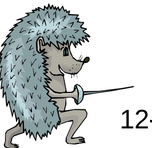
Development Methodology

Version 061013

What shall we learn in school today?

- Why your java programs fail
  - The solution
    - thinking in a way corresponding to how your program will work
- Methodology
  - Some useful tips
- Dialectics – making conflicts drive the development
  - early conflicts are less dangerous
  - people with complementary competence is fruitful
  - complementary views help see the whole picture
- The need for harmonization

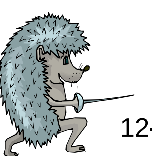


Why your normal Java program fails

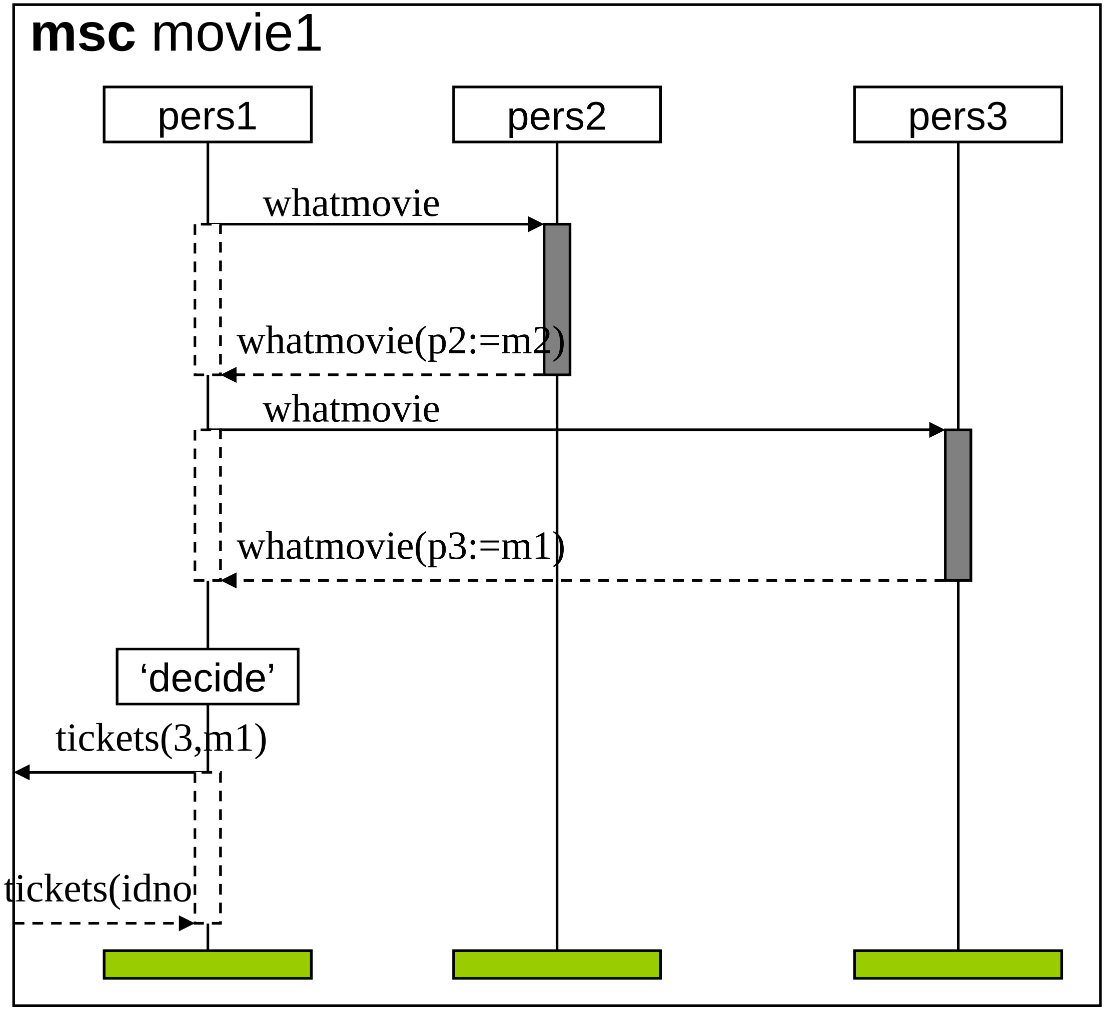
- or how to think in correspondence with how the computer works

Agreeing on which movie to watch

- A group of persons are going to agree on which movie to watch this evening
- There is only a small number of movies (less than the number of persons). One can assume that the decision can be based on democratic principles: the movie with most votes win.
- We will use three different ways of communication:
  - (half-) duplex two-party telephony (synchronizing communication)
  - conference call (synchronous communication)
  - SMS (asynchronous communication)

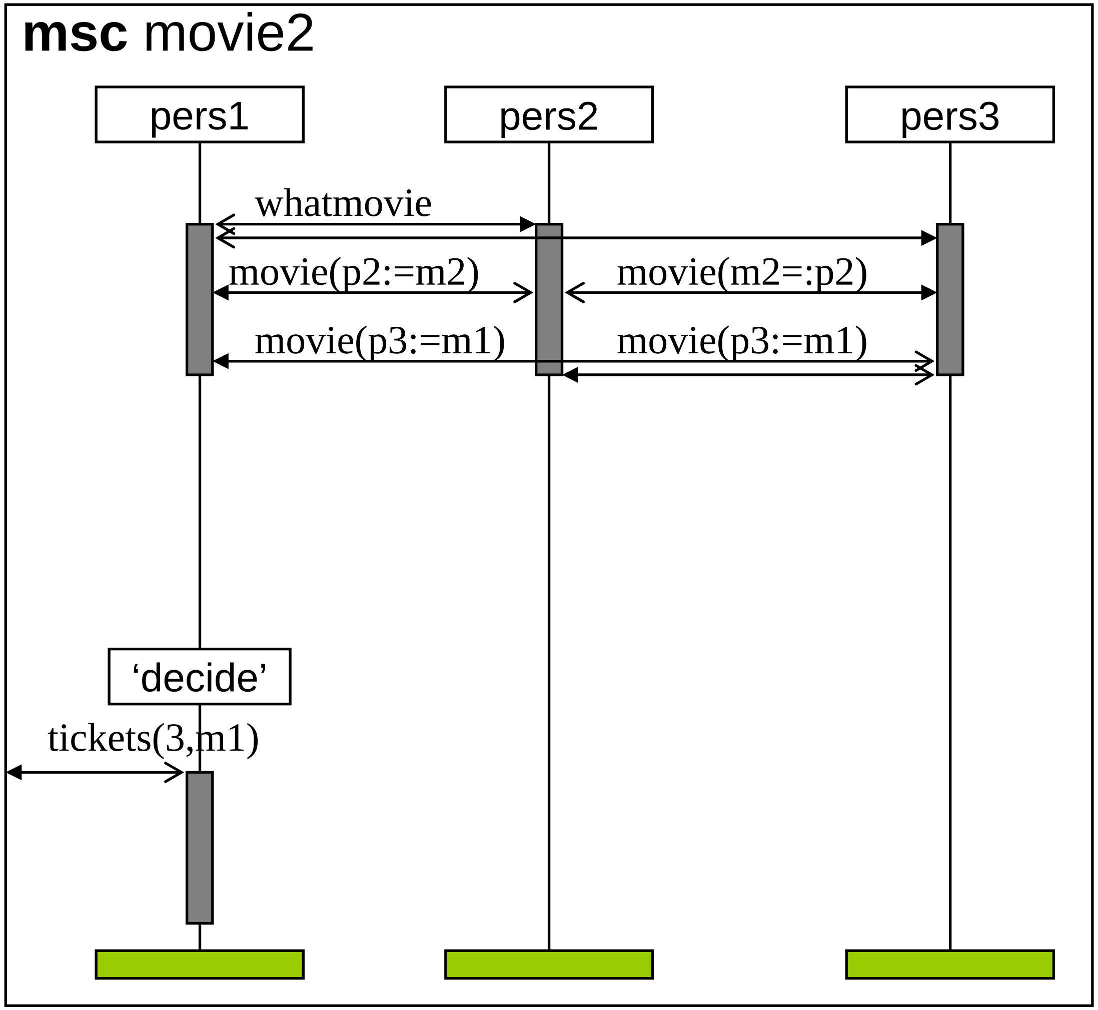


Synchronizing communication



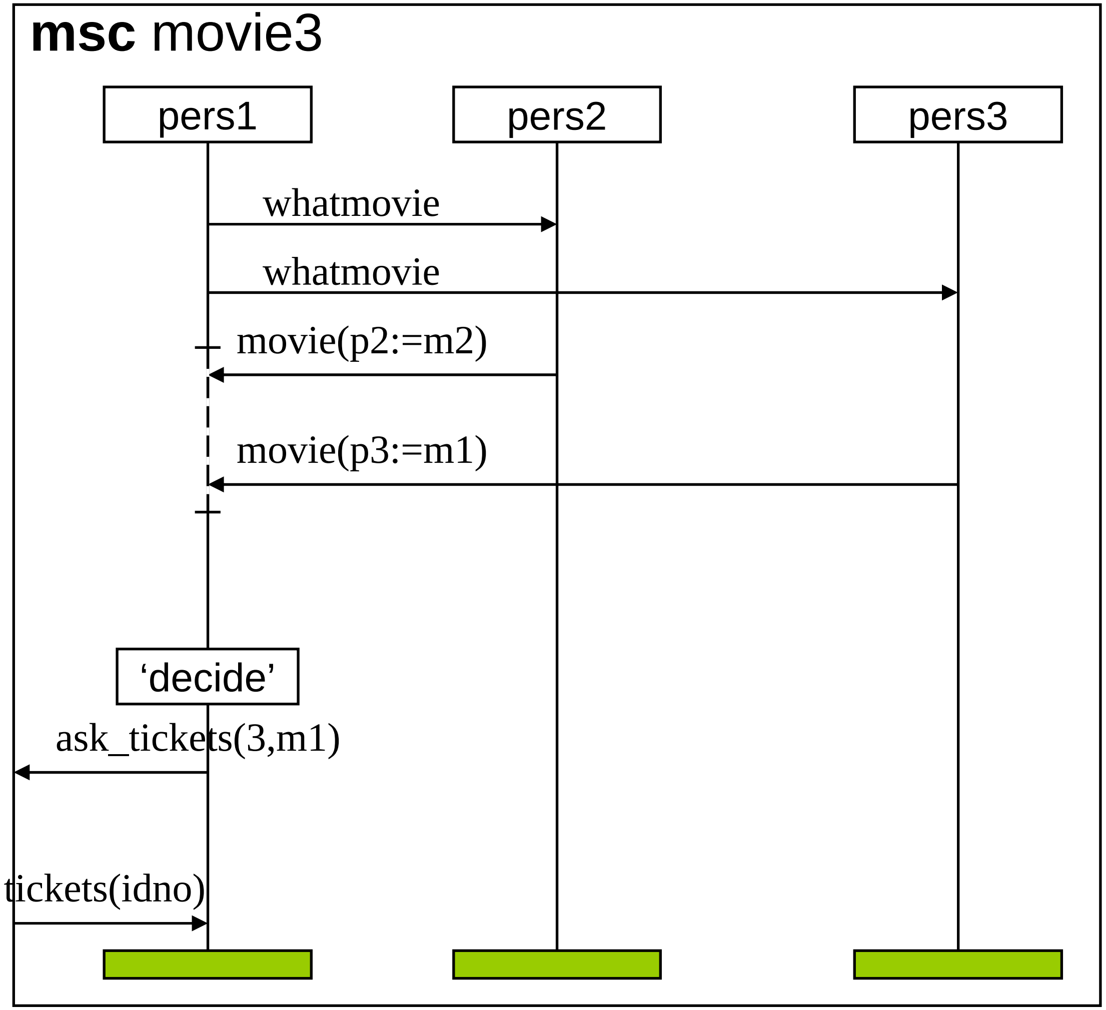
- *pers1* is the master
- *pers2* and *pers3* are slaves
- *pers1* cannot perform anything while *pers2* and *pers3* are trying to decide for themselves

Synchronous communication



- *pers1* is the central
- *pers2* and *pers2* are co-workers
- neither of the persons can do anything while the communication lasts
- (this is informal MSC since MSC-2000 have no mechanisms for synchronous

Asynchronous communication

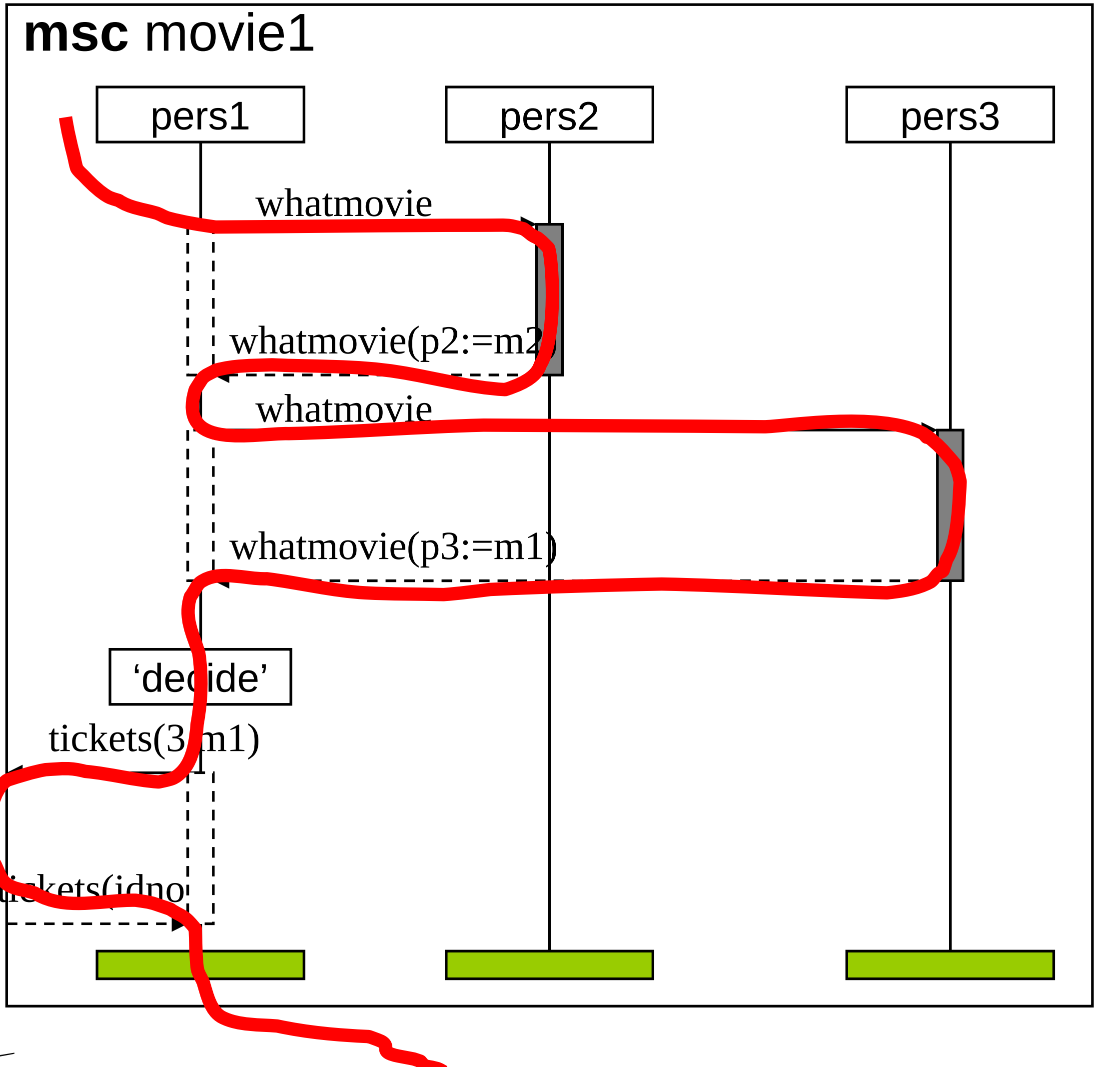


- *pers1* is the central
- *pers2* and *pers2* are co-workers
- *pers1* can do other kinds of work while *pers2* and *pers3* decide their opinions
- *pers2* and *pers3* can make up their opinion in parallel

