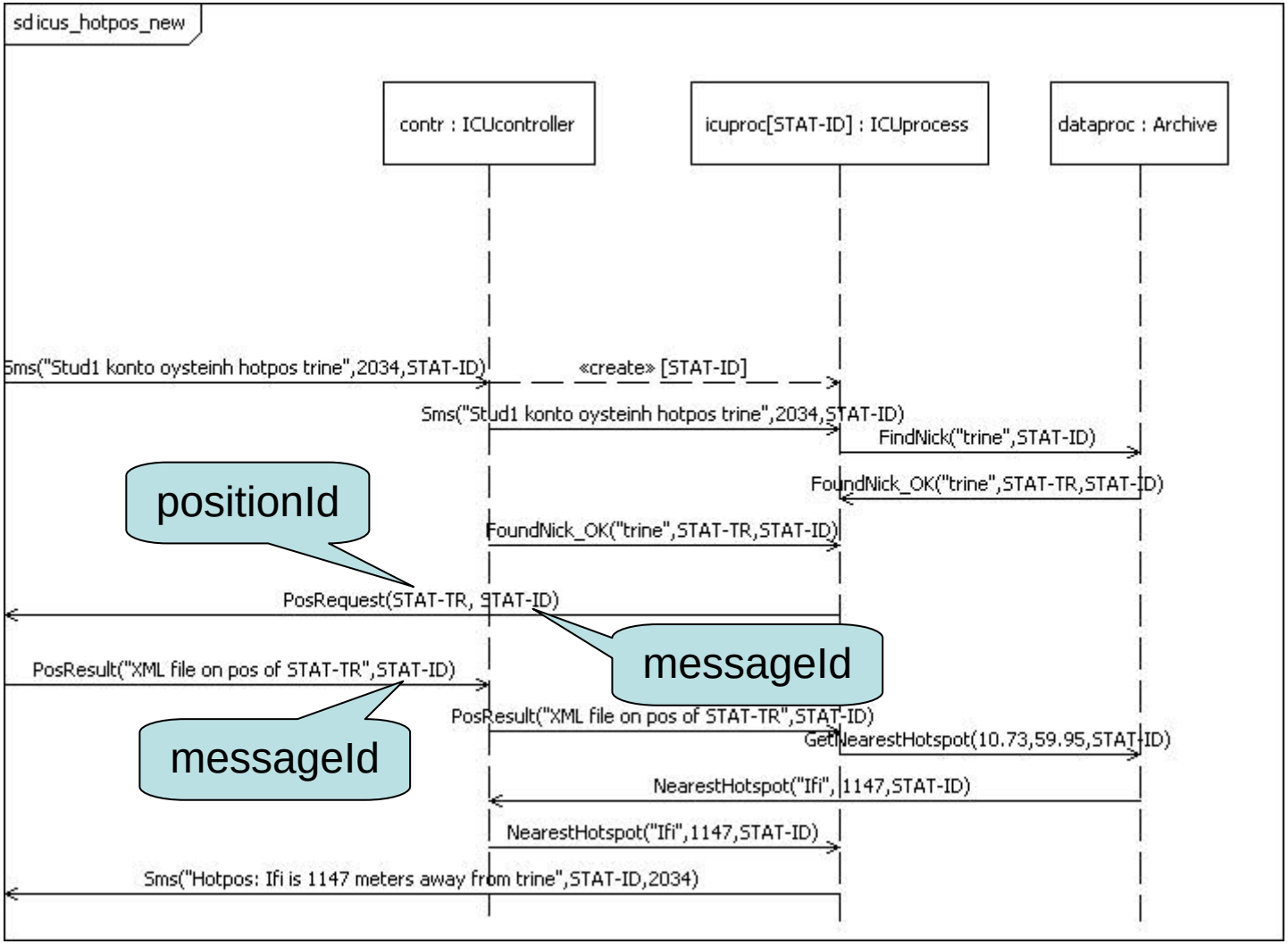
Parameters:
`messageId` - The messageId to set.

INF 5150

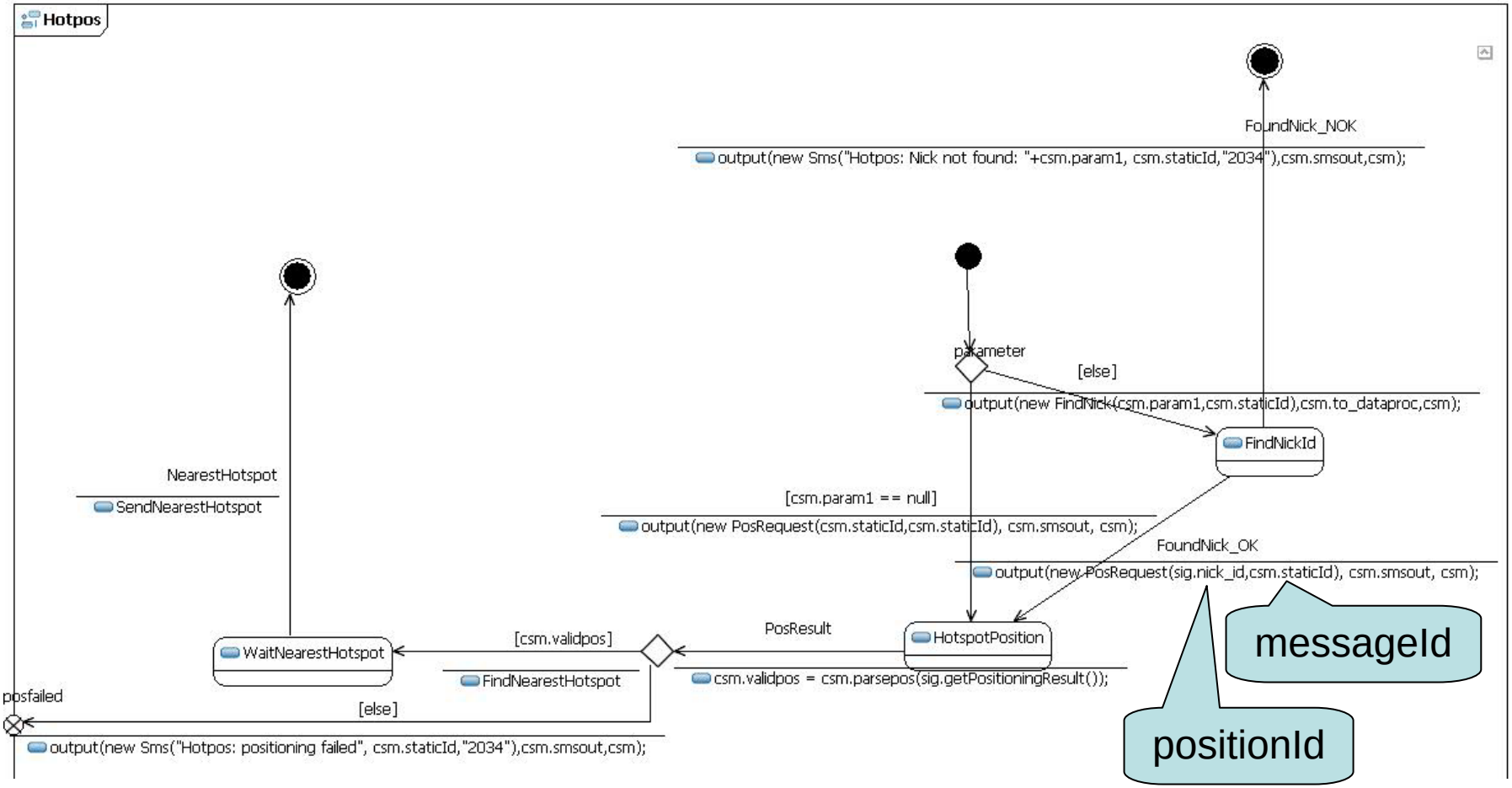
Exploring the SMSMediator interface

- When the documentation is less than satisfactory (and it always is), one has to experiment with the interface
- We hypothesize that PATS have had the same need to tag the communication as we have
  - thus we hope that the *messageId* of *PosRequest* (that we can choose) is returned as the *messageId* of the corresponding *PosResult*
- Through experimentation we assert that this is indeed the case!

