ES Software Engineering

Lecture 5
Use case modeling and Application logic design

In the previous lecture

1. Interfaces
2. Package diagrams
3. State machine
4. States, transitions
5. Concurrent states, pseudo states

Plan of the lecture

1. Deployment diagram
2. Modeling Use Cases
3. Defining actors and Use Cases
4. Application logic design
5. Problem of attacks which keep the functionality of the system
6. Data Interfaces

Deployment diagram

Deployment diagram – is used for modeling large systems. It is particularly important for the design issues of embedded systems, because it represents the relationship between the software layer (artifacts) and the equipment (nodes).

Deployment diagram

Deployment diagram

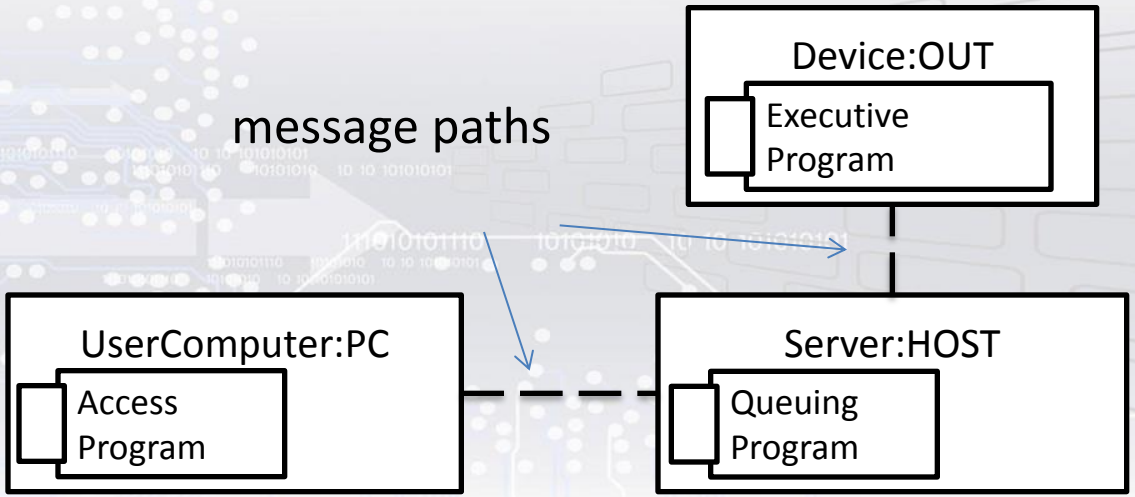
1. Software – represented by:

- artifacts – compiled and executable components
- data
- libraries

2. Hardware – presented with nodes which represent devices. The nodes are connected with rails/paths representing a defined way of communicating.

Deployment diagram

Deployment diagram – example



Use case modeling

Use case diagram

"case" – means case when a system is used to fulfill user's requirements

"use case" – an abstract unit of functionality that system delivers to the user. Use cases consider analysis of all parts of a system which does not take into account a non-functional requirements of the system.

"actor" – a role that human, device or another system plays when interacting with the system

Use case modeling

Examples:

1. Actors:

- humans (operators, supervisors) who interact with a system
- internal parts of a system – they can change a system's state but their states can't be changed by the system
- external systems

2. Use cases:

- calibration
- measures
- calibration

Use case modeling

Recognizing actors:

- Who will use the system and its functionality?
- Who should supervise the system?
- What resources does the system posses (devices/people)
- Which systems does the system communicate with?
- Who or what gets the results of the work of the system?

Use case modeling

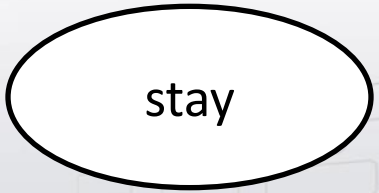
Recognizing Use Cases:

- Analysis of system functions from user's point of view
- Should the actor be informed about events in the system?
- Does the actor need to generate information in the system?
- What is the input and output data of the system?
- System bottle necks

Use case modeling

Use case modeling – notation:

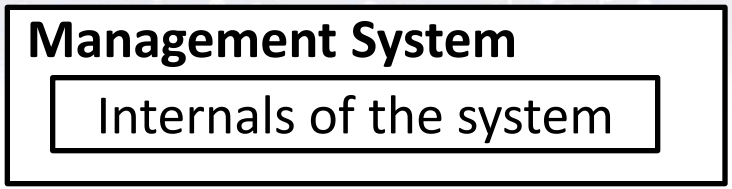
a) 

or


Use Case

b) 
actor

user

c)


system name

Use case modeling

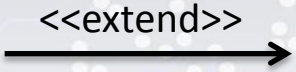
Use case modeling – notation:

a) 

reuse block (separate
or used by several Use Cases)

b) 

interaction

c) 