Threads

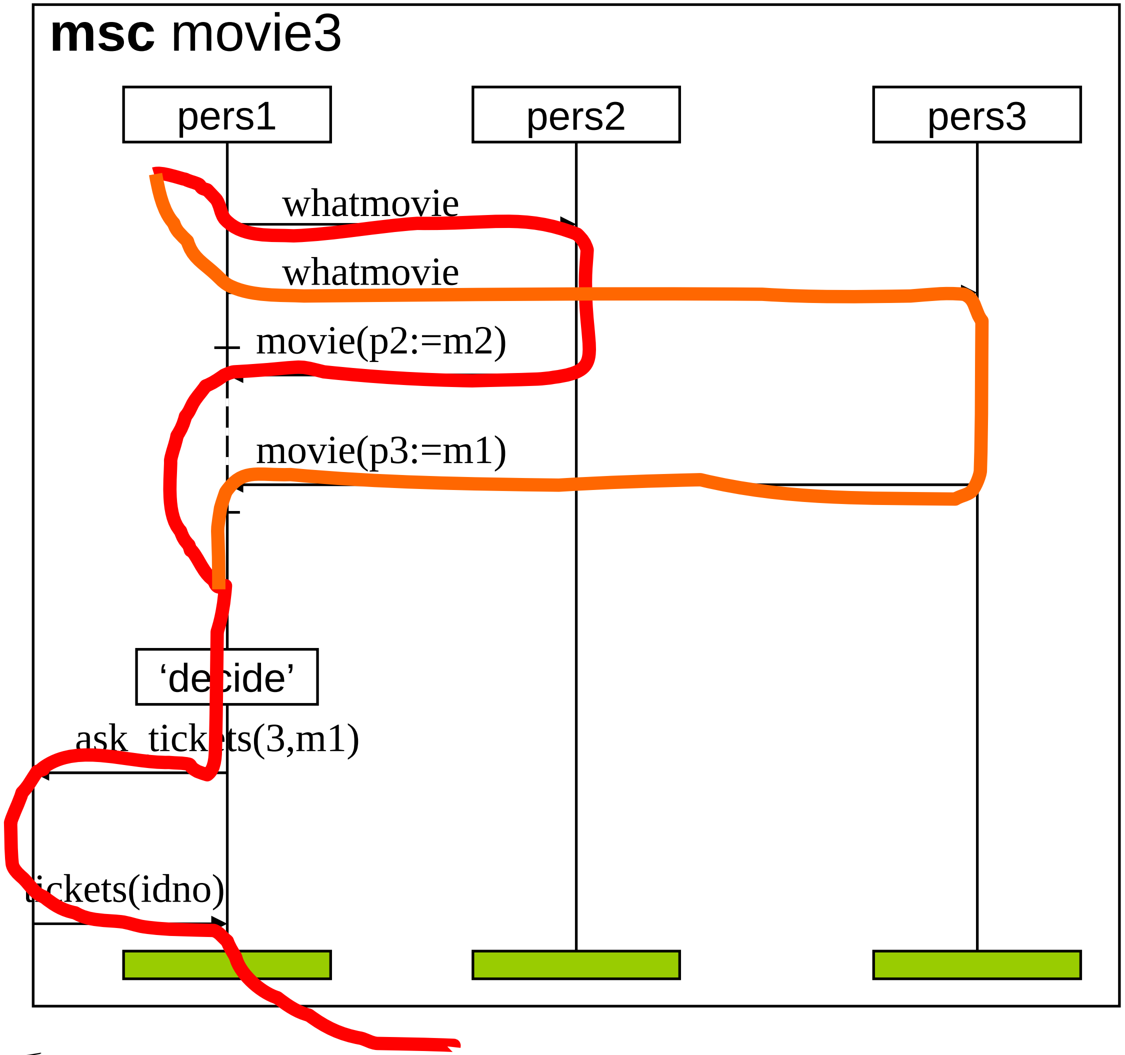
- Threads are flows of control
  - the metaphor is that the threads go through the web of objects like a thread in the fabric of a shirt that is sewn
- Threads are said to be “light weight processes”?!
  - threads are not operating system tasks
  - threads refer to the same address space (object space)
  - threads must be considered concurrent
- What is the canonical mental model of threads?
  - this is a very hard question, and we shall try and look at this ....
- Are there simple ways to ensure thread-safe programming in Java?
  - there is no simple way, but some approaches are safer than others
- Threads can be used to enforce priority
  - but be conscious about what you can achieve through priority

Threads 1



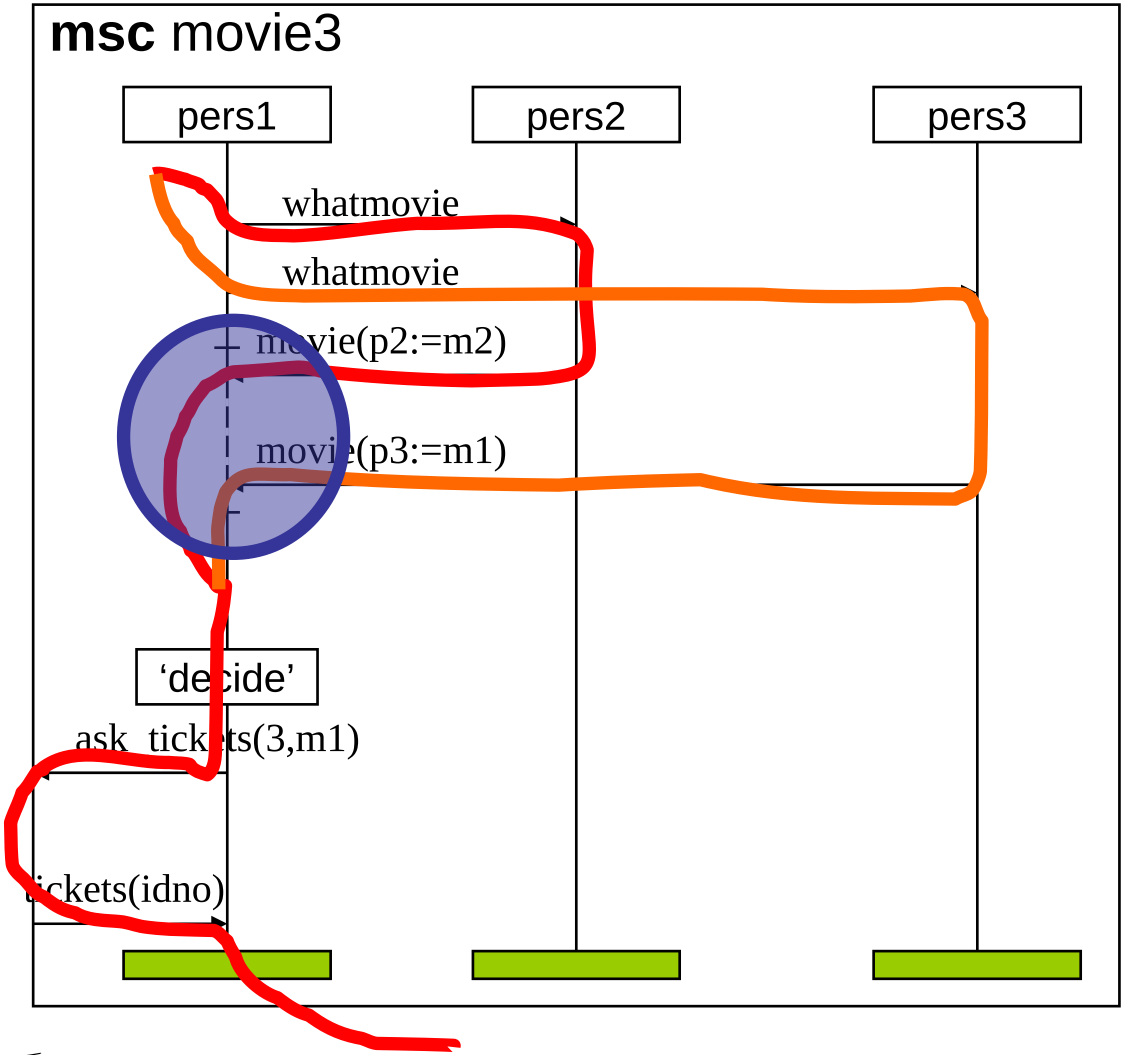
- one thread
- in fact the whole system is sequential!
- anybody can program this in Java

Threads 2



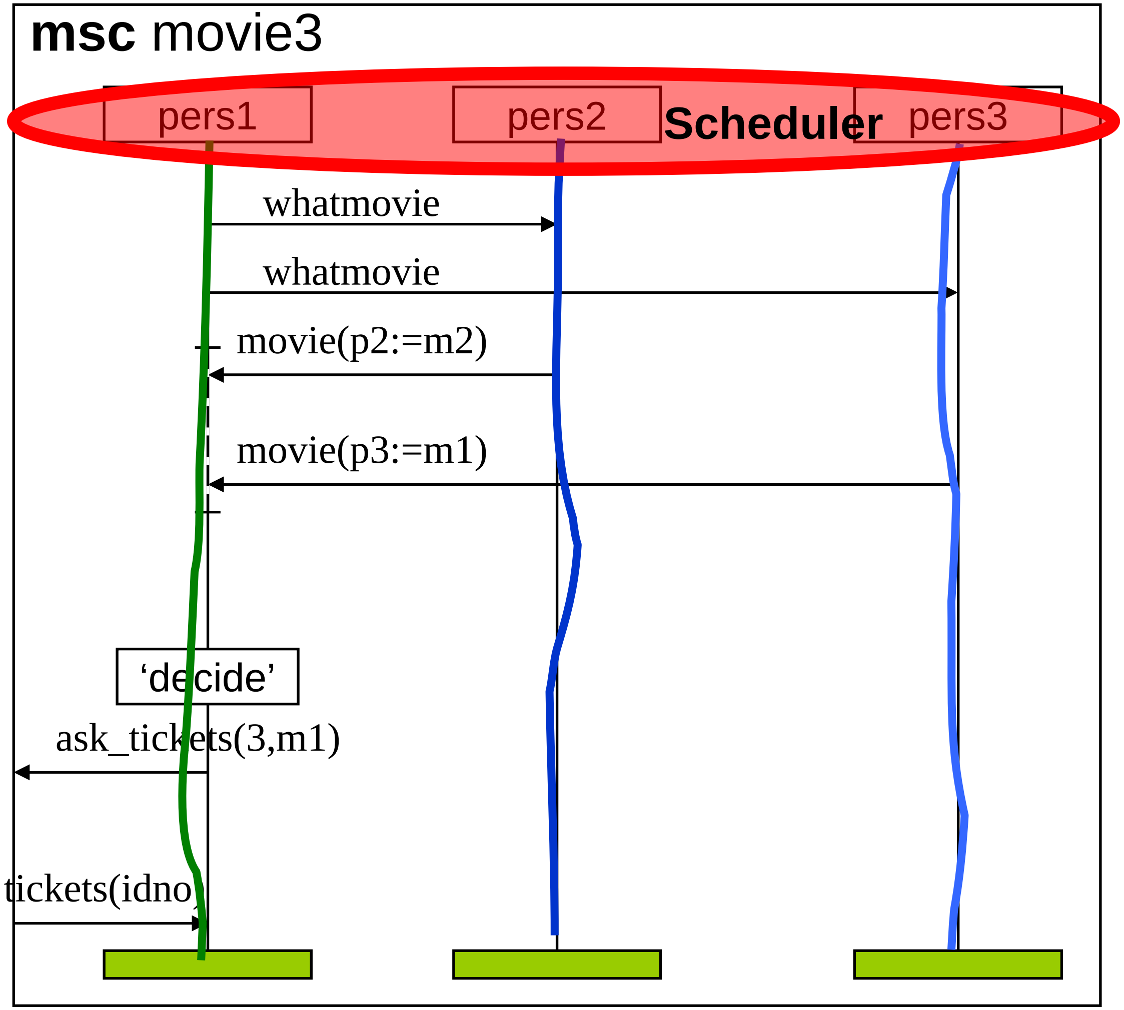
- there are two independent threads of control
- in fact there could be even more since *pers2* and *pers3* could have had other business to attend to!
- as it is, it is a fairly simple "fork" / "join" and quite simple to program
- such a local fork and join is still almost sequential

Threads 2 (more)



- Problems
  - technical
  - conceptual
- If *pers1* following messages *movie* also updates the count for each movie, there is a concurrent update problem
- Who are the threads? Are they concepts?

Threads 3 (JavaFrame / UML / SDL)



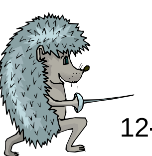
- *pers1*, *pers2* and *pers3* are all ActiveObject
- they are StateMachines
- *pers1* is Leader
- *pers2*,*pers3* are Followers
- There is one (or more Threads) controlled by Schedulers
- Schedulers are hidden for the programmer

Object Orientation

- The objects are the performers / executors
- They themselves perform their methods
- In Java in fact the Threads are executing the methods
- This means that the same object may be executed from different Threads, but conceptually being one active object in itself

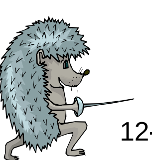
Why we make errors with Threads in Java

- You use another Thread to achieve higher speed
  - usually wrong, if it is on the same machine, it will slow the machine down, not speed it up
- You use several Threads, but lose track of them because they are not associated closely with concepts
  - You use several Threads, but your concept of the ActiveObjects are not associated with them
- You are using a synchronizing approach and believe that the program is essentially sequential, but alas...
  - another programmer does the same, but your Threads interact without synchronization on some obscure common object
- You know about Thread problems and use synchronized methods to a large degree
  - either you run into deadlock, or very inefficient programs



Why use several Threads in Java?

