Working with objects:
Designing Distributed Systems for Reuse

Trygve Reenskaug, Odd Arild Lehne

OOPSLA '97 tutorial # 26
Atlanta, Georgia, 5 October 1997

TASKON
Work Environments

Taskon, Gaustadalléen 21, N-0371 Oslo 3 Norway. Tel. + (47) 22 95 86 31 Telefax: + (47) 22 60 44 27

1.
Working with objects

Designing Distributed Systems for Reuse

OOPSLA Tutorial # 26

Trygve Reenskaug, Odd Arild Lehne
Taskon, Oslo

odda@taskon.no
trygve@taskon.no

1.1
Motto:

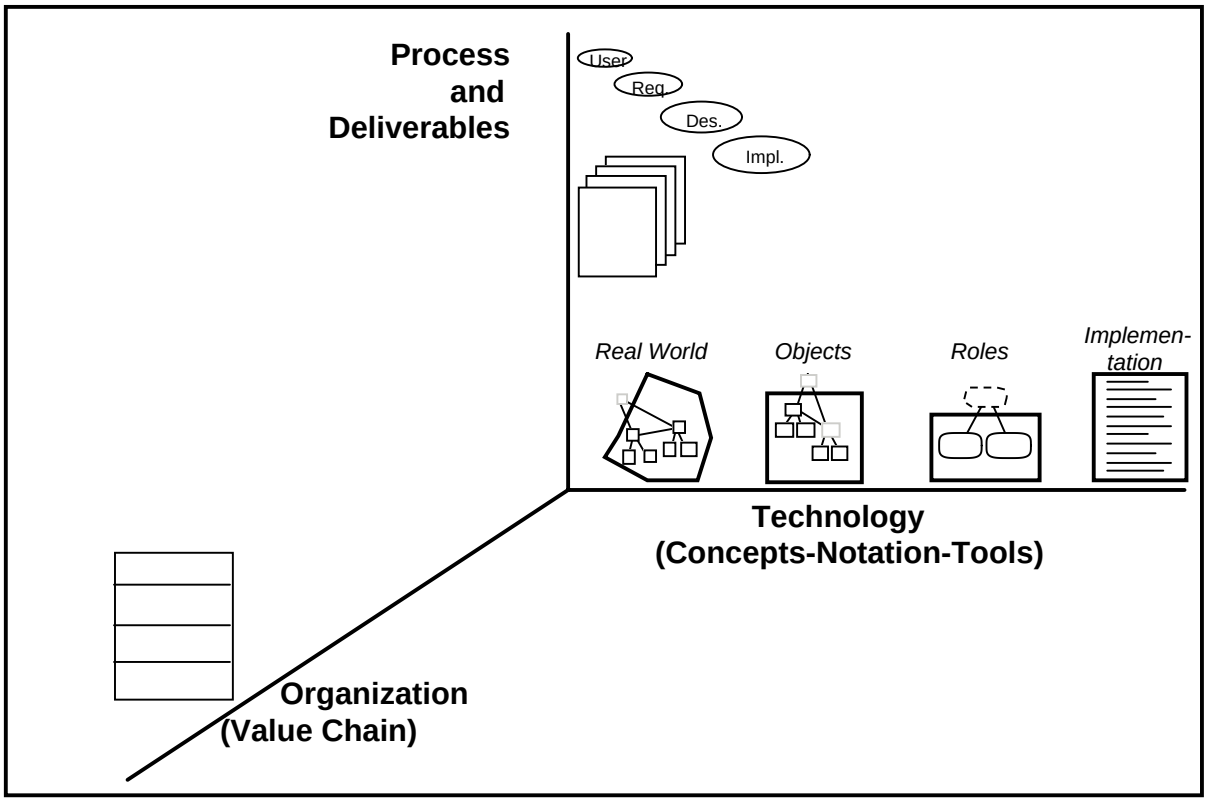
There are two ways of constructing a software design:

- **One way is to make it so simple that there are obviously no deficiencies**

- **and the other way is to make it so complicated that there are no obvious deficiencies.**