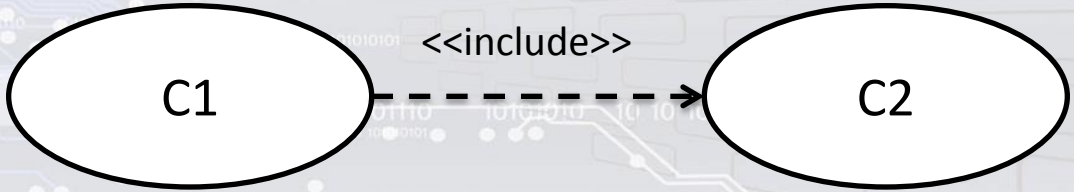
relation (extend or include) between
Use Cases or between
a Use Case and a reuse block

Use case modeling

<<extend>> and <<include>>

Are relationships which determine the order of defining Use Cases.

Example 1 <<include>>: case C1 is first in the sequence, meaning it's the base one:



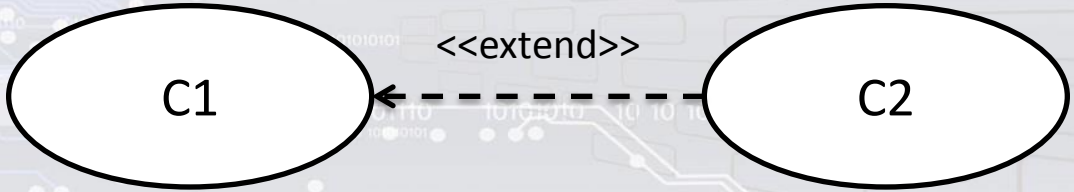
C1 always turns on C2

This is a basic sequence, which always occurs.

Use case modeling

<<extend>> and <<include>>

Example 2 <<extend>>: case C1 is first in the sequence, meaning it's the base one:

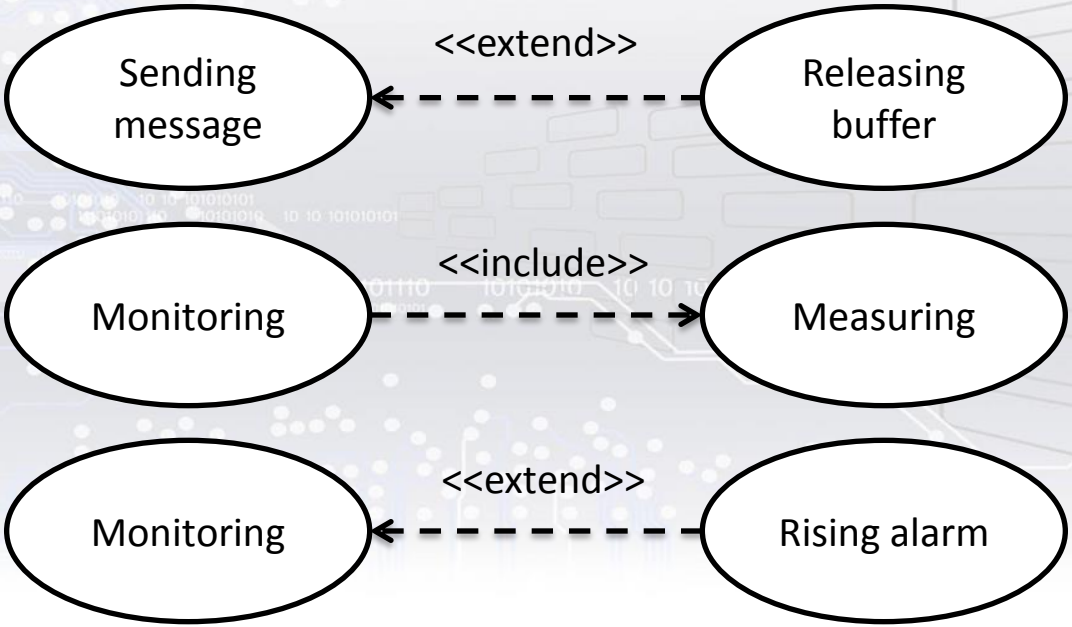


C1 is sometimes extended with C2

This is an optional sequence, which doesn't always occur.

