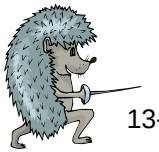
UNIVERSITY
OF OSLO

System Development Part 1

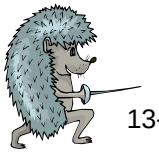
Version 051014

INF 5150



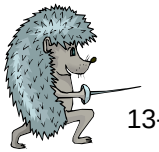
What shall we learn in school today?

- Why your java programs fail
  - The solution
    - thinking in a way corresponding to how your program will work
- 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



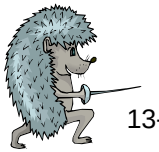
Why your normal Java program fails

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

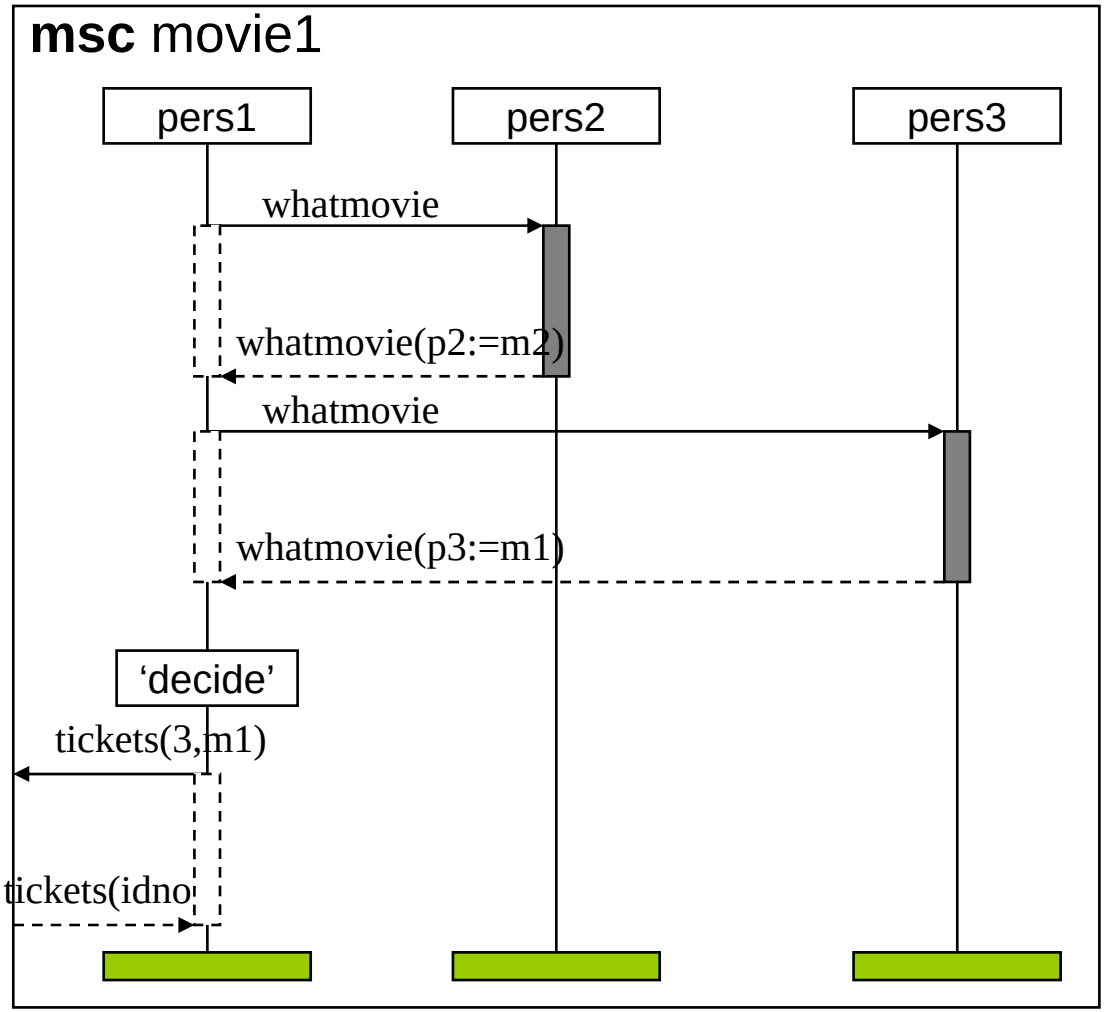


Agreeing on which movie to watch

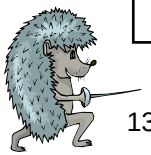
- A group of persons are going to agree on which movie to watch this evening
- There is only a small number of movies (less than the number of persons). One can assume that the decision can be based on democratic principles: the movie with most votes win.
- We will use three different ways of communication: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



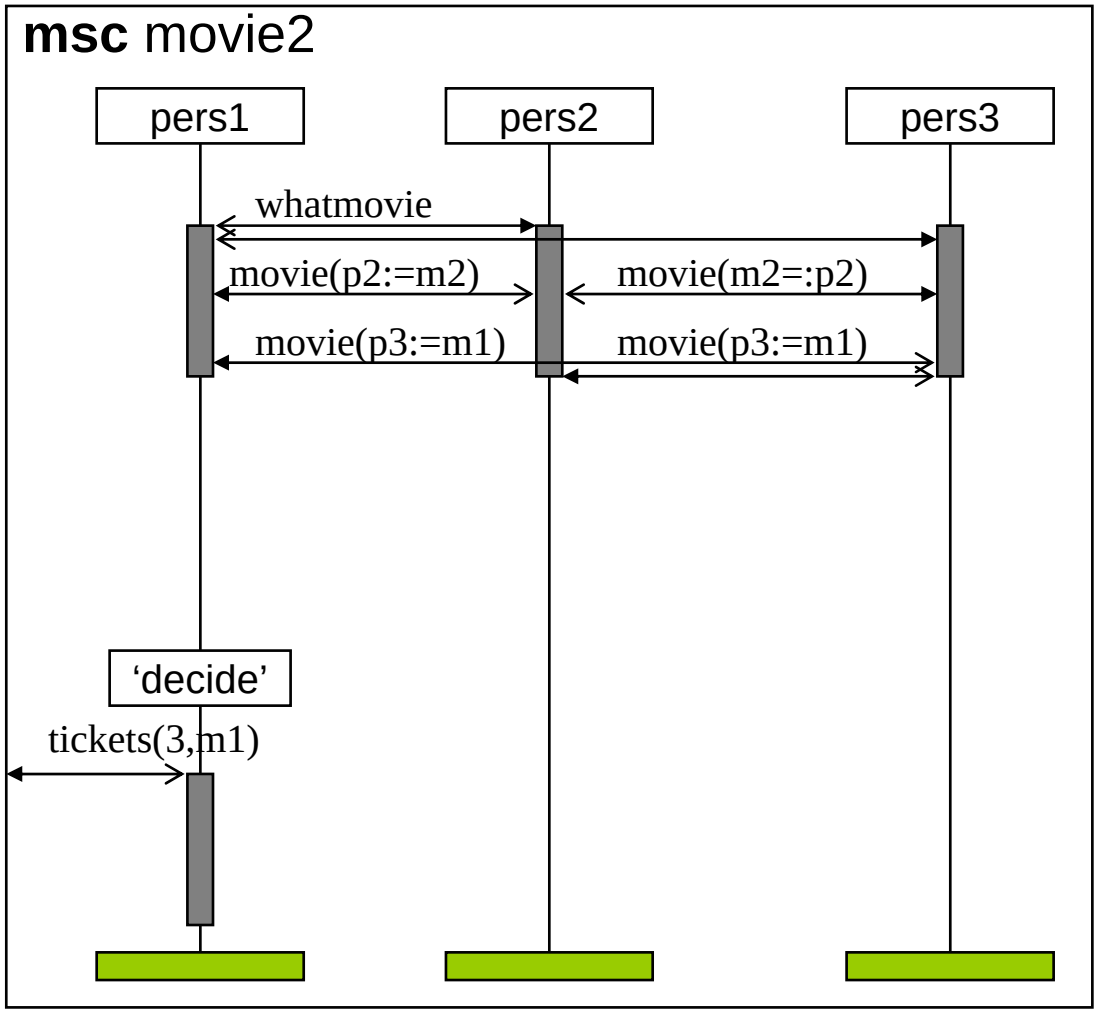
Synchronizing communication



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



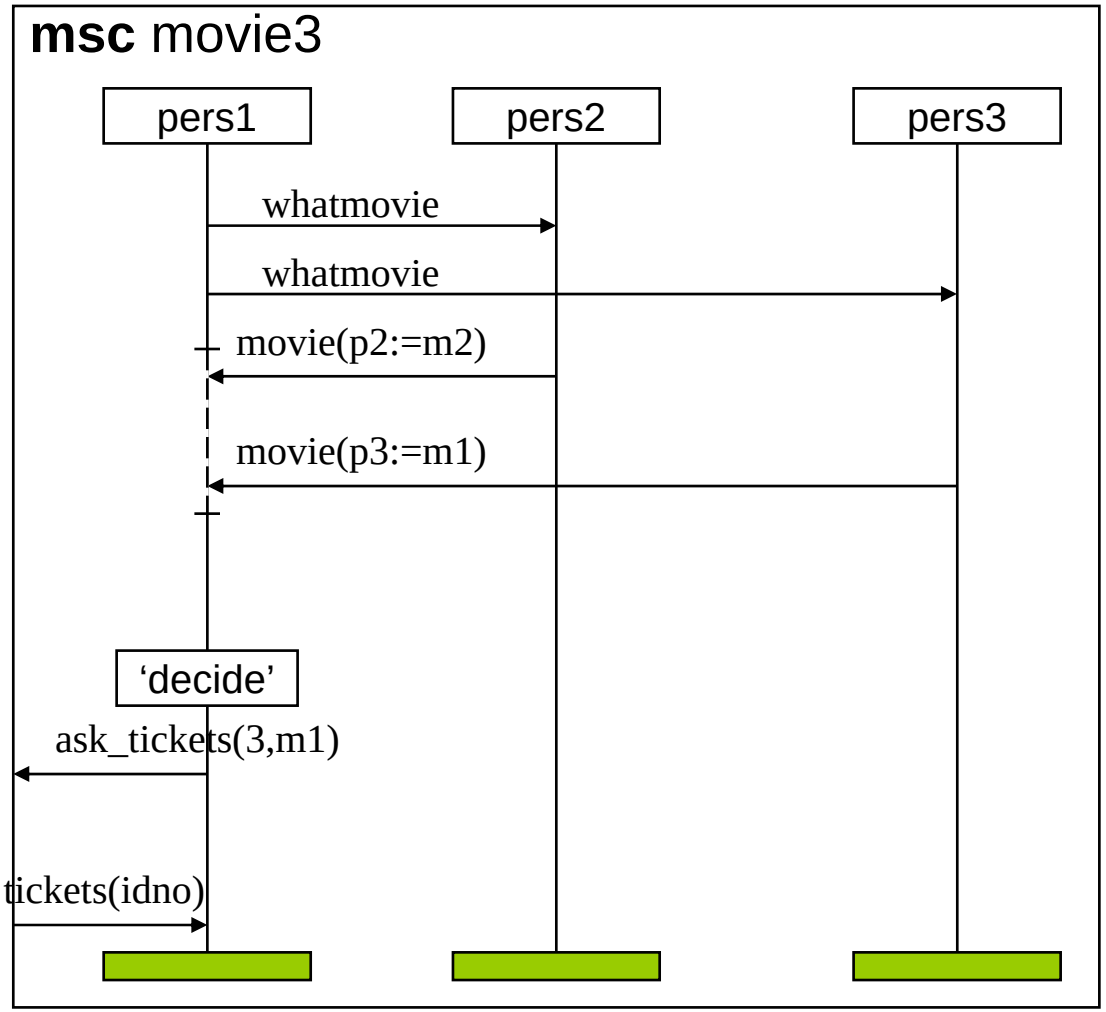
Synchronous communication



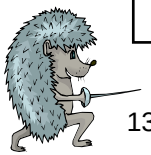
- *pers1* is the central
- *pers2* and *pers2* are co-workers
- neither of the persons can do anything while the communication lasts

- (this is informal MSC since MSC-2000 have no mechanisms for synchronous

Asynchronous communication

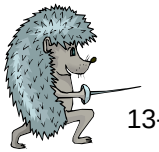


- *pers1* is the central
- *pers2* and *pers2* are co-workers
- *pers1* can do other kinds of work while *pers2* and *pers3* decide their opinions
- *pers2* and *pers3* can make up their opinion in parallel

