UNIVERSITY
OF OSLO

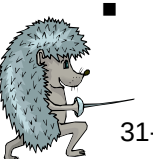
INF-5150 2006
by Øystein Haugen and Ketil Stølen
plus assistants Atle Refsdal, Marius Furulund

Version 060901

INF 5150

Øystein Haugen <oysteinh@ifi.uio.no>

- 80-81: UiO, Research assistant for Kristen Nygård
  - 81 : IN 105 together with Bjørn Kirkerud
- 81-84: Norwegian Computing Center, Simula-machine
- 84-88: SimTech, typographical applications
- 88-90: ABB Technology, SDL, prototype SDL tool, ATC
- 89-97: SISU project, methodology, V&V, ITU
  - 93: Engineering Real Time Systems
  - 96: Integrated Methodology -> TIMe
- 96-00: Rapporteur ITU for MSC
- 97: Practitioners' verification of SDL systems (dr. scient.)
- 97- 03: Ericsson, NorARC
- 98- 03: Ifi, UiO as Part time Associate Professor
  - IN-TIME (98) IN-RTIMe (99) IN-RTIMe (2000) INFUIT (2001 og 2002)
- 99- : Participates in OMG wrt. UML 2.0
  - Responsible for UML 2.x chapter on Interactions
- 04 - : Associate Professor at Ifi

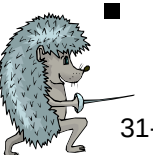


Ketil Stølen <ketil.stolen@sintef.no>

- Leader of Group for Quality and Security Technology at SINTEF
- Professor II at IFI
- Background from University of Manchester (4 years); Technical University Munich (5 years); Institute for Energy Technology (3 years); Norwegian Defence Research Establishment (1 year); SINTEF (6 years)
- PhD in formal methods
- Leading role in the development of the Focus method - a formal method providing the basic foundation for the refinement part of this course
- Lead the development of the CORAS methodology for model-based security analysis providing the basic foundation for the security part of this course
- Is currently managing research projects with a total budget of 35 million NOK

Atle Refsdal <atler@ifi.uio.no>

- Education:
  - Electrical engineer, Gjøvik Ingeniørhøyskole
  - Master thesis in Language, Logic and Information (Språk, Logikk og Informasjon – SLI) at the Faculty of Arts, University of Oslo
- Previous employees:
  - Beijer Electronics, Drammen. Industrial automation/Programmable Logical Controllers
  - Computas AS, Oslo/Lysaker. Knowledge Systems "Saksbehandlingssystemer".
- Currently employed at the Department of Informatics as a PhD student in the SARDAS project
  - where Ketil Stølen, Øystein Haugen, Birger Møller-Pedersen and Rolv Bræk are supervisors
  - we describe/analyze availability using UML or UML-like models
- Took INF-5150 in 2003
- Was assistant to INF-5150 in 2004 and 2005



Marius Furulund <kristf@ifi.uio.no>

- Education:
  - Finishes Master of Informatics, May 2007
    - Thesis on group estimation at Simula Research Center with supervisors Kjetil Moløkken-Østvold and Magne Jørgensen
  - Spent one year at San Diego State University
- Previous employees:
  - Ifi, Student assistant:
    - INF-2120 Spring 2006
    - INF-1000, INF1010, INF3120
  - Dagbladet:
    - working part time with the sport section
- Took INF-5150 in 2004

Books and Curriculum

- We will produce and refer to written material to support the lectures
  - Rumbaugh, Jacobson, Booch: *UML Reference Manual, Second Edition*, 2004. Addison-Wesley. ISBN: 0321245628. Link to Addison-Wesley on this book.
  - Pilone, Dan: *UML 2.0 in a Nutshell*, 2005. O'Reilly Media. ISBN: 0-596-00795-7. Read it on Safari.
  - OMG: *UML Profile for Modeling QoS and Fault Tolerance*, 2005. OMG. http://www.omg.org/docs/ptc/05-05-02.pdf.
  - Haugen, Møller-Pedersen, Weigert: *Structural Modeling with UML 2.0*, 2003. Kluwer. ISBN: 1-4020-7501-4. We have picked out one chapter, but also other chapters are interesting. chapter in pdf.
  - Haugen, Husa, Runde, Stølen: *STAIRS towards formal design with sequence diagrams*, 2005. SoSyM, Springer Online. STAIRS article.
  - Runde, Haugen, Stølen: *The Pragmatics of STAIRS*, 2006. Springer-Verlag. LNCS 4111. STAIRS tutorial.
- The slides of the lectures will be made available on the web in Acrobat format
- The lectures are part of the curriculum
- INF 5150 will use the required planning pages : http://www.ifi.uio.no/INF5150/

Goal: Unassailable IT-Systems

- The course INF-UIT aims at teaching the students
  - how software is made unassailable meaning that
    - the software is easily analyzed with respect to reliability and dependability
    - the software is easily maintained
- The overall goal is to explain
  - how practical software development can benefit from theories about
    - state machines
    - refinement
    - formal reasoning
    - modularity
    - security and related matters

Practical details

- When?
  - Lecture: Friday 9.15 - 12.00 (Lille Aud.)
  - Exercises: Tuesdays 10.15 – 12.00 (3A)
- Language: Norwegian/English
- Exam
  - Credits: 10 studiepoeng
  - Form: written
  - Grades: A - F
- Obligatory Exercises
  - There will be one obligatory exercise done in groups of 5-6
  - The students may be asked to explain details in their solution
  - The obligatory exercise will have two drops

Unassailable IT-Systems

- Unassailable?
- IT?
- Systems?

Unassailable

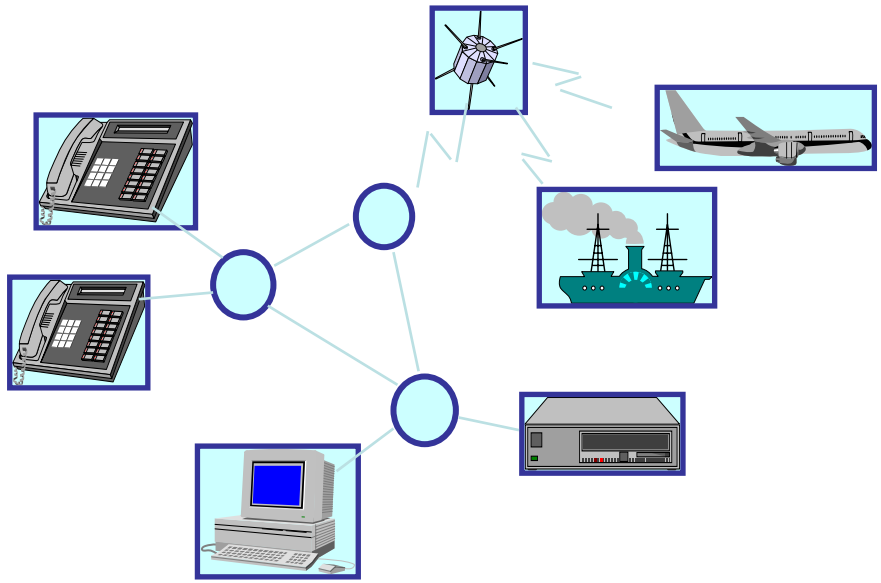
- Not assailable : not liable to doubt, attack, or question
- Where is this important?
  - for all software?
    - to some extent, but possibly less than one would like to think
  - for some critical software
    - telecom
    - surveillance (of patients, of production processes)
    - within computers themselves
- This course is not concerned with attacks that come from hackers towards data bases with sensitive content
  - we are concerned with helping software to perform desirably even in unexpected situations

IT?

- Information Technology
  - using computers
  - with emphasis on practical systems
  - with emphasis on behavior
- Engineering
  - Well acknowledged and asserted techniques
  - Creativity only when and where needed
  - Replication of earlier efforts
  - Pragmatics as well as theory

Systems?