-- C. A. R. Hoare

1.2
Three dimensions of software development



1.3
Incidental and planned reuse

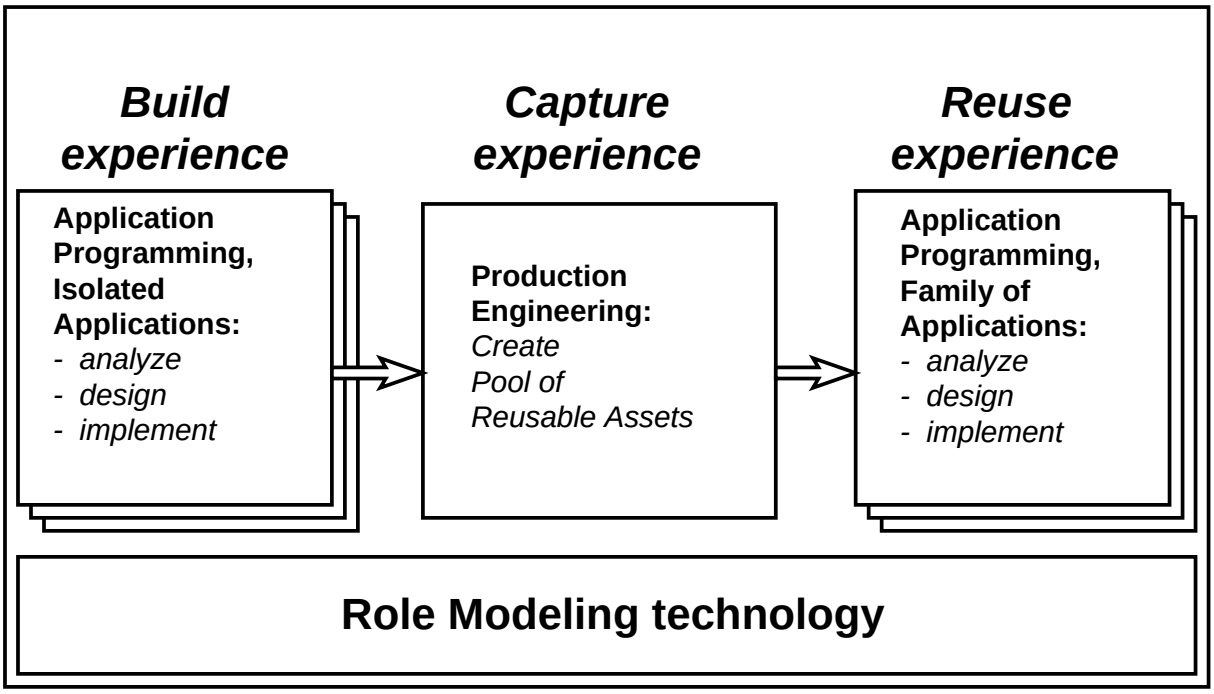
Incidental reuse (ad hoc):

- ***search existing solutions for applicable ideas, models, code, ...***

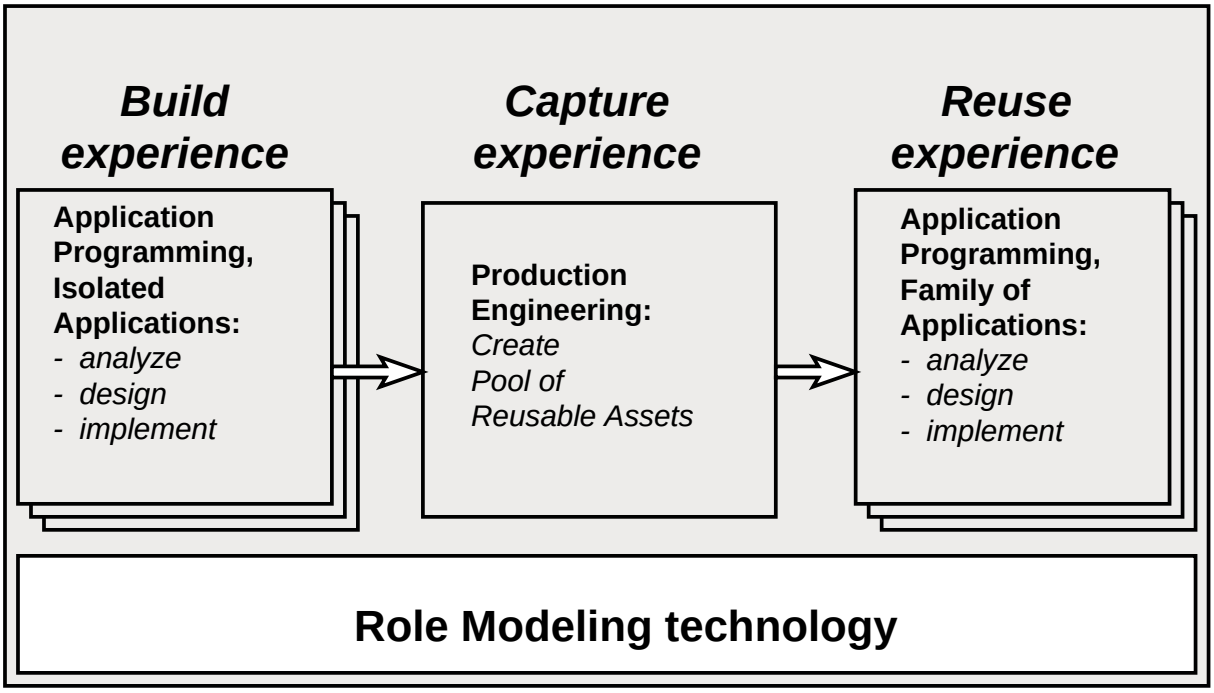
Planned reuse (product development):