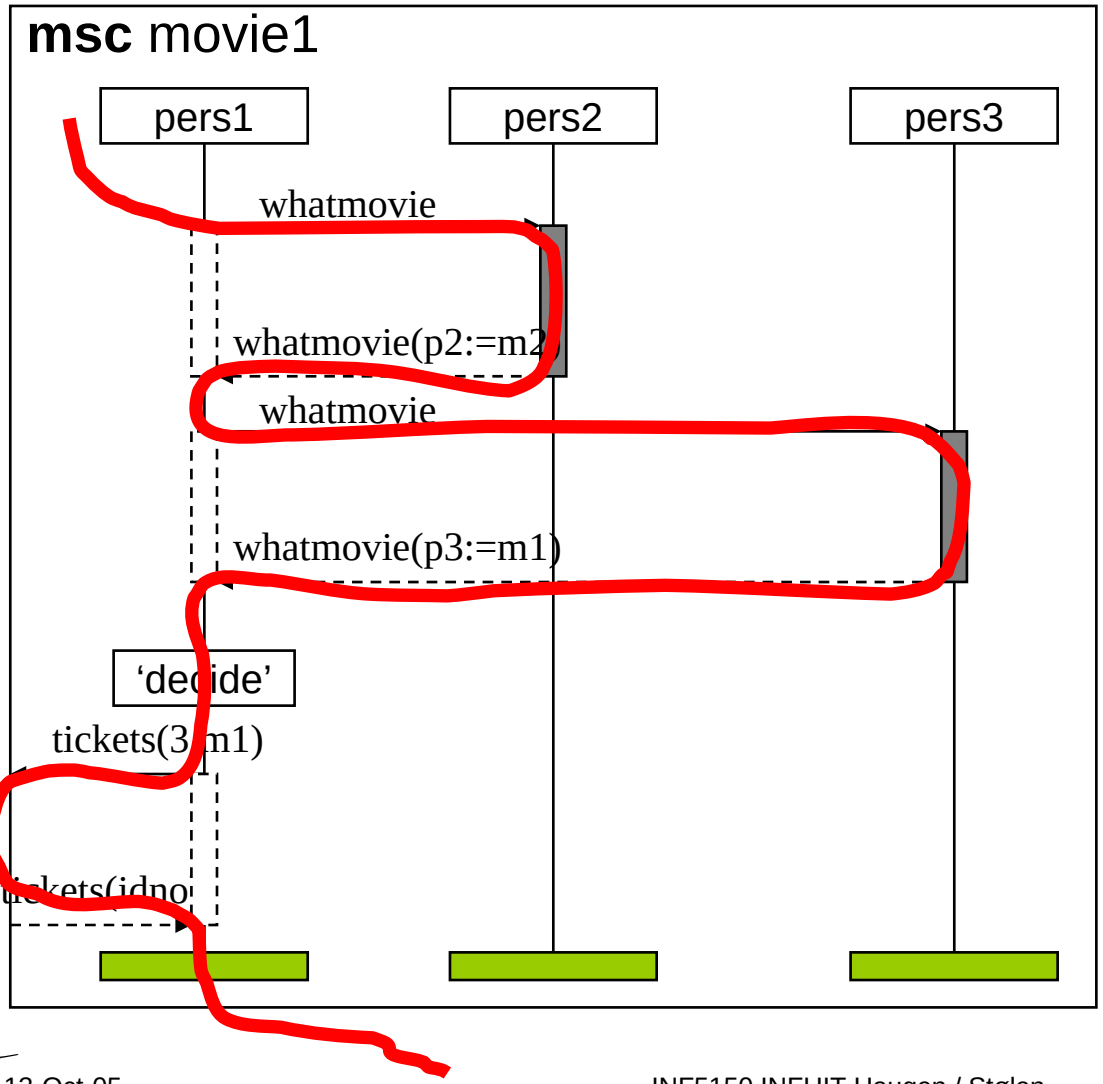


Threads

- 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

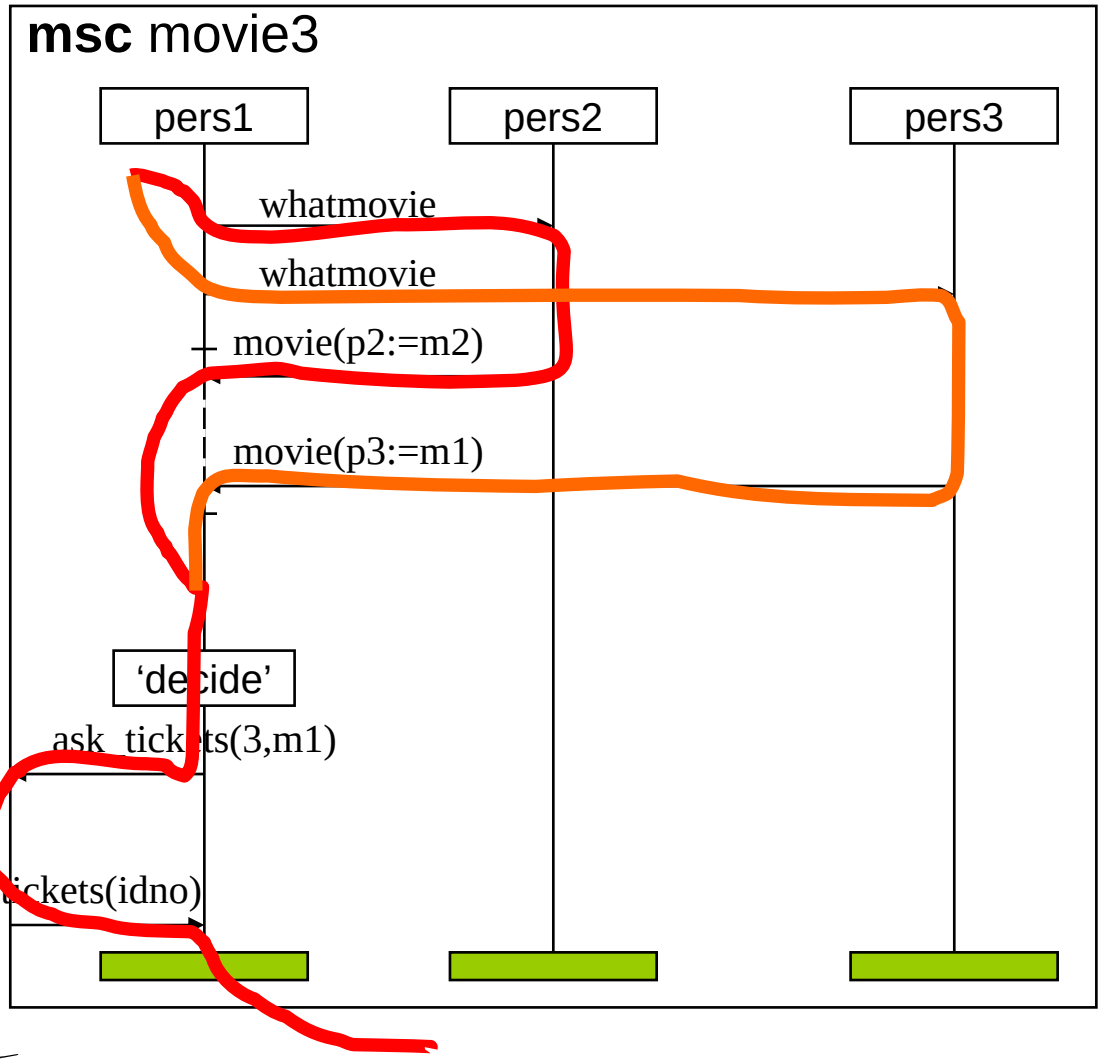


Threads 1



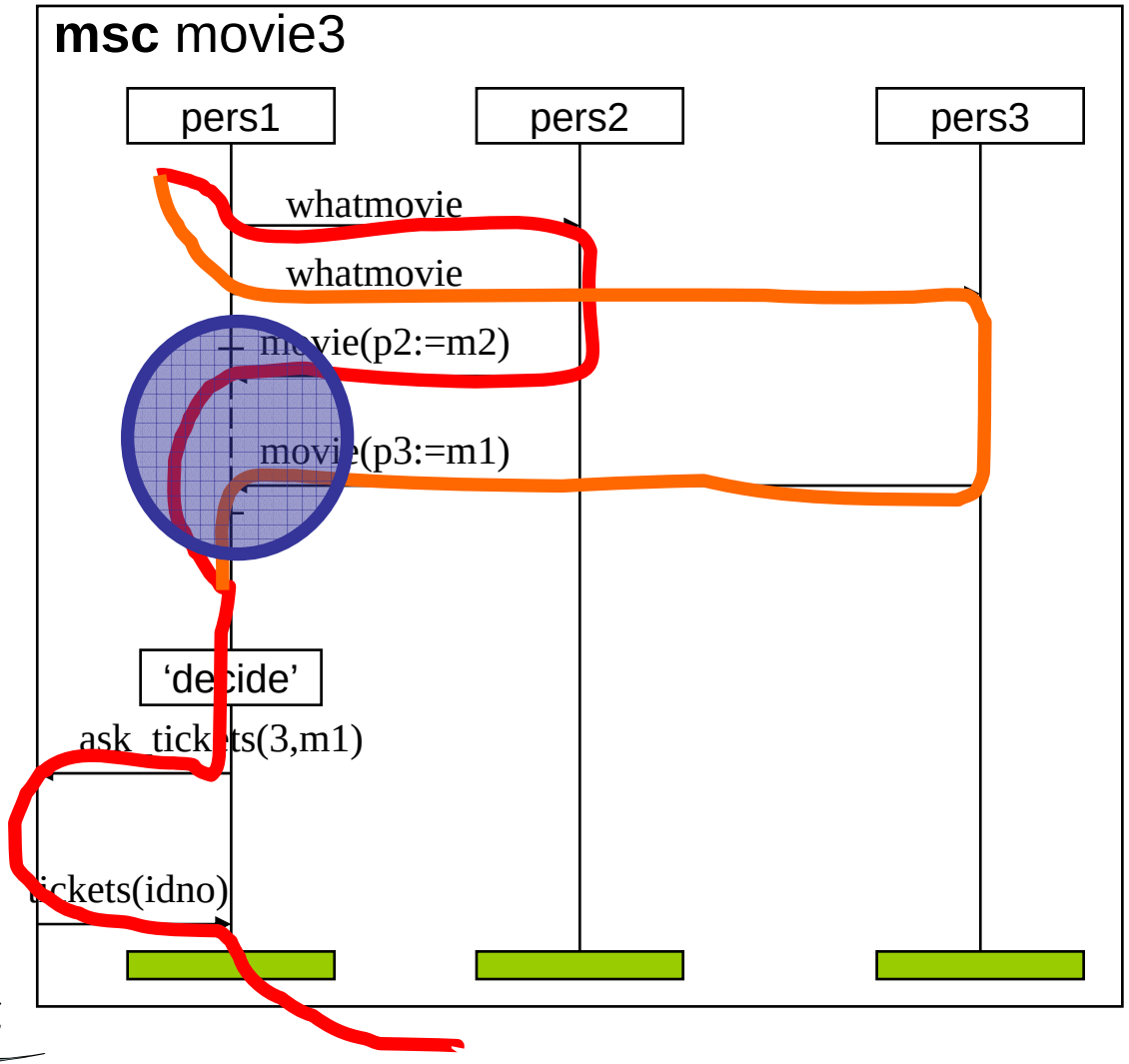
- one thread
- in fact the whole system is sequential!
- anybody can program this in Java

Threads 2



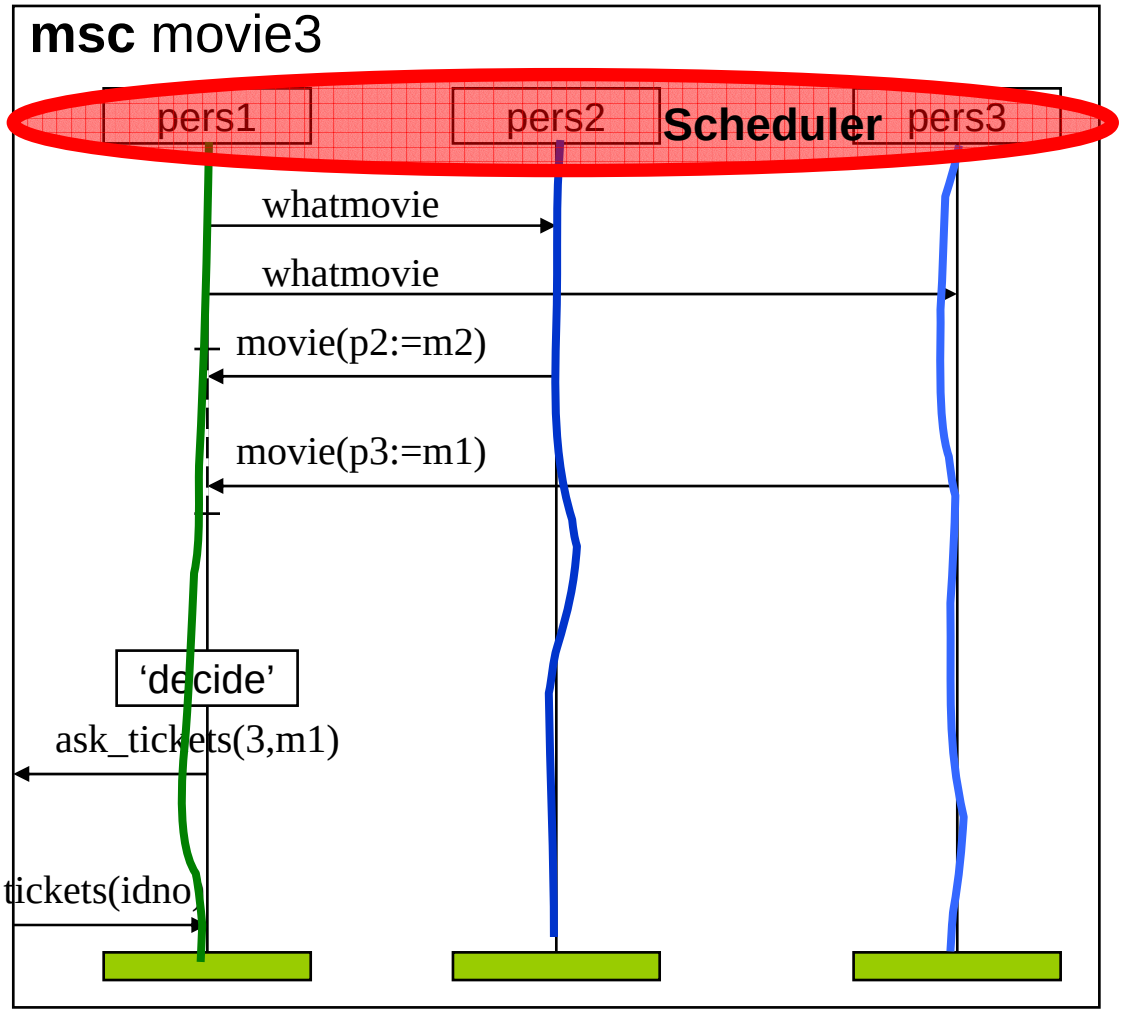
- there are two independent threads of control
- in fact there could be even more since *pers2* and *pers3* could have had other business to attend to!
- as it is, it is a fairly simple "fork" / "join" and quite simple to program
- such a local fork and join is still almost sequential

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?

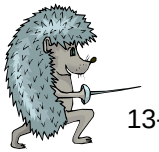
Threads 3 (JavaFrame / UML / SDL)



- *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

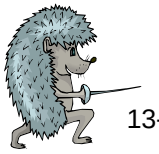
Object Orientation

- 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



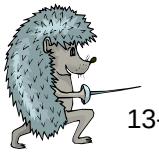
Why we make errors with Threads in Java

- 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



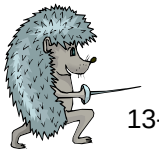
Why use several Threads in Java?

- 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)