- distributed
- concurrent
- real-time
  - In synchrony with real life
  - often small amounts of time for each service e.g. Automatic Train Control
  - the actual durations may or may not be significant
- reactive
- heterogeneous
- complex

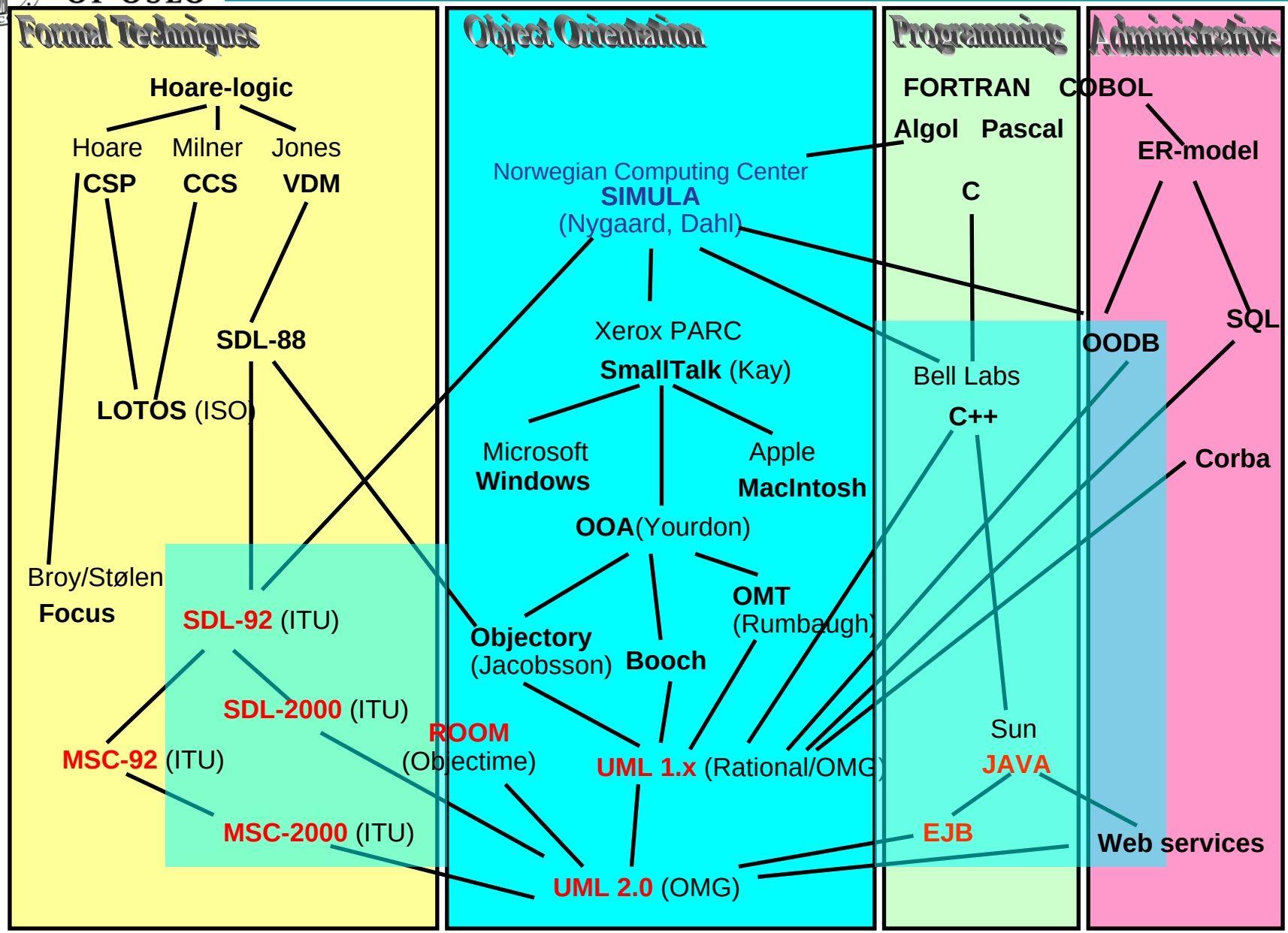


Lecture plan - INF-5150 Autumn 2006

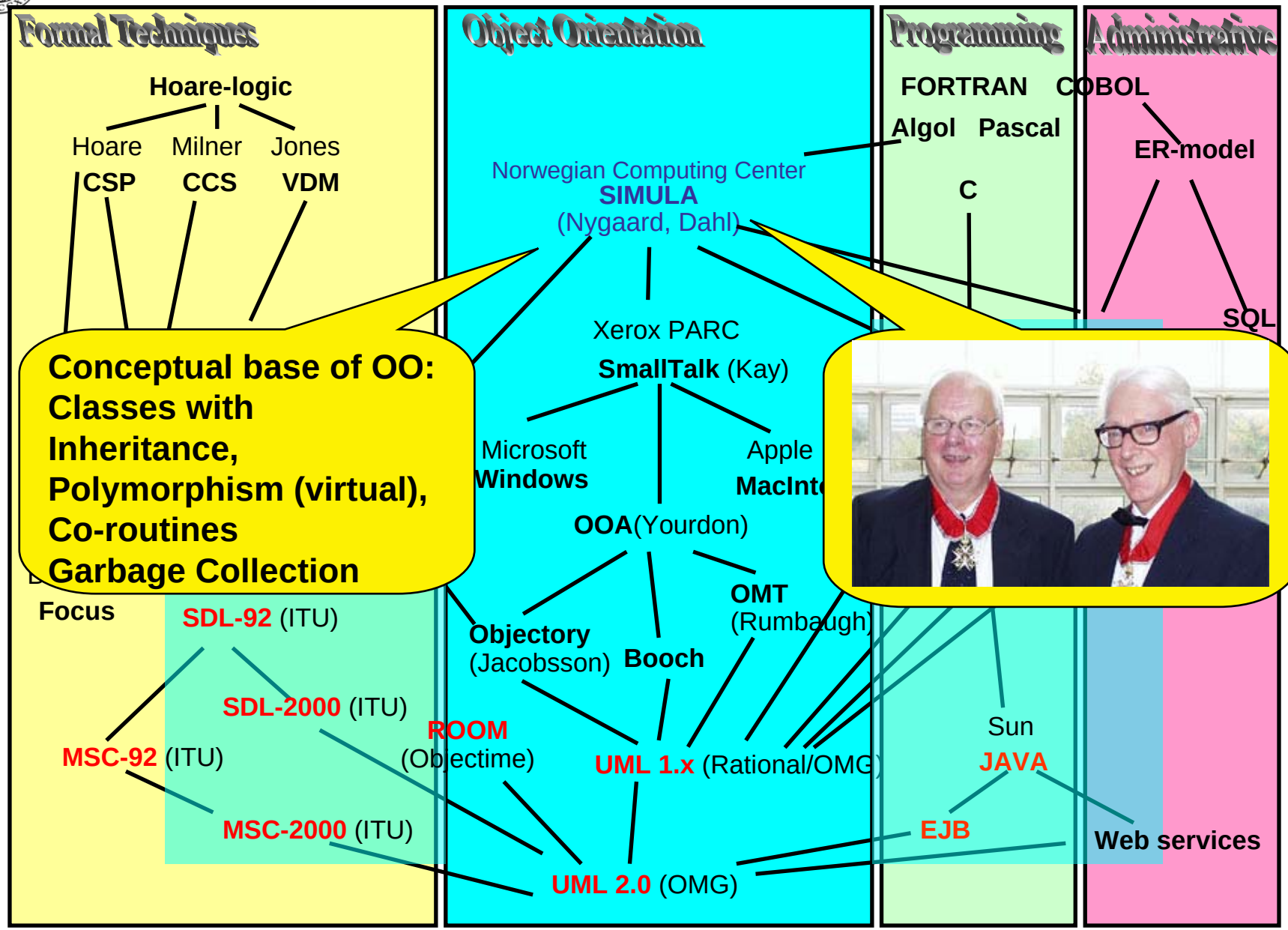
Teaching plan (INF5150 - autumn 2006)

Date	Teacher	Place	Topic	Lecture notes / comments
01.09.2006	Haugen	Lille Aud	Introduction	Everyone must attend in order to follow the course THE PLAN IS NOT TOTALLY FIXED. Changes may appear throughout the semester. In this plan you will find links to the lectures most often before the lecture is held.
08.09.2006	Haugen	Lille Aud	UML Sequence Diagrams	
15.09.2006	Haugen	Lille Aud.	UML State Machines	
22.09.2006	Stølen	Lille Aud	Refinement 1	
29.09.2006	Stølen	Lille Aud	Refinement 2	
06.10.2006	Guest lecturer	Lille Aud.	to be announced	Since both Haugen and Stølen have other engagements this Friday we plan to invite a guest speaker on a relevant subject
13.10.2006	Haugen	Lille aud.	Development Methodology	Also: HARD DEADLINE for Obligatory Exercise Drop 1 We follow an absolute policy: The groups must deliver their drop before 23.59. There are no exceptions. Any excuses (force majeur) will then be considered by the teachers in their evaluation.
20.10.2006	Haugen/Stølen	Lille Aud	Walk-through of Drop 1	This procedure is followed: • A Group presents their deliverable • Another (designated) criticizes the deliverable • The group teacher(s) give their evaluation • The lecturers supplements and suggests a grade (A-F) • The next group presents.
27.10.2006	Refsdal	Lille Aud.	Refinement 3	
03.11.2006	Stølen	Lille Aud.	Security Analysis 1	
10.11.2006	Stølen	Lille Aud	Security Analysis 2	
17.11.2006	Stølen	Lille Aud.	Security Analysis 3	Also: HARD DEADLINE for Obligatory Exercise Drop 2
24.11.2006	Haugen	Lille Aud.	Testing with UML	
01.12.2006	Haugen / Stølen	Lille Aud.	Walk-through of Drop 2	Same procedure as for Drop 1, but who criticizes whom has been reversed.
05.12.2006	EKSAMEN			Eksamensplasseringen vil kunne endres når oppmeldingene er helt klare. Følg med på linken her. Det er det som gjelder ikke hva som står i denne oversikten.