- ***Understand the business***
- ***Understand the developers***
- ***Understand applicable solutions***
- ***Create reusable designs and code***

1.4
Reusable component lifecycle



2.
Role Modeling
Focus On Object Collaboration



2.1
Four aspects of OO modeling

Interface modeling
The interface is
a what abstraction
What does this object look like?

Role modeling
The role is
the why abstraction
Why do we have this object?

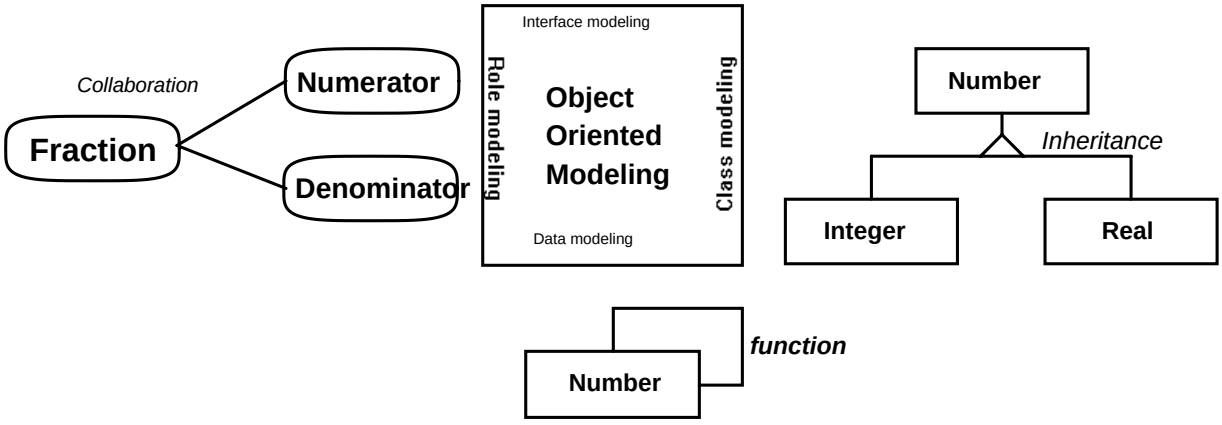
Object Oriented Modeling

Class modeling
The class is
the how abstraction
How is this object implemented?

Data modeling
The entity is
another what abstraction
What does it mean?

2.1.1
Simple examples

```
interface Number {
    Number add (Number num);
    Number subtract (Number num);
    Number multiply (Number num);
    Number divide (Number num);
```