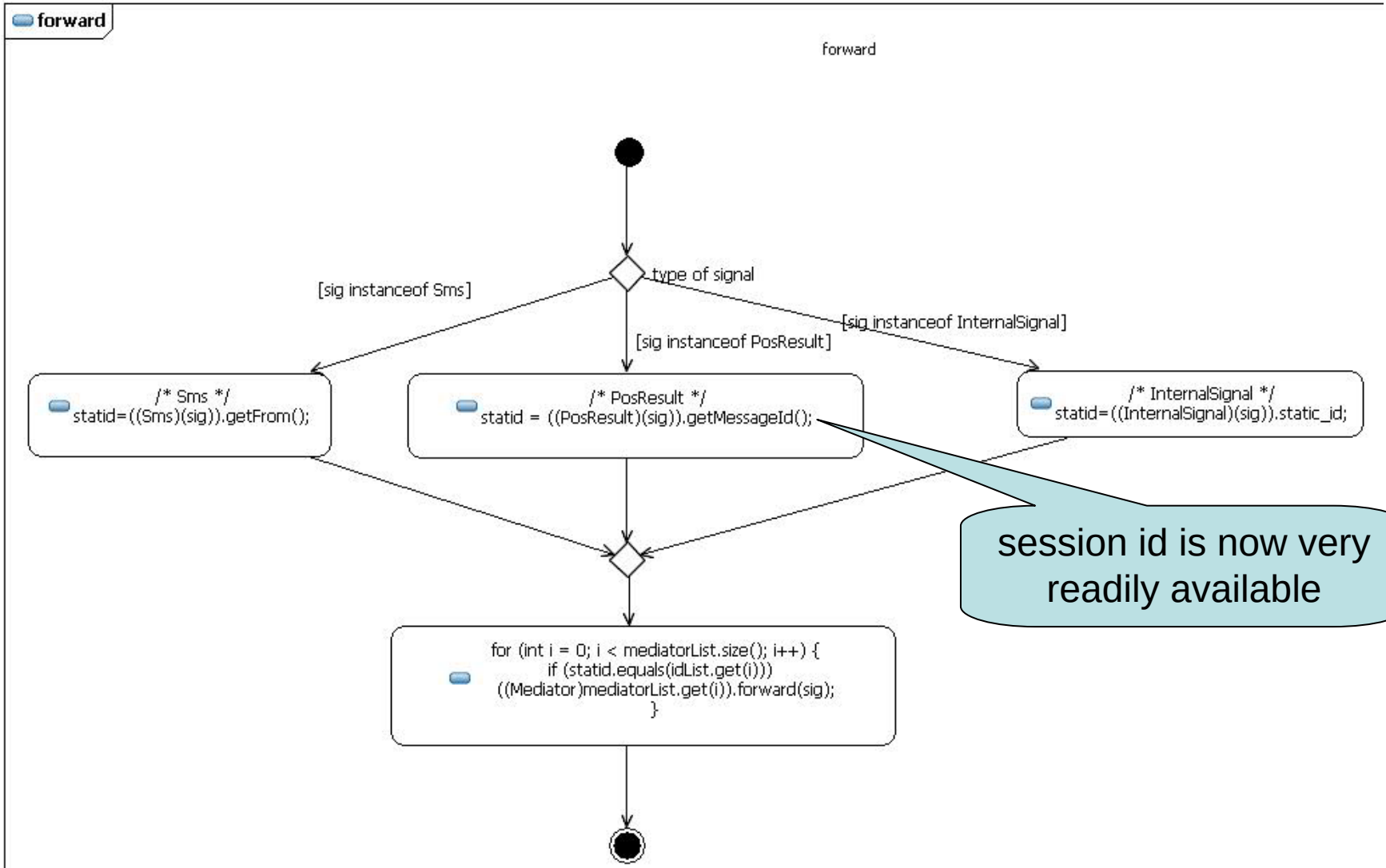
hotpos nickname



Hotpos submachine



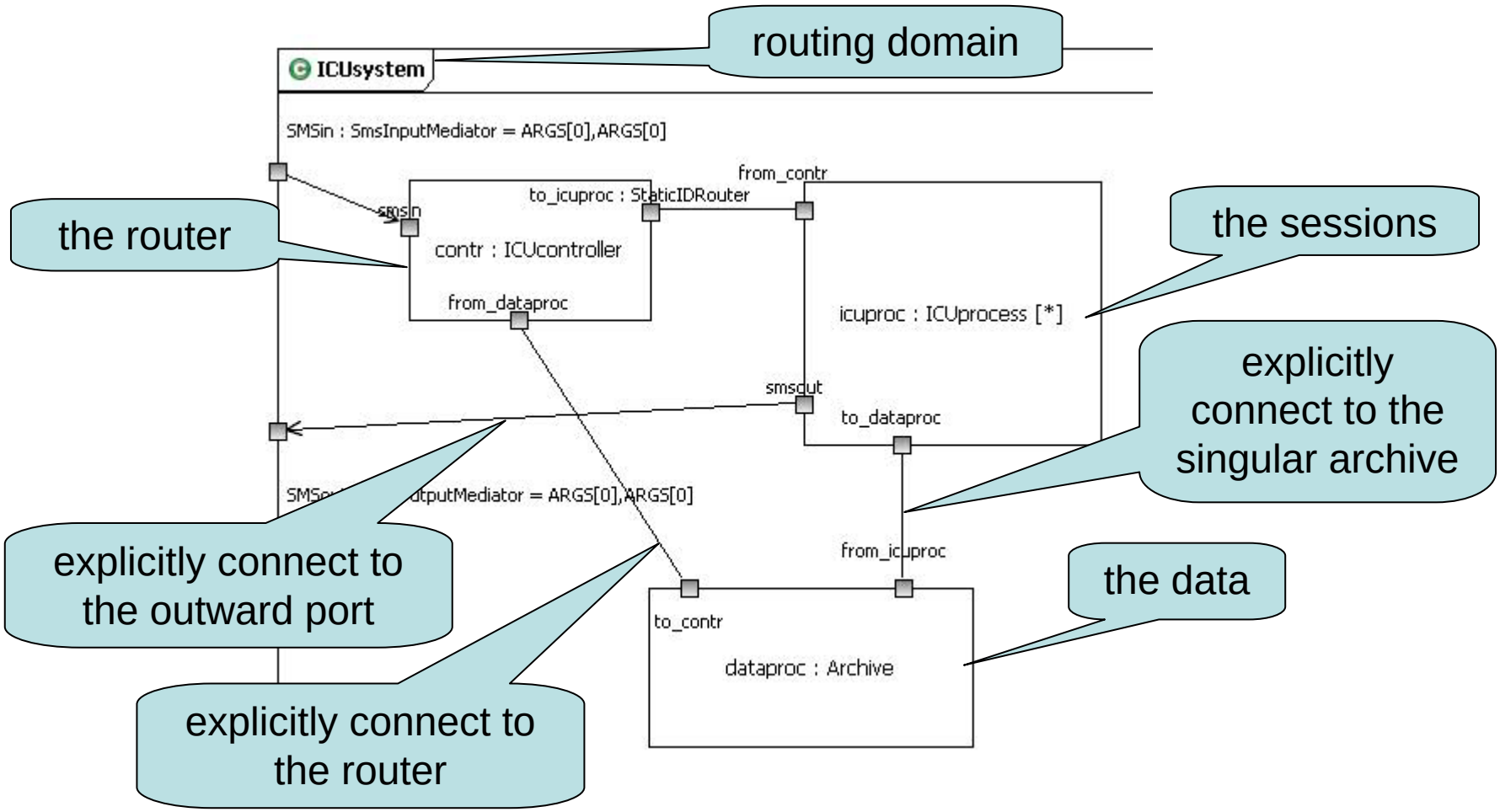
Adapted *forward* doing the routing



More on the problems of routing

... just to get a feeling for what is normally behind the scenes

The receptionist-session-archive architecture



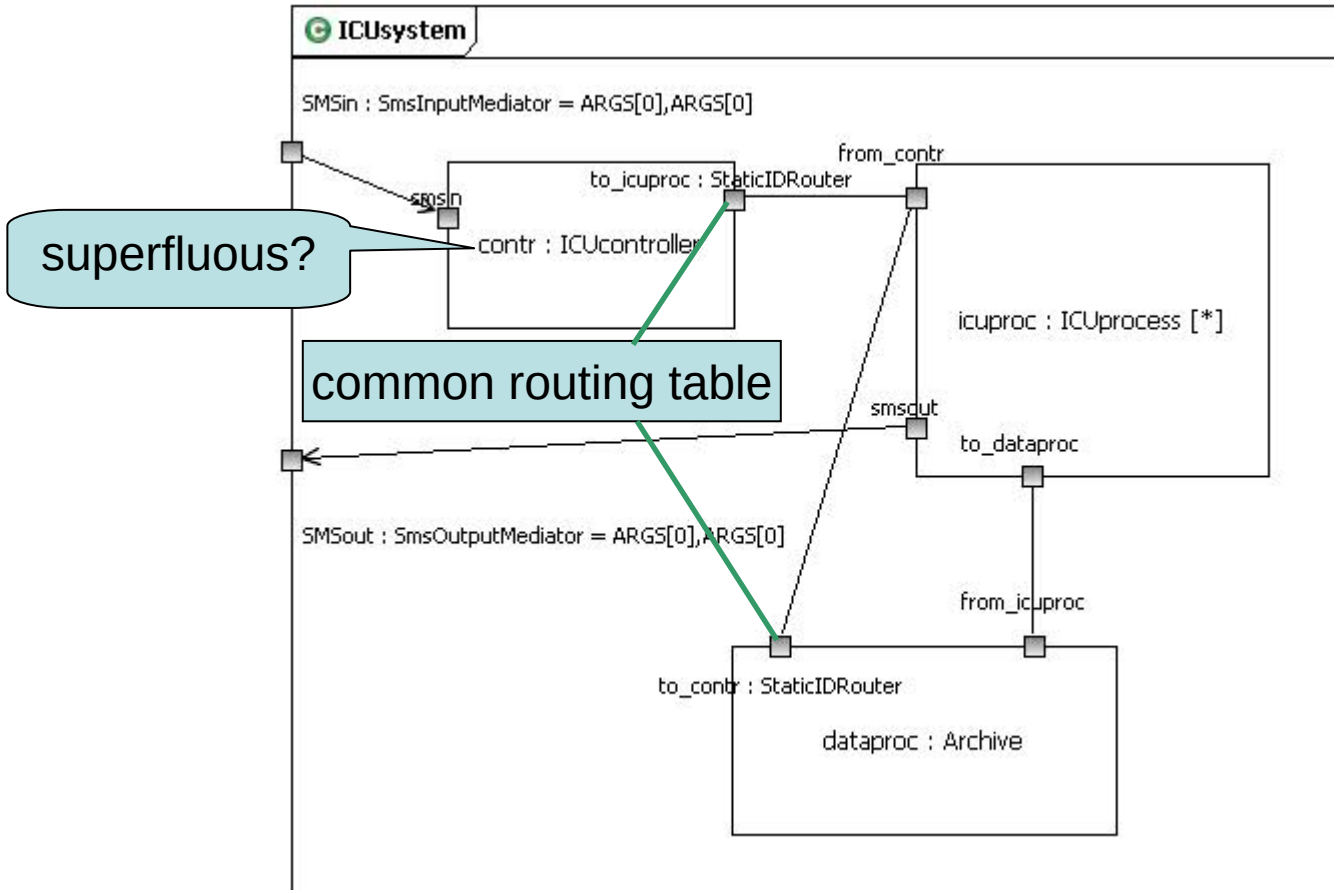
Our architecture's routing strategy

- Local addressing within the routing domain
  - in fact each process may only send to its own outward ports
  - the routing domain sets up the connections
- Explicit connection to either singular parts or port, or to the (single) router
  - In ICU we connect to
    - *contr:ICUcontroller* which is the router (and a singular part)
    - *dataproc:Archive* which is a singular part
    - *SMSout:SmsOutputMediator* which is a singular port
- Routing may take into account any information
  - but it is quite normal that the routing is done on a table where an identifier is mapped to an address
    - the address in our case is a port