- There are real external stimuli that should be handled according to interrupts
  - it would be better if all (or many) interrupts could be handled by the same Thread since Threads consume resources
- There are some parts of the system that requires better priority than the rest
  - Giving priority could give improved performance
    - duration of transitions vary considerably
  - Certain urgent operations are done in time,
  - but priorities should not be used in reasoning about the overall functionality
- The system is physically distributed over several machines
  - Then it is obvious that we need more than one JVM (Java Virtual Machine)

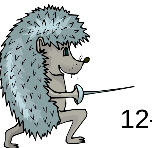


UNIVERSITY OF OSLO

Why UML 2 / JavaFrame is different

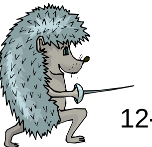
- The predominant model of UML 2 State Machines / JavaFrame is that of telecom:
  - concurrency is an opportunity, not a mere threat
- Execution logic is tied to the programming concepts
- Execution performance discriminates between the programmers' level and the execution platform
  - Threads are dealt with separately from the functional logic
- High degree of independence implies:
  - parallel design possible
  - modifiability / flexibility
  - early simulation / prototyping
  - known validation approaches
- In short: dependability with less efforts

INF 5150



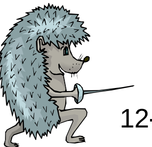
Methodology

originally from
Bræk&Haugen
”Engineering Real Time Systems” from 1993
made in the SISU project



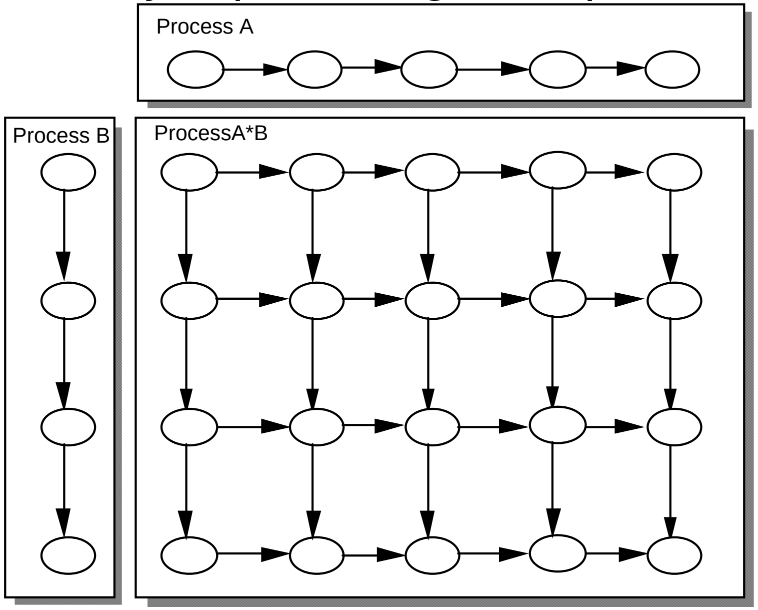
The goals of the design

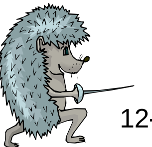
- ***Readable*** in the deep semantic sense that it supports collective understanding in a project team. It should support unambiguous communication among project members and in-depth understanding by the individual.
- ***Analyzable*** in the sense that properties can be derived and compared with requirements.
- ***Implementable*** in the sense that the described functionality can be implemented in a way that satisfies non-functional requirements.



S-rules on concurrency

- Model independent and parallel behaviors as separate processes.
  - Parallelism is a real world fact
  - Parallel processes do not interfere with each other
  - Concurrent processes help to achieve encapsulation and modularity
  - Concurrency implies a logical separation of substance



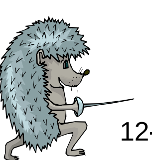


S-rules on structuring

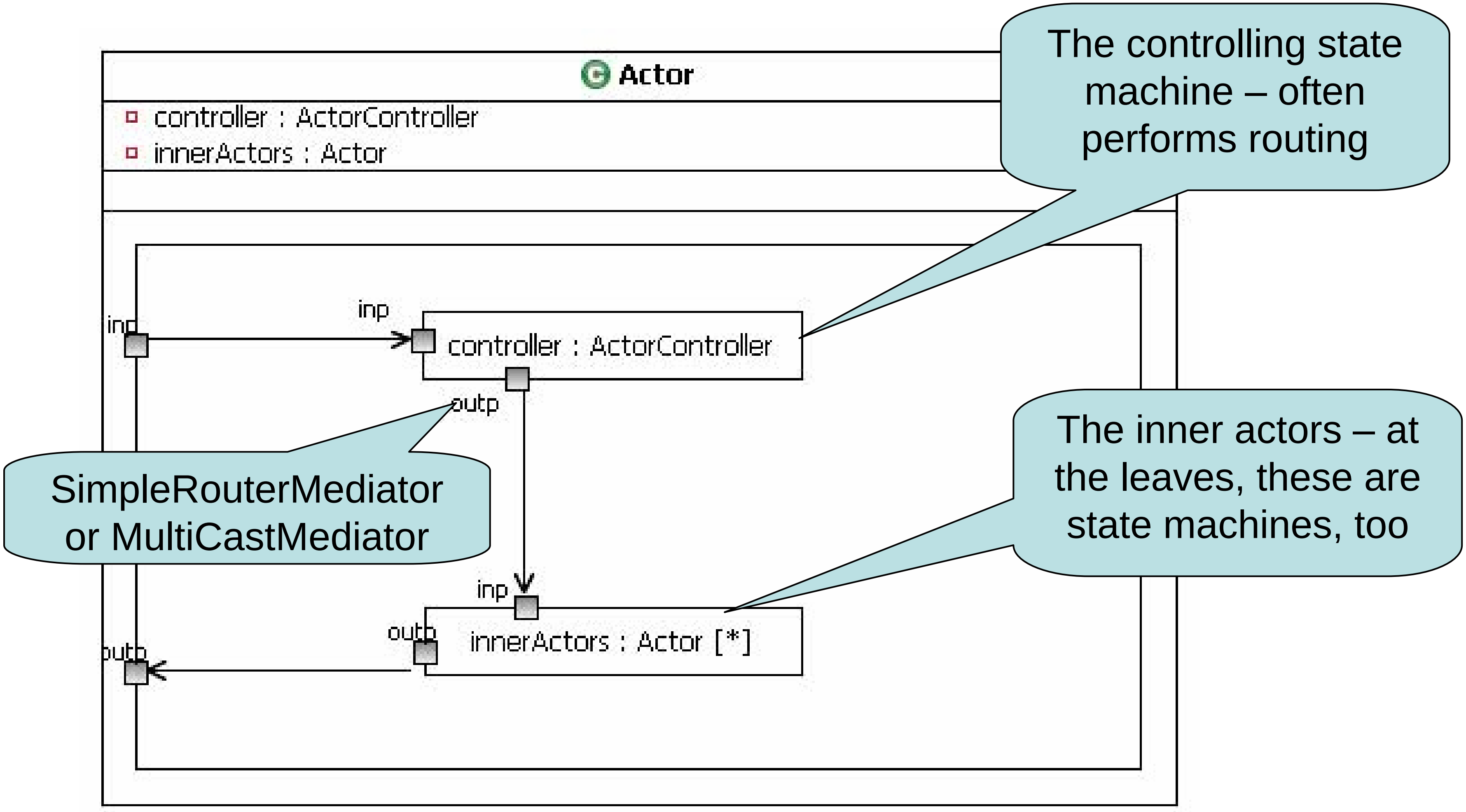
- interconnections
  - Use one channel and/or signal route to carry each independent and concurrent interaction dialogue.
- system and environment
  - For the elements at the periphery of your concern, place them inside the system if you wish to describe their behavior in detail
  - If you are merely interested in their signal interface, place them in the environment which means they will not be identified explicitly in UML

S-rules on purpose of composite structures

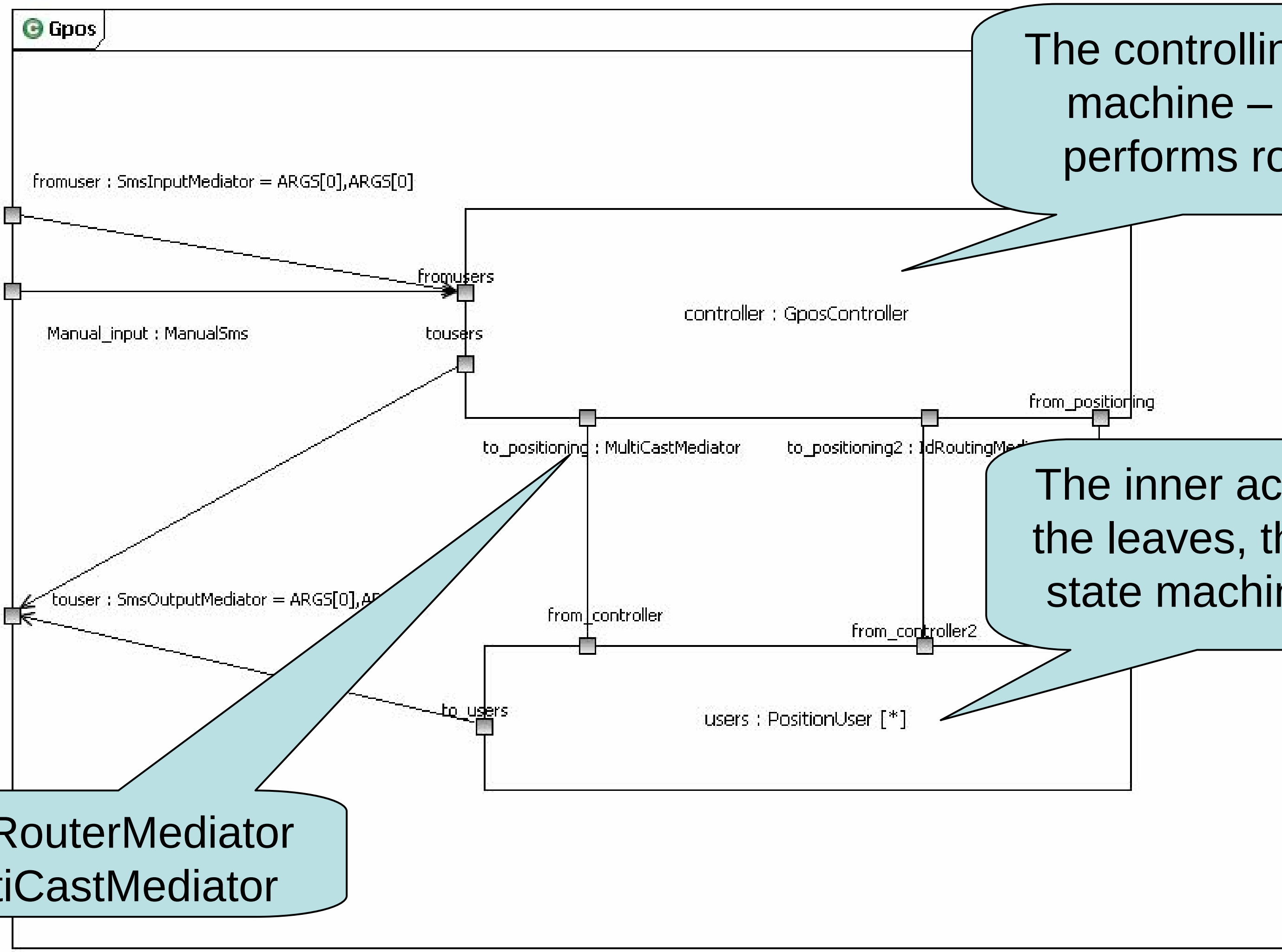
- Gradual approach to detail
- Units of reuse and repetition;
- Encapsulation of layering;
- Encapsulation of independent adaptation and change;
- Limited scope of process creation and communication;
- Correspondence with the physical system.



The Actor architecture



Does the pattern apply to the Basic Service?



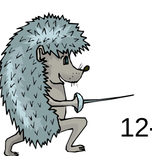
The controlling state machine – often performs routing

The inner actors – at the leaves, these are state machines, too

SimpleRouterMediator or MultiCastMediator

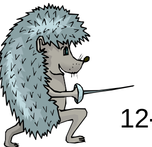
The purpose of Ports

- Ports represent interfaces
  - that are separate from their owner
- Ports may therefore remain while the owner is exchanged
  - change implementation during runtime without having to traverse the whole architecture to update the connectors
- Ports represent the only way to communicate with their owner
  - encapsulation
- Ports are often used for simple routing
  - while more complicated routing is done by state machines
- Use Ports always
  - even though UML 2 does not require them

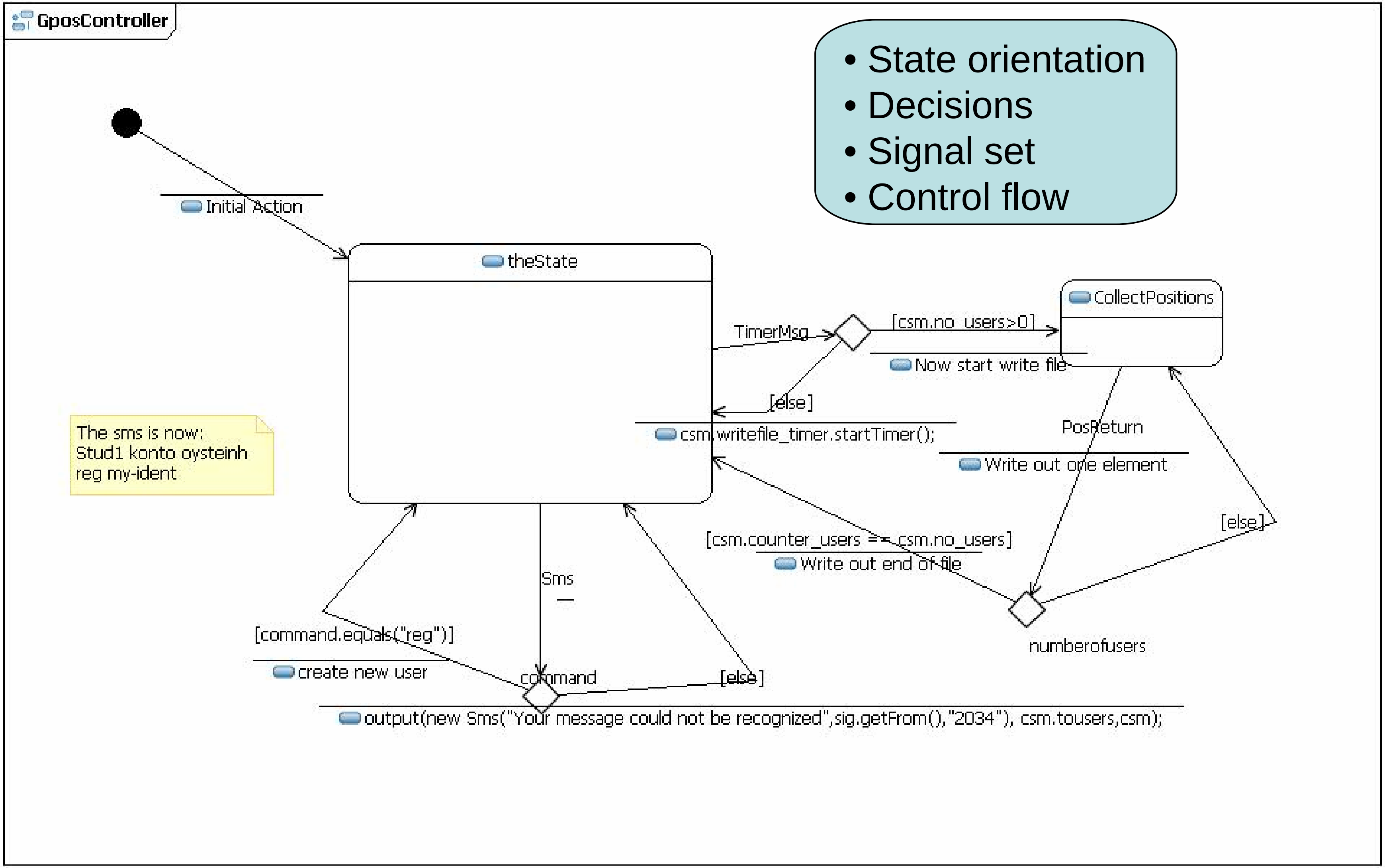


S-rules on state machines

- state orientation
  - Represent what the environment may distinguish as control states of the process, as states in the process graph.
- decisions
  - Critically review all decisions to ensure that they are not symptoms of undesirable state hiding.
- signal set
  - Represent what the environment may distinguish as different control signals by different signal types
- control flow
  - Branch on input signals in states rather than on decisions.