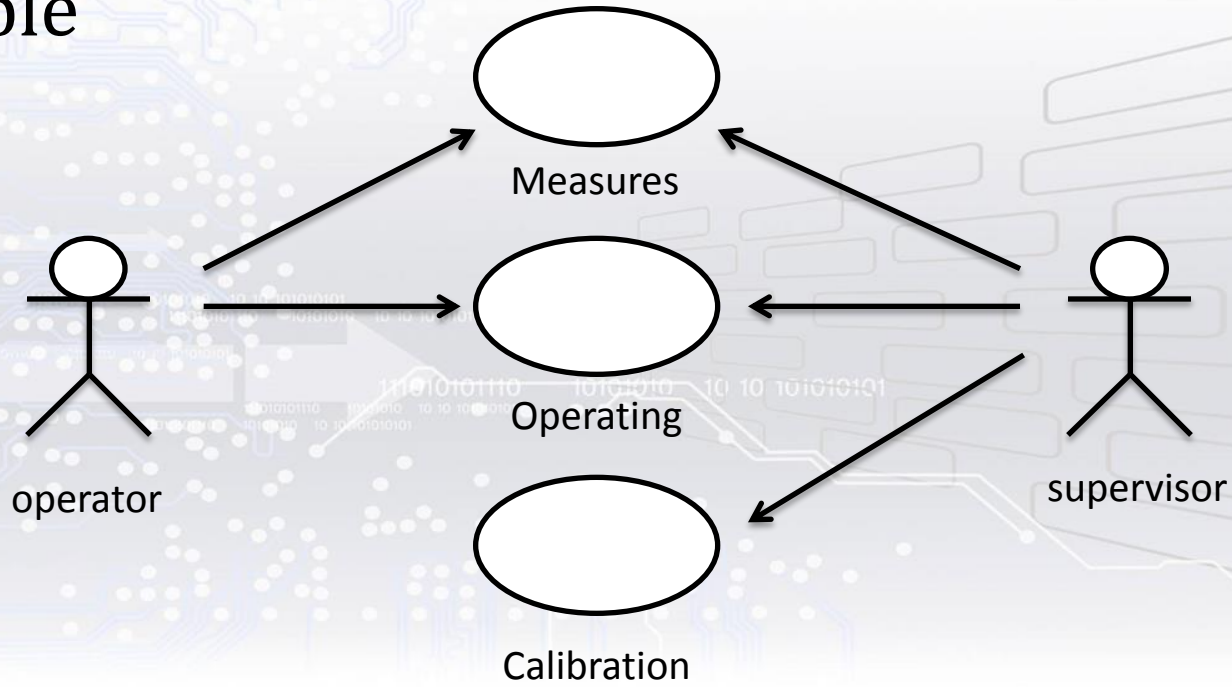
Use case modeling

Examples



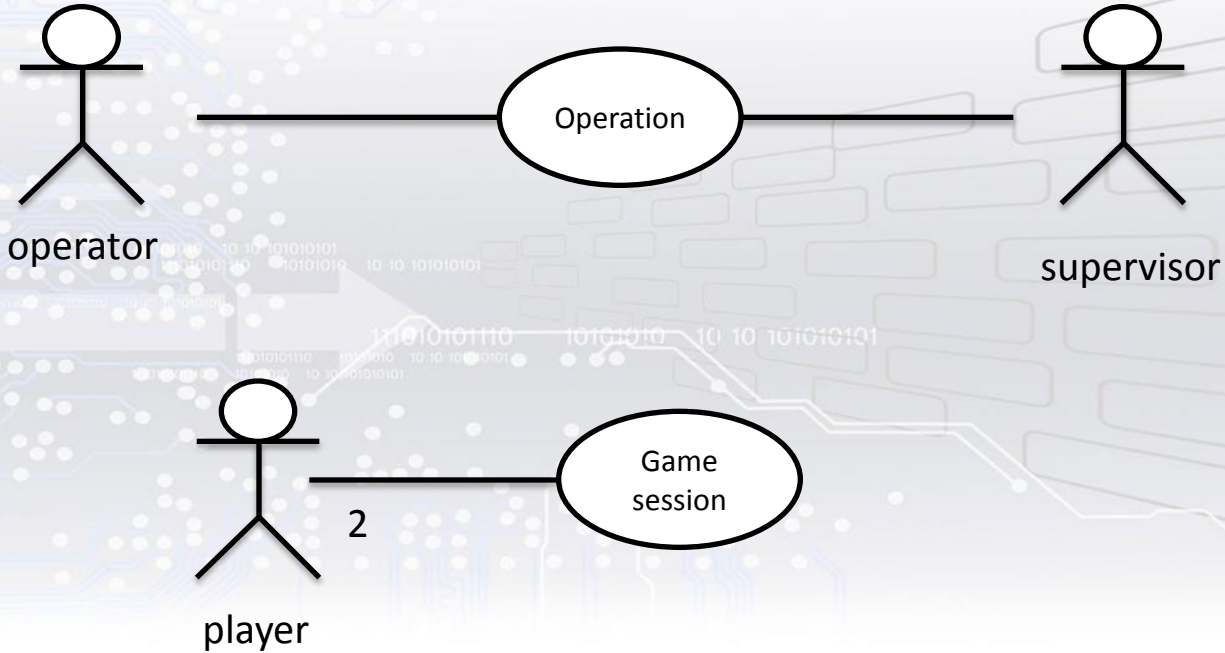
Use case modeling

Example



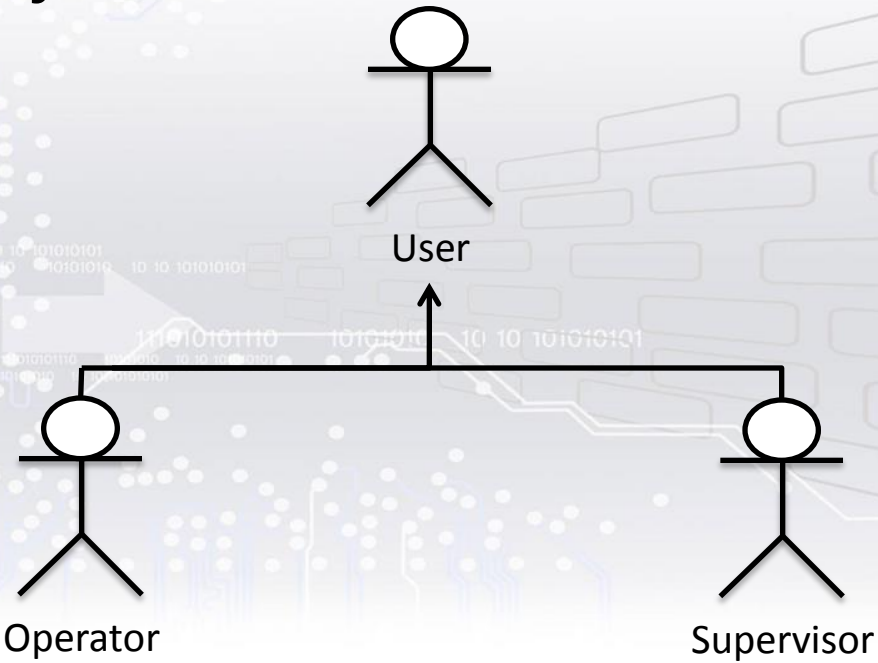
Use case modeling

Many actors in one use case



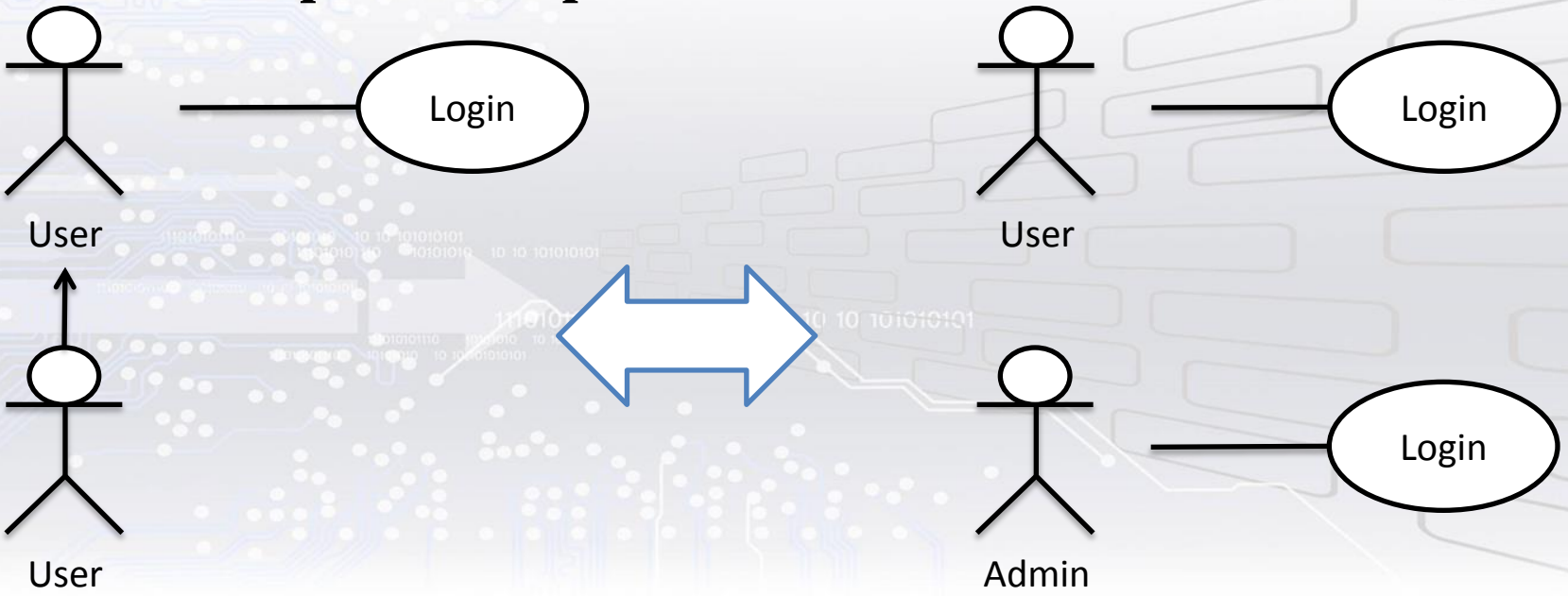