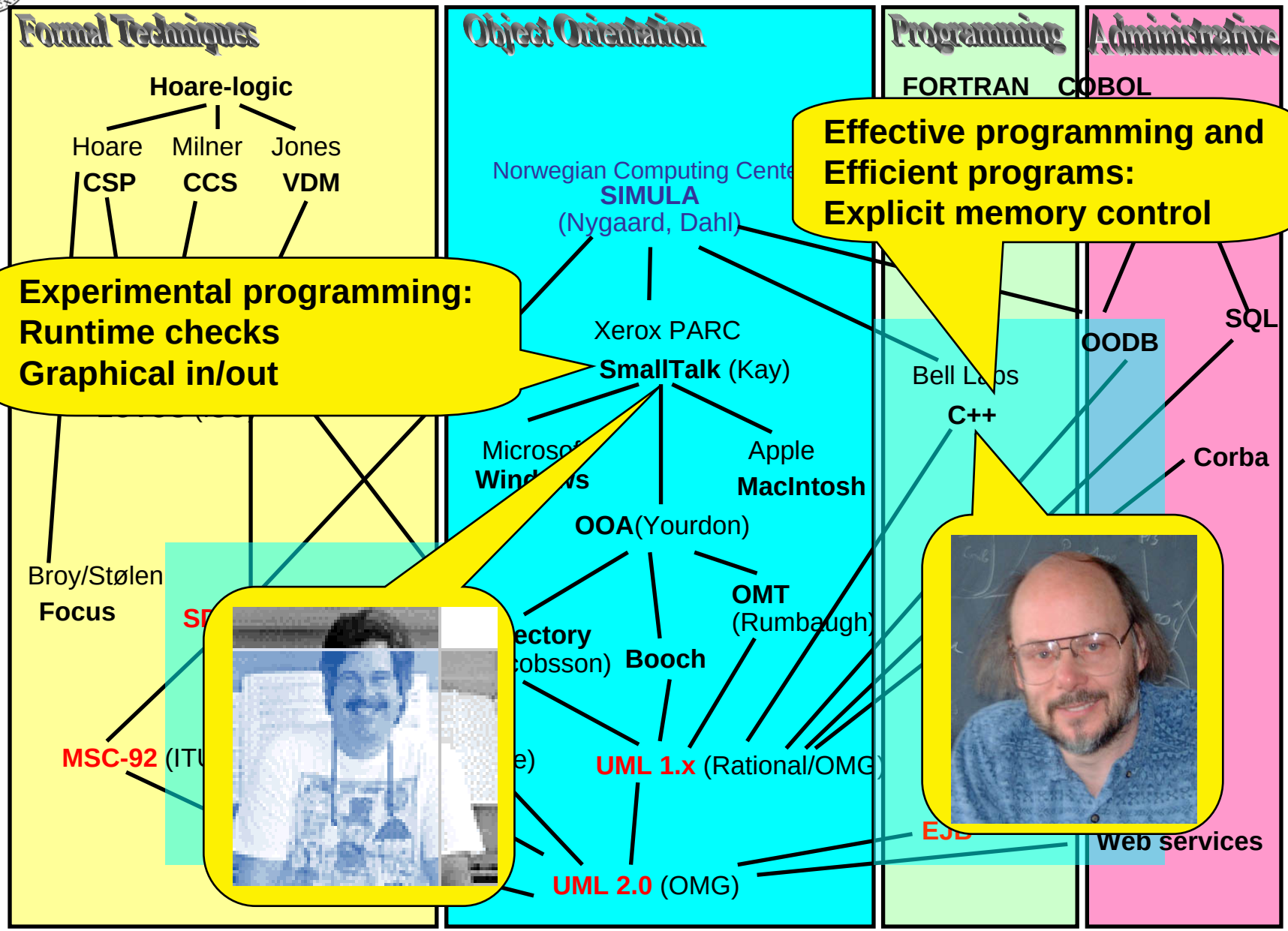
UML 2.0 in the history of languages



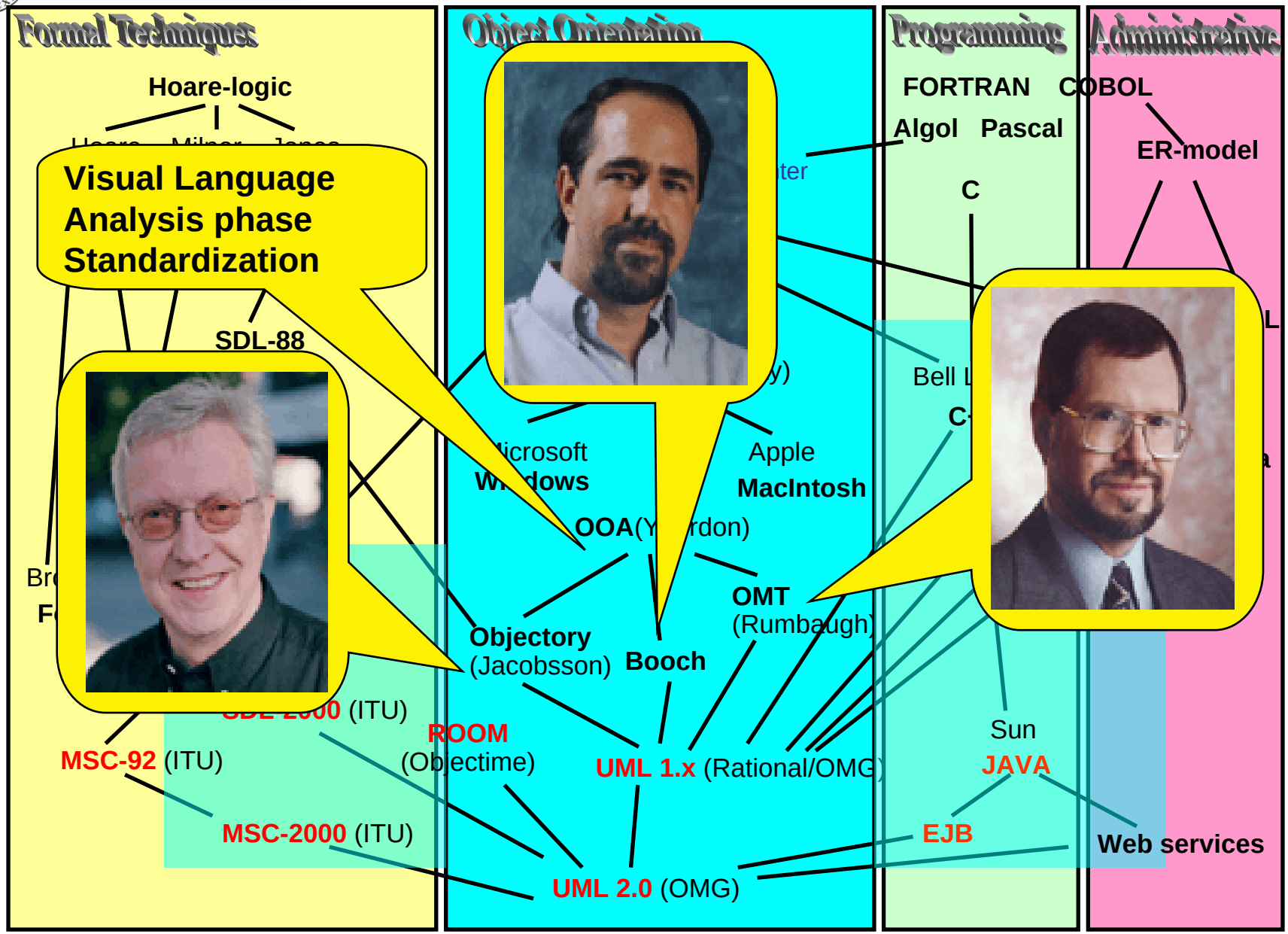
The founding fathers



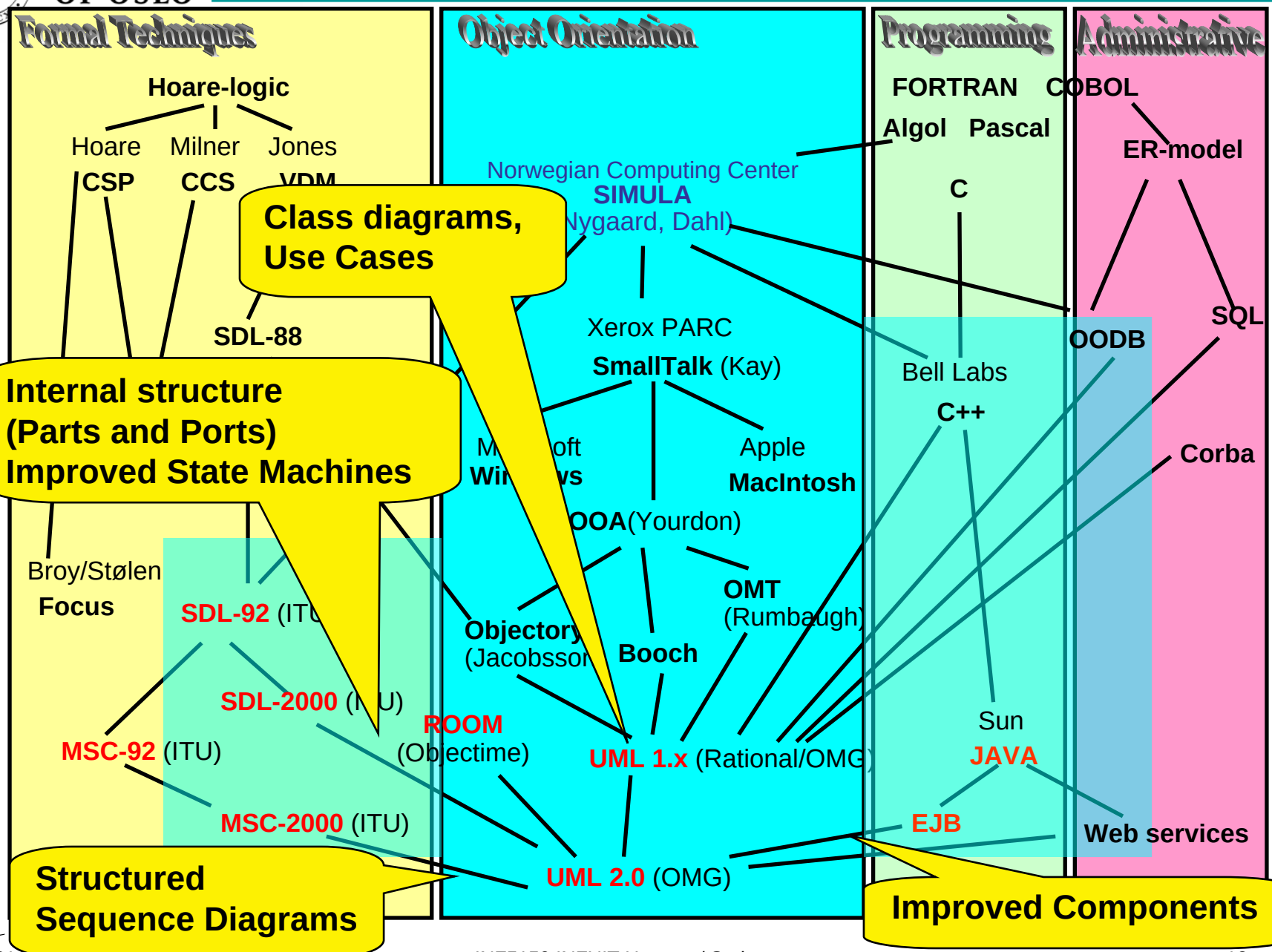
Making OO Popular and Commercial



The Three Amigos



Influences on UML 2.0



What language(s) to use?

- Requirements
  - used in practice for real engineering
  - expressive
  - visual
  - precise
  - trendy
- Alternatives
  - java (Sun)
    - possibly supplied with selected libraries
  - SDL (ITU)
  - MSC (ITU)
  - UML 1.x (OMG)
  - UML 2.0 (OMG)

Why choosing UML 2?

- Pro
  - UML is definitely trendy wrt. modeling languages
  - UML is standardized by open standardization organization (OMG)
  - UML 2.0 has most features of MSC and SDL
  - UML 2.0 is more precise and executable than UML 1.x
  - UML 2.0 is supported by more than one tool, and can be expressed through any drawing tool like Powerpoint, Visio, Framemaker