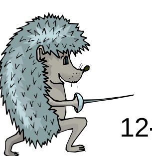
Does this apply to GposController?



- State orientation
- Decisions
- Signal set
- Control flow

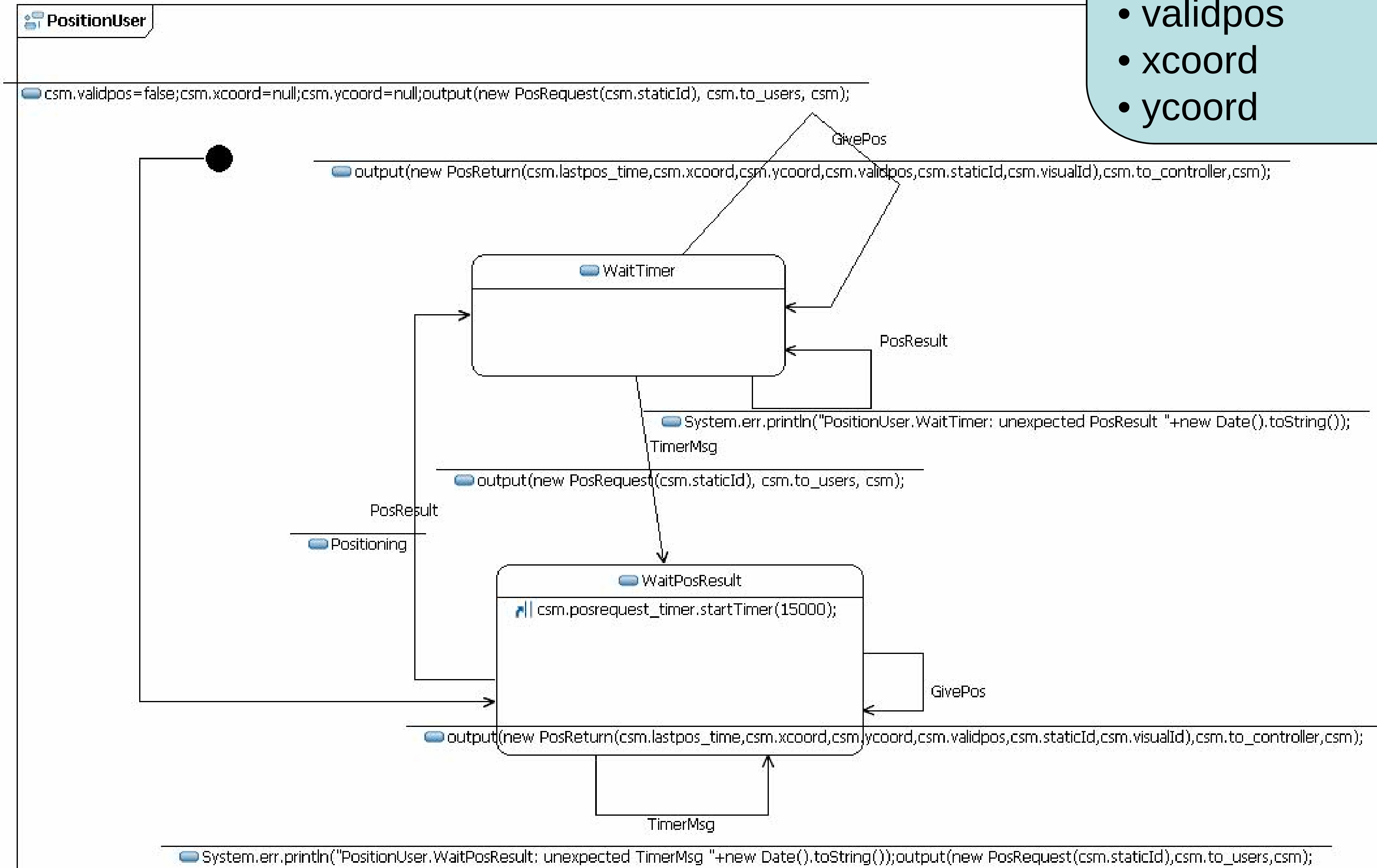
S-rules for the use of data

- Proper use of data
  - non-decisive data
    - when the process graph structure is not dependent on the data values
  - context knowledge
    - to keep information about the situation and structure of the environment
  - loop control data
    - to control loops that are not terminated by specific signals
- shared data
  - Introduce special processes to encapsulate shared data.
  - Encapsulate data needing independent access in separate processes.



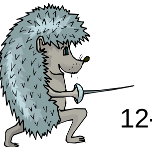
Does this apply to PositionUser?

- *staticId*
- *visualId*
- lastpos_time
- TimerMsg posreq
- validpos
- xcoord
- ycoord

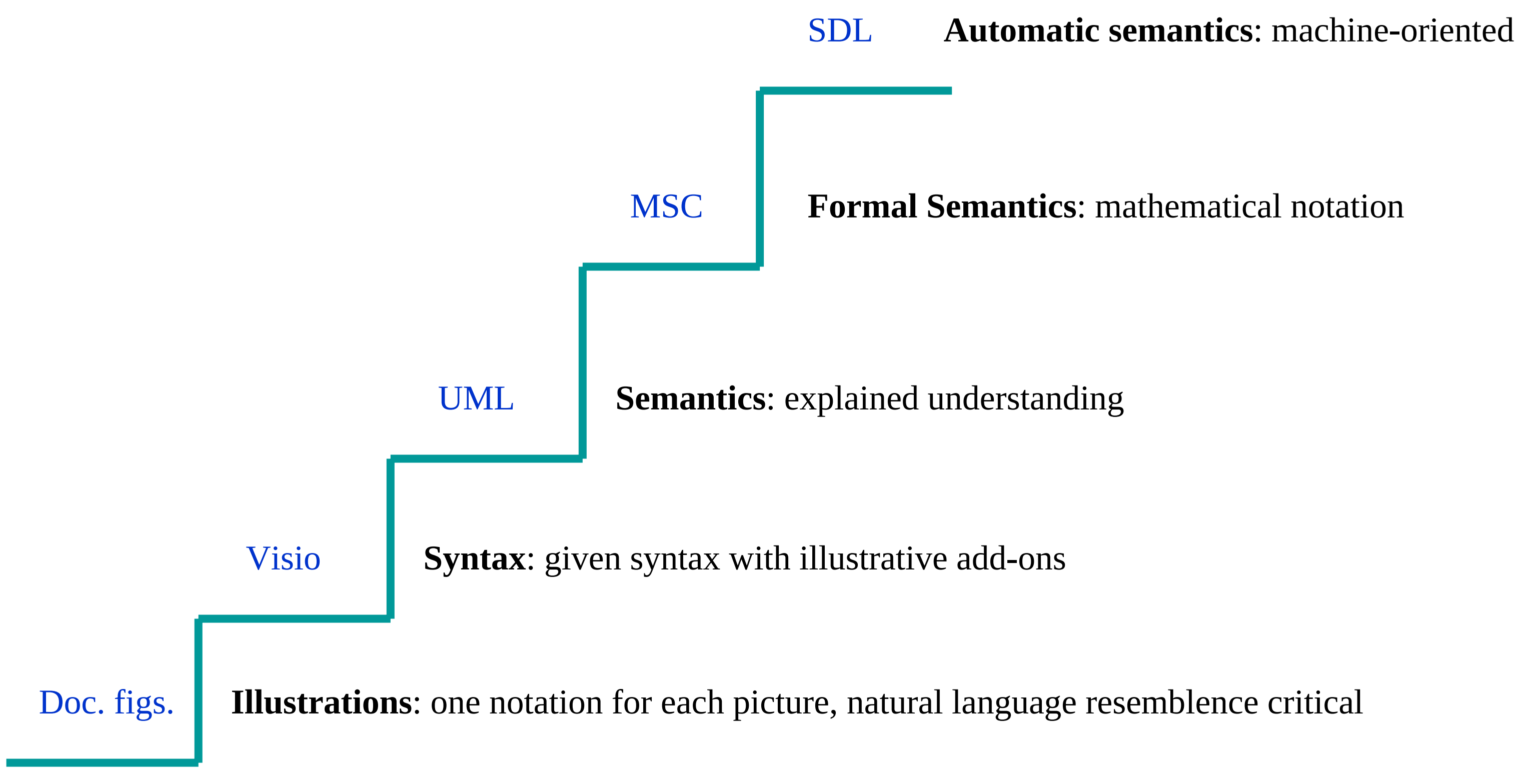


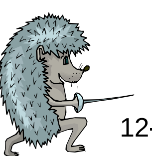
Dialectic System Development

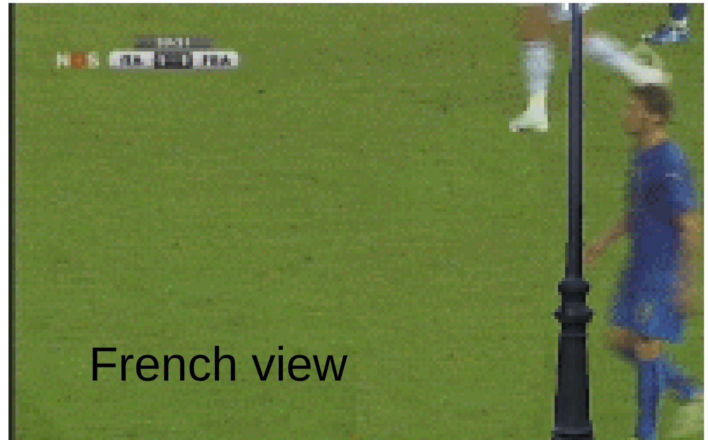
how to take advantage of conflicts



The language maturity staircase

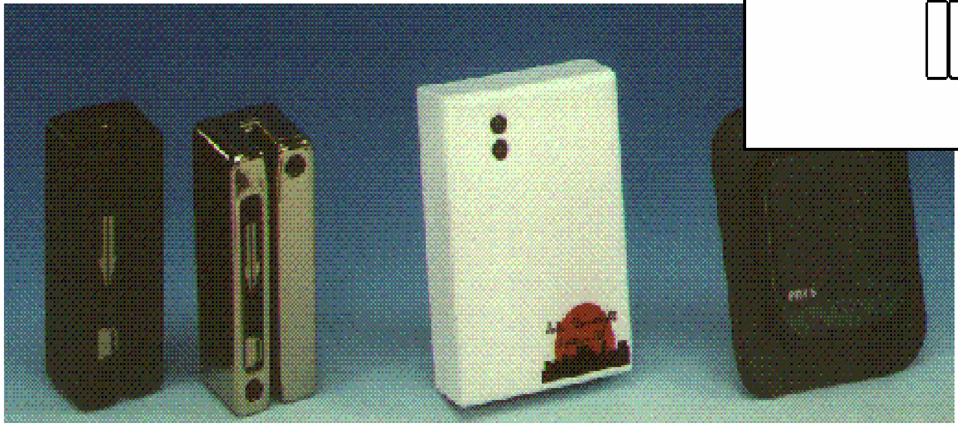


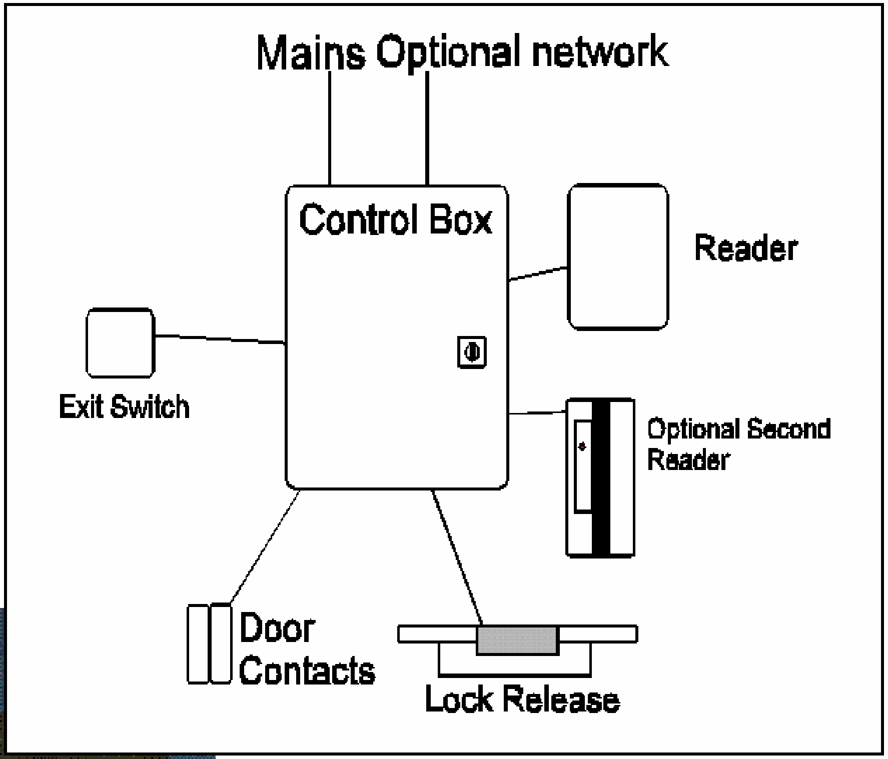




http://www.theregister.co.uk/2006/07/13/zidane_headbutt_outrage/

Access Control System



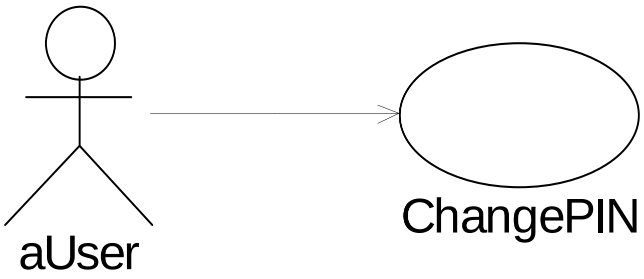


Domain Statement

- Area of concern
  - Access control has to do with controlling the access of users to access zones. Only a user with known identity and correct access right shall be allowed to enter into an access zone. Other users shall be denied access.
- Stakeholders
  - Users of the system, those responsible for the security of the access zones.
- Services
  - The user will enter an access zone through an access point.
  - A supervisor will have the ability to insert new users in the system.
  - Users shall be able to change their secret code.
    - The authentication of a user shall be established by some means for secret personal identification (code). The authorisation is based upon the user identity and access rights associated with the user.