Use case modeling

Actor hierarchy



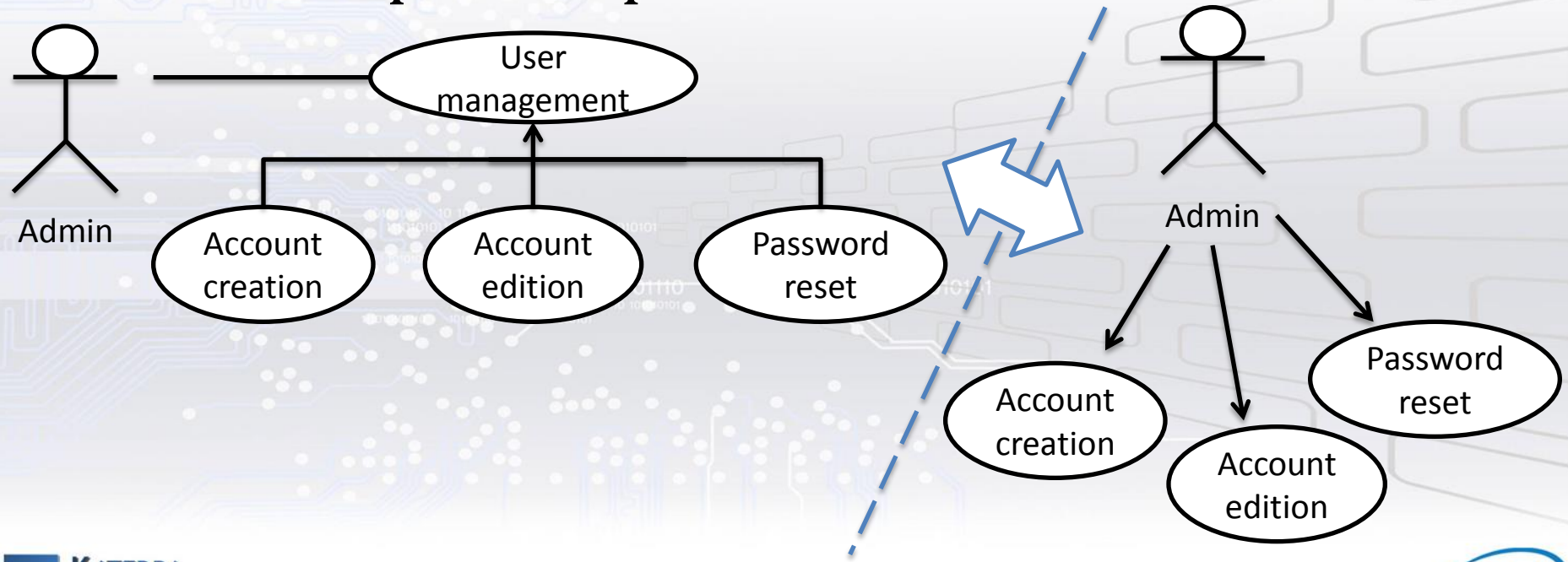
Use case modeling

Relationships example



Use case modeling

Relationships example



Use case modeling

Use case kinds

1. Overview / detailed
- only generic use cases
- specific use cases with interactions

2. Essential / real
- used in analysis
- used in design

Use case modeling

Scenario

Use Case modeling requires an analysis of possible scenarios of events. During such an analysis the following should be taken into account:

- main event sequence
- sub-activities
- exceptional situations

Use case modeling

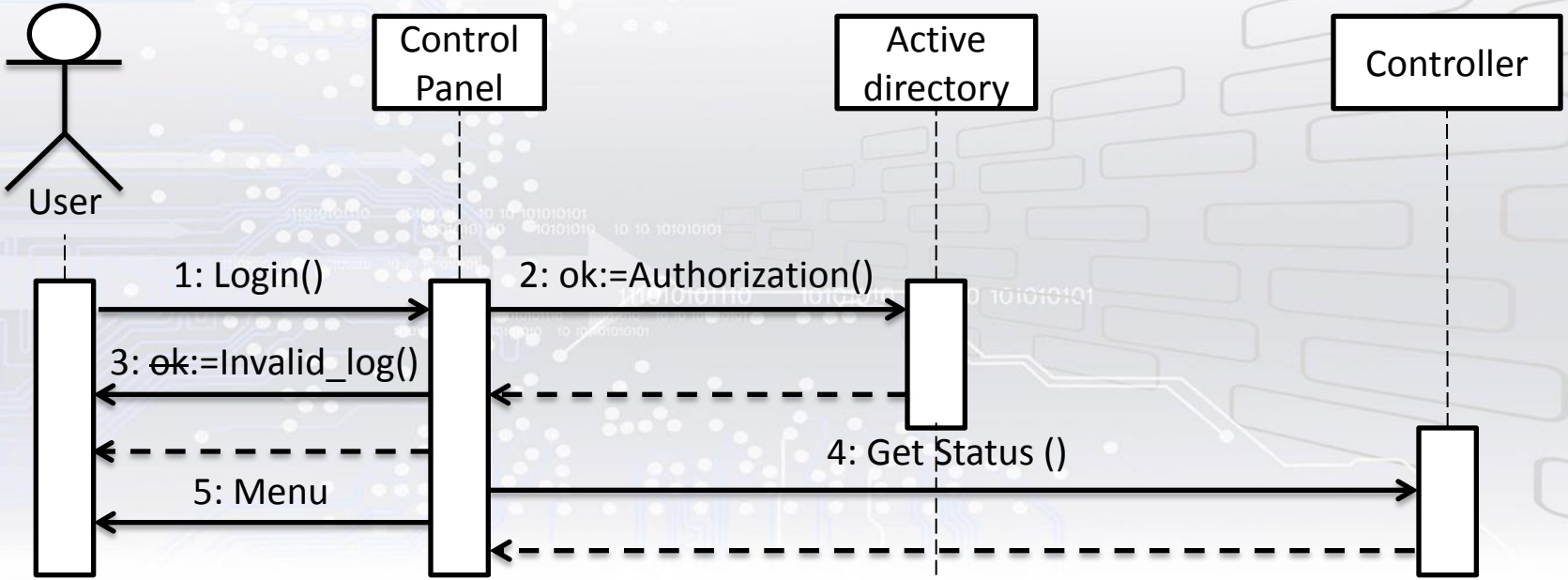
Example scenario

„Order placement”

1. Customer authentication
2. Order negotiation
3. Order authorization

Use case modeling

Scenario – interaction diagram



Use case modeling

Use case model usage:

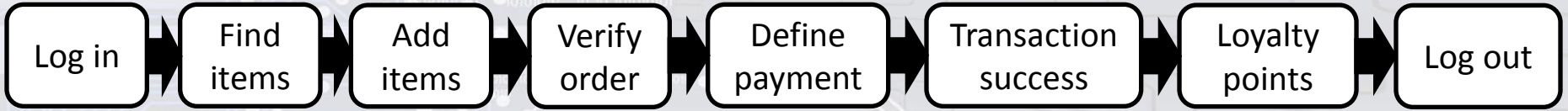
- application logic design
- user interface design
- testing

Use case modeling

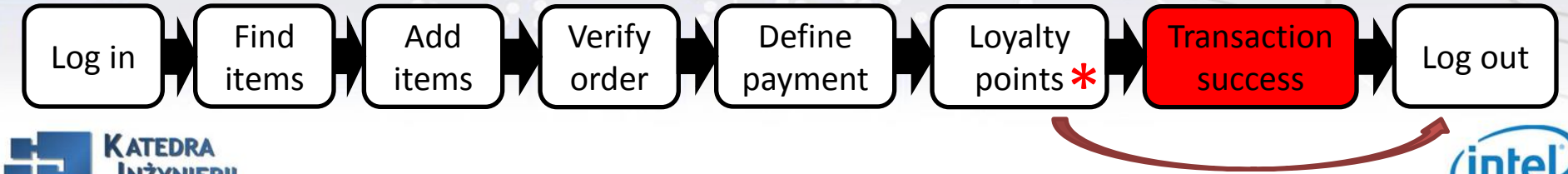
Application logic design – genesis

LOGIC

A business process example – ordering



A broken business process example



Application logic design

A broken business process example:

- Early loyalty points without paying (free items)
- Manipulation of business process
- Theft, fraud, financial loss

Logic Defect:

A defect that exposes a business process to manipulation from attacker who causes undesirable results of process sequences without disrupting application continuity.