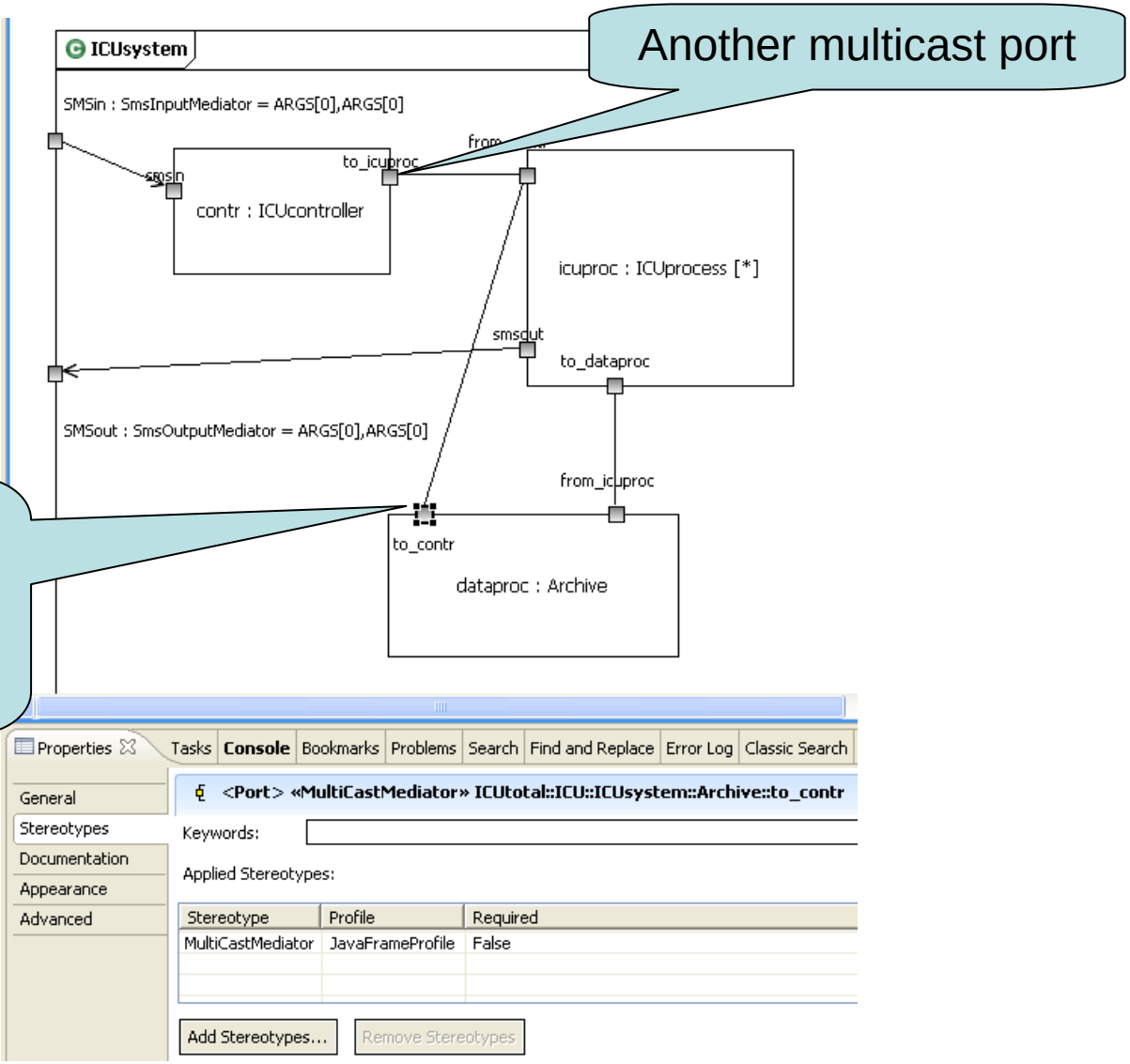
Alternative 1: The global address space

- A global address space means that for the whole system
  - any process (or its ports) has a unique address
  - any such address can be reached
- This is similar to the web and its URL
- This presumes that
  - there is an underlying system of routing
  - with the effect that logically there is a connection between any two processes of the whole system

Alternative 1: The global routing table



Alternative 2: Multicast / broadcast



Comparison

- Alt 0: Local addressing
  - local logic – but more logic through explicit connections
  - easy to make several instantiations
  - possible bottleneck at the router
- Alt 1: Global addressing
  - simple logic, especially when returning answers
  - requires underlying routing system
  - more global reasoning which may mean more difficult distribution
- Alt 2: Multicast / broadcast
  - no routing, the process decides for each message
  - simple communication
  - each process does a lot of futile work, but this may not be important if there are enough concurrent resources available

Agile modeling and session identifier

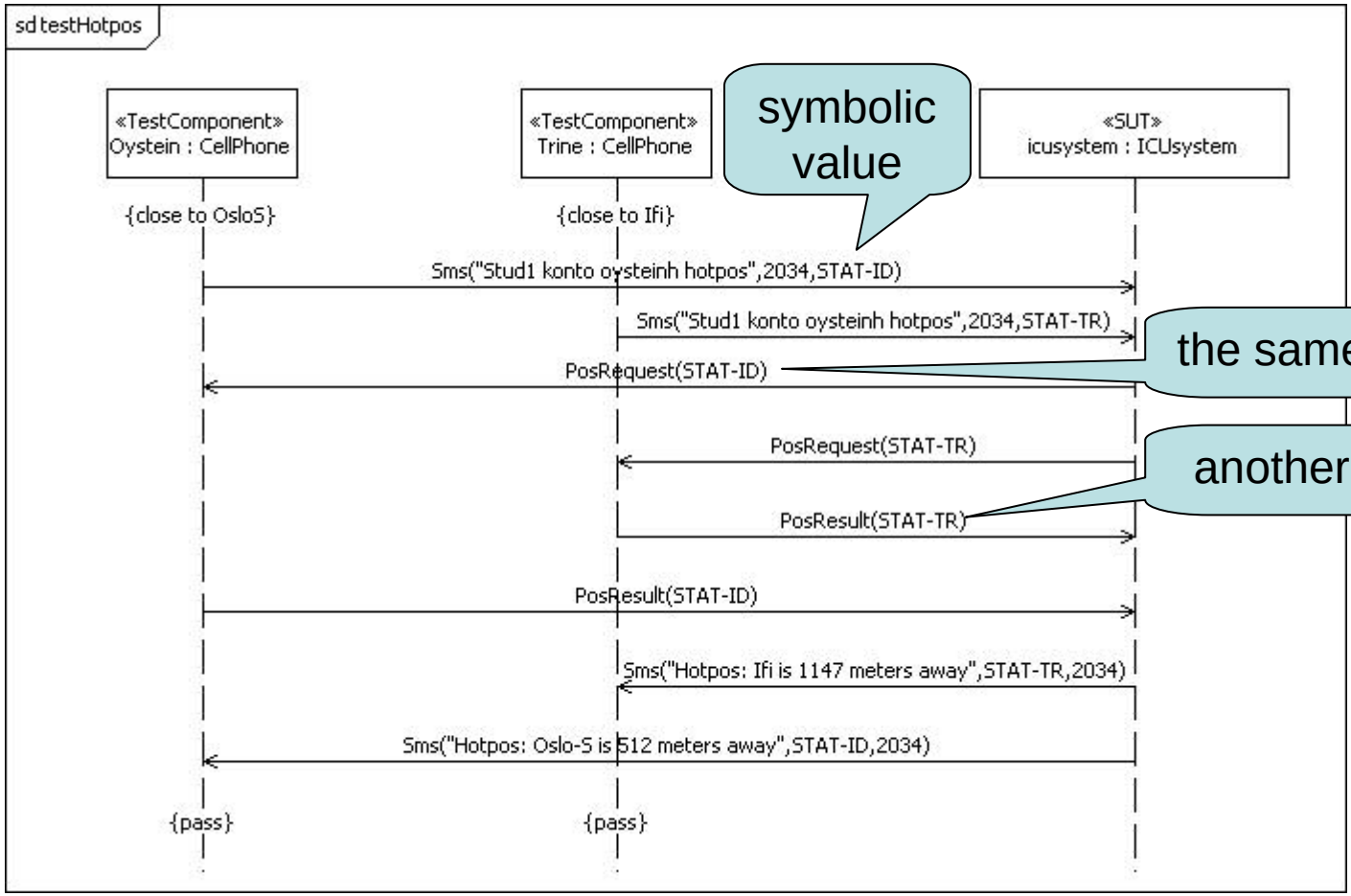
- We have used Static Id as our session identifier
  - This meant that the same GSM may not invoke more than one session at any point in time
  - Not a very tough restriction, but unnecessary and cumbersome to check
- We chose Static Id as session identifier
  - since it was the easiest choice from a system where there were no sessions
  - we had not discovered the augmented features of the SMSPorts
- It is typical for incremental development that
  - early decisions must be reviewed in light of new findings
  - and the system re-engineered
  - Here we may choose to go for unique session numbers