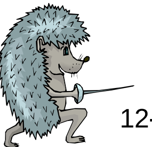
Service: Change PIN

- Informal specification:
  - ”Users shall be able to change their secret code”

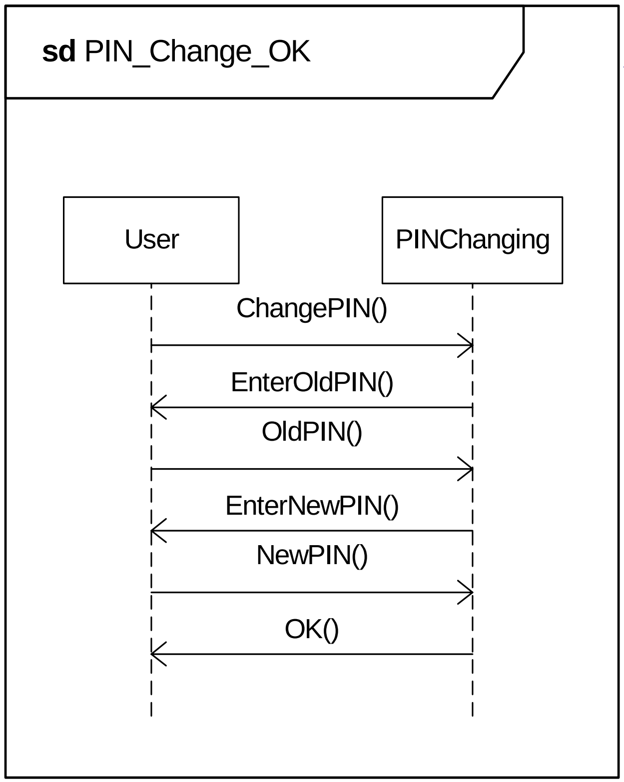


Make More Precise

- formalize
  - move the description to a more formal language
- refine
  - narrow
    - add more properties to make it less ambiguous
  - supplement
    - add new aspects, consider supplementary scenarios



Improve Precision: Service and Role orientation



formalizing

Consistent?

narrowing

service PIN Change

- Users shall be able to change their personal identification
- The User shall be able to choose his new PIN
- The Card shall be validated by the old PIN before a new PIN can be given. The new PIN shall subsequently also be validated.

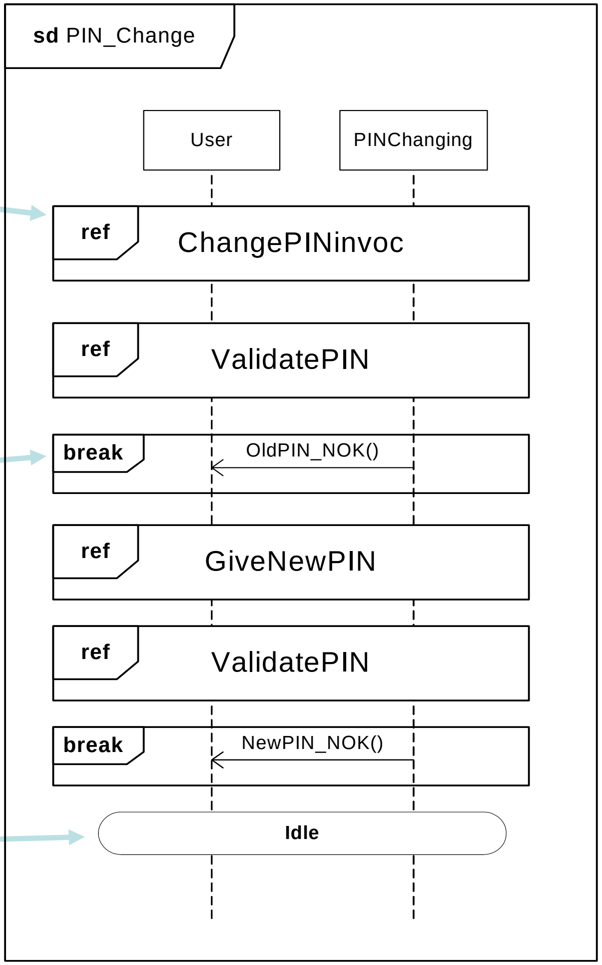
supplementing

Supplementing

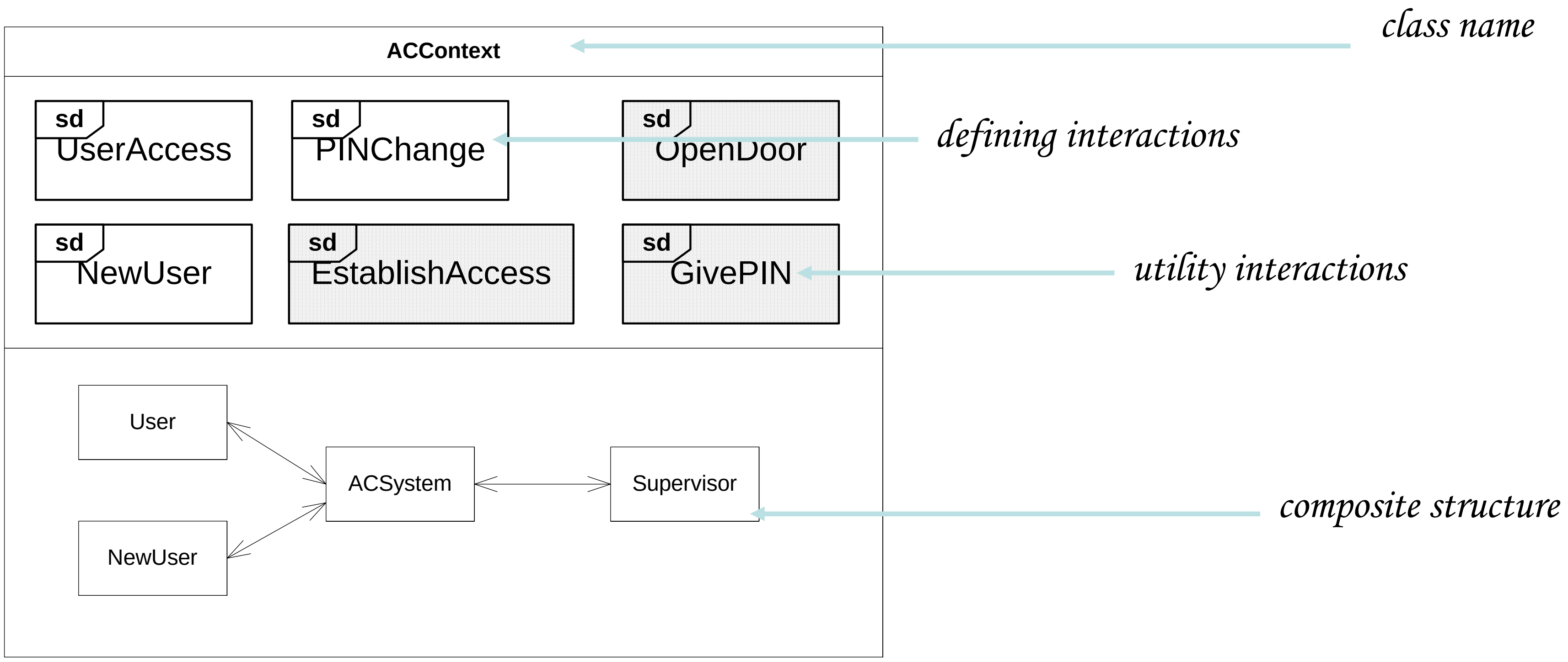
Interaction occurrence (use)