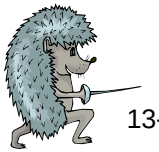
Why UML 2 / JavaFrame is different

- 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

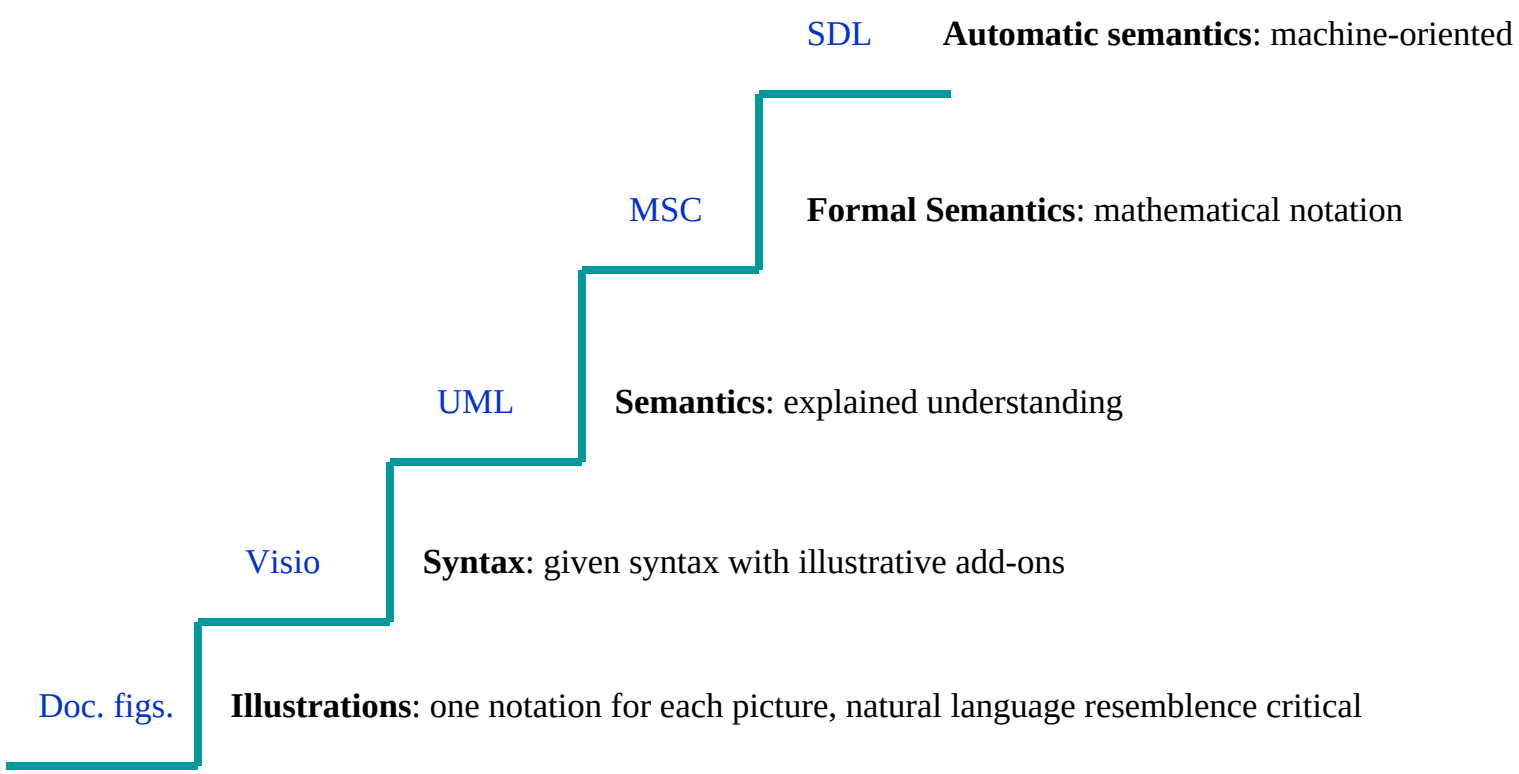


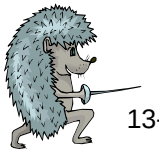
Dialectic System Development

how to take advantage of conflicts

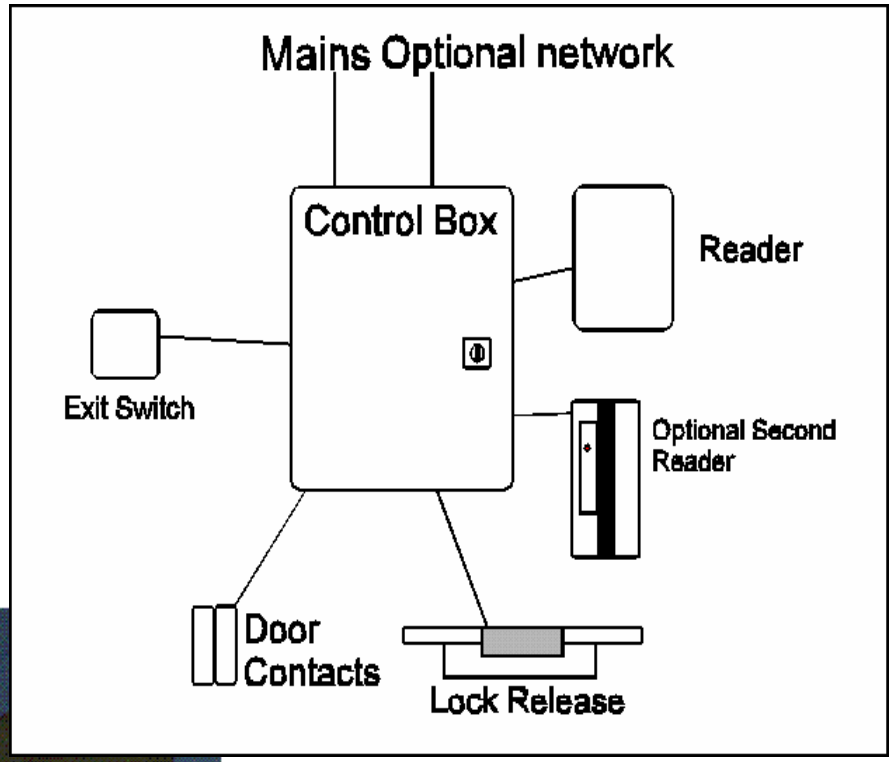


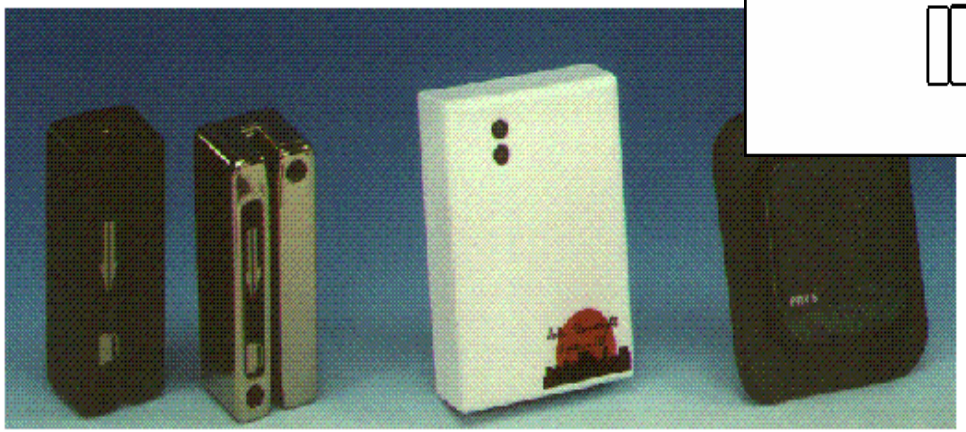
The language maturity staircase

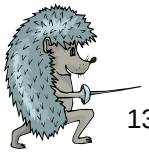




Access Control System

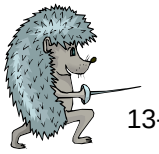






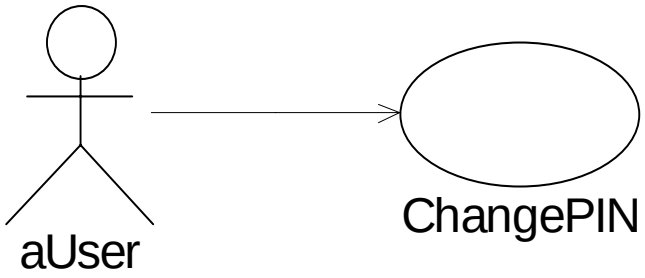
Domain Statement

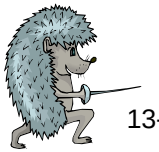
- Area of concern
  - Access control has to do with controlling the access of users to access zones. Only a user with known identity and correct access right shall be allowed to enter into an access zone. Other users shall be denied access.
- Stakeholders
  - Users of the system, those responsible for 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

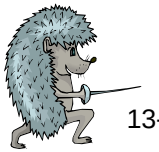
- Informal specification:
  - ”Users shall be able to change their secret code”



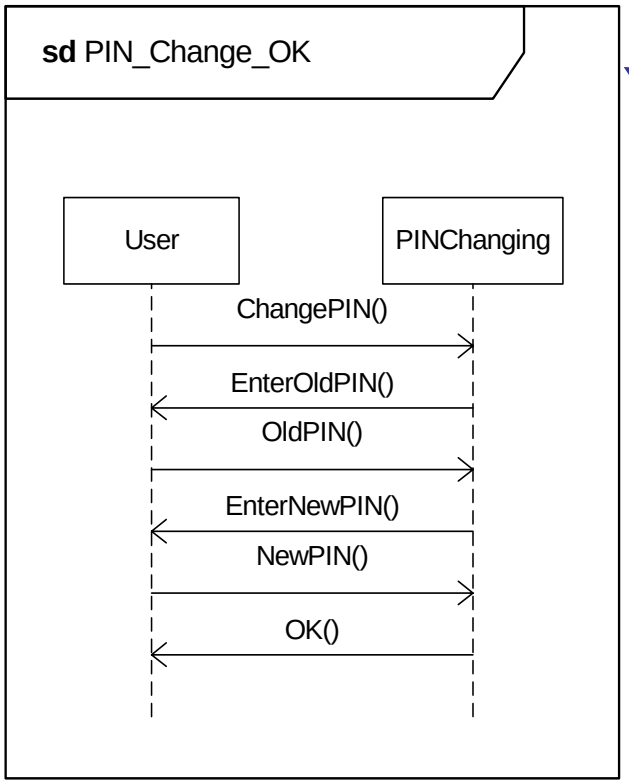


Make More Precise

- 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?

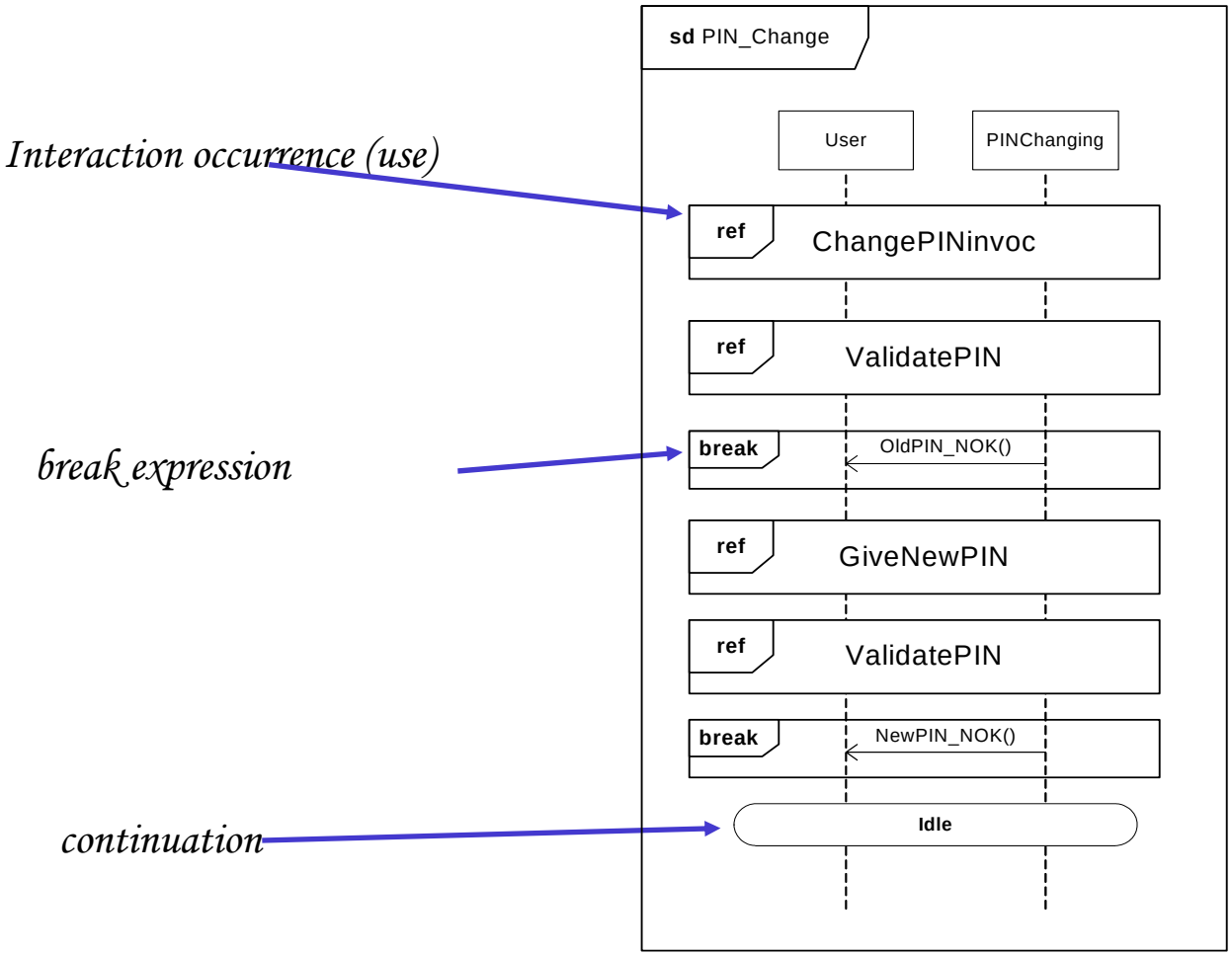
narrowing

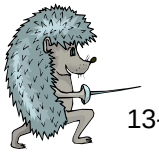
service PIN Change

- 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

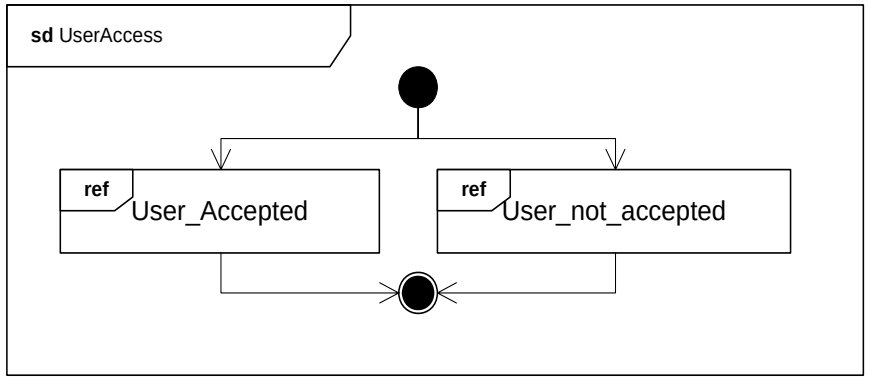
Supplementing

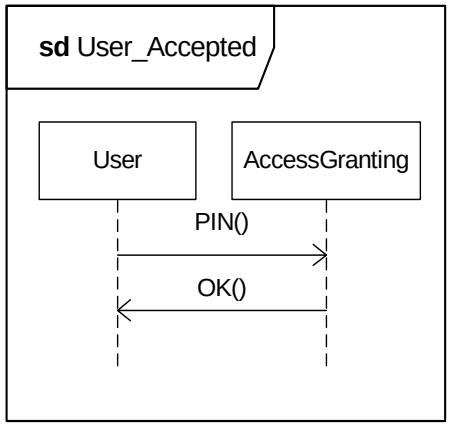


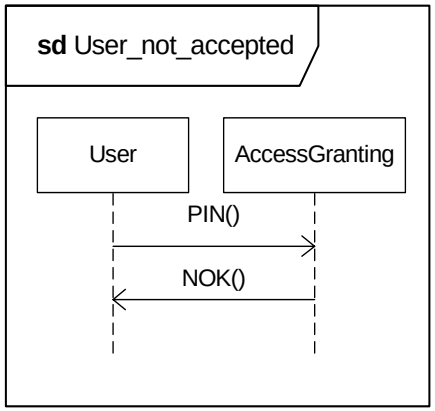


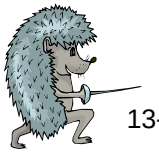
UserAccess

The user will enter an access zone through an access point



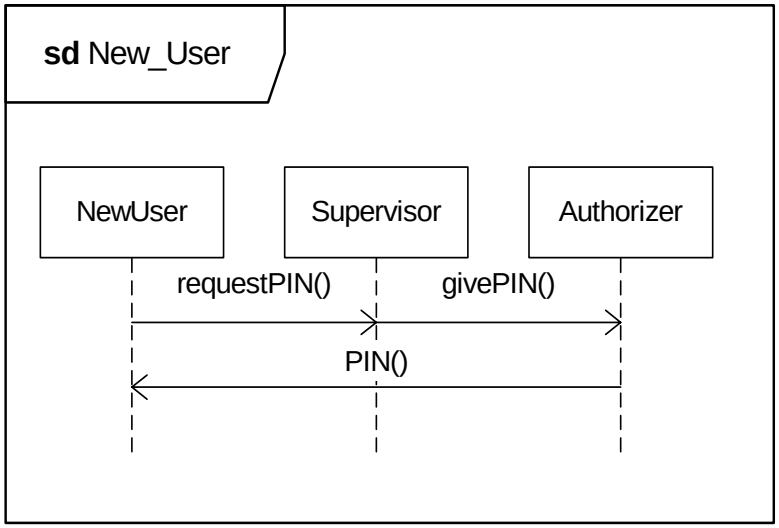






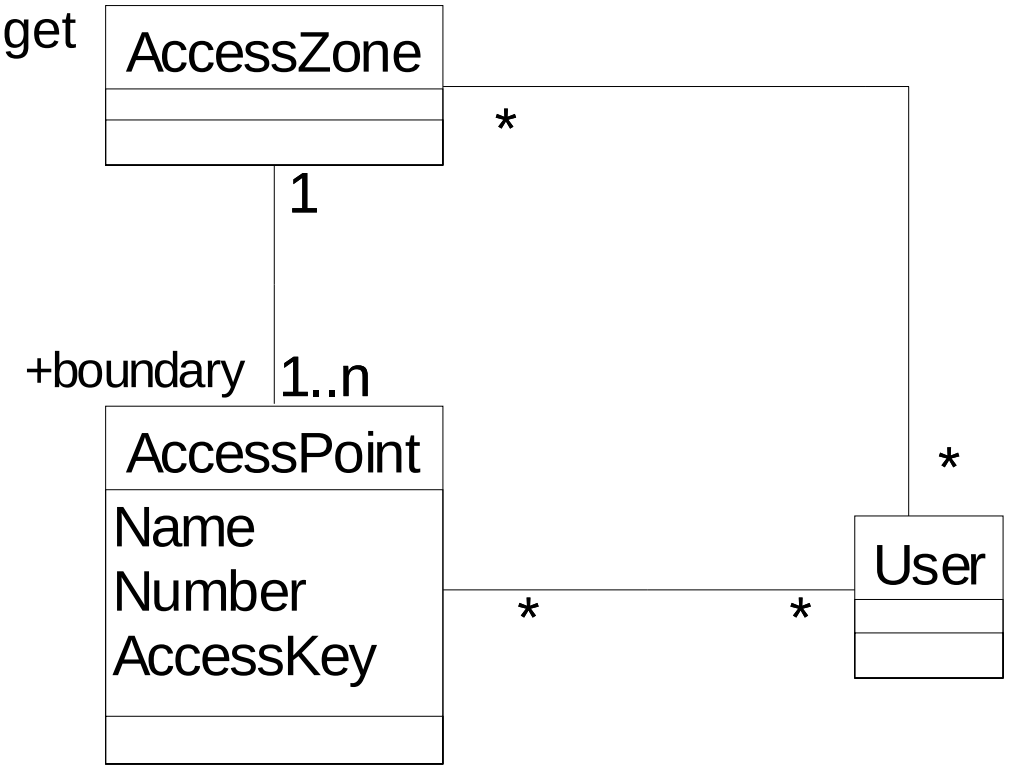
New User

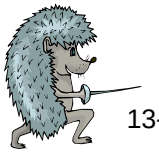
A supervisor will have the ability to insert new users in the system