Even More Testing with U2TP
- Focusing on describing test data
- what data to test

Testing again – focusing on data

- How to describe test data
  - wildcards
  - data pools, data partitions and data selectors
- Principles for selecting data
  - Equivalence Class Partitioning
  - Boundary Value Analysis
  - Classification Tree Method
- Preamble and Postamble

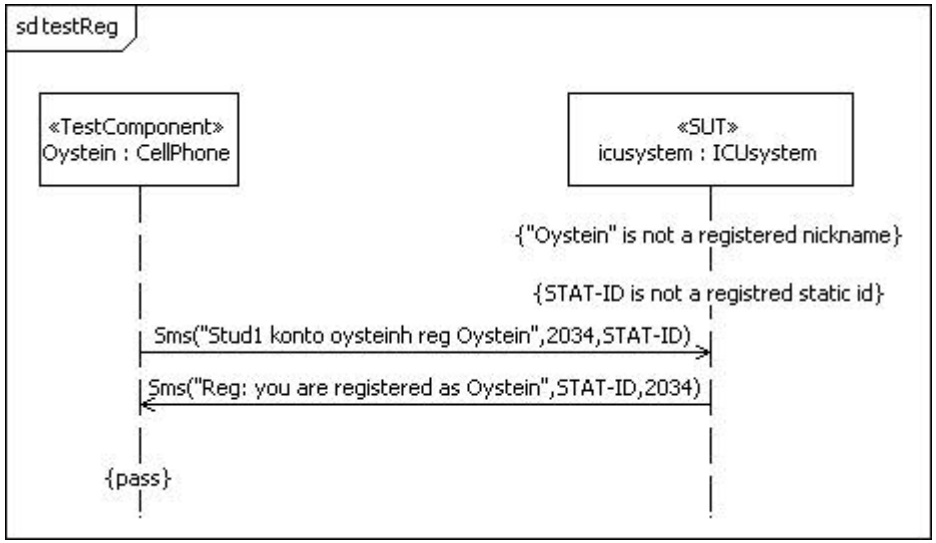
Wildcards (1) – symbolic values

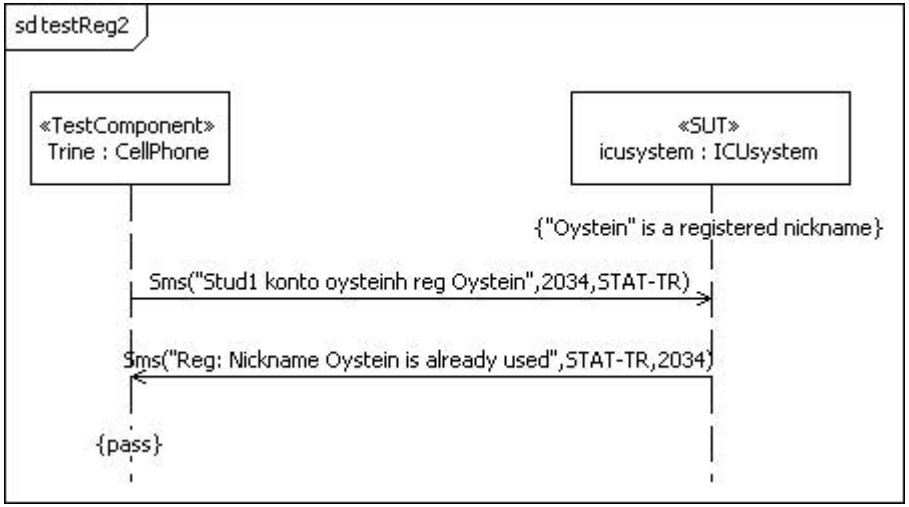


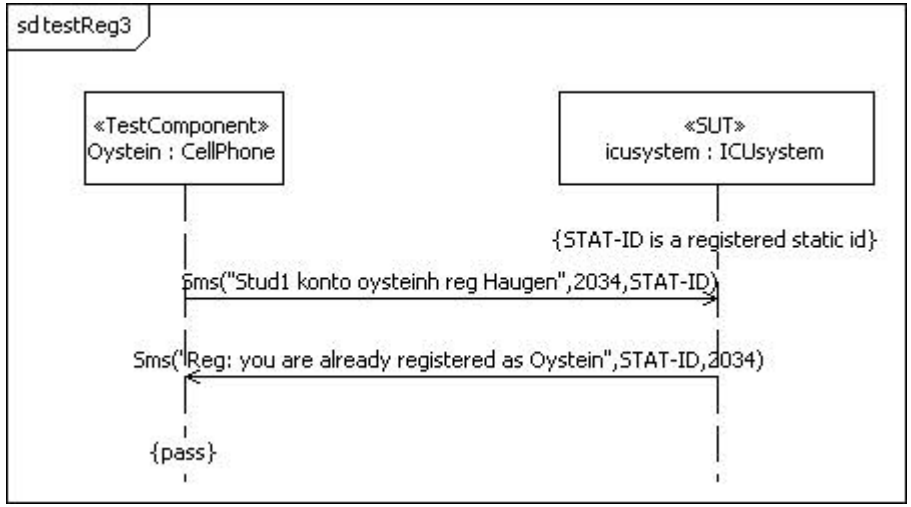
Wildcards (2) – explaining symbolic values

- Symbolic values are the same as an instance with a wildcard value
  - String STAT-ID = *
    - where the asterisk designates that the value itself is of no importance
  - String STAT-TR = *
    - another string value (not necessarily distinct from STAT-ID)
- We could also have said:
  - Sms(message="Stud1 konto oysteinh hotpos",to="2034",from=*)
    - again where the asterisk designates ”whatever value”
    - the disadvantage is that now we cannot easily refer to that value later
  - Sms("Stud1 konto oysteinh hotpos",2034,A-CGHDWQ)
    - this becomes almost too concrete with little to gain