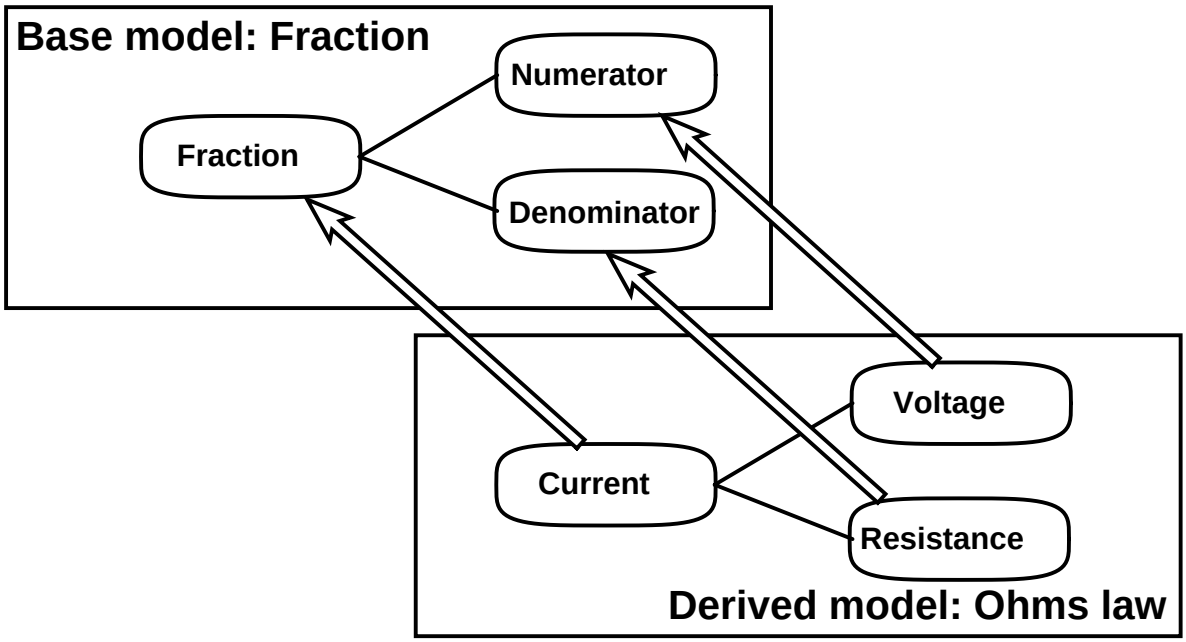
2.2
Main features of role modeling

¤ We only show relevant

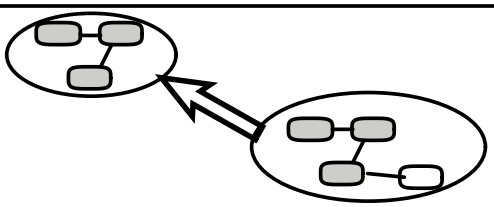
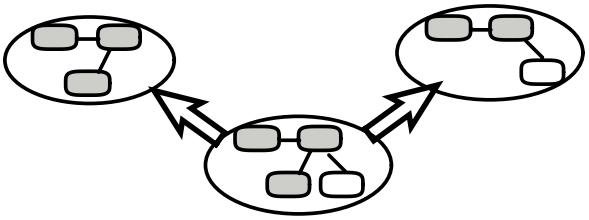
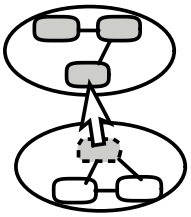
-- objects
-- aspects
-- details

¤ We abstract object identity
to its position in the pattern

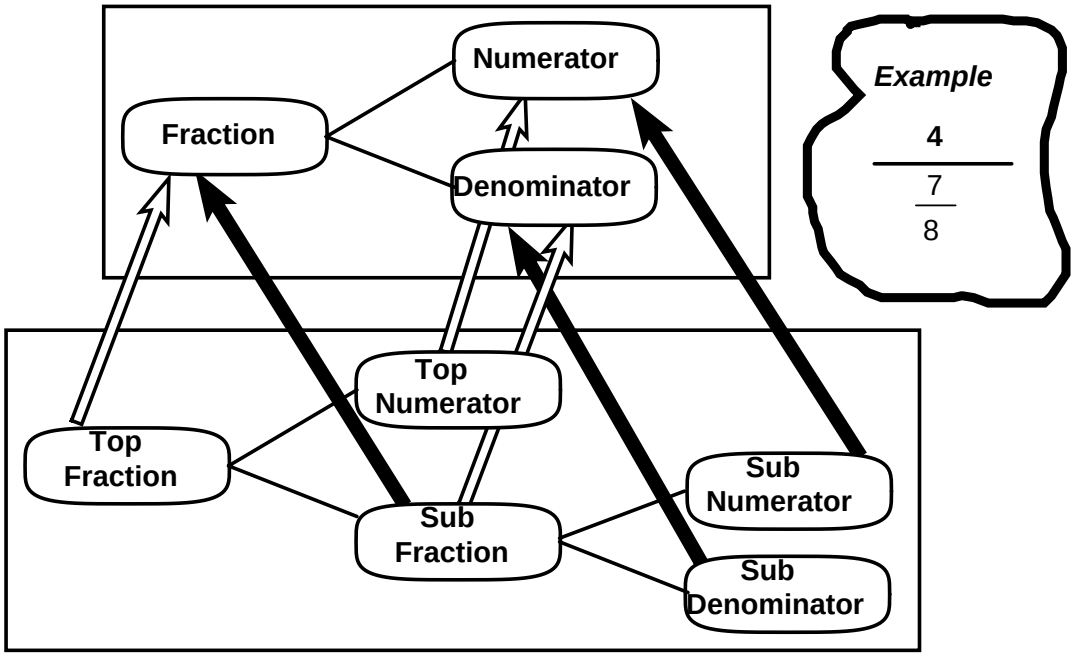
2.2.1
Model inheritance: *Specialization*