break expression

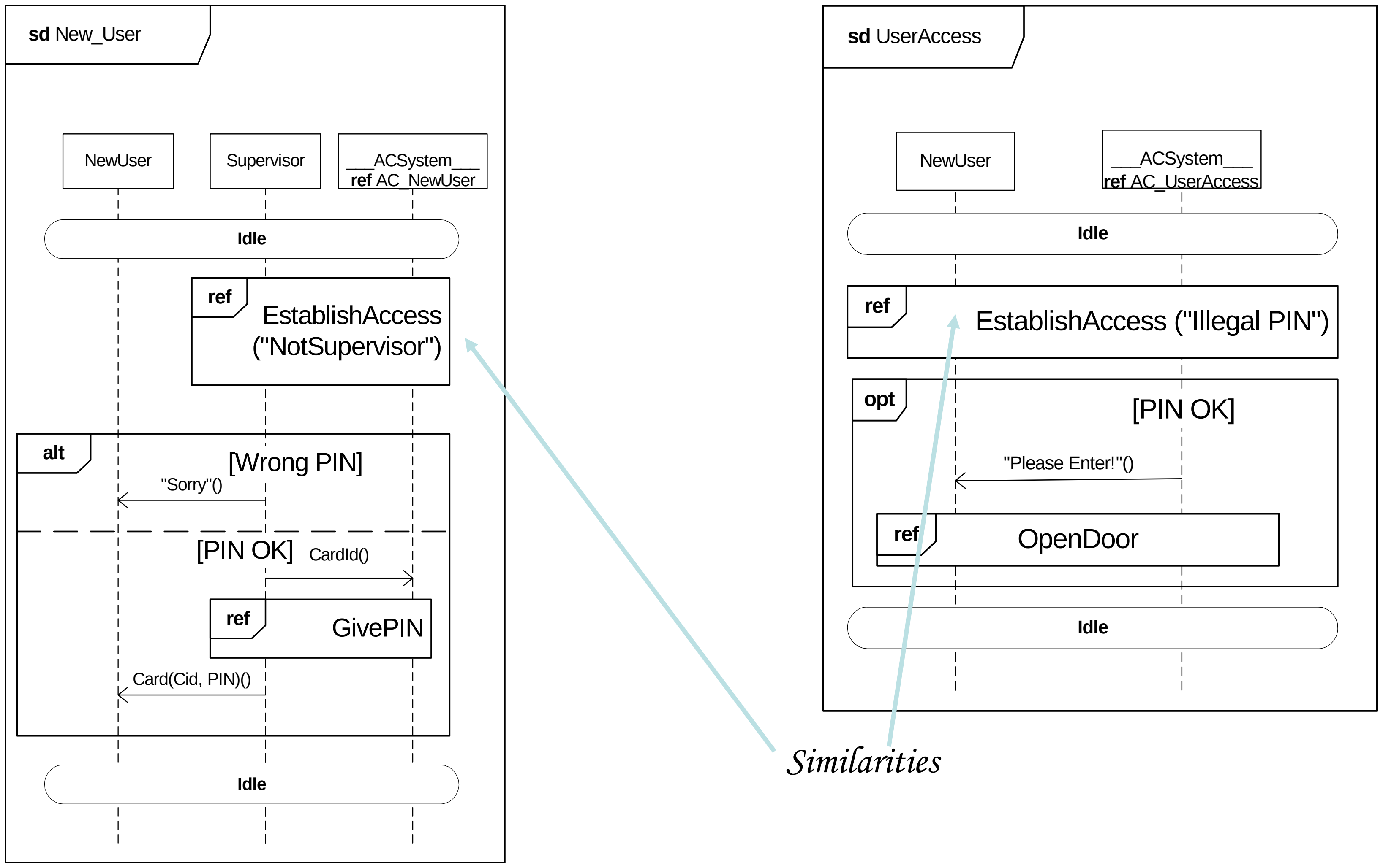
continuation



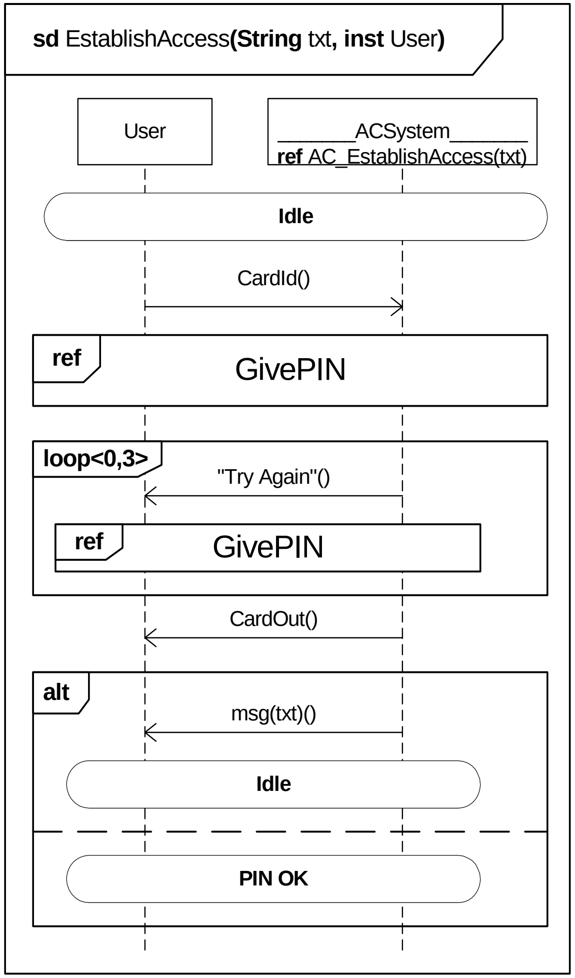
The Access Control Context as UML Class



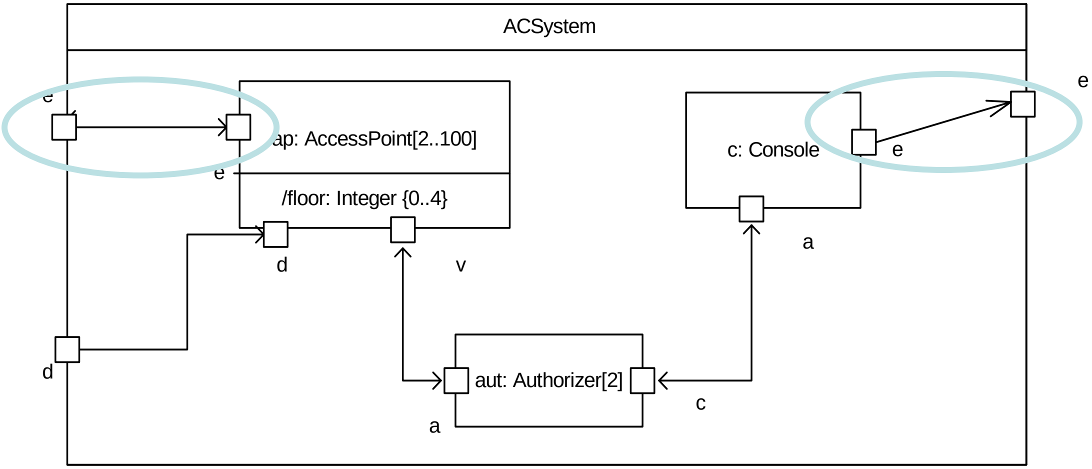
System services



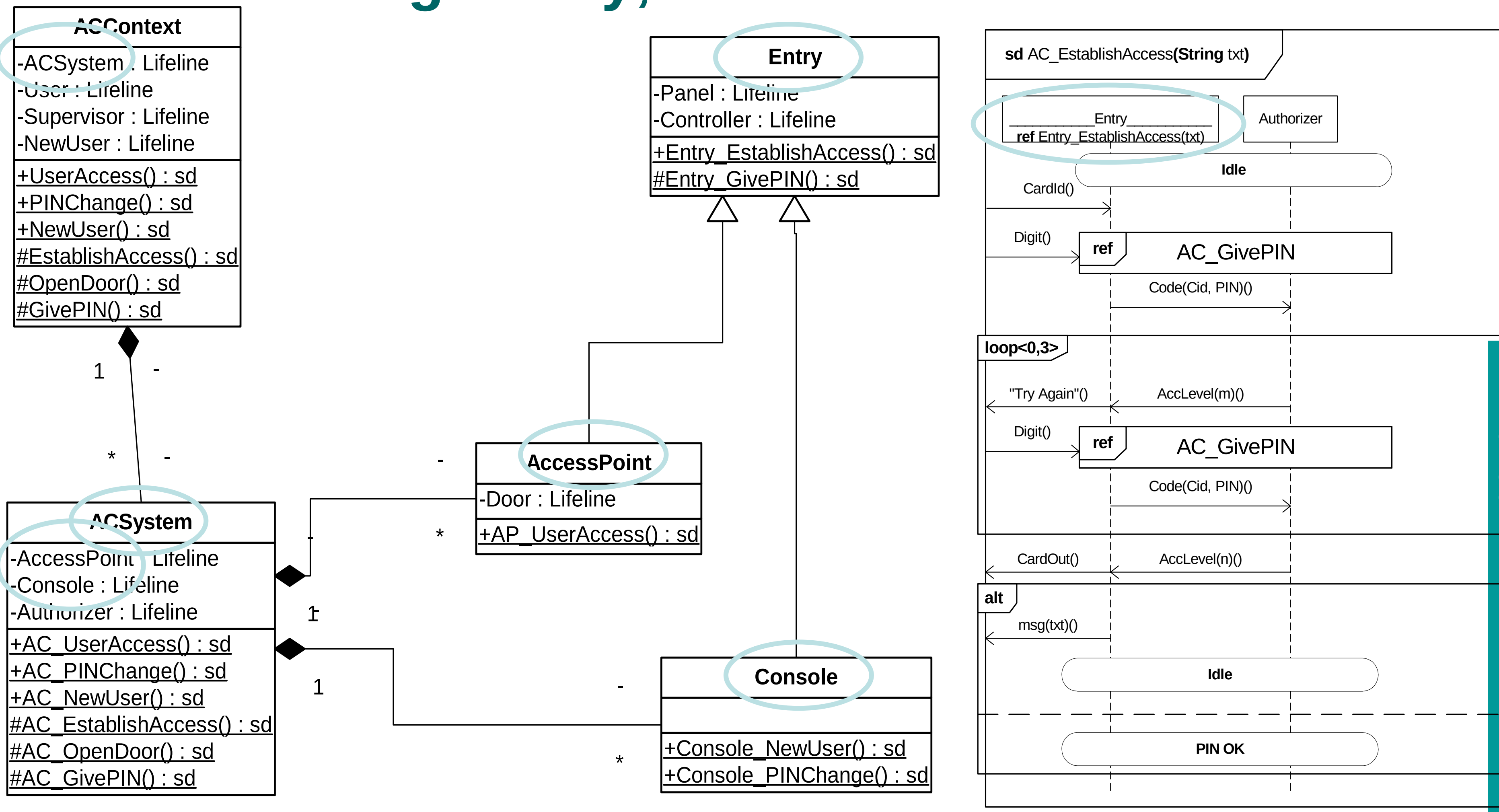
Need for generalization: Entry



- On what connectors is EstablishAccess applied?
  - between the AccessPoint and a normal User
  - between the Console and the Supervisor user