Improve Precision: (Domain) Object Models

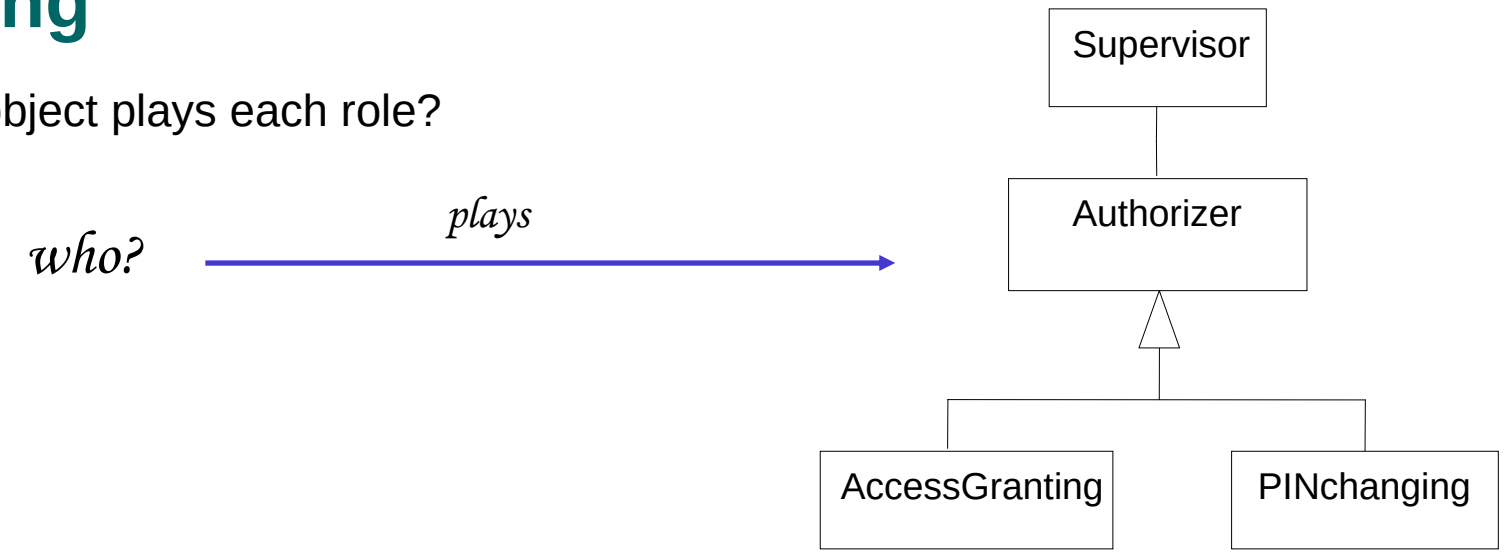
- By simply applying UML to the domain statement, we may get a first class model

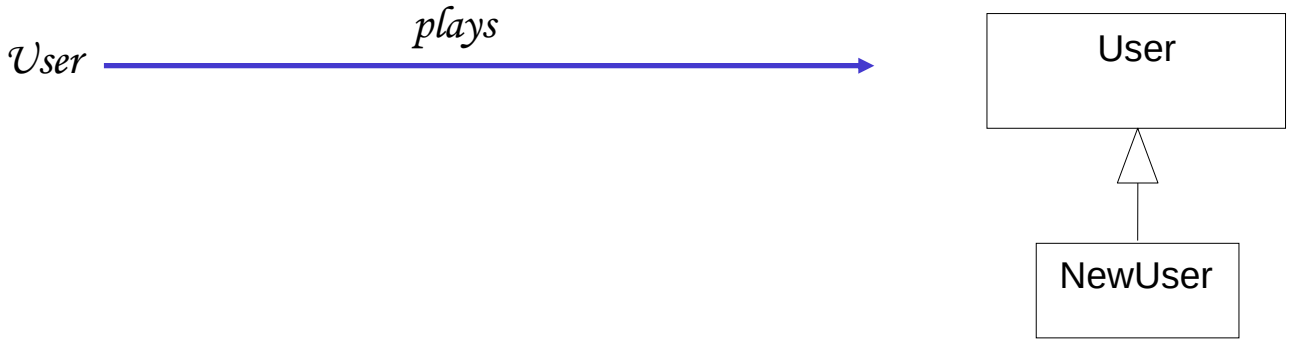




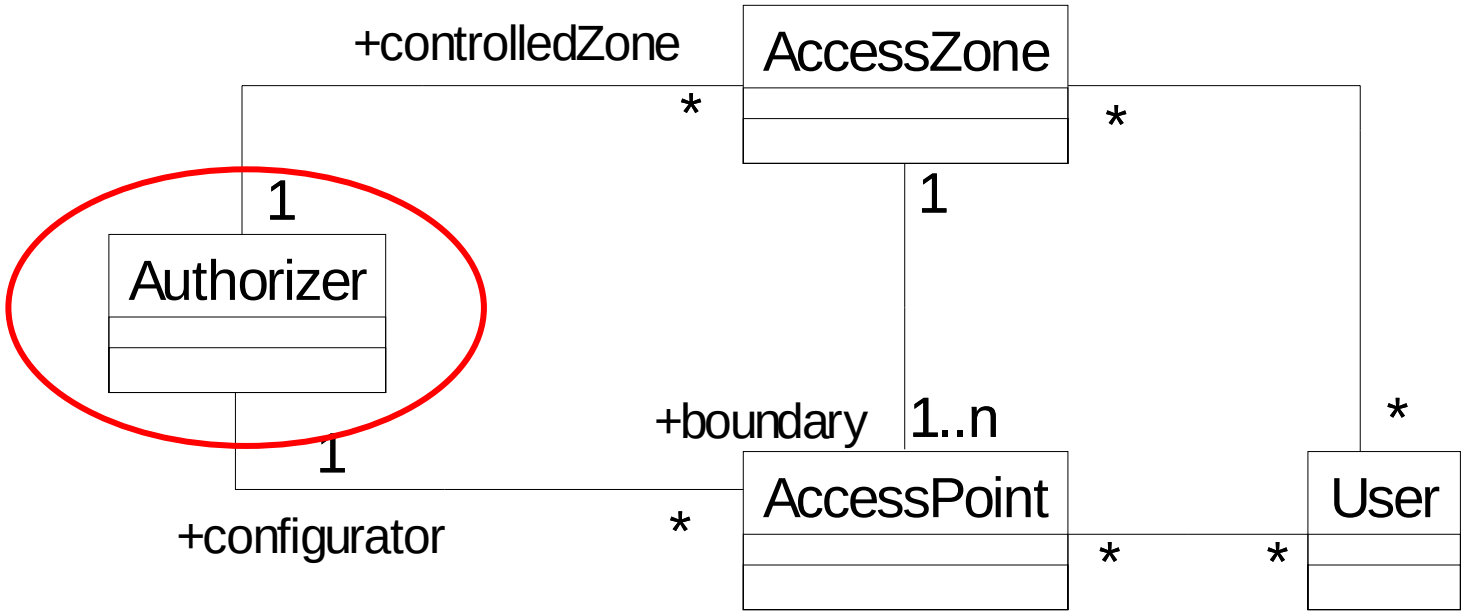
Casting

Which object plays each role?

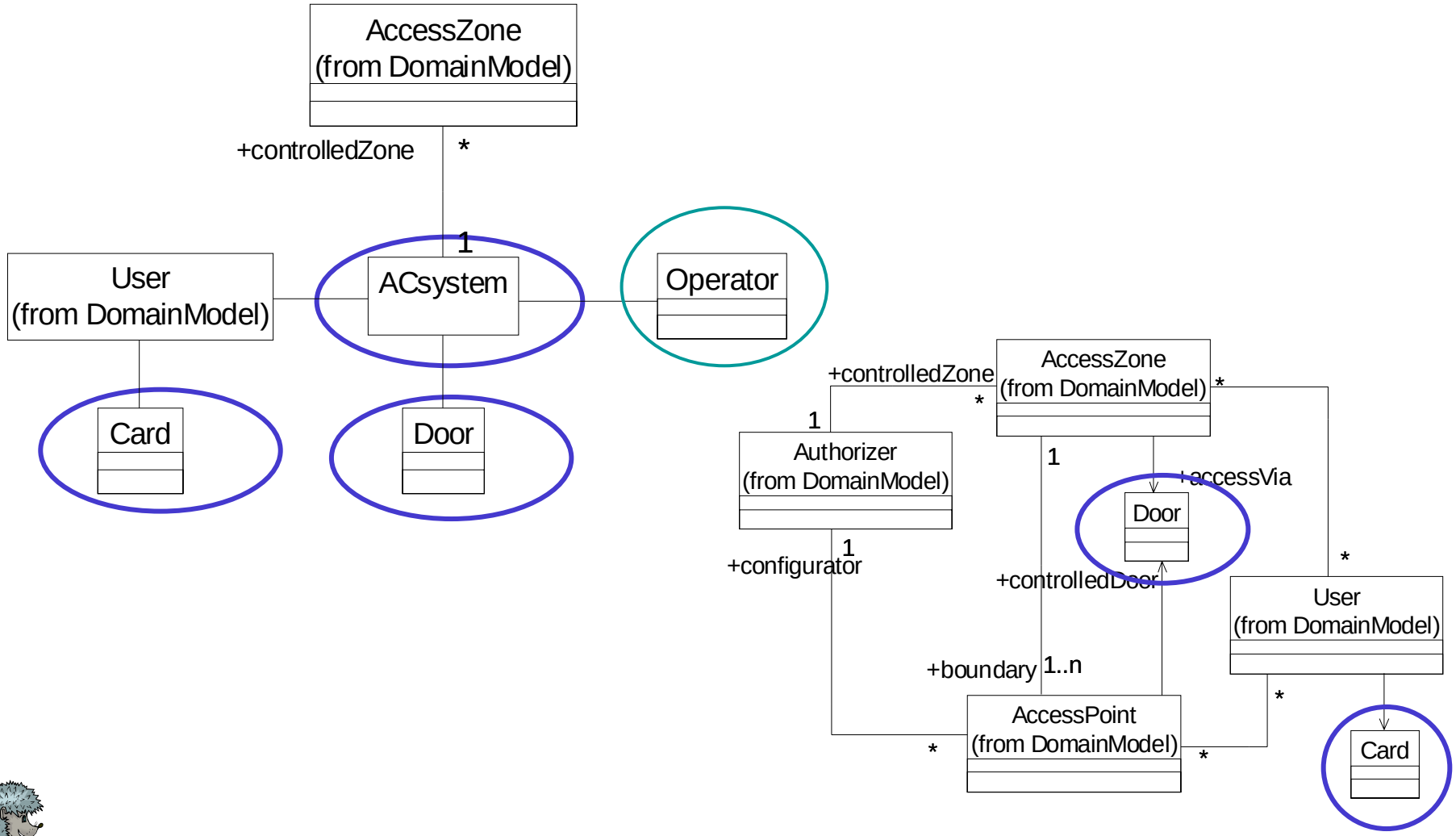




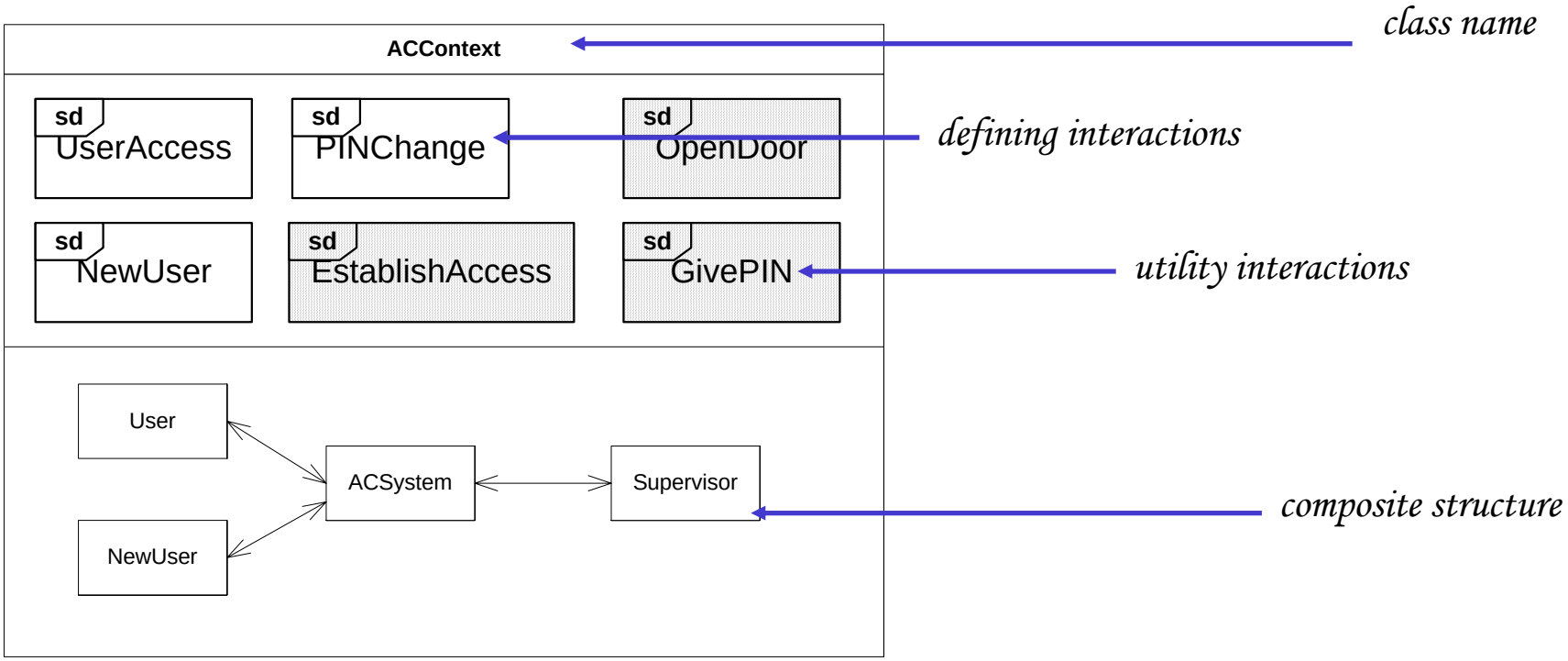
Harmonizing with object model (UML)