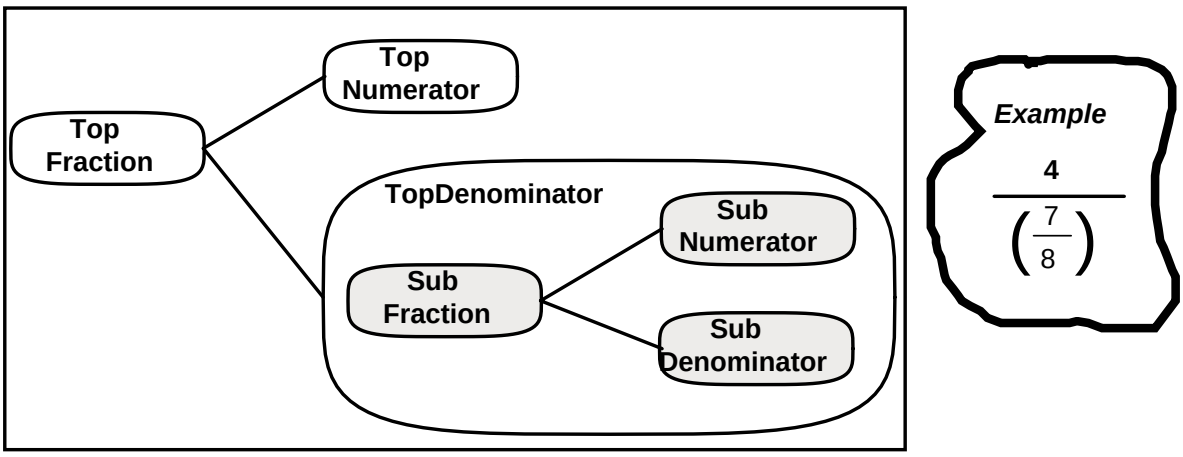
2.2.2
Separation of concern:
Three uses of synthesis

1. **Specialization** **-- generalization**	
2. **Composition on** **same level of abstraction**	
3. **Aggregation**	

2.2.3
Composition on same level of abstraction



2.2.4
Aggregation



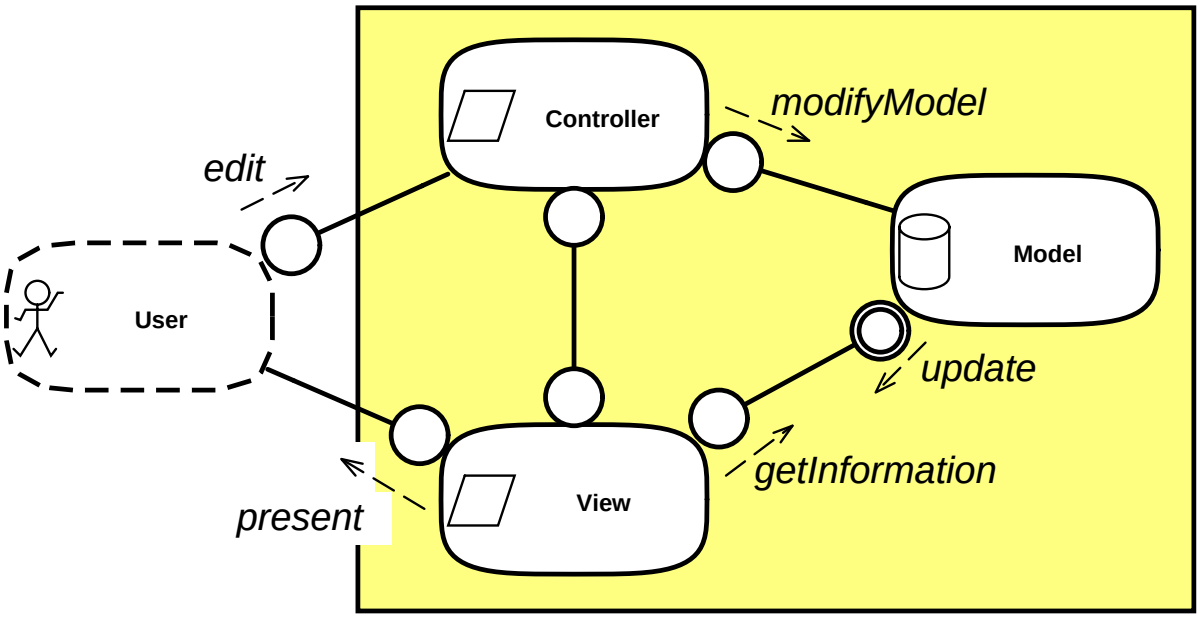
2.2.5
Important observations:

* ***A model is created for a purpose.***
* ***A model is never complete.***
* ***We think in multiple models, always trying to choose the best model for our purpose.***
* ***We tend to think in hierarchical models, even though the world is rarely hierarchical.***

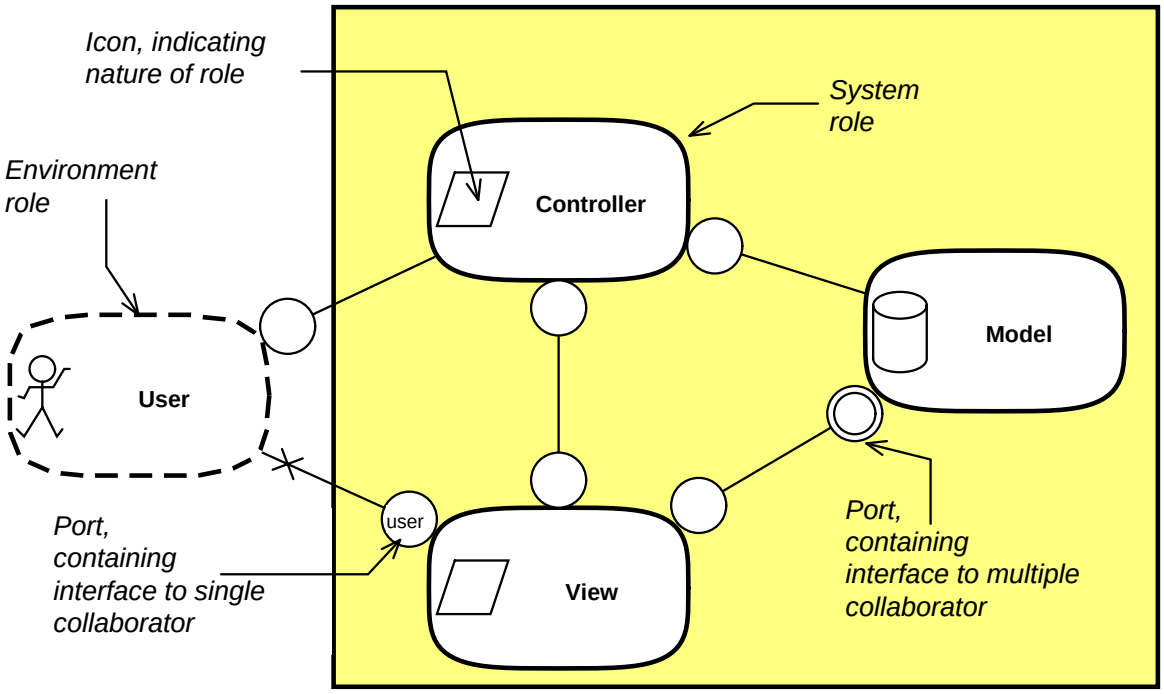
2.3
Role Model Advantages

- **Role models describe object patterns**
- **Modeling distribution**
- **Separation of Concern**
- **Reuse through model inheritance**
- **Seamless bridge to implementation**

2.4
Role models describe object patterns



2.4.1
Role Model Collaboration view notation