Testing *registration*





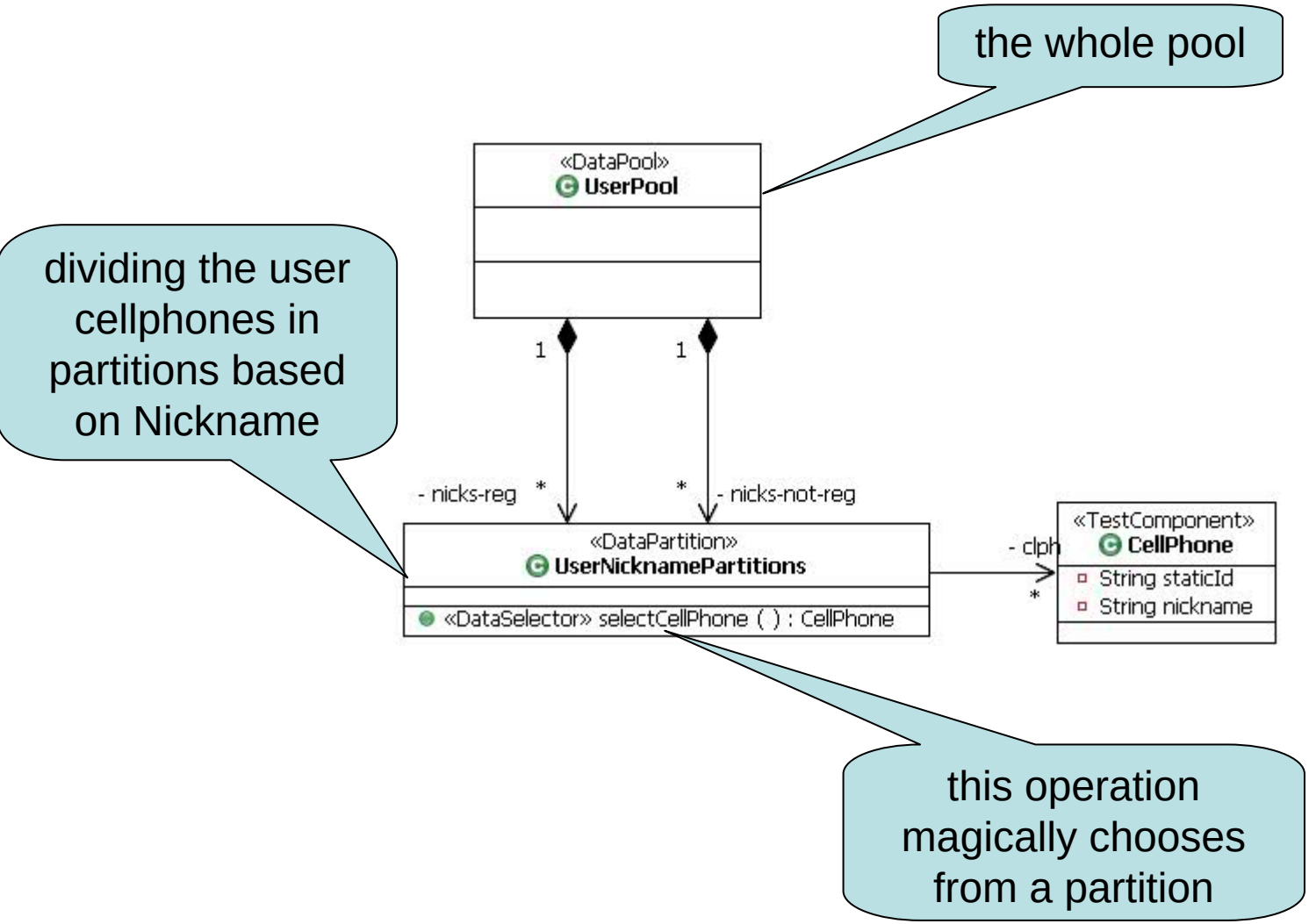


We need a pool of users

- We need a data pool of users
  - where some have new nicknames and static id
  - some have old nicknames and new static id
  - some have new nicknames and old static id
  - some have old nicknames and old static id

instance	nickname	static id
Oystein	Oystein	STAT-ID
Trine	Trine	STAT-TR
Sverre	Sverre	STAT-SV
Sigurd	Sigurd	STAT-FS

How to define a data pool of Users



Equivalence Class Partitioning

- Equivalence partitioning is based on the premise that
  - the inputs and outputs of a component can be partitioned into partitions that, according to the component's specification, will be treated similarly by the component.
- Thus the result of testing a single value from an equivalence partition is considered representative of the complete partition
- In ICU:
  - provided neither Oystein nor Trine has ever registered, it is of no concern which of the two users are applied for the test of registration

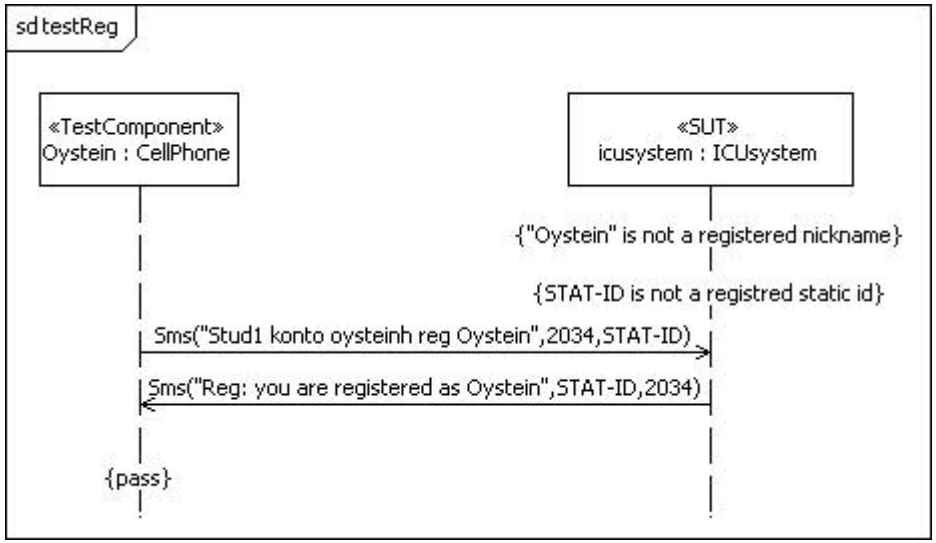
Boundary Value Analysis

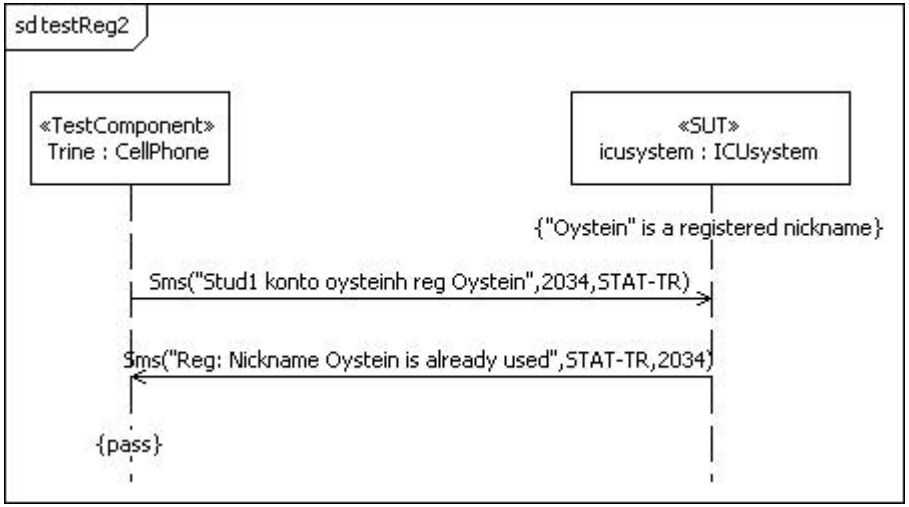
- Boundary Value Analysis is based on the following premise.
  - Firstly, that the inputs and outputs of a component can be partitioned into partitions that, according to the component's specification, will be treated similarly by the component and,
  - Secondly, that developers are prone to making errors in their treatment of the boundaries of these classes.
- Thus test cases are generated to exercise these boundaries.
- In ICU:
  - There are no boundaries to name spaces
  - For Hotpos, the problems should occur where the distances to several hotspots are the same