  - UML 2.0 is now, UML 1.x is history soon
- Con
  - Full UML 2 is hardly supported by any tool, yet
  - Real programmers do not use modeling languages anyway

UML Diagrams

- UML diagrams:
  - Use case diagram
  - Static structure diagrams:
    - Class / object diagram
    - Collaboration
    - Composite structure diagram
  - Behavior diagrams:
    - Sequence diagram
    - Communication diagram
    - State diagram
    - Activity diagram
  - Implementation diagrams:
    - Component diagram
    - Deployment diagram

Use:

Identifying main system functions

Domain and application modeling

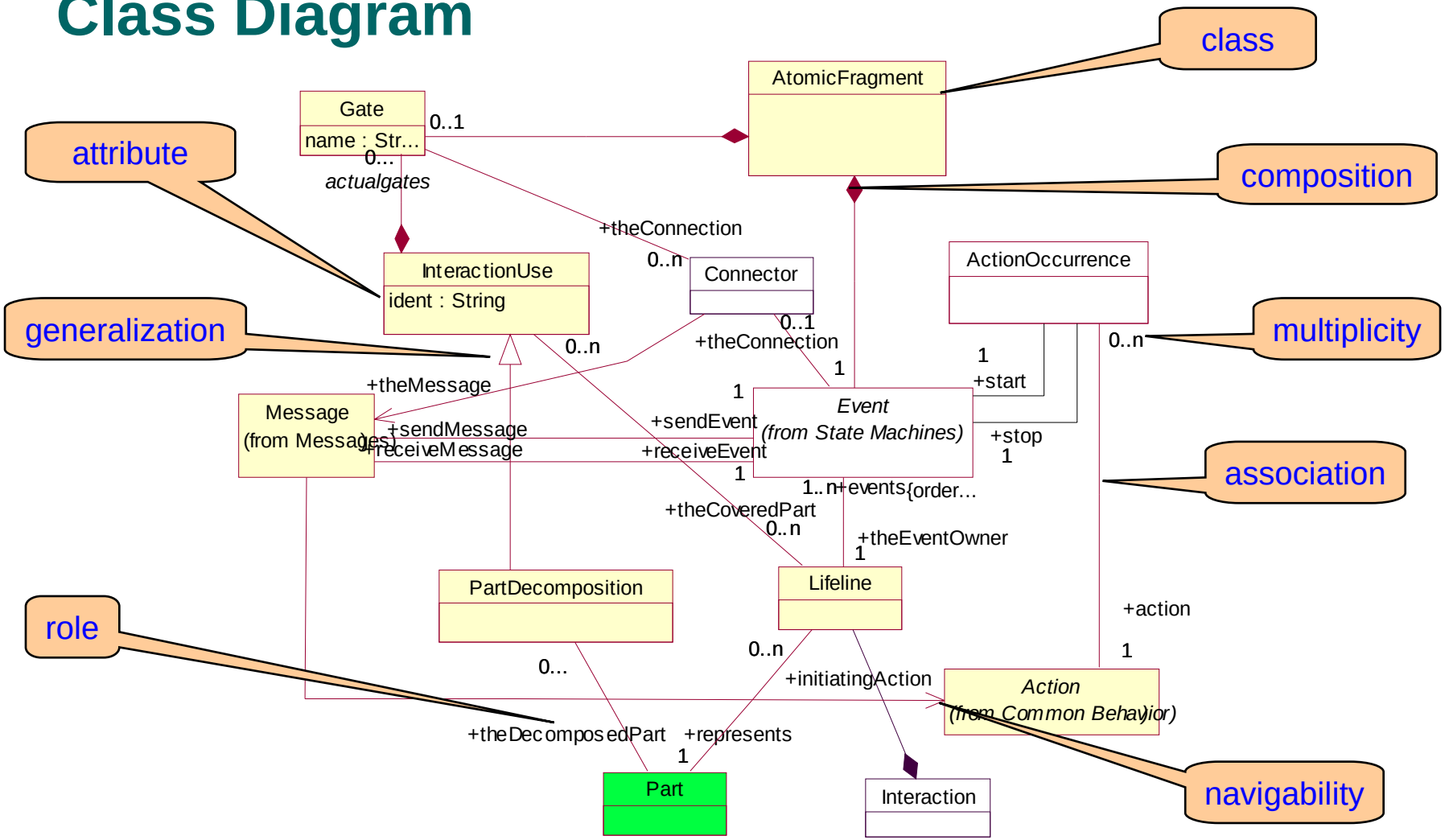
internal structure of objects

Interactions between objects

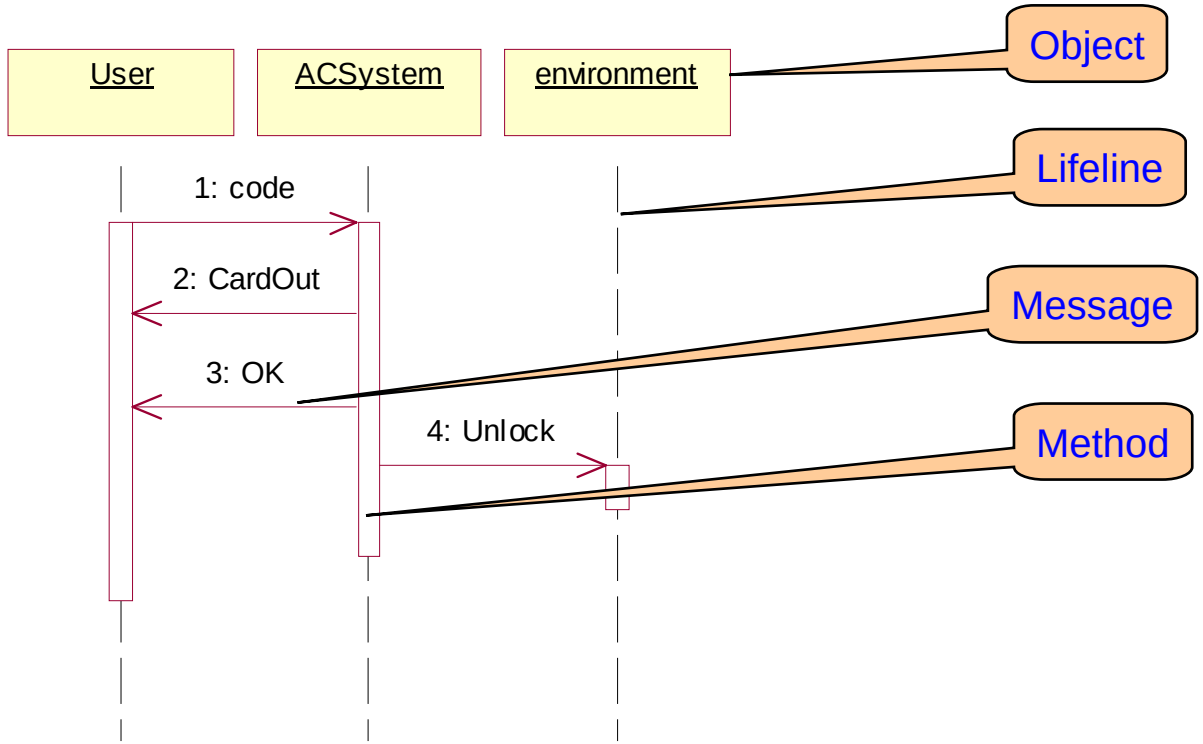
Class behaviour (state oriented)
Ditto (action oriented)