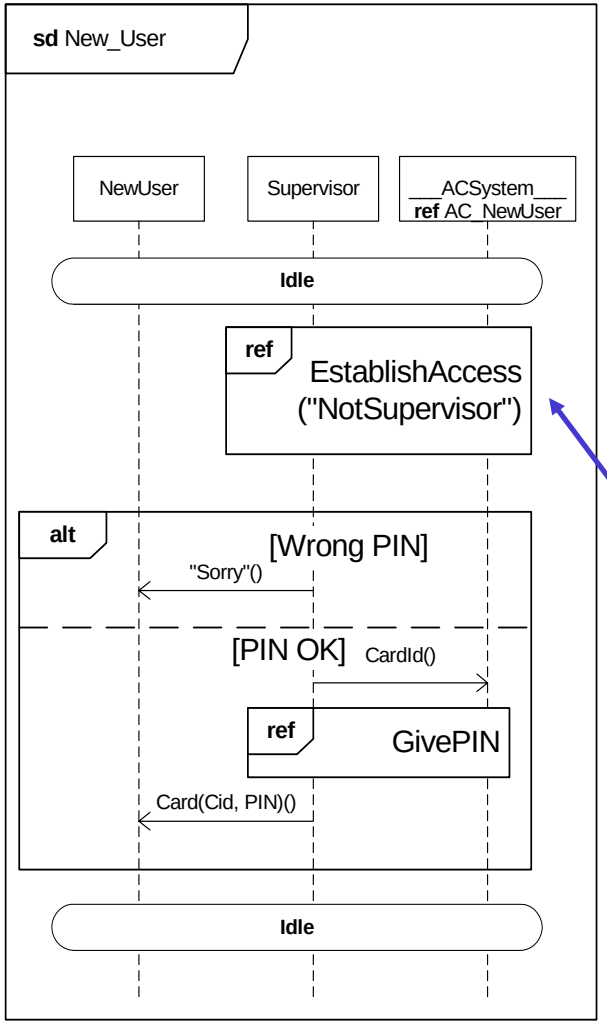
System orientation – becoming more specific

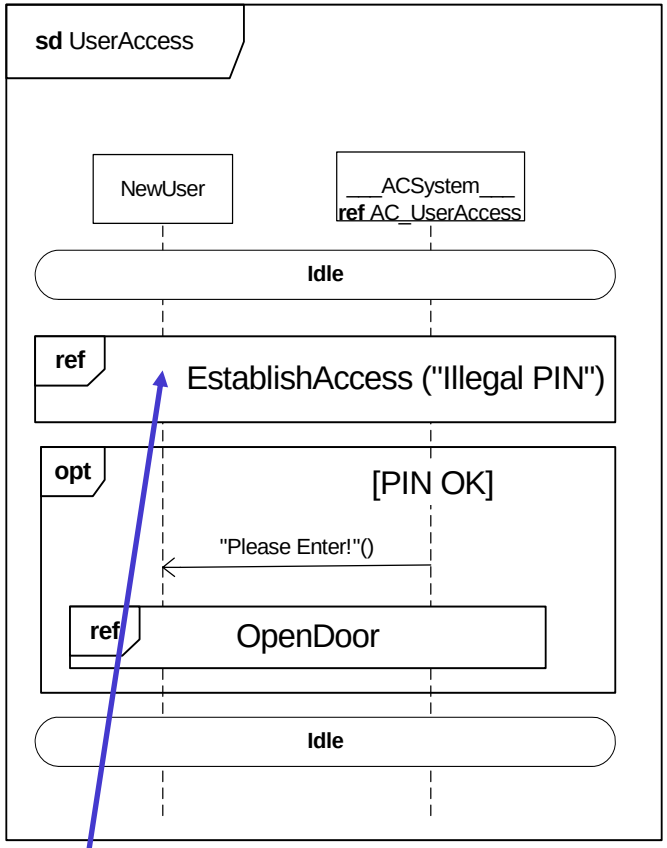


The Access Control Context as UML Class

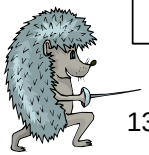


System services

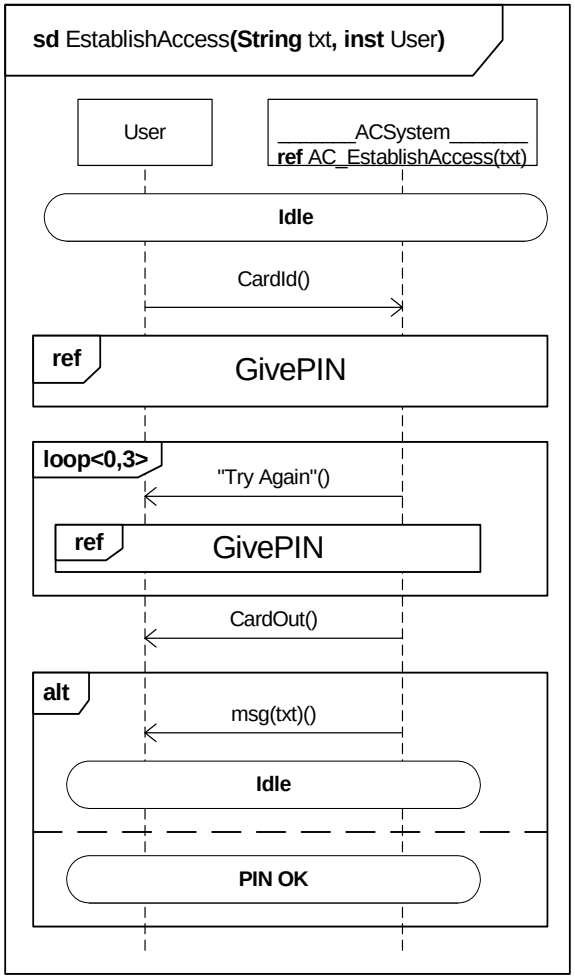




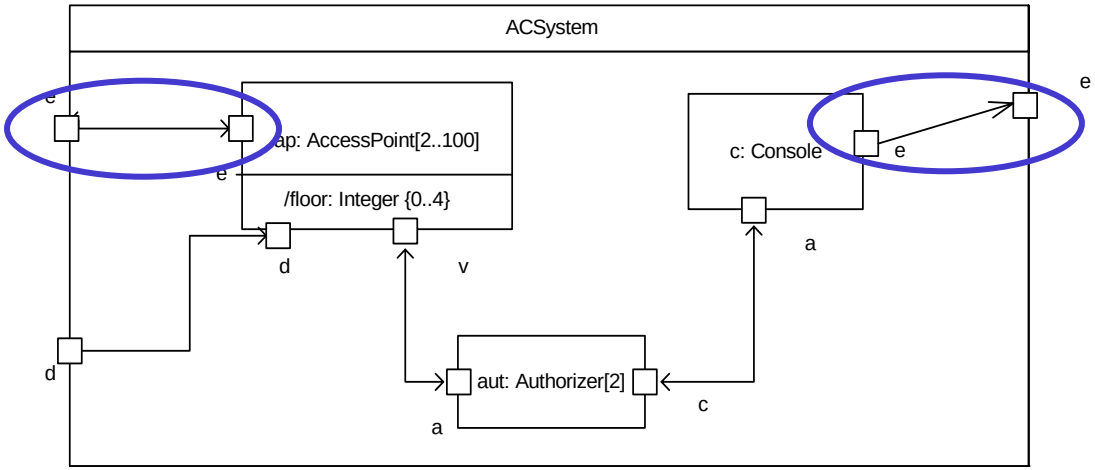
Similarities

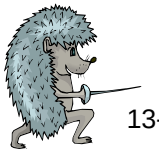


Need for generalization: Entry

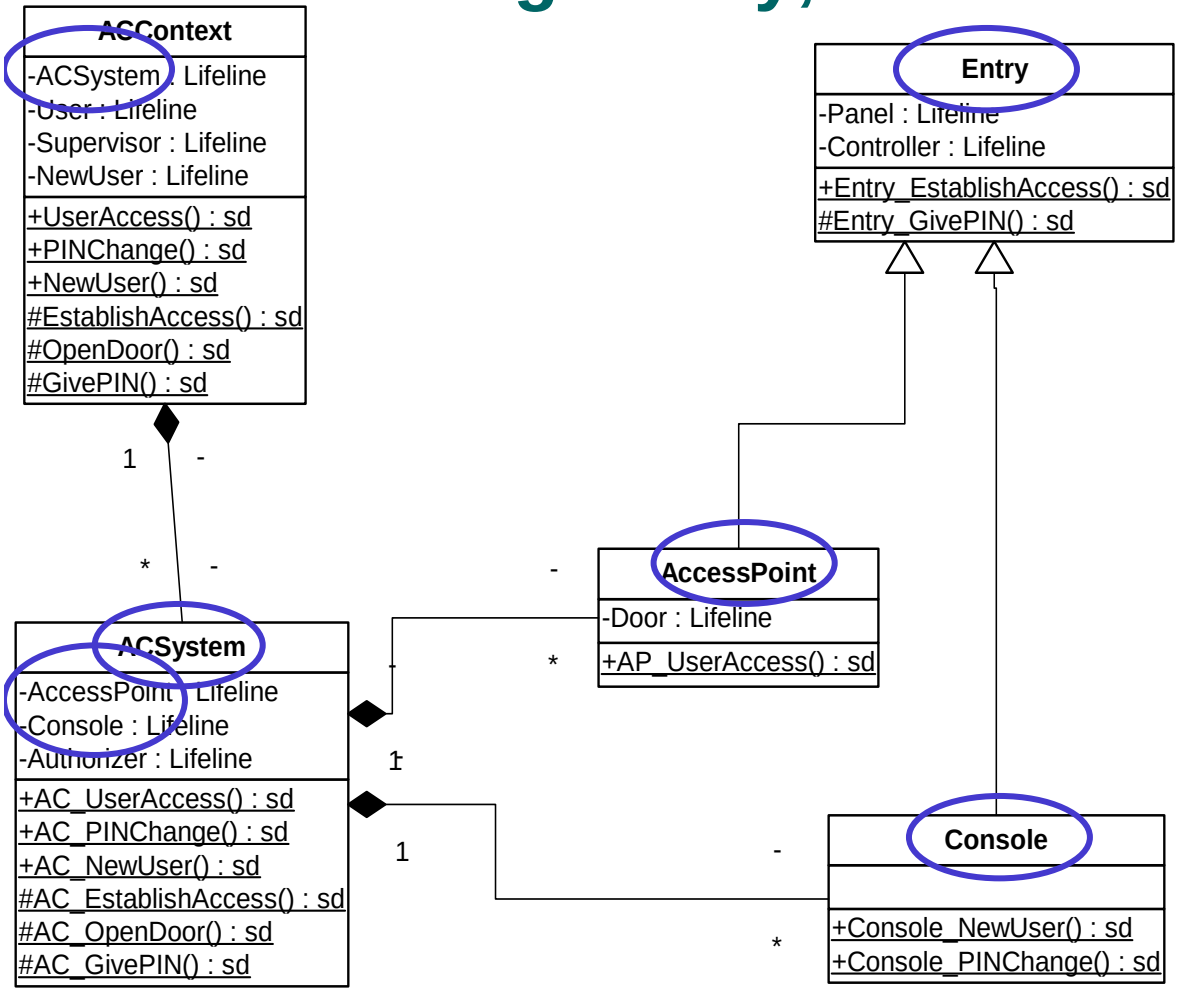


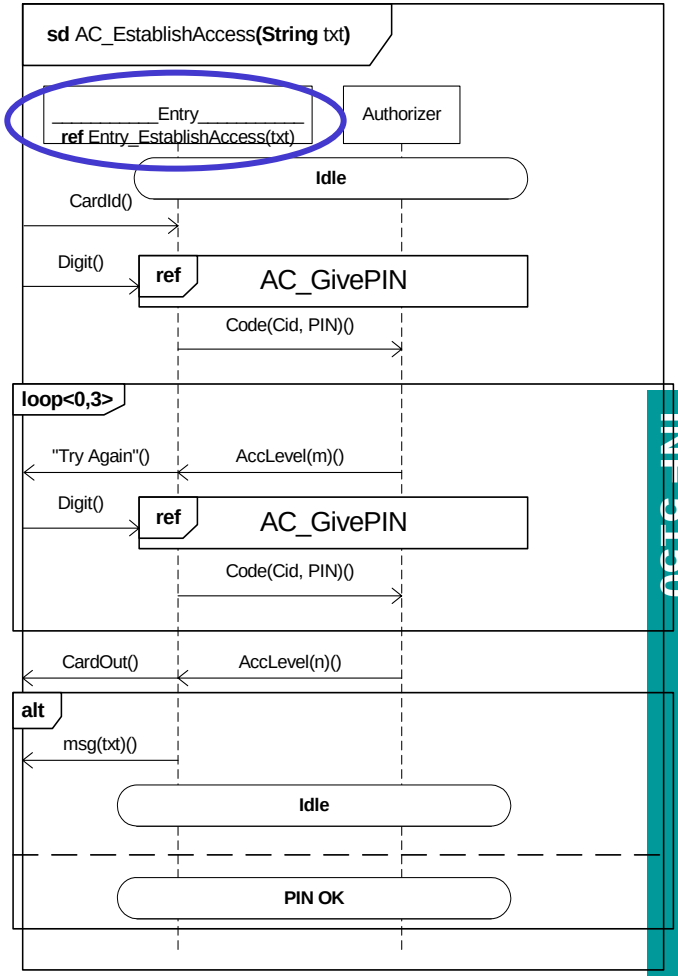
- On what connectors is EstablishAccess applied?
  - between the AccessPoint and a normal User
  - between the Console and the Supervisor user

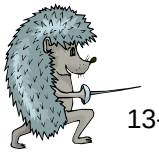




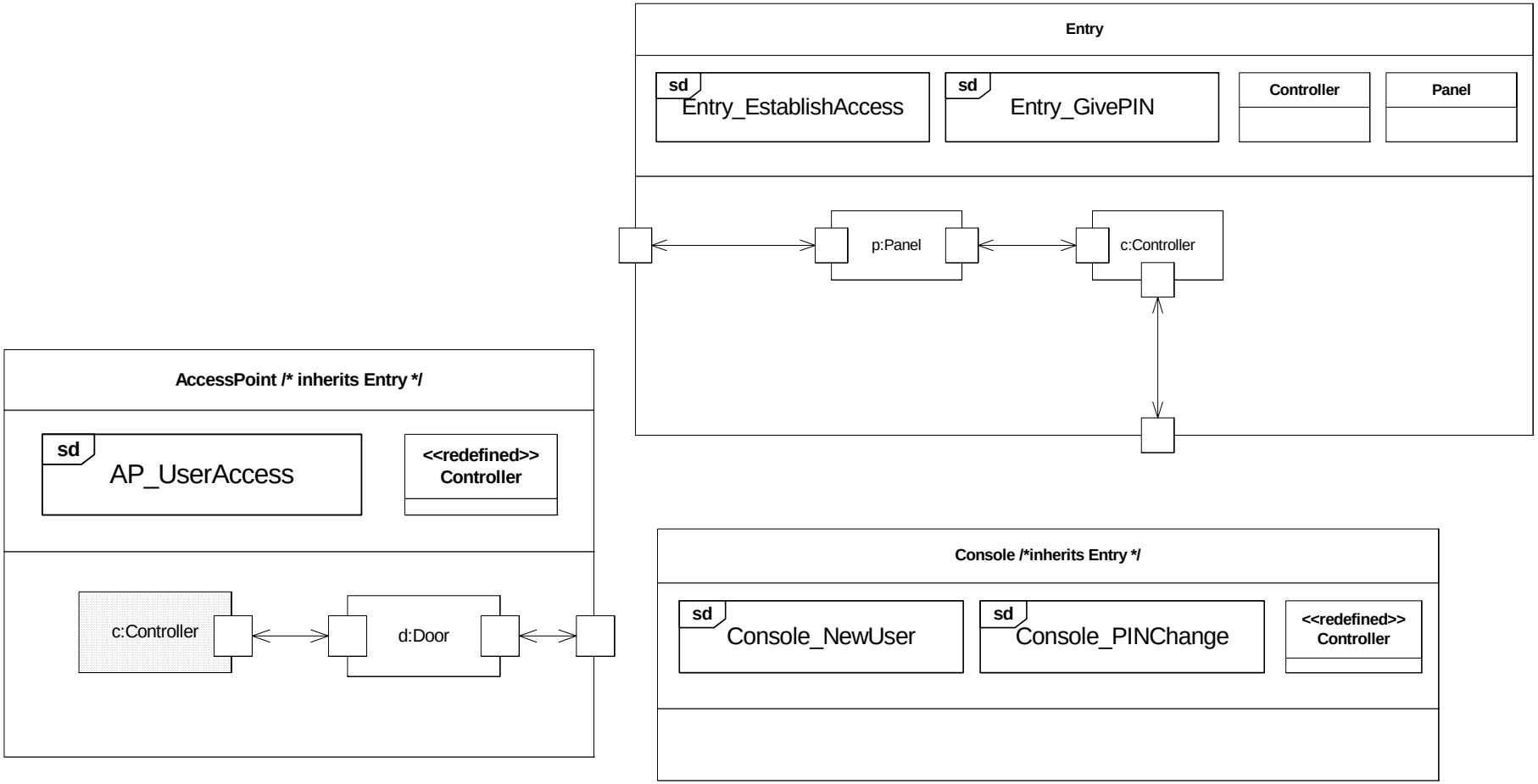
Harmonizing: Entry, AccessPoint and Console