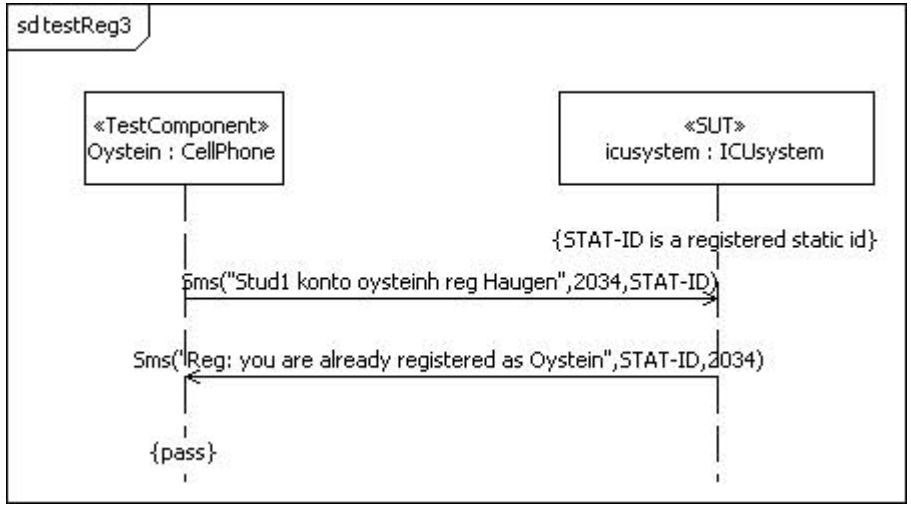
Classification Tree Method

- As for classification-tree method, the input domain of a test object is regarded under various aspects assessed as relevant for the test.
  - For each aspect, disjoint and complete classifications are formed.
  - Classes resulting from these classifications may be further classified – even recursively.
- The stepwise partition of the input domain by means of classifications is represented graphically in the form of a tree.