For software structure
For hardware/software structure

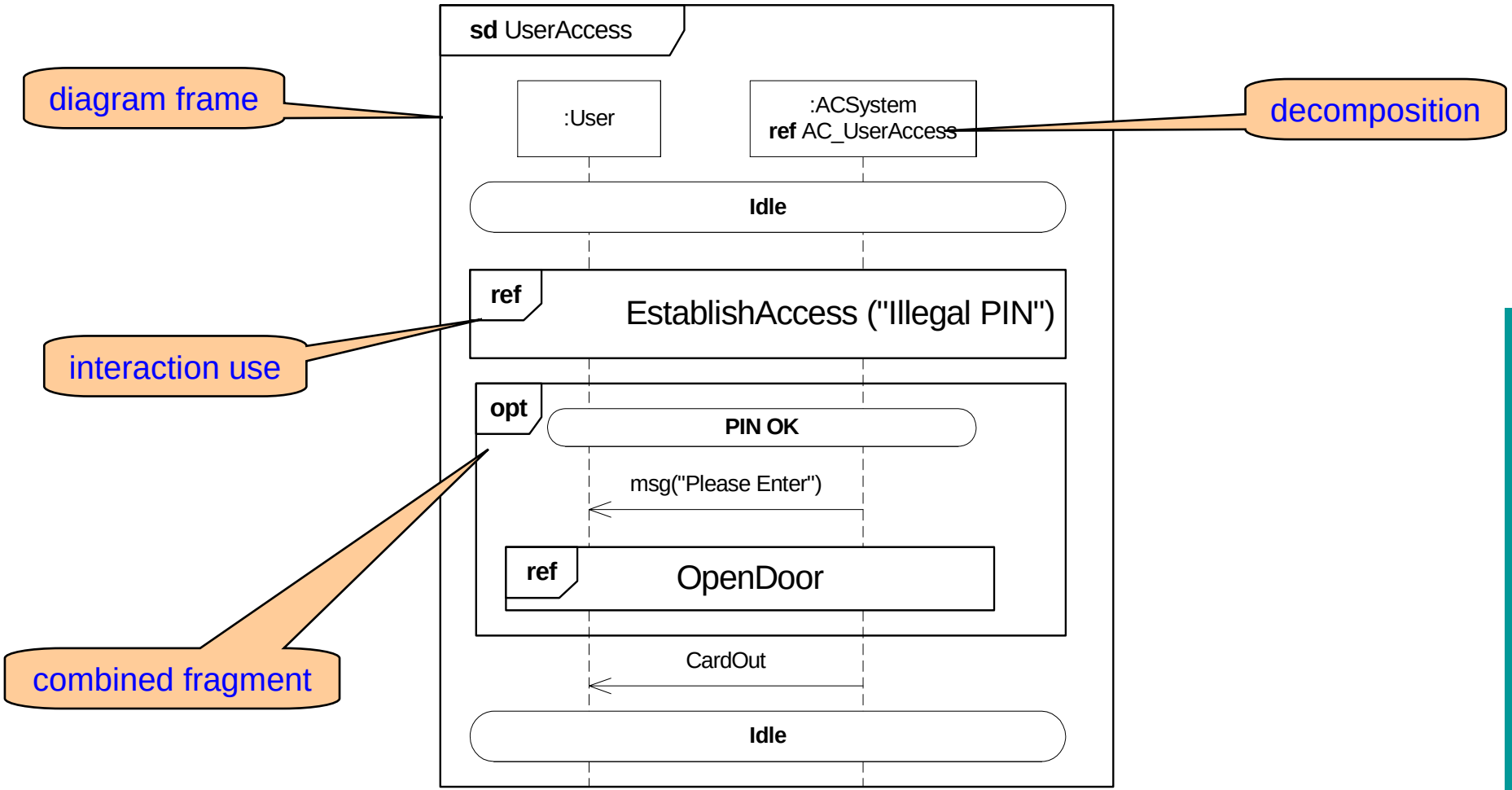
Class Diagram



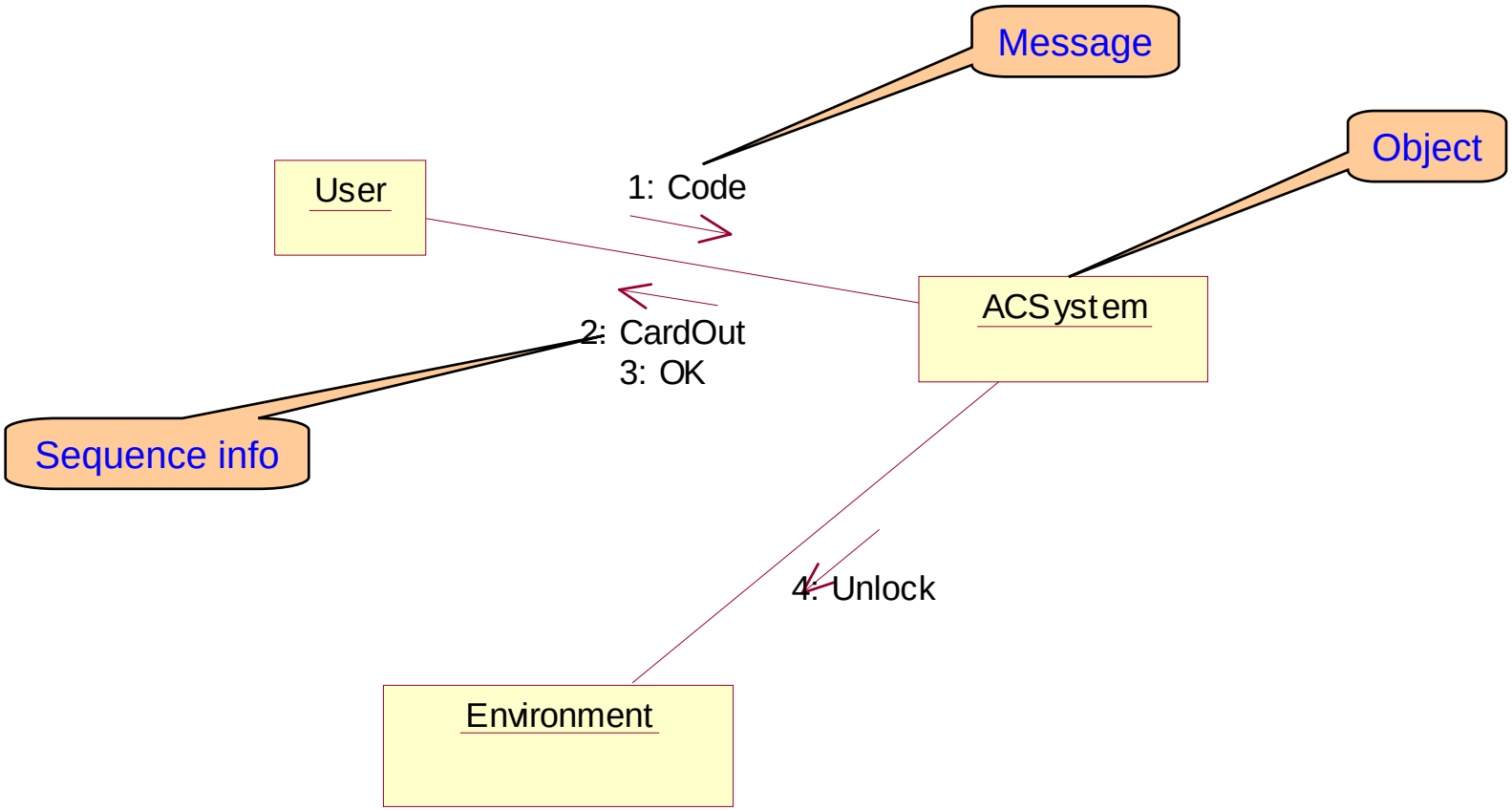
Sequence Diagram (UML 1.x)