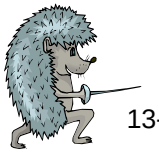




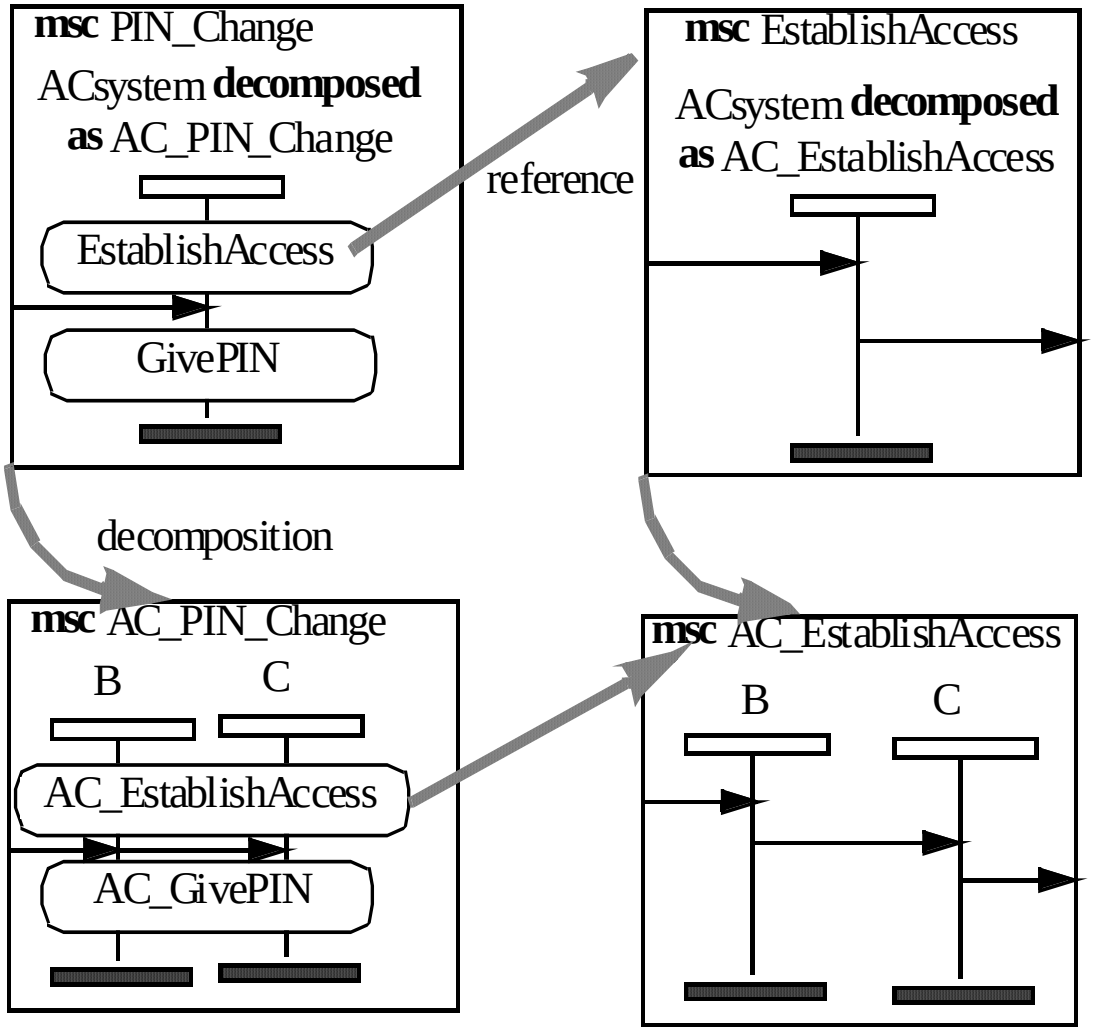


The Entry class hierarchy

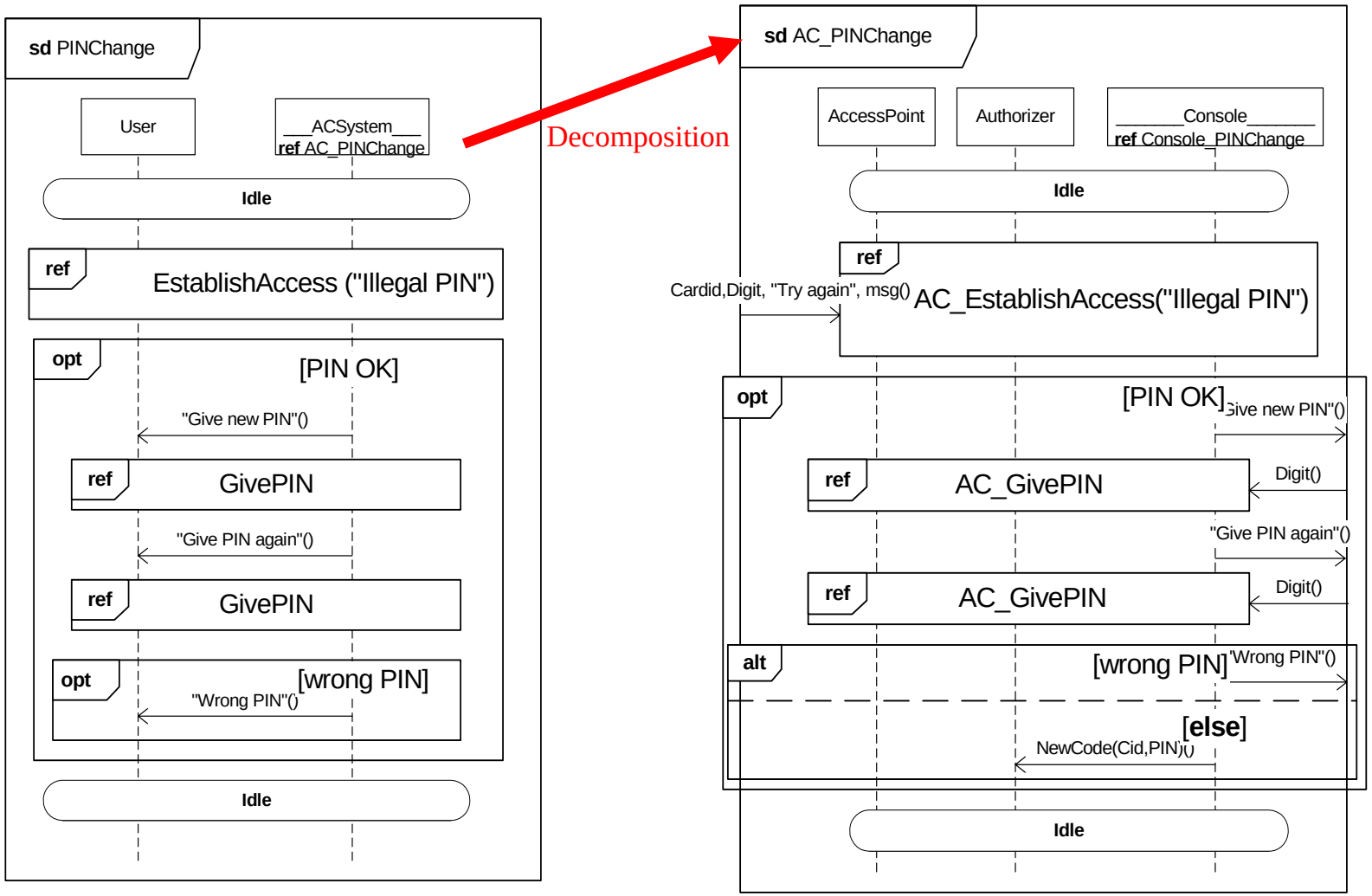




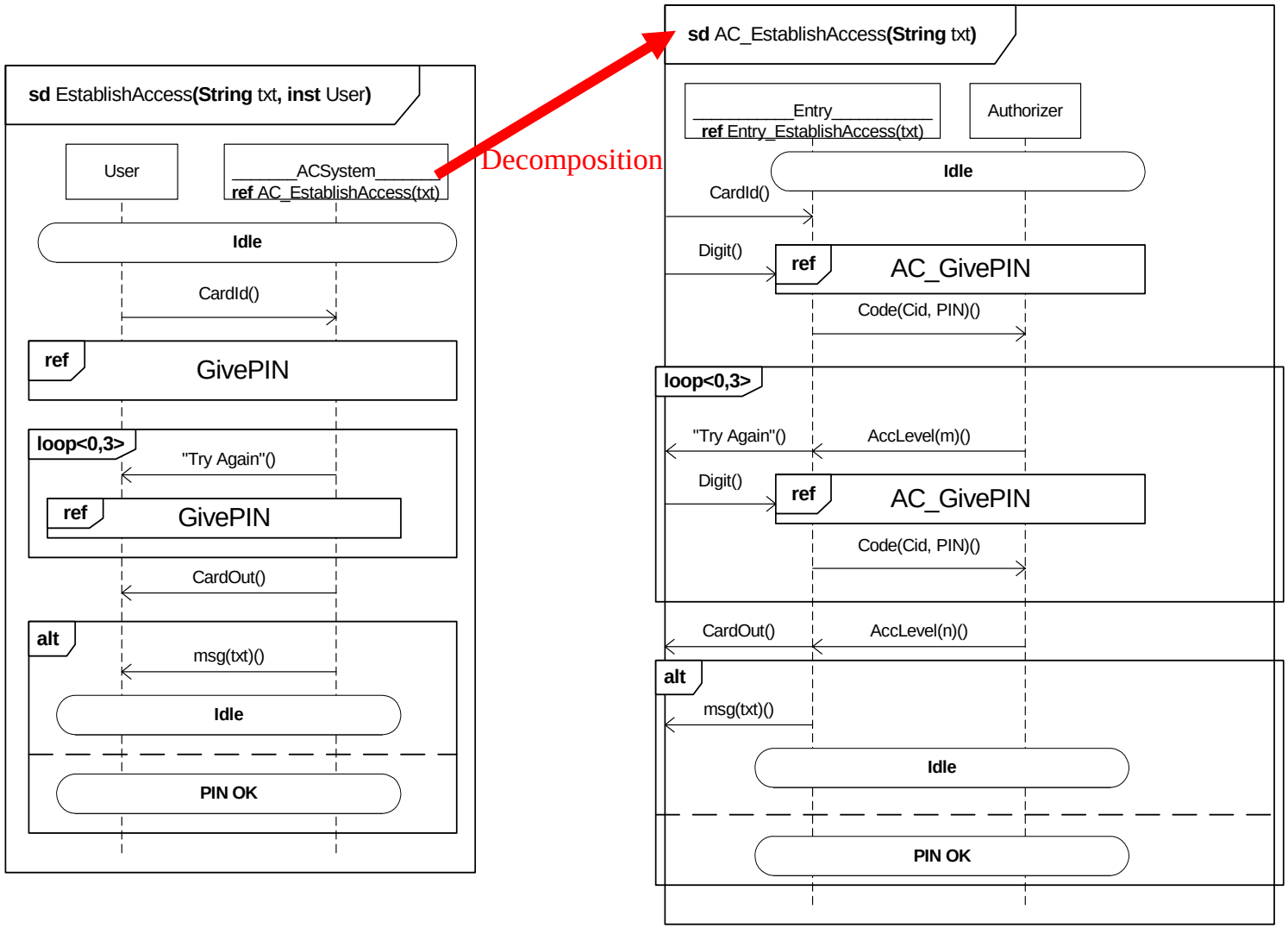
Detailing through commutative decomposition

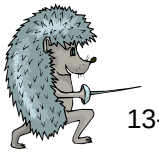


Change PIN

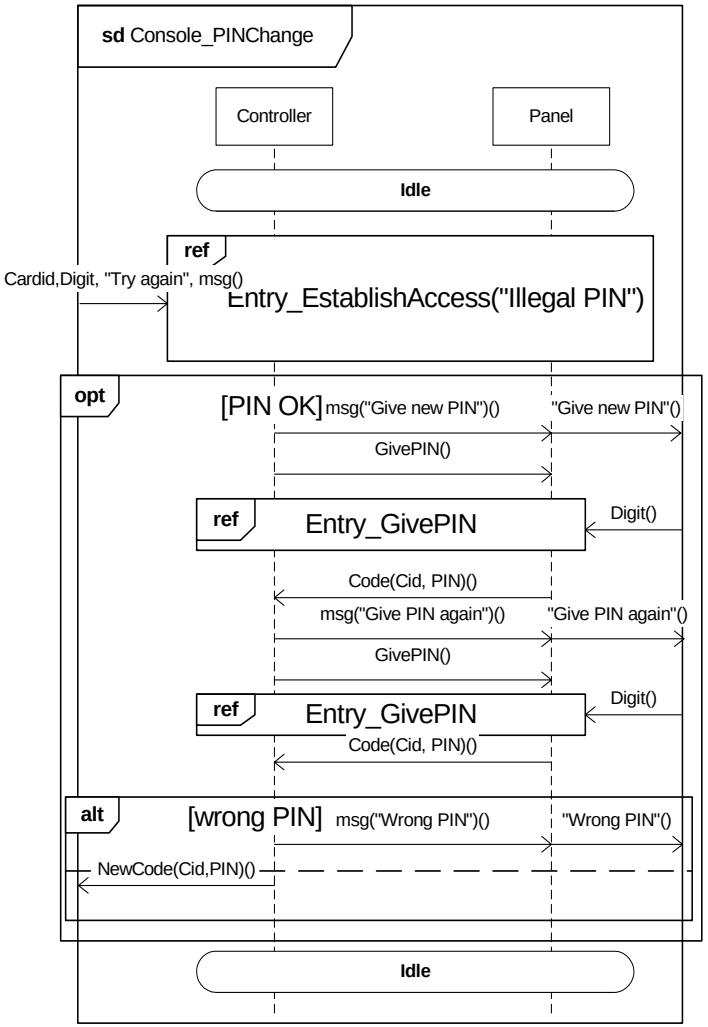

sd PINChange
User
___ACSystem___
ref AC_PINChange
Idle
ref
EstablishAccess ("Illegal PIN")
opt
[PIN OK]
"Give new PIN"()
ref
GivePIN
"Give PIN again"()
ref
GivePIN
opt
[wrong PIN]
"Wrong PIN"()
Idle
Decomposition
sd AC_PINChange
AccessPoint
Authorizer
_____Console_____
ref Console_PINChange
Idle
ref
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

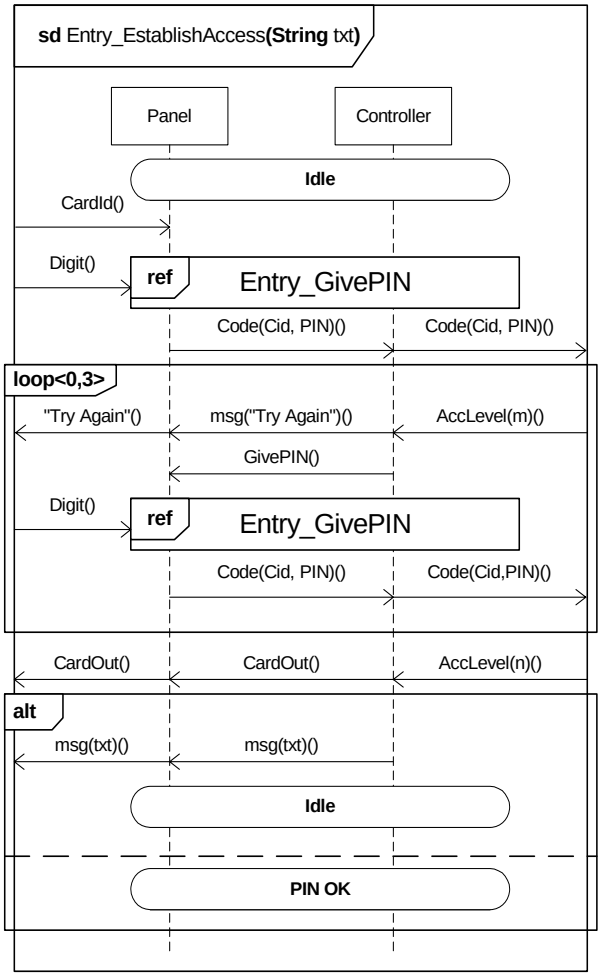
Commutative Decomposition

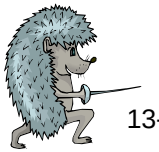




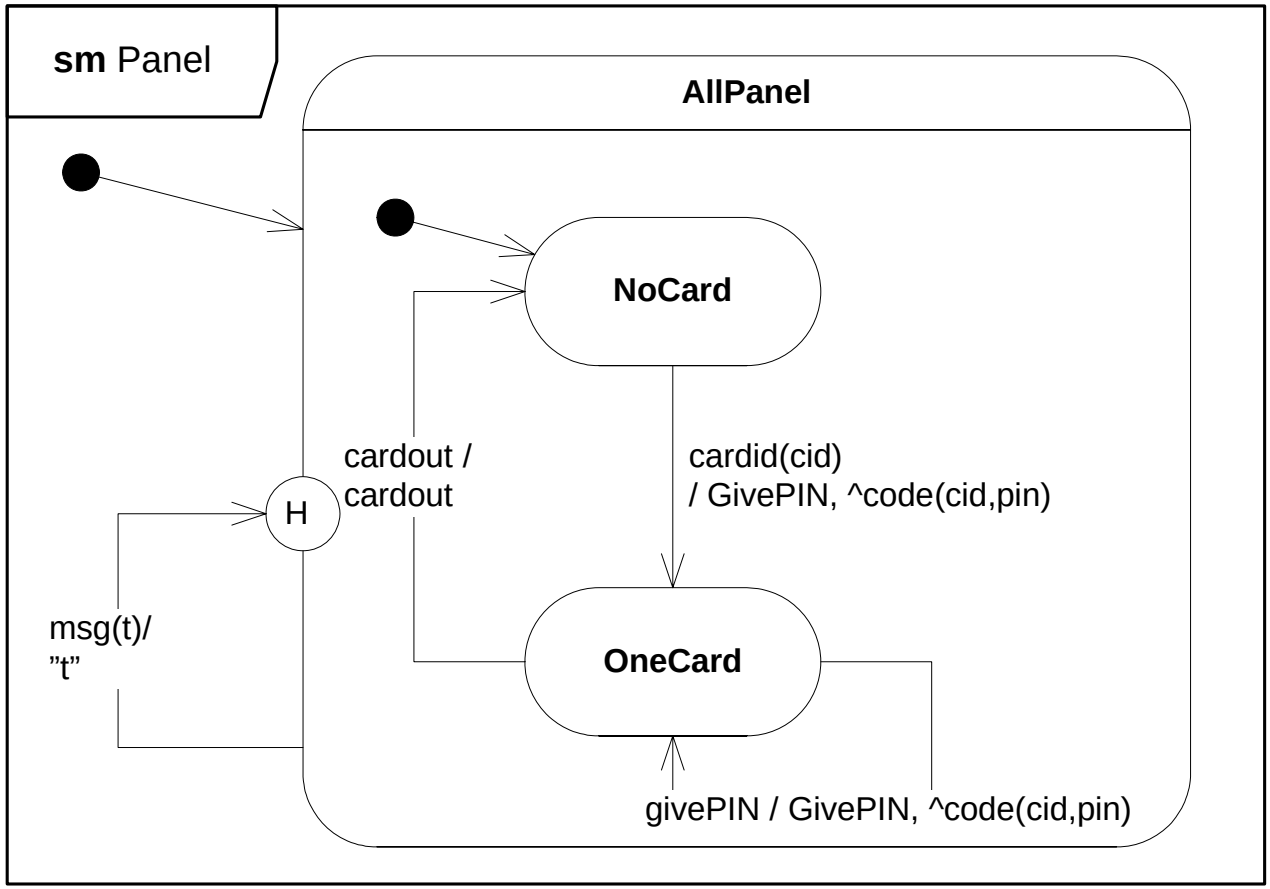
Verification 1: Model checking PIN Change in Panel



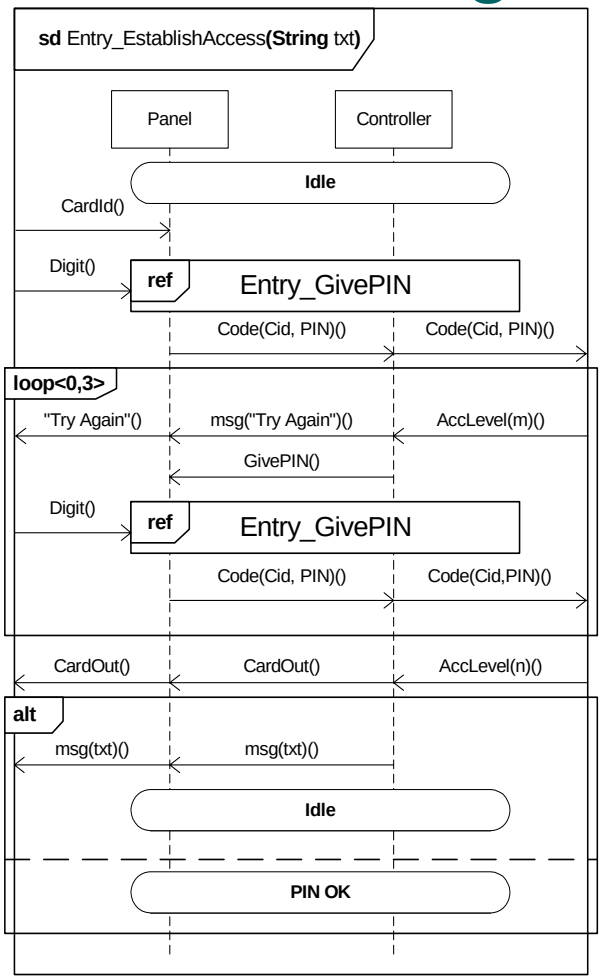


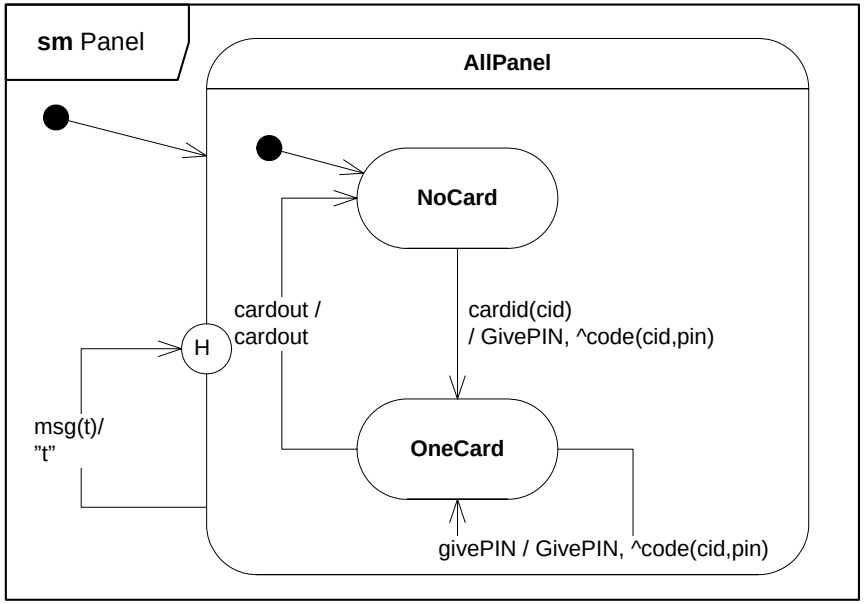


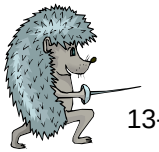
Panel: UML State Machine, GivePIN as a method



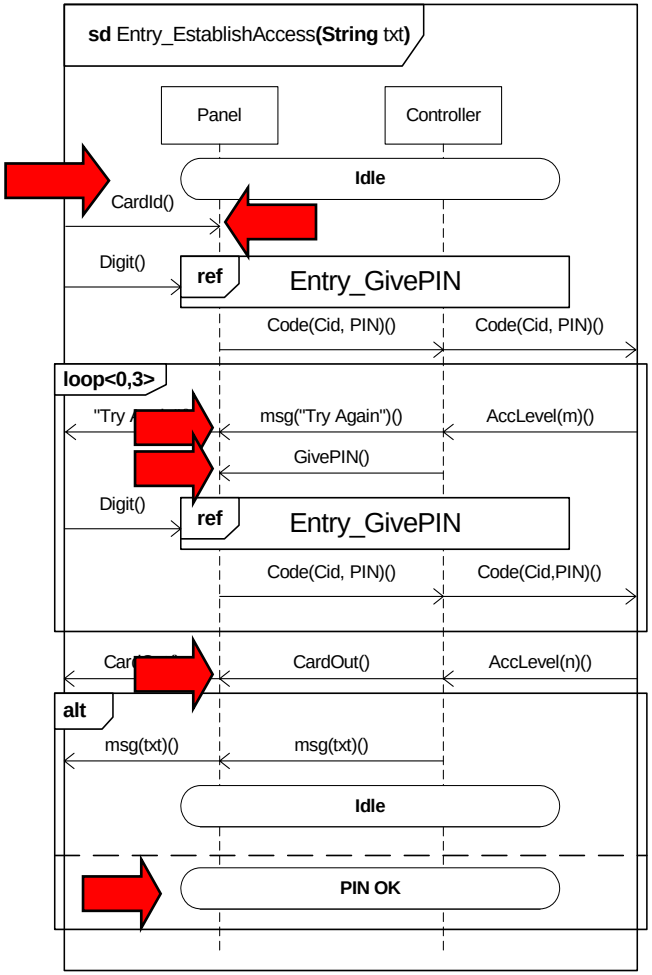
Model checking continued....

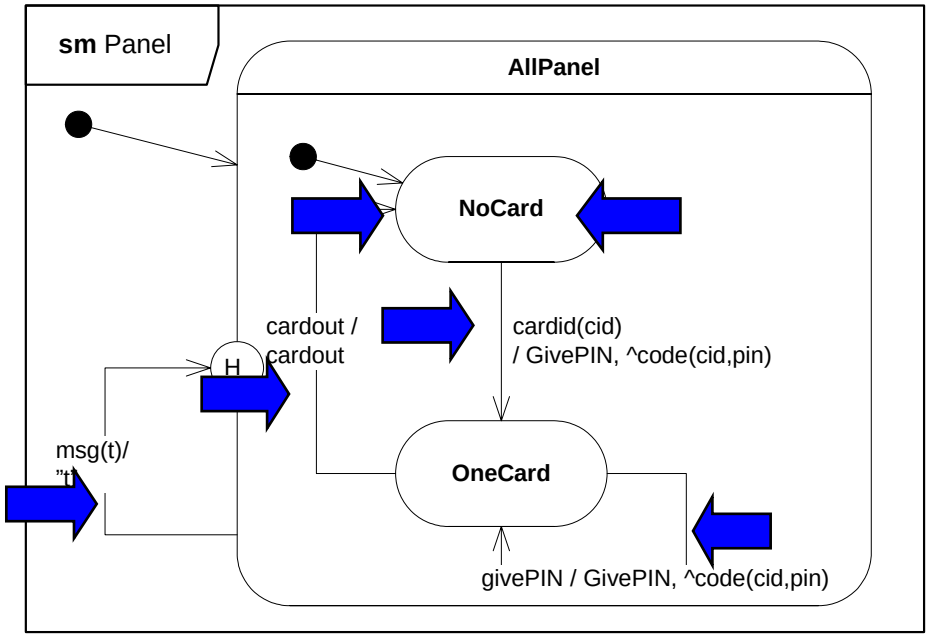


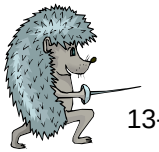




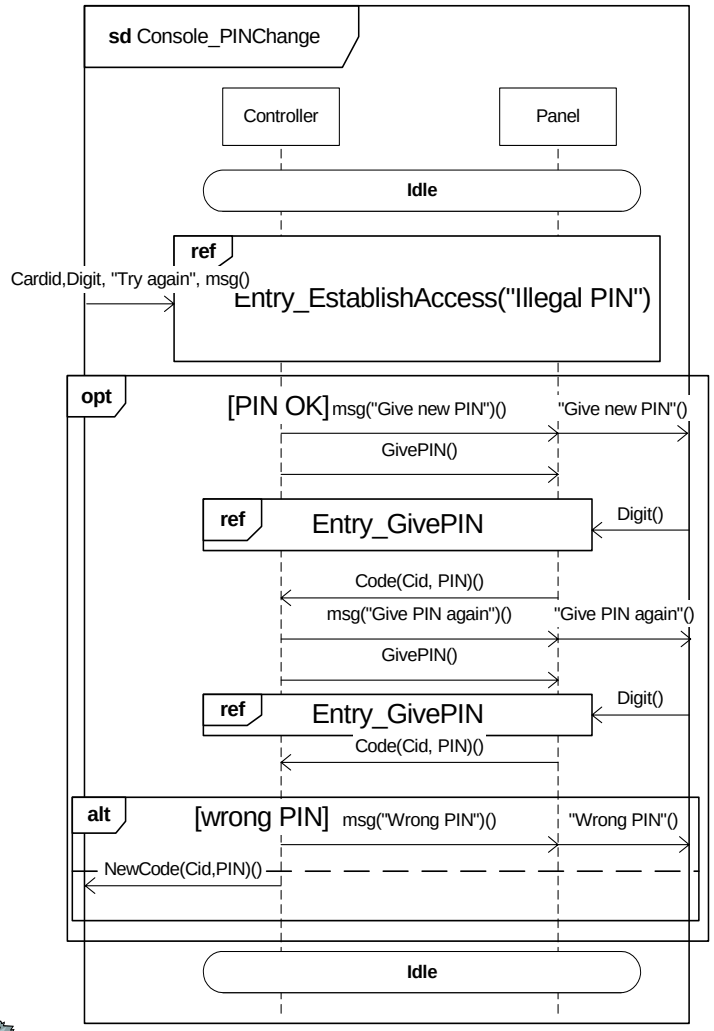
Model checking continued....