Application logic design

Application logic (AL)

„Design components of a system with a steps procedure of executing business processes in a piece of software”.

1. Class model completion and refinement
2. Fitting AL to system architecture
3. Joining classes, interfaces and components
4. Component organization

Application logic design

Ad. 1. Class model completion

- Container class addition
  - entity management (adding, ordering, remove)
  - cross-entity data validation
  - aggregation
- Class hierarchy complement (find missing classes)
  - generalization
  - specialization

Application logic design

Ad. 1. Class model refinement

- Entity classes
- Entity identification
  - int or GUID
- Properties or fields (attributes)
  - read-only or read-write?
  - write-only? write-once?
  - const?
  - derived properties – expressions
- Feature visibility
  - private, protected, public
  - internal? friend?
  - class, static

Application logic design

Properties refinement:

- General types (integer, float)
- Specific types (int32/int64, single/double)
- Fixed-point real numbers (currency, decimal)
- Date and time?
- Collections(multi-value properties)
  - multiplicity: [*], [n..*]
  - capacity – does not guarantee minimal item count
  - list, dictionary (indexed access)
  - vector list or linked list(access / modification efficiency)

Application logic design

Operations refinement

- Operations vs. functions (<<realize>> relationship)
- Function parameters
  - in, out, inout
  - default values
  - overloaded functions
- Result types
  - two or more result values – out parameters needed
- Operation kind: regular, virtual, abstract, (class, static), (new, override)

Application logic design

Application logic considering system architecture

- Single application logic model?
- Dividing logic to layers
- Complexity management (packages)

Application logic design

Interfaces (as data type, not user interface)

Multiple interfaces and multiple inheritance

- multiple inheritance available only in few languages (e.g. C++)
- multiple class parents implemented as interfaces
- each interface must be implemented in each class separately
- implementation via delegation

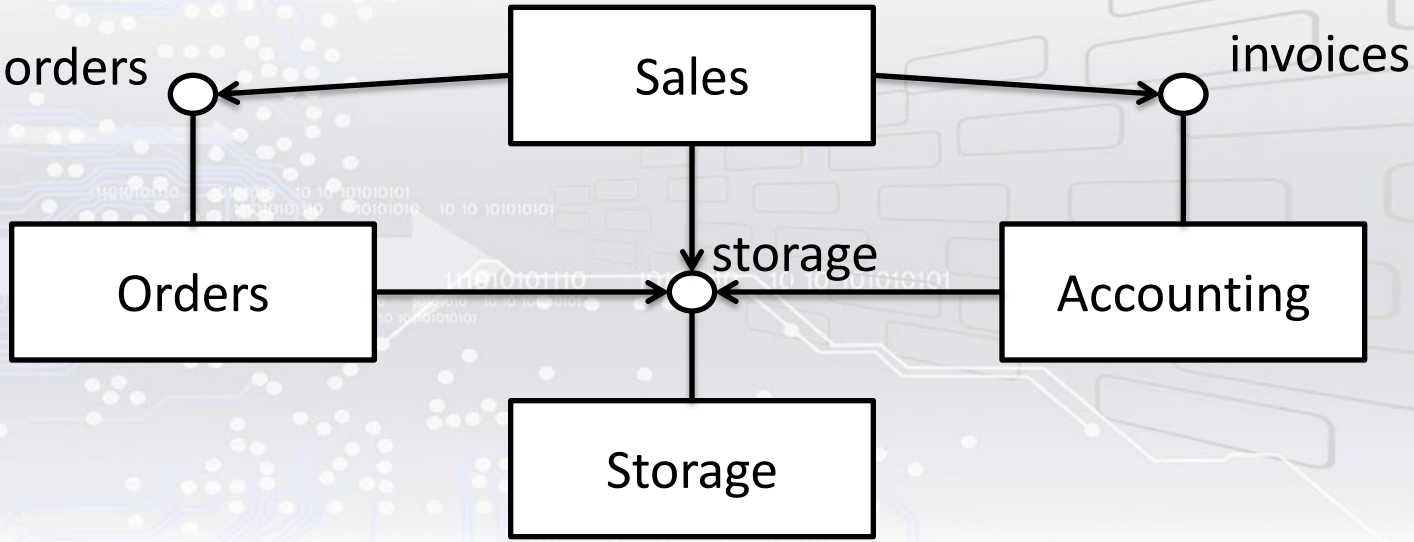
Application logic design

Classes – interfaces – components

- Classes at the lower infrastructure layer
- Components at the higher layer
- Implemented component = library (DLL)
- Components joined through interfaces
- Interface implementation in a class
- At least one implementing class in a component