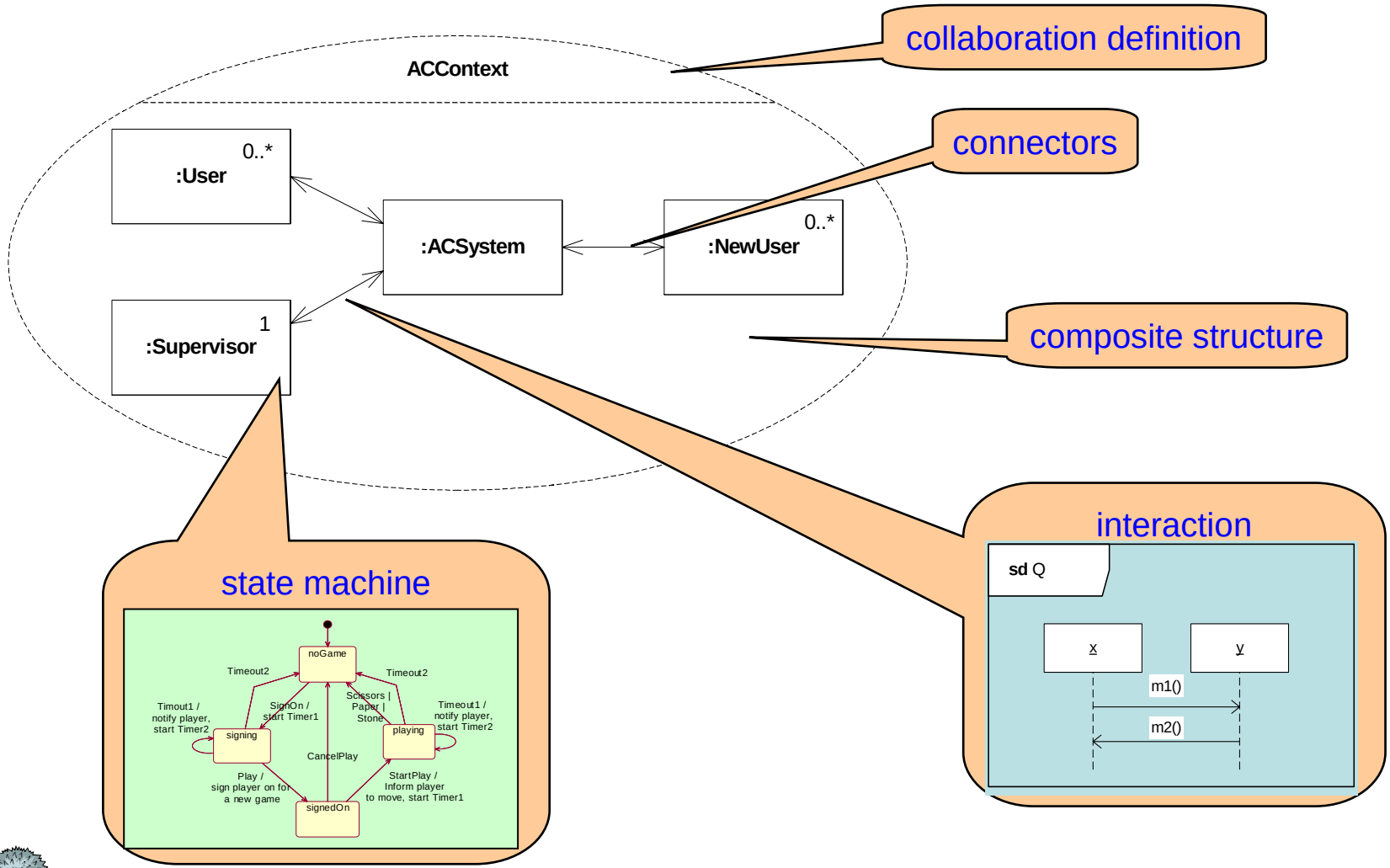
Sequence Diagram (UML 2)



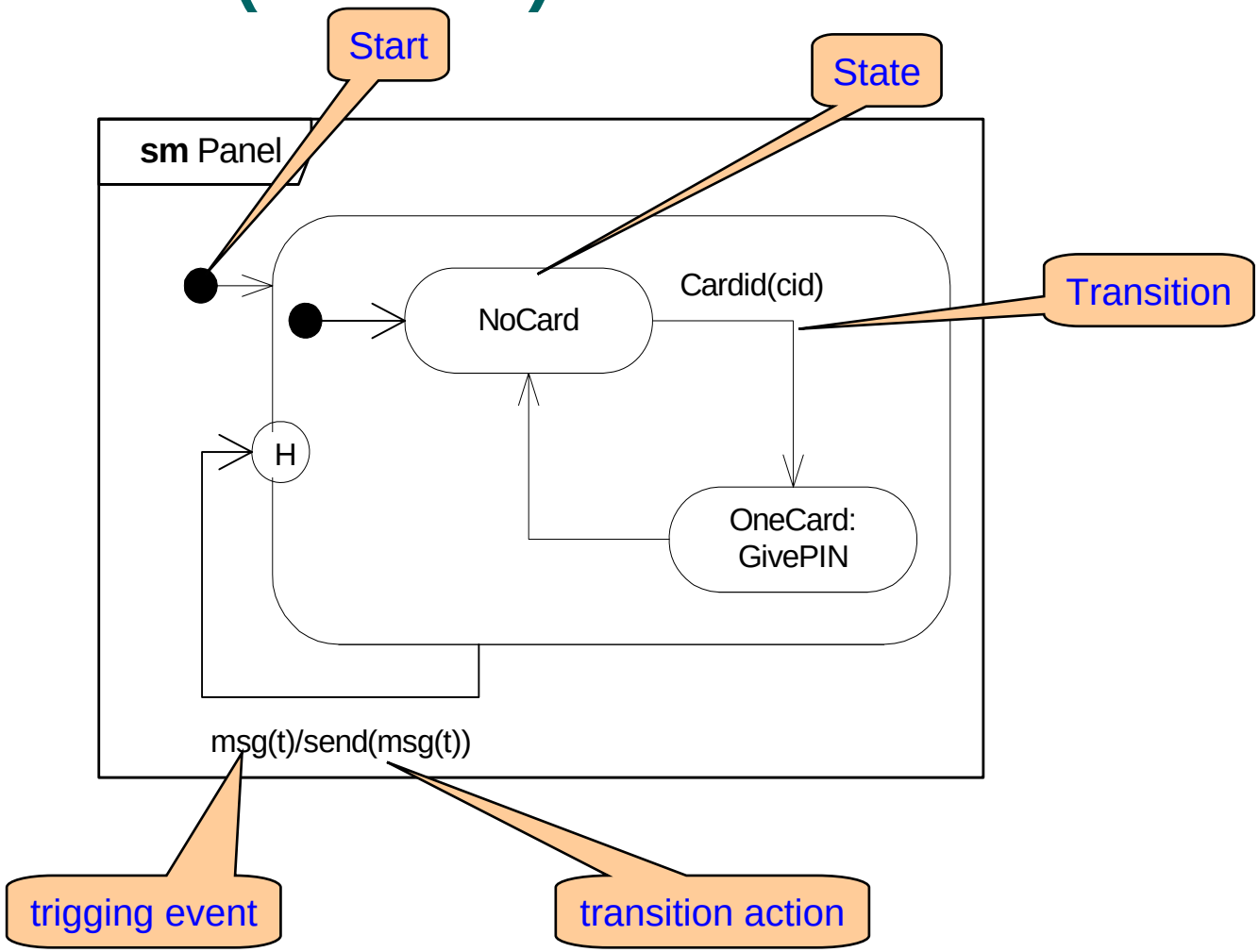
Collaboration diagram (UML 1.x)
Communication diagram (UML 2.0)



Collaborations in UML 2



State Machines (UML 2.0)



How important are languages?

- Not very important
  - "Syntactic sugar"
- Very important
  - "Understanding through describing"