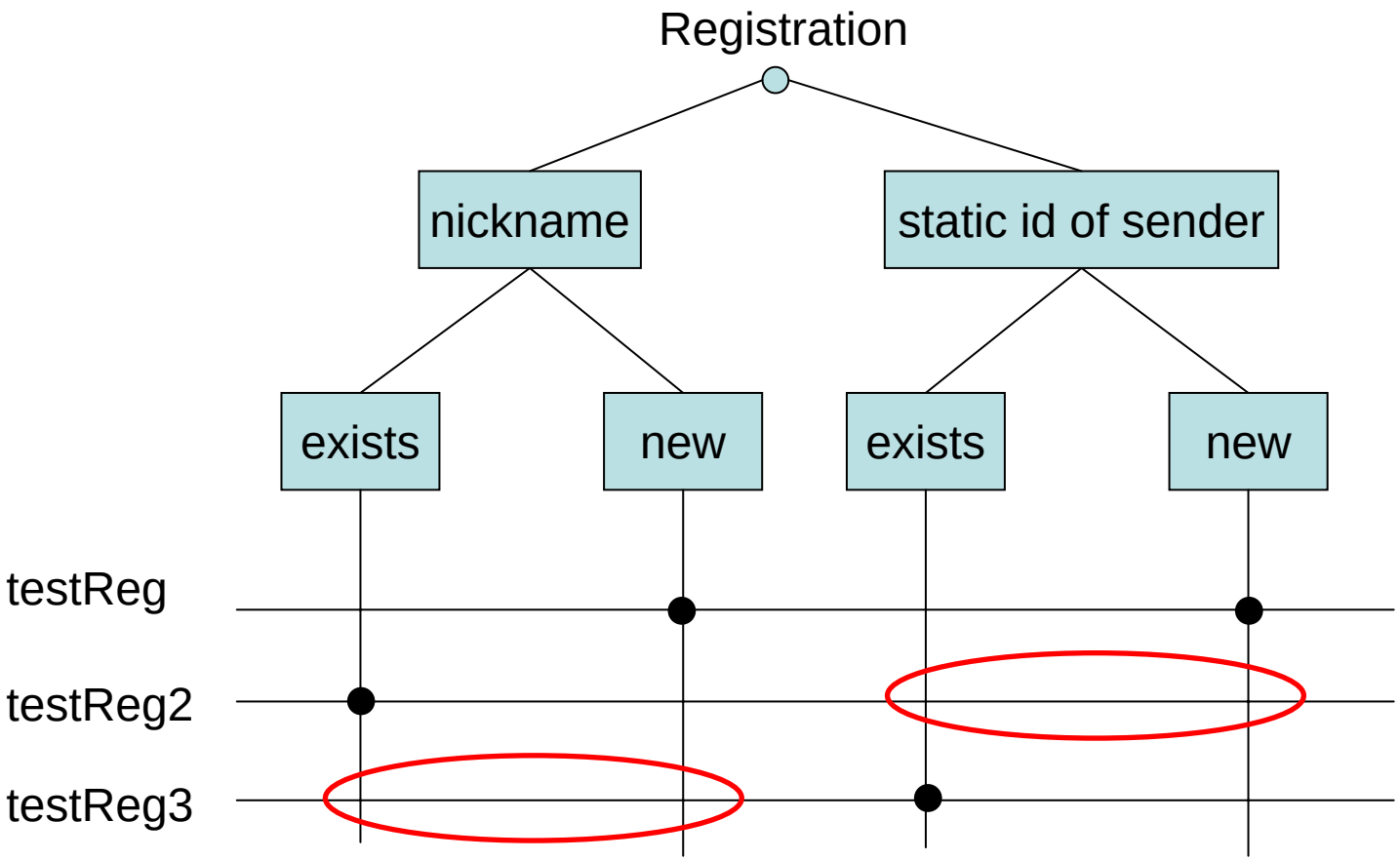
Testing *registration*







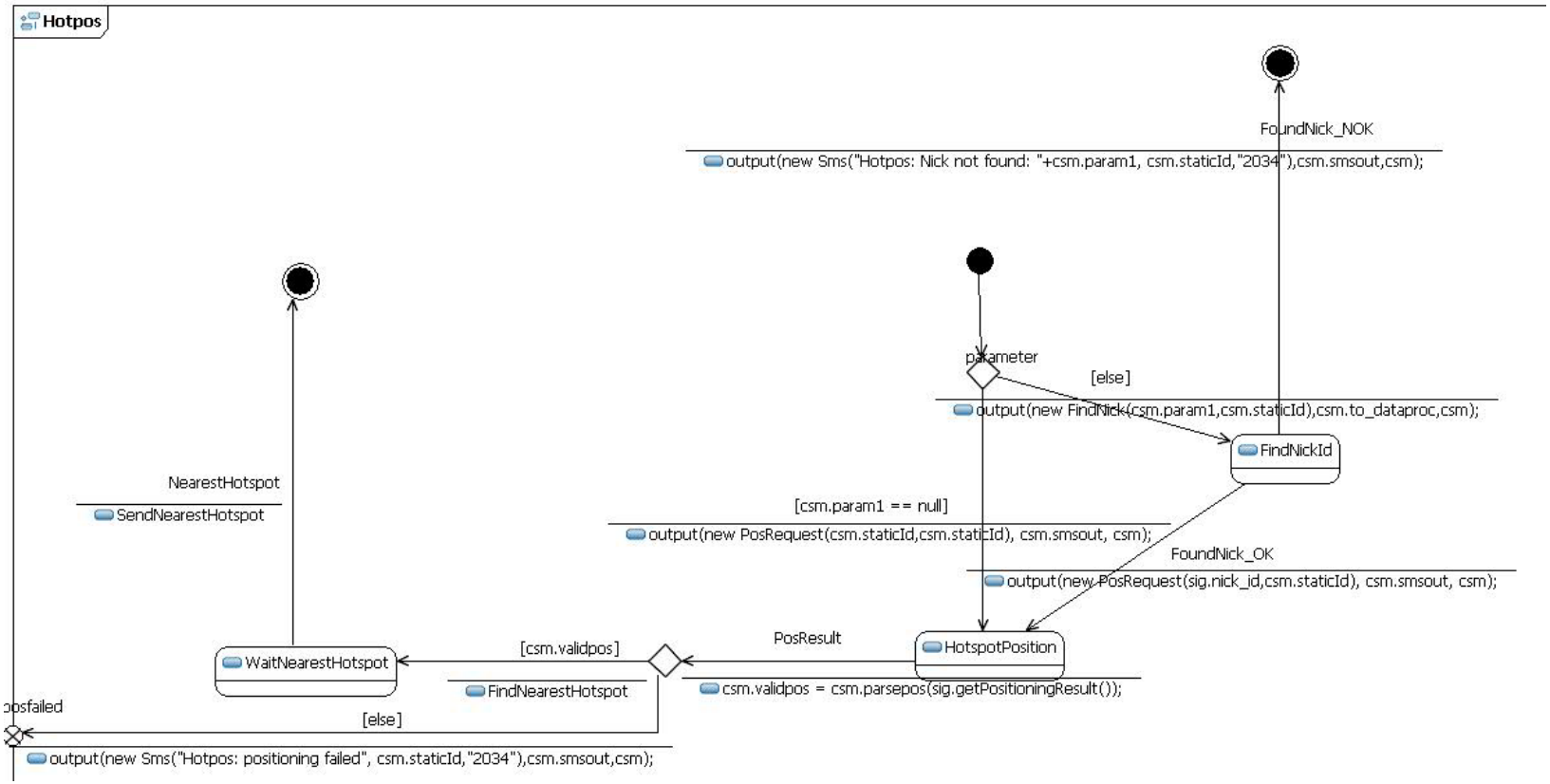
ICU registration classification tree



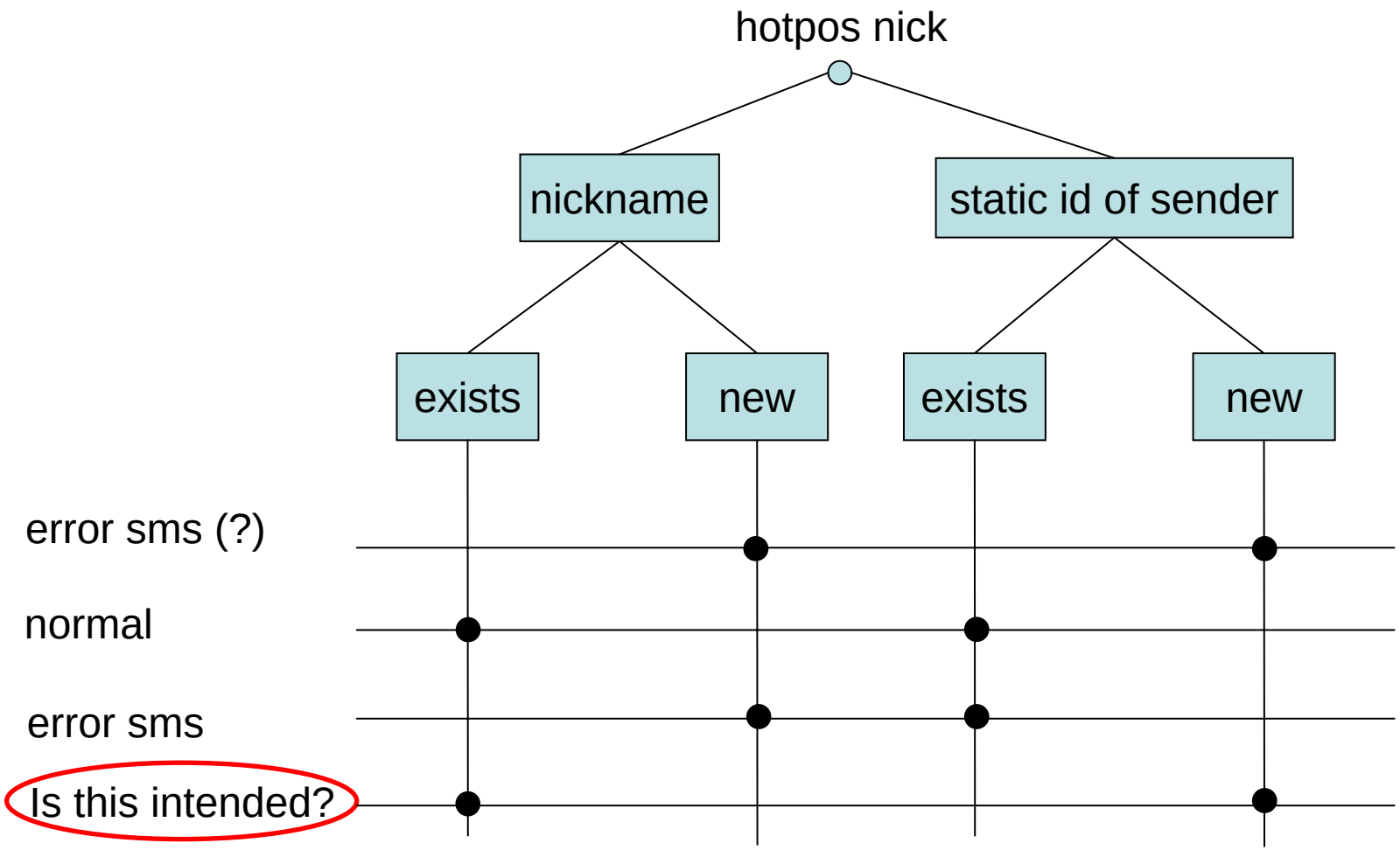
Preamble and Postamble

- A test preamble is a description of how to get the test system into a situation where the next test can be executed
- A test postamble is a description of how to clean-up after the test
- A combined test may often be done such that the tests normally make up each others preamble
  - TestReg will make CellPhone(Oystein,STAT-ID) registered
  - TestReg2 or TestReg3 have then their preconditions satisfied

hotpos [nickname]



ICU *hotpos* classification tree



Unintended cases: the positioning stranger

- The systematic testing reveals:
  - A complete stranger may position any one registered as long as he/she knows their nickname
- What was the real intention behind registration?
  - That the ones inside can see others inside
  - Not that anybody can see the insiders and nobody can see the outsiders!
  - Remedy: Only registered users can position others
- Systematic testing reveals
  - not only errors in the design and the implementation
  - but also problems with the requirements
    - there were inconclusive traces that should have been negative

Summary Data-oriented Testing

- Not every possible data combination can be tested
- Therefore we need to group the data
  - such that the values in a group can be considered equal
- Apply analysis to form the value groups
  - Equivalence Class Partitioning
  - Boundary Value Analysis
  - Classification Tree Method
- Any systematic approach will be better than nothing!
- UML Testing Profile offers the following concepts:
  - Data Pool
  - Data Partition
  - Data Selector