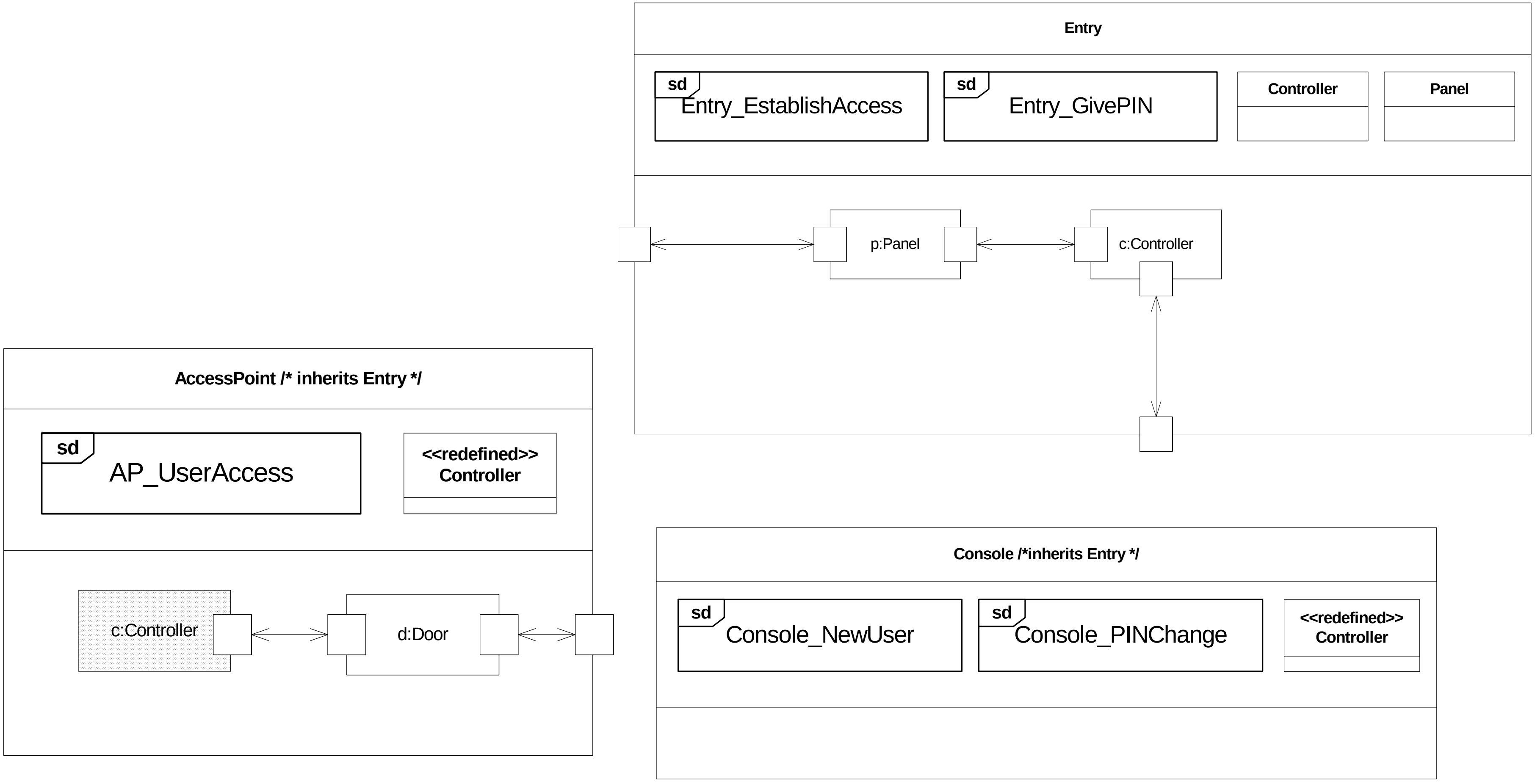


Harmonizing: Entry, AccessPoint and Console

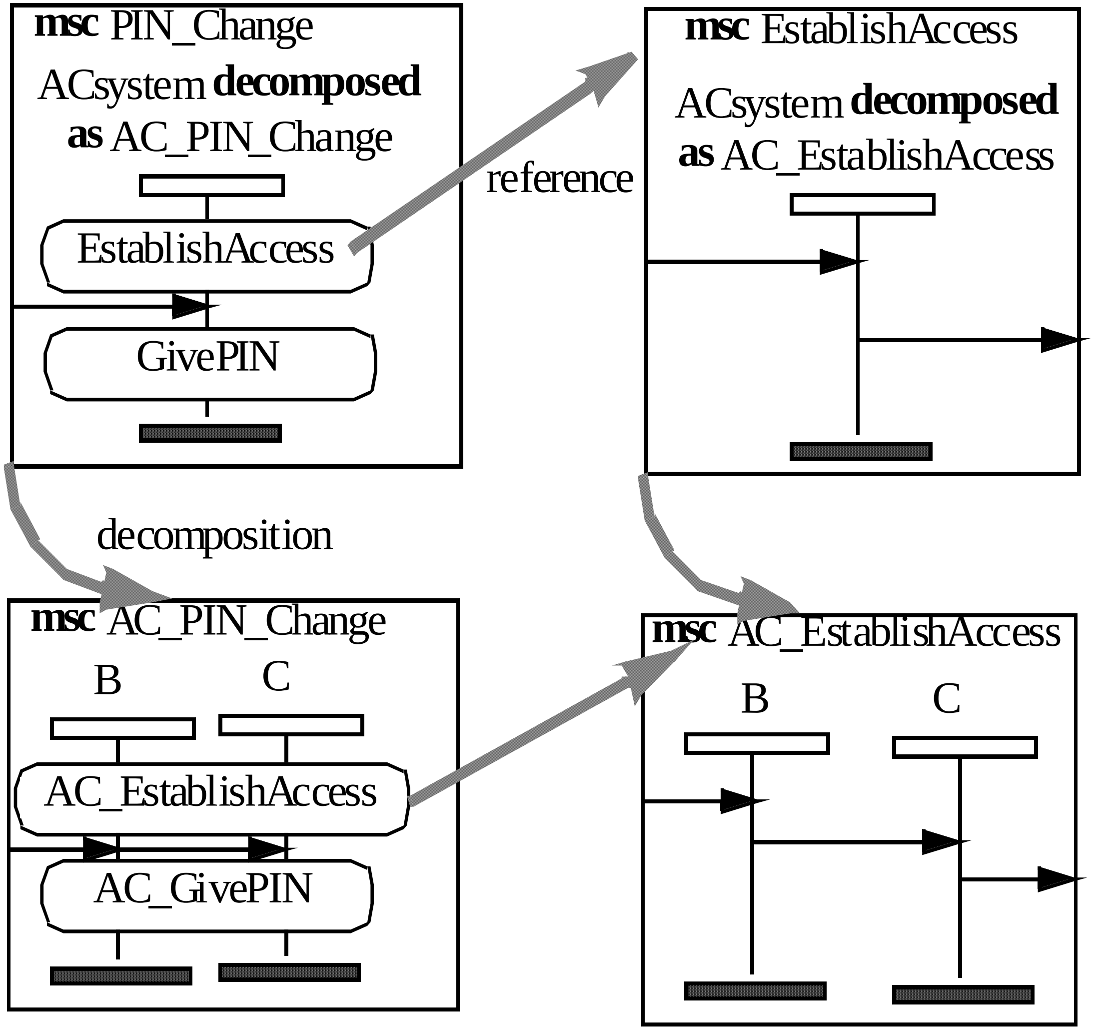


INF 5150

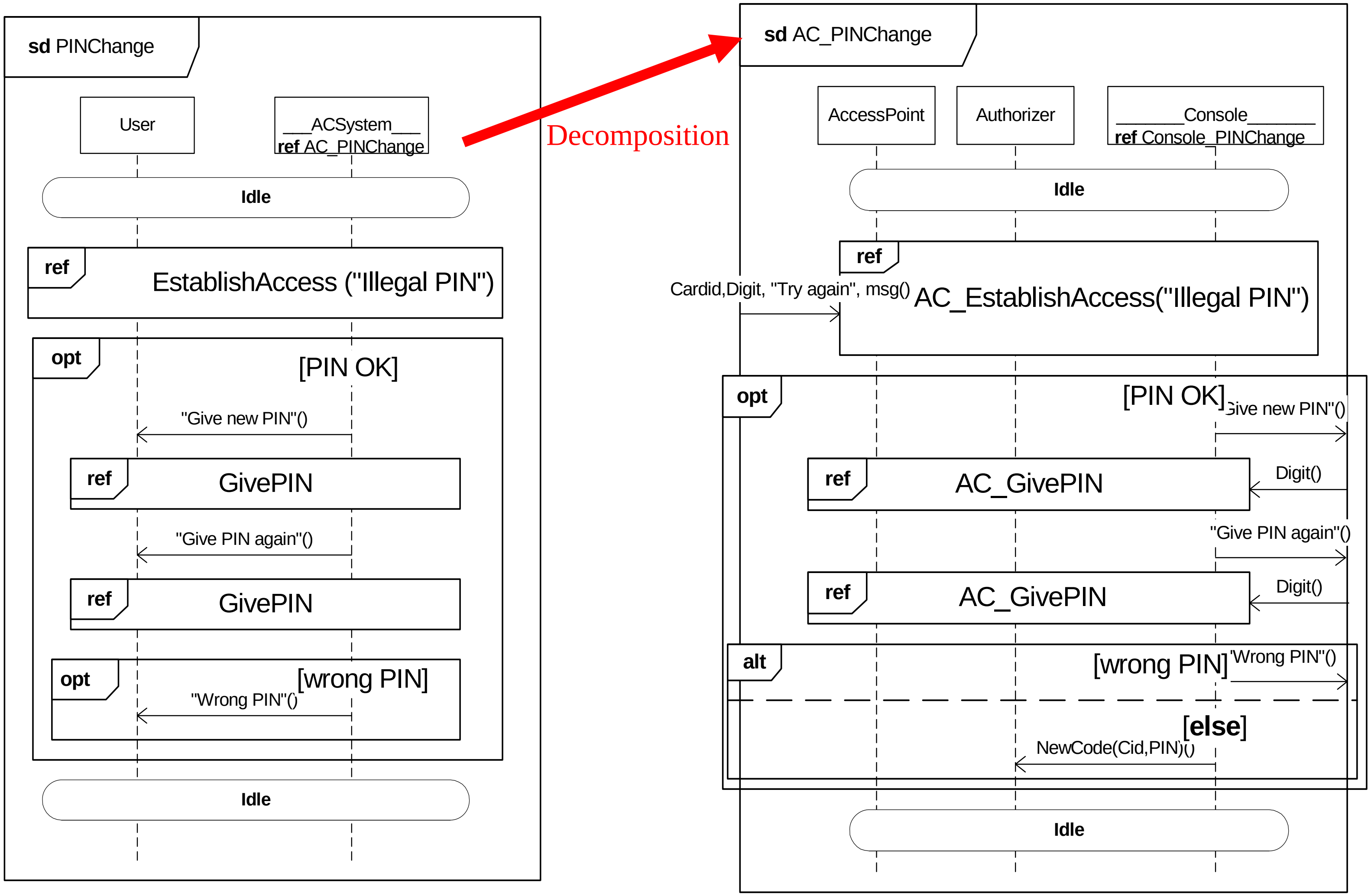
The Entry class hierarchy



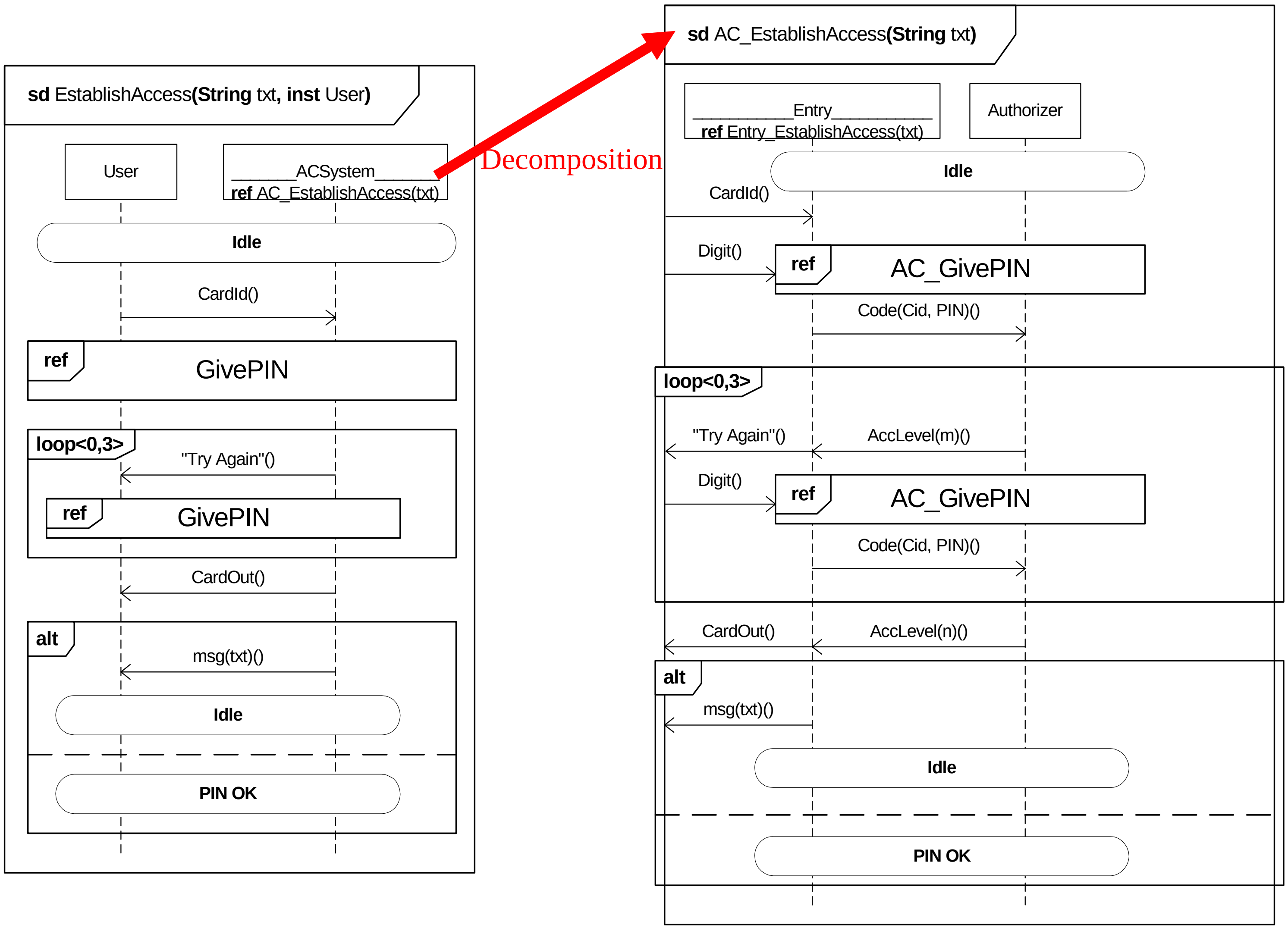
Detailing through commutative decomposition



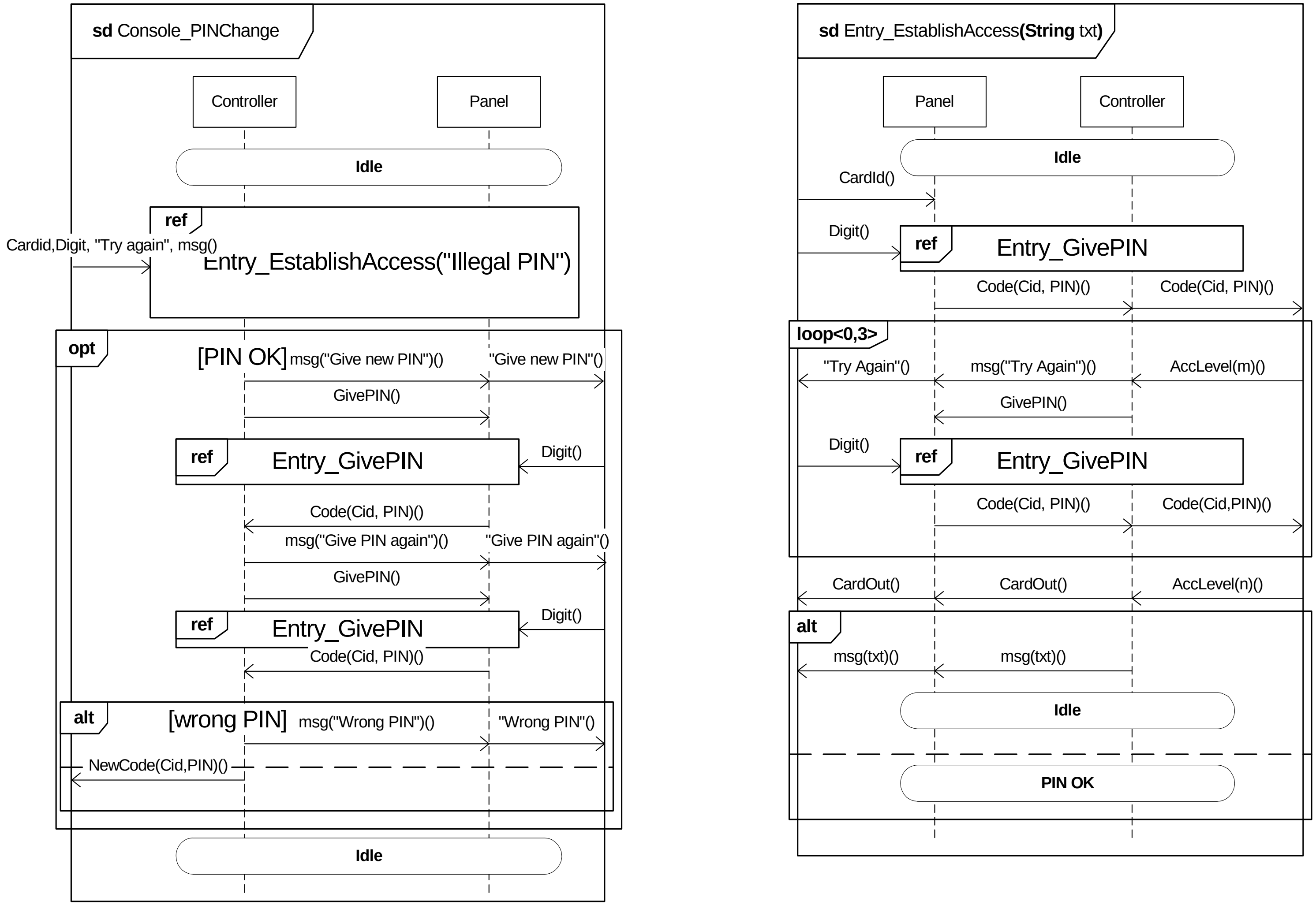
Change PIN


sd PINChange
User
___ACSystem___
ref AC_PINChange
Idle
ref
EstablishAccess ("Illegal PIN")
opt
[PIN OK]
"Give new PIN"()
ref
GivePIN
"Give PIN again"()
ref
GivePIN
opt
[wrong PIN]
"Wrong PIN"()
Idle
Decomposition
sd AC_PINChange
AccessPoint
Authorizer
_______Console_______
ref Console_PINChange
Idle
ref
Cardid,Digit, "Try again", msg()
AC_EstablishAccess("Illegal PIN")
opt
[PIN OK]
"Give new PIN"()
ref
AC_GivePIN
Digit()
"Give PIN again"()
ref
AC_GivePIN
Digit()
alt
[wrong PIN]
"Wrong PIN"()
[else]
NewCode(Cid,PIN)()
Idle