Application logic design

Component diagram



Application logic design

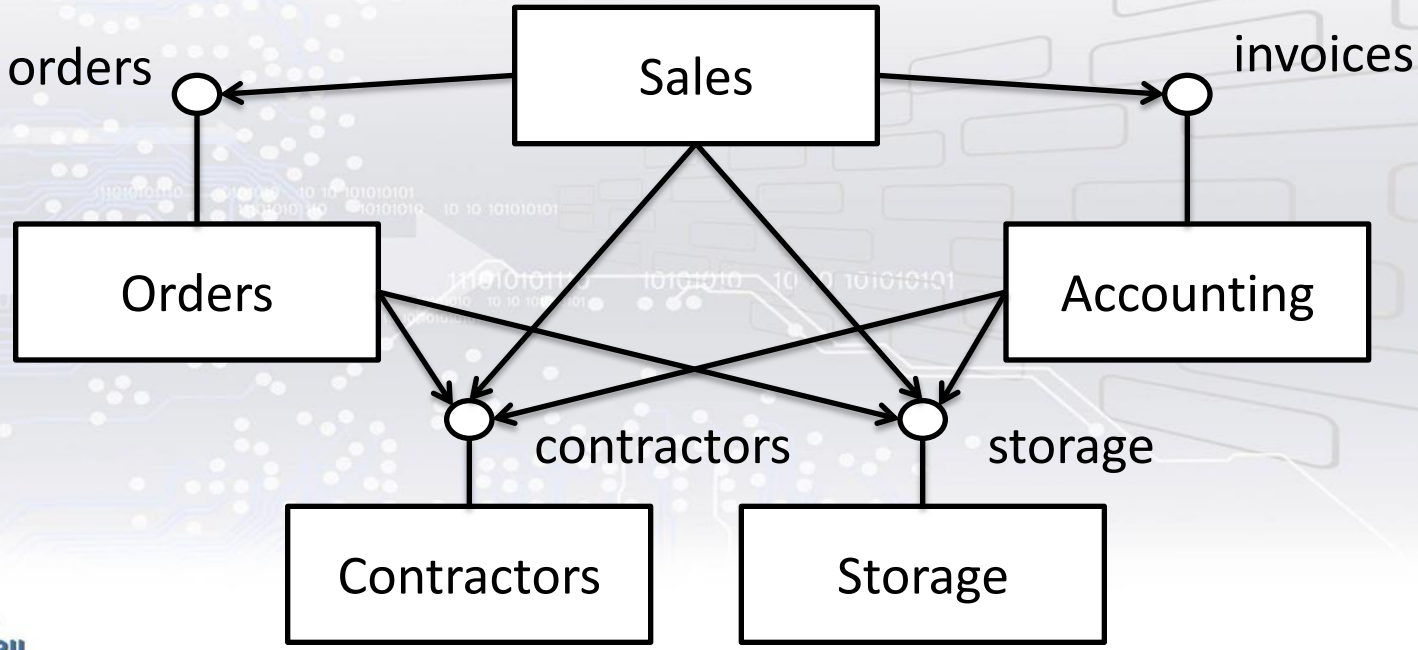
Assigning classes to components

Component “Orders”

- class “Order Manager”
  - implement interface “Orders”
- class “Order”
- class “Article” -> to component “Storage”
- class “Customer” -> to component “Contractors”
  - a new component is needed!

Application logic design

Better component diagram



Use case modeling

Use case model usage:

- application logic design
- user interface design
  - Layout
  - Window navigation diagram
  - Aesthetics
  - Consistency
  - Content
- testing
  - Methods
  - V-model
  - Maintenance

Bibliography

[1] http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-wdrozenia%2c1%2c20.html

[2] Jarosław Kuchta, *Use case modeling*, Embedded Systems Software Engineering, 2015

[3] Grady Boch, James Rumbaugh, Ivar Jacobson, *Unified Modeling User Guide* (book, pdf)

[4] http://www.slideshare.net/RafalLos/defying-logic-business-logic-testing-with-automation

[5] Jarosław Kuchta, *Application logic design*, Embedded Systems Software Engineering, 2015

[6] Roger S. Pressman: *Software Engineering. A Practitioner's Approach* (book, PDF)

[7] Dennis A., Wixom B.H., TegardenD., *Systems Analysis & Design. An Object-Oriented Approach with UML*, John Wiley and Sons, USA, 2002