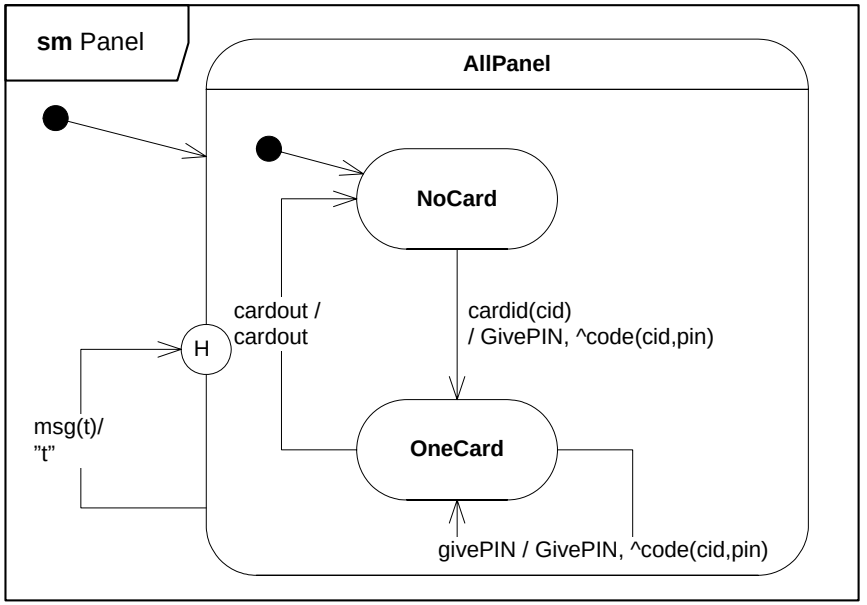




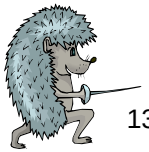


Model checking continued.... ....

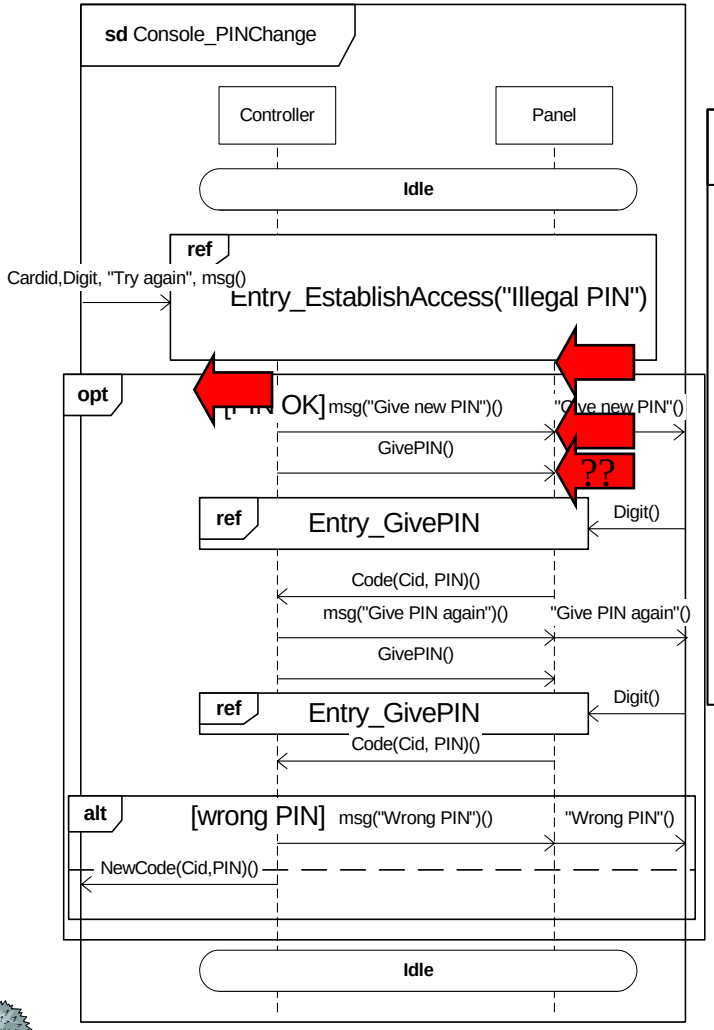


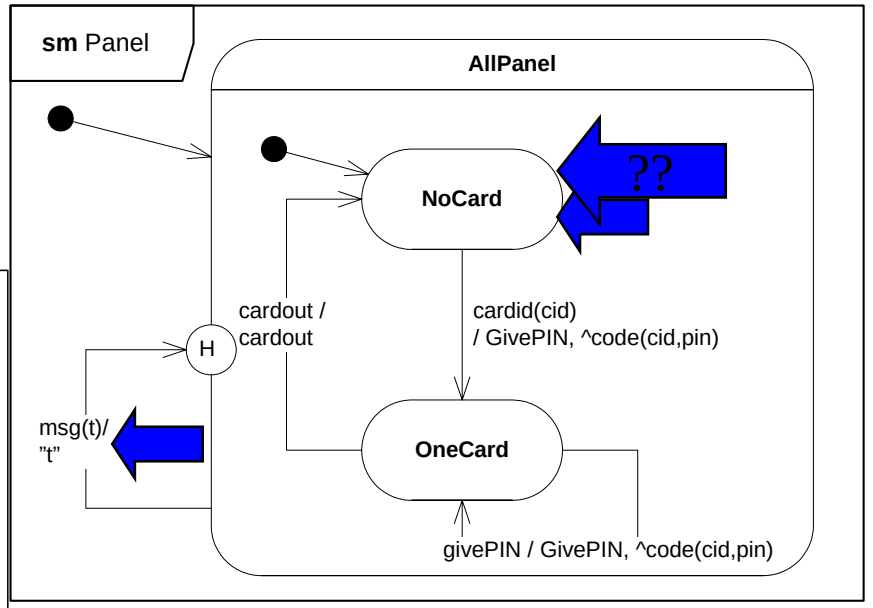


When *EstablishAccess* has elapsed, *Panel* is in state *NoCard*, but it receives *GivePIN*!

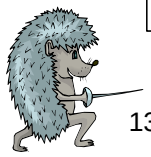


Model checking continued.... ....



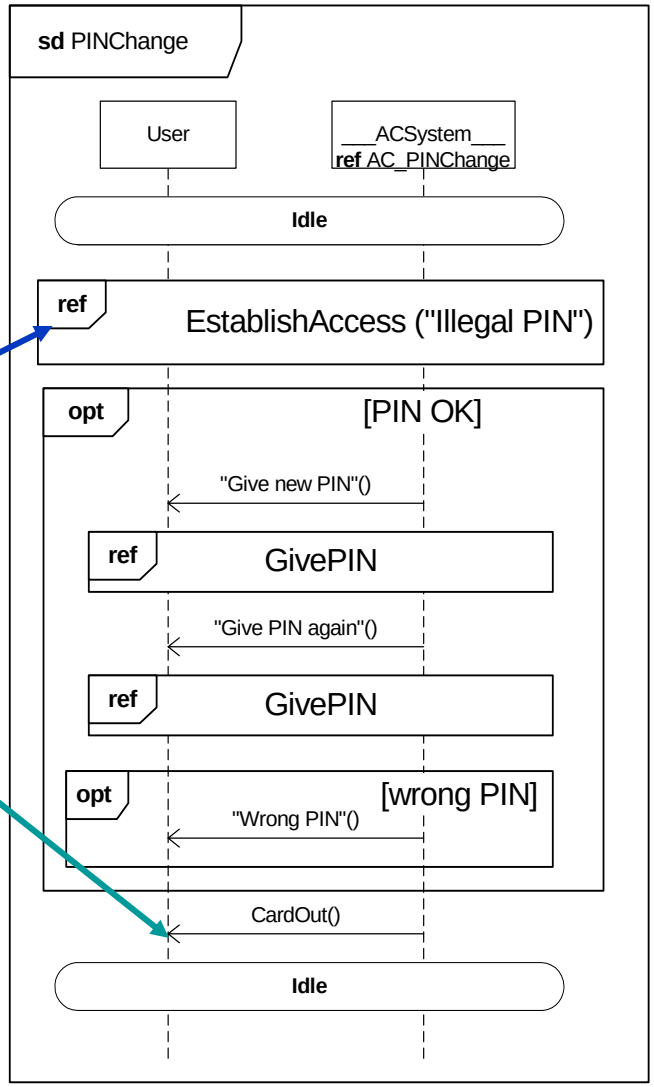


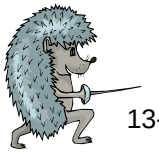
When *EstablishAccess* has elapsed, *Panel* is in state *NoCard*, but it receives *GivePIN*!



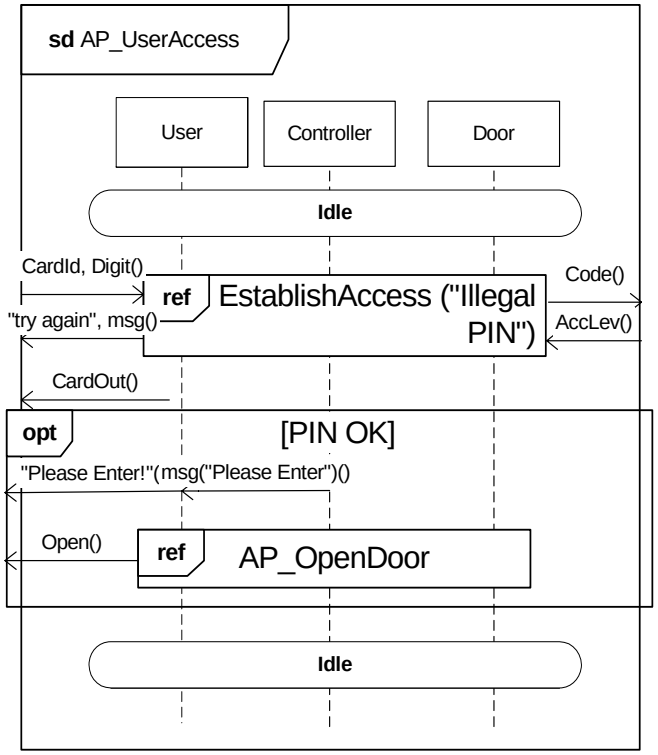
Harmonizing

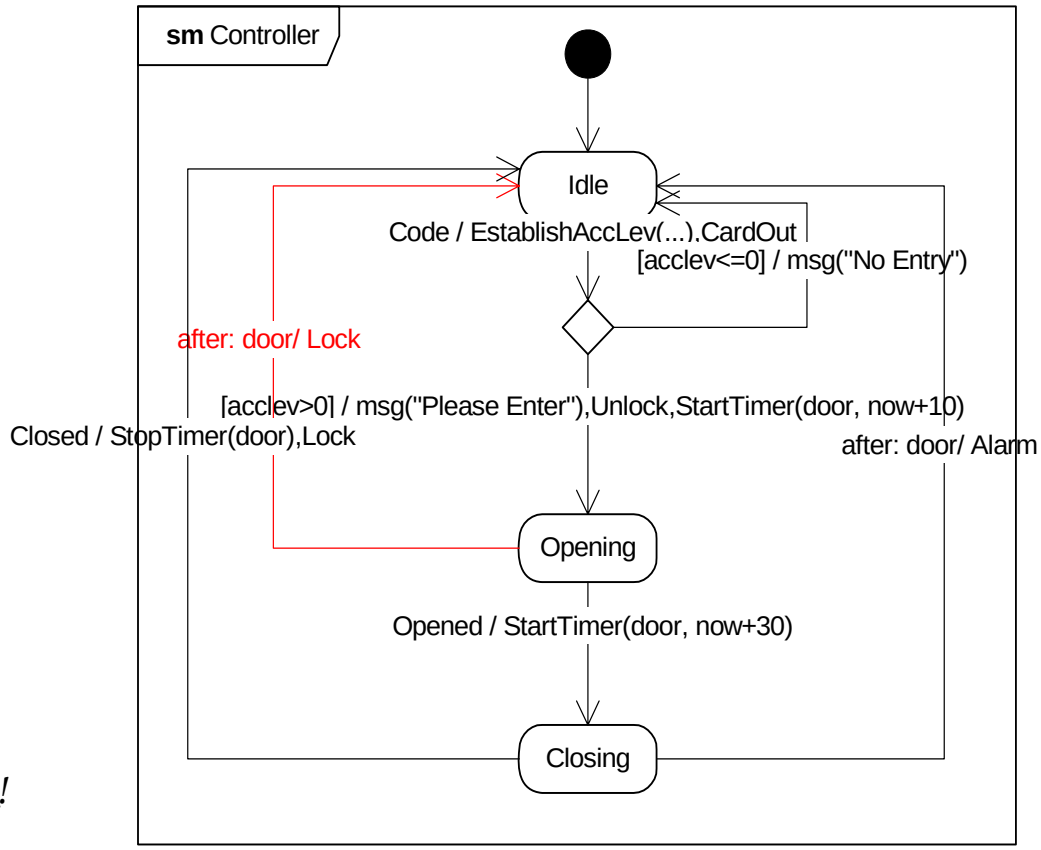
We decide to move CardOut from EstablishAccess to the end of PIN_Change





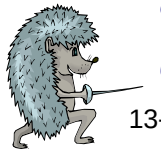
Verification 2: AccessPoint's Controller





sd: User Access vs sm: Controller = OK!

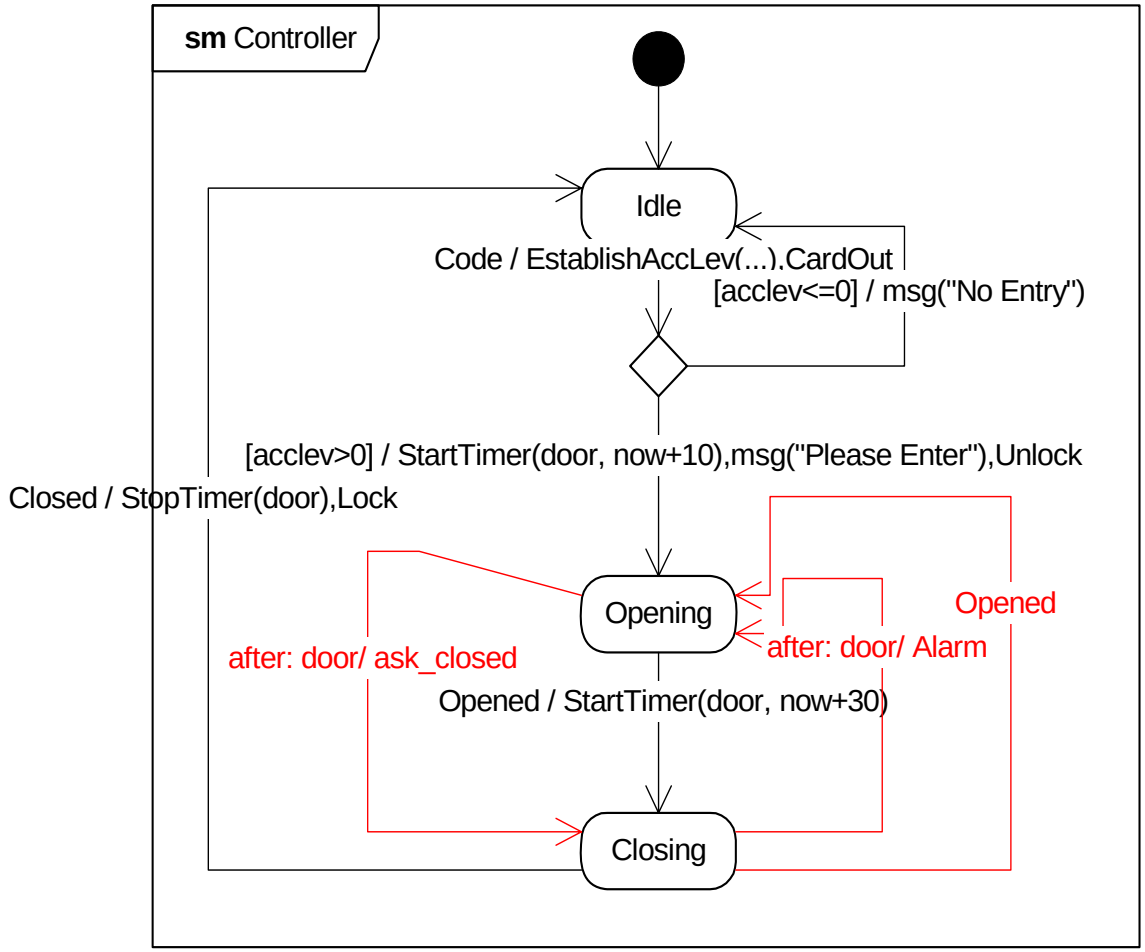
Are we then certain that AccessPoint's Controller is perfect?



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