Methodology

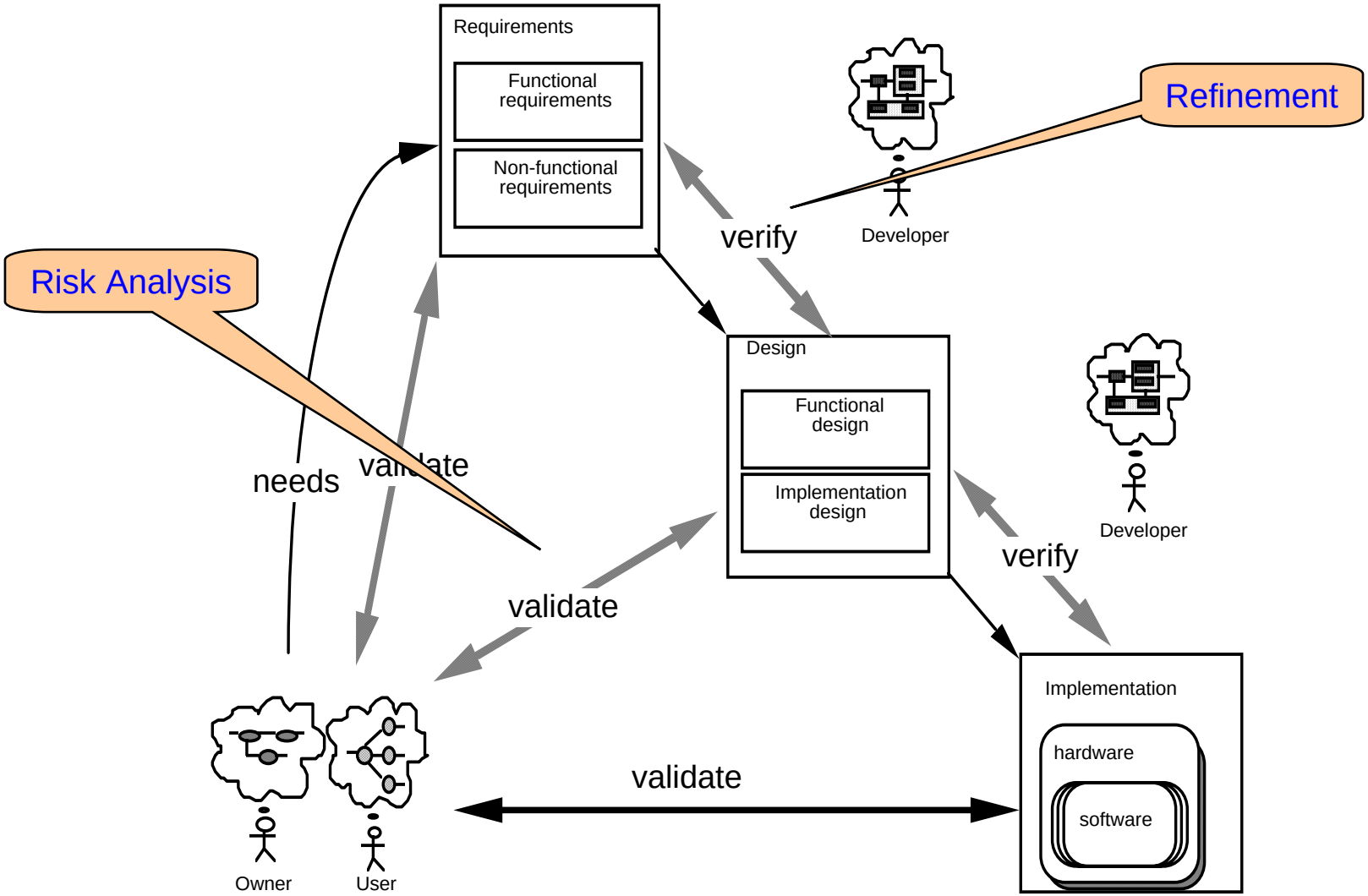
- A good language helps a lot
  - but is hardly sufficient
  - you need to know how to use the language also
- A good method is hard to find
  - easy to understand
  - easy to believe in
  - easy to follow
  - easy to modify
  - easy to get positive effects

  - easy to cheat?
  - easy to overlook?
  - easy to misuse?
  - hard to evaluate?

Verification and Validation

- Barry Boehm, 1981:
  - Verification: To establish the truth of correspondence between a software product and its specification (from the Latin veritas, “truth”). Are we building the product right?
  - Validation: To establish the fitness or worth of a software product for its operational mission (from the Latin valere, “to be worth”). Are we building the right product?
- Quality
  - process quality **= meeting the specification**
  - system quality **= playing the role required by the environment.**
- Quality assurance
  - Constructive **methods that aim to generate the right results in the first place**
  - Corrective **methods that aim to detect errors and make corrections.**

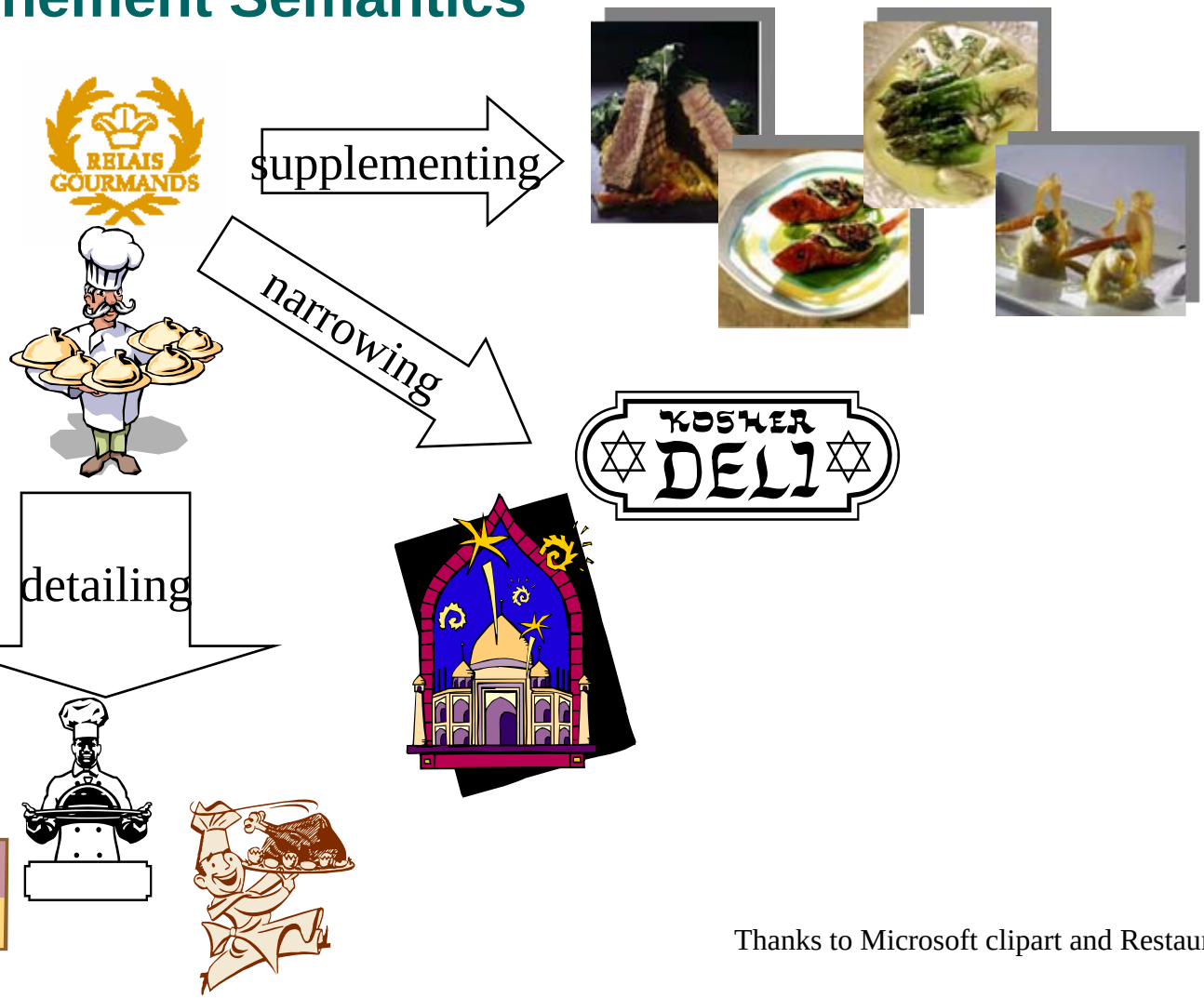
Development model



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

STAIRS – Steps To Analyze Interactions with Refinement Semantics



Thanks to Microsoft clipart and Restaurant Bagatelle's web-site

Refinement

- Refine = to free (as metal, sugar, or oil) from impurities or unwanted material
  - here: to make more exact, to reduce the set of legal solutions
  - in particular: to reduce the set of legal histories
- The role of histories
  - Histories model system runs
  - Specifications are modeled by sets of histories
- The need for a precise semantics
  - Syntax, Semantics, Pragmatics
- The assumption/guarantee paradigm
  - The assumption describes the properties of the environment in which the specified component is supposed to run
  - The guarantee characterizes the constraints that the specified component is required to fulfill whenever the specified component is executed in an environment that satisfies the assumption

Three main notions of refinement

- Property refinement
  - *requirements engineering:* requirements are added to the specification in the order they are captured and formalized
  - *incremental development:* requirements are designed and implemented in a step-wise incremental manner
- Interface refinement
  - *type implementation:* introducing more implementation-dependent data types
  - *change of granularity:* replacing one step of interaction by several, or the other way around
- Conditional refinement
  - *imposing boundedness:* replacing unbounded resources by implementable bounded resources
  - *change of protocol:* replacing abstract communication protocols by more implementation-oriented communication protocols