Commutative Decomposition

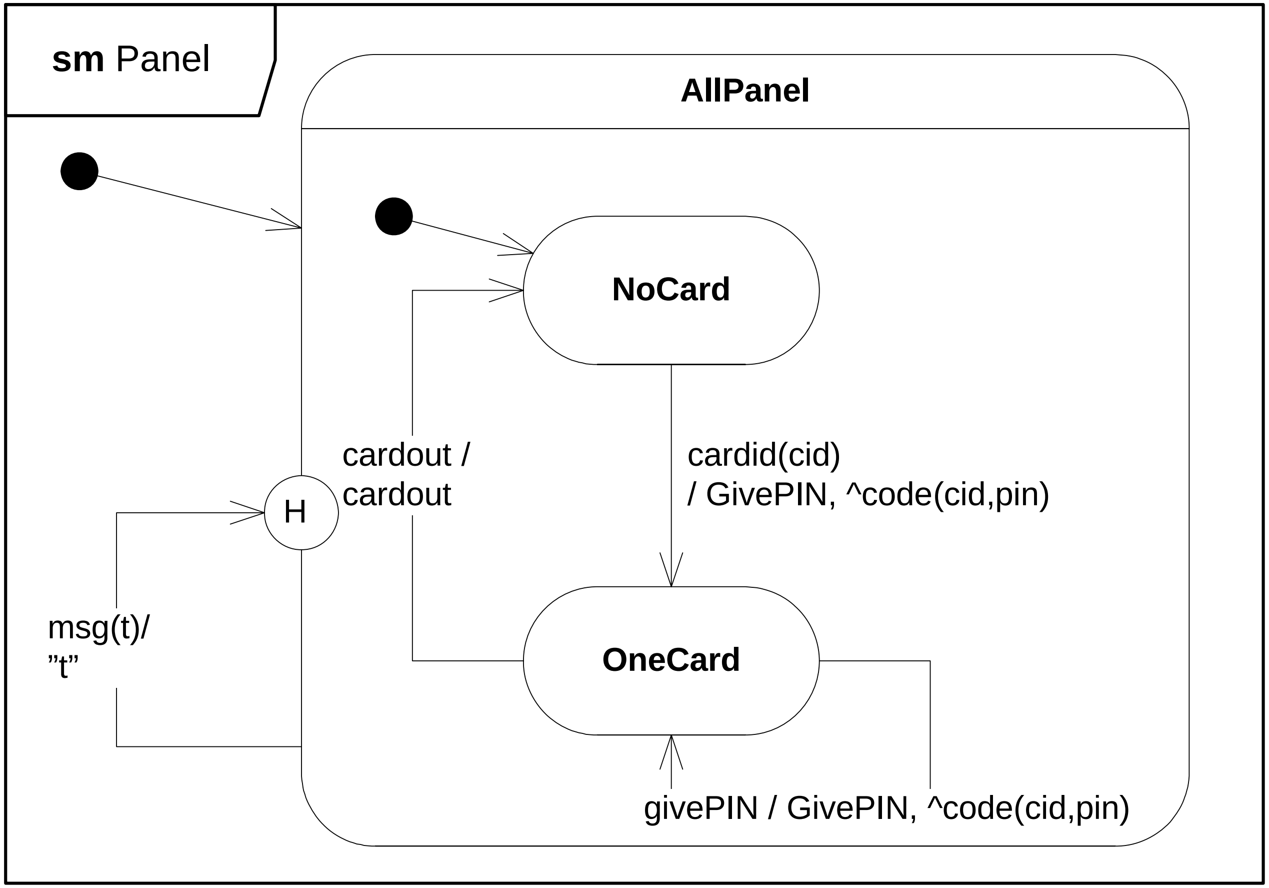


Verification 1: Model checking PIN Change in Panel

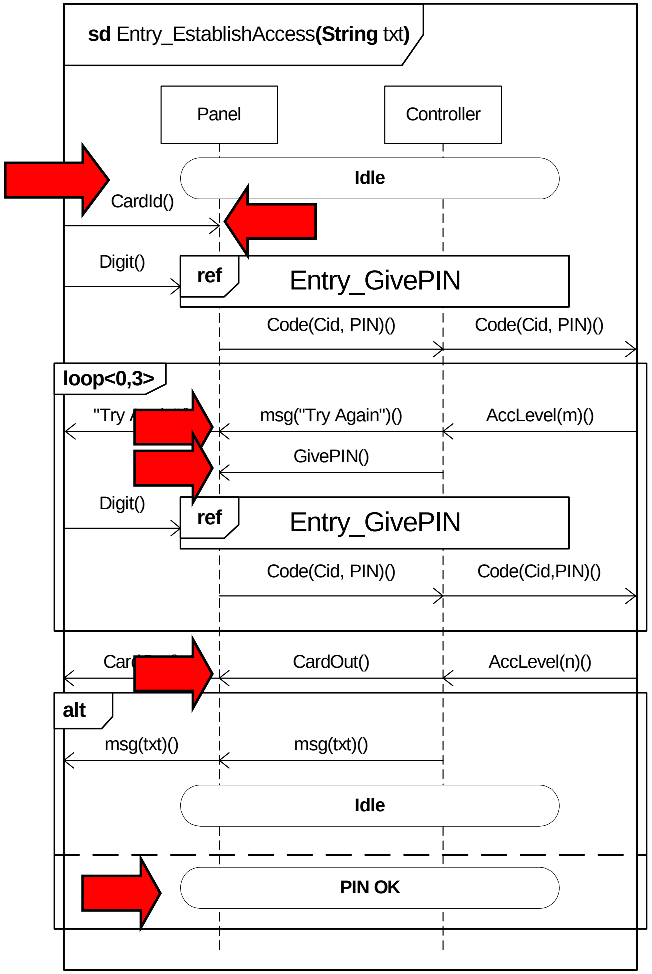


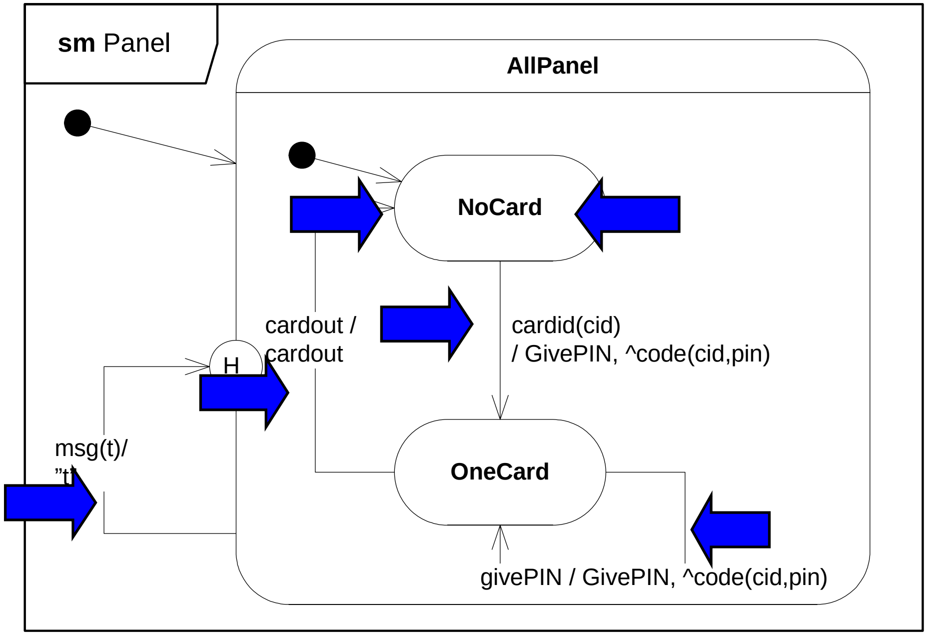
INF 5150

Panel: UML State Machine, GivePIN as a method

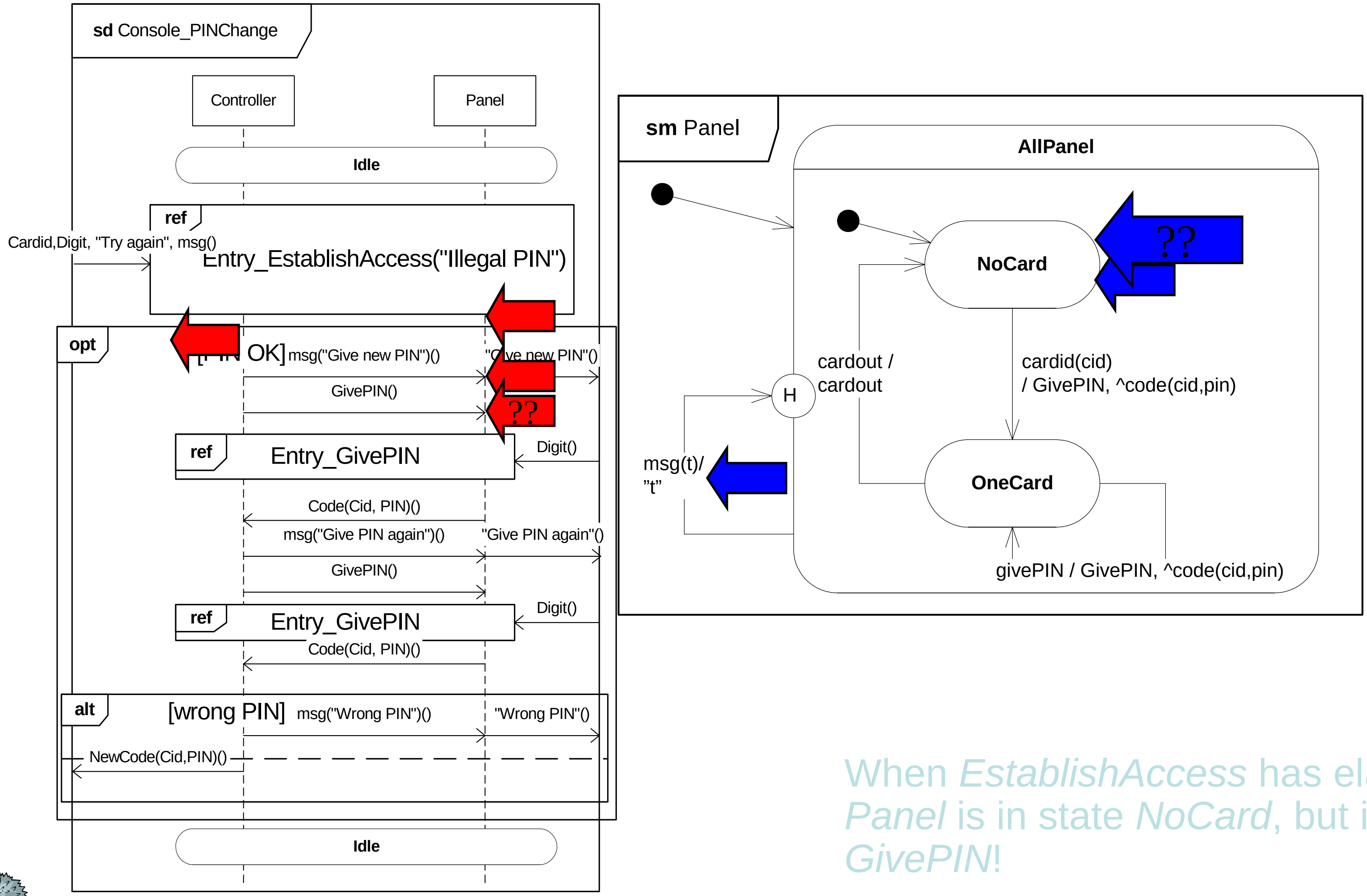


Model checking continued....





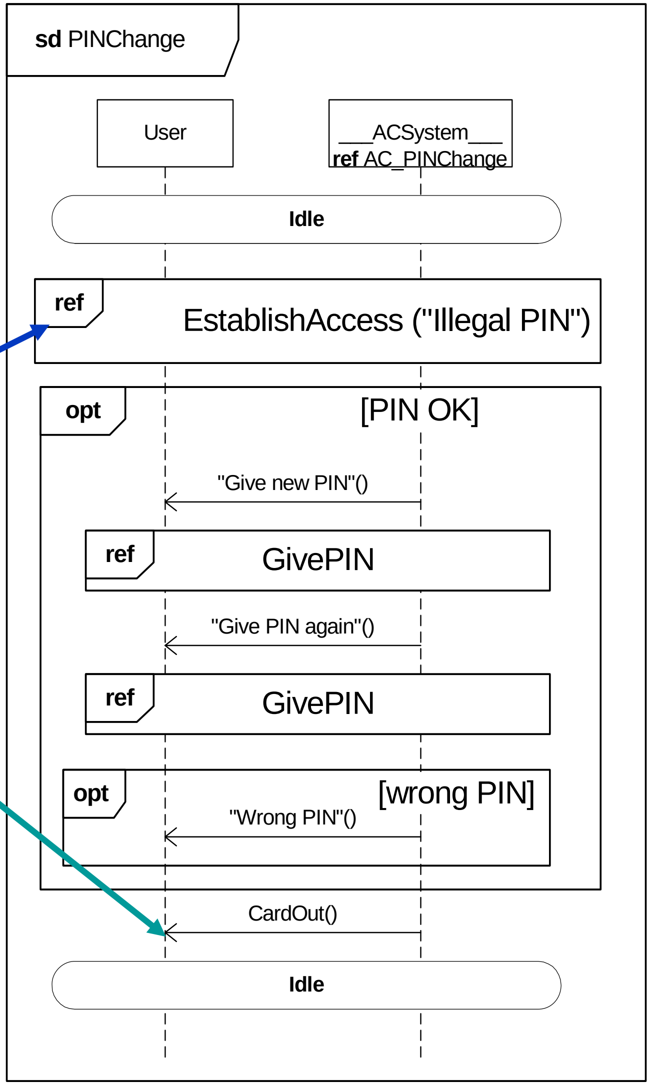
Model checking continued.... ....



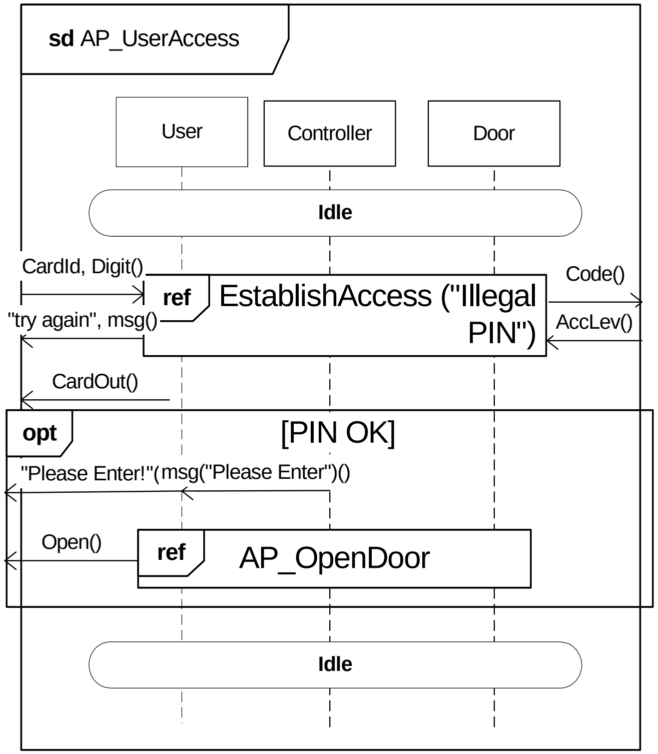
When *EstablishAccess* has elapsed, *Panel* is in state *NoCard*, but it receives *GivePIN*!

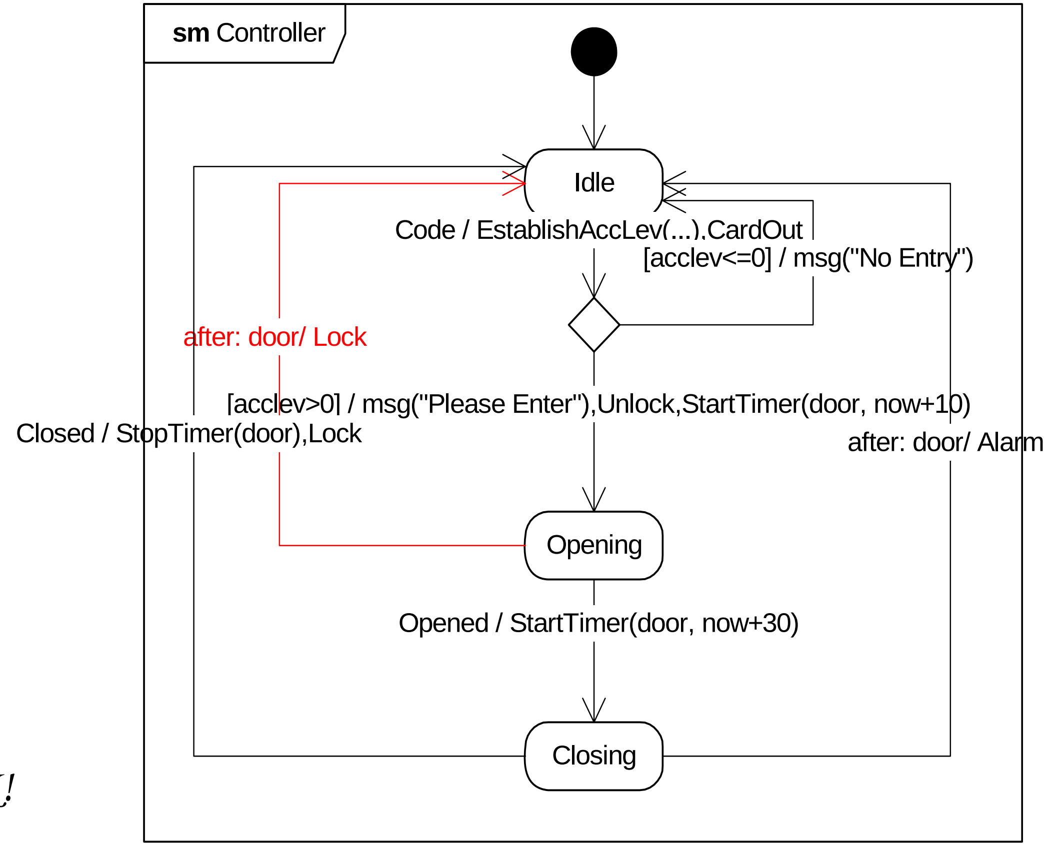
Harmonizing

We decide to move CardOut from EstablishAccess to the end of PIN_Change



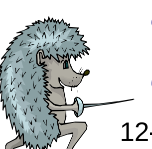
Verification 2: AccessPoint’s Controller





sd: User Access vs sm: Controller = OK!

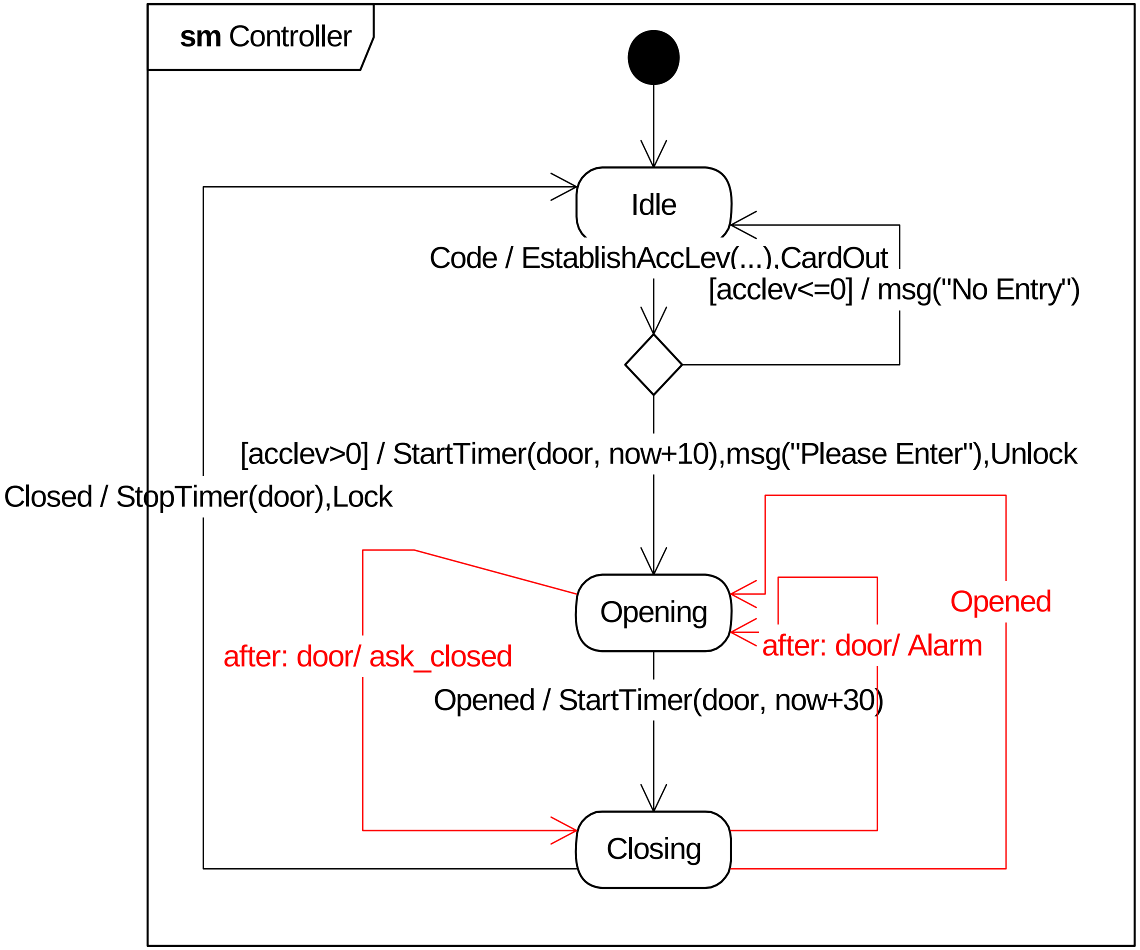
Are we then certain that AccessPoint’s Controller is perfect?



The User opens the door exactly when the timer expires. door+opened in input port

Verification 3: Detecting default transitions

- *Sequence Diagrams are not suited to uncover all possible variants of interaction*
- *State Machines (JavaFrame or UML 2) supported by automatic techniques can find unwanted signaling combinations*
- *There are several techniques to evaluate projections of processes to uncover the complexity of the software*



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