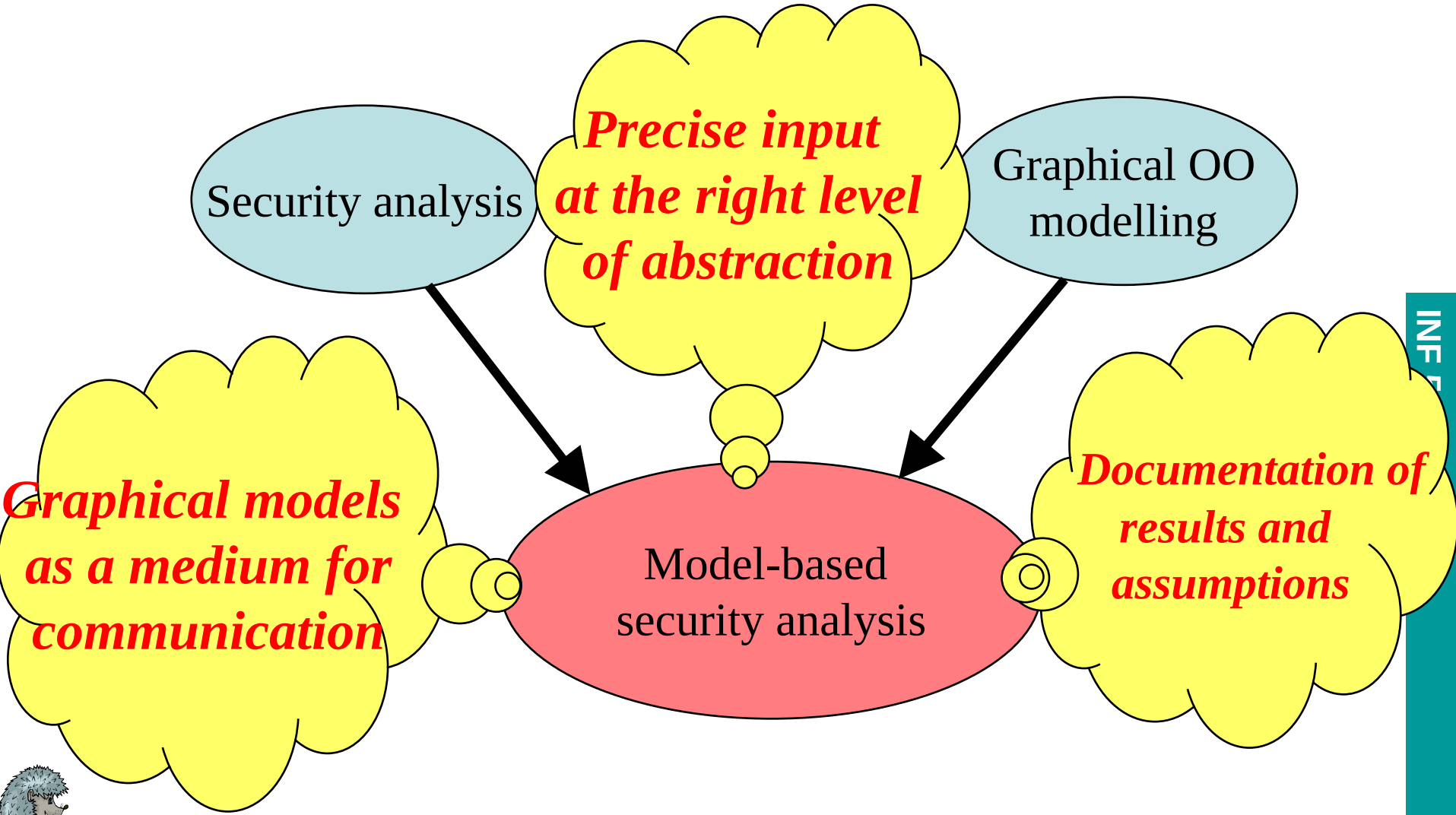
Objectives for the lectures on refinement

- The lectures on refinement will
  - motivate and explain the basic instruments and principles for defining notions of refinement
    - this includes
      - using histories to model executions
      - the notion of an observer
      - understanding the assumption/guarantee principle
  - explain the following refinement concepts in a UML setting
    - property refinement
    - interface refinement
    - conditional refinement
  - demonstrate refinement in examples
- The exercises on refinement will
  - train you in the art of refining, and prepare you for the exam

Model-based security analysis

- Risk analysis is a systematic use of available information to
  - determine how often specified events may occur
  - the magnitude of their consequences
- Model-based security analysis is the tight integration of state-of-the art modeling methodology in the security risk analysis process
- Model-based security analysis is motivated by
  - Precision improves the quality of security analysis results
  - Graphical UML-like diagrams are well-suited as a medium for communication between stakeholders involved in a security analysis; the danger of throwing away time and resources on misconceptions is reduced
  - The need to formalize the assumptions on which the analysis depends; this reduces maintenance costs by increasing the possibilities for reuse
  - Provides a basis for tight integration of security analysis in the system development process; this may considerably reduce development costs since undesirable solutions are weeded out at an early stage

Three dimensions of model-based security analysis



Requirements analysis versus security analysis

Requirements analysis

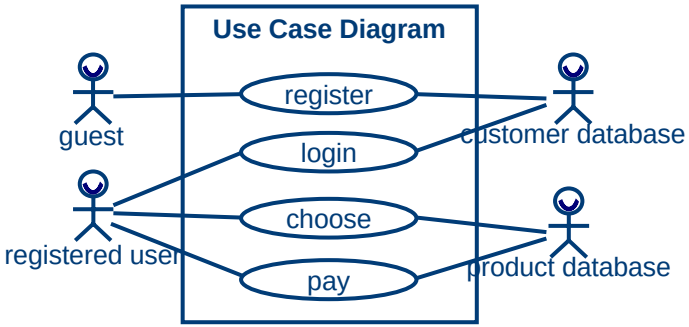
Properties

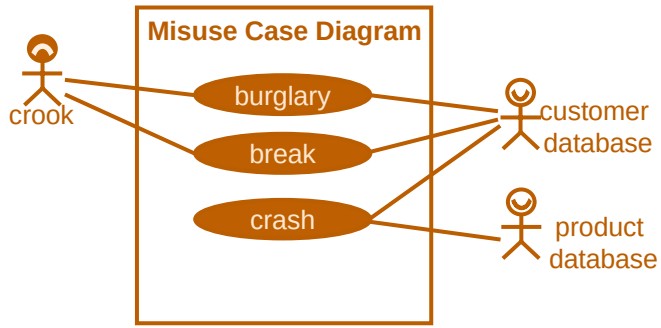
Actors

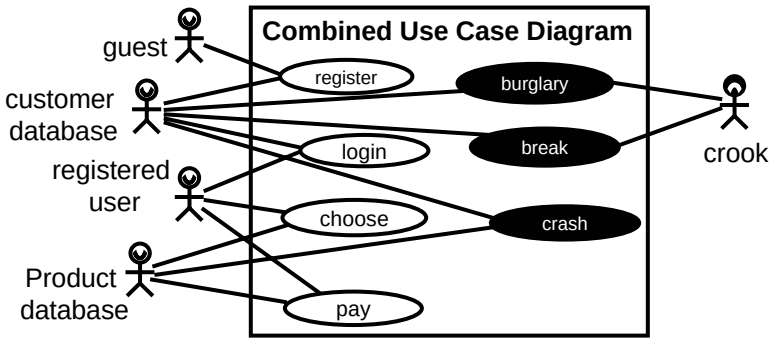
Security analysis

Vulnerability

Attacker







Objectives for the lectures on security analysis

- classify notions of dependability
- introduce, motivate and explain the basic notions and principles for risk management in general and security risk analysis in particular
- relate risk management to system development
- describe the various processes involved in risk management
- motivate and illustrate model-based security analysis
- present relevant standards
- demonstrate the usage of concrete analysis methodology

Obligatory Exercise: I See You Plus (1)

- Based on project from INF2120 Spring 2006
- We will give the INF5150 students access to two solutions to the INF2120 project along with the brief evaluation of these solutions by the teachers of INF2120.
- The solutions are executable such that INF5150 students can experiment with their solutions directly.
- None of the INF2120 solutions are perfect.
- They are not even perfect solutions to the INF2120 project task, but they are good attempts.
- The INF5150 exercise is not identical to the INF2120 task, but similar.

Obligatory Exercise: I See You Plus (2)

- Basing the INF5150 exercise on the INF2120 one simulates real life where projects seldom start from scratch.
- Rather new products build on old solutions with additions, modifications and corrections. This is also what we shall do in INF5150 this year.
- We have never tried this approach before.

Obligatory Exercise: I See You Plus (3)

- ”I See You Plus” is a personal surveillence system
- You may position the GSM mobile phones of your buddies and place their positions on GoogleEarth
- You may also get positioning information on SMS relative to predefined hotspots
- The solution has some implementation-oriented requirements that go beyond the solutions given by the INF2120 groups.
- Security analysis will also be associated with this project

Obligatory Exercise: Tools (1)

- Rational Software Modeler (based on Eclipse 3.0)
  - runs on Linux as well as Windows
  - students can obtain copies for free
- Sequence Diagram editor (SeDi3) plugin
  - the best sequence diagram editor there is (*Andreas Limyr / Frank Davidsen*)
  - tightly integrated with RSM – works on the same repository
- UML to JavaFrame transformer as plugin to RSM
  - push a button – executable UML! (*Asbjørn Willersrud*)
- Supplied interfaces (UML ports) to
  - PATS-lab for SMS-sending and positioning
    - Use only Telenor subscription mobiles!

Obligatory Exercise: Tools (2)

- The CORAS-tool available as open source (LGPL-license):
  - http://coras.sourceforge.net/
- Based on other open software (Apache Cocoon, eXist XML database)
- Created by Sintef

Obligatory exercise: Procedures

- Project groups
  - we put them together in a couple of weeks
- Drop 1:
  - Mandatory guidance by the assistants
  - Hard deadline at 23.59
  - Presentation with projector
  - Criticism by another group (written) and by the assistants
  - Public grading by lecturers
    - but this has no effect on the final grade
- Drop 2:
  - A baseline is given by the assistants
  - otherwise it is very much the same story as Drop 1

INFUIT (INF 5150) in a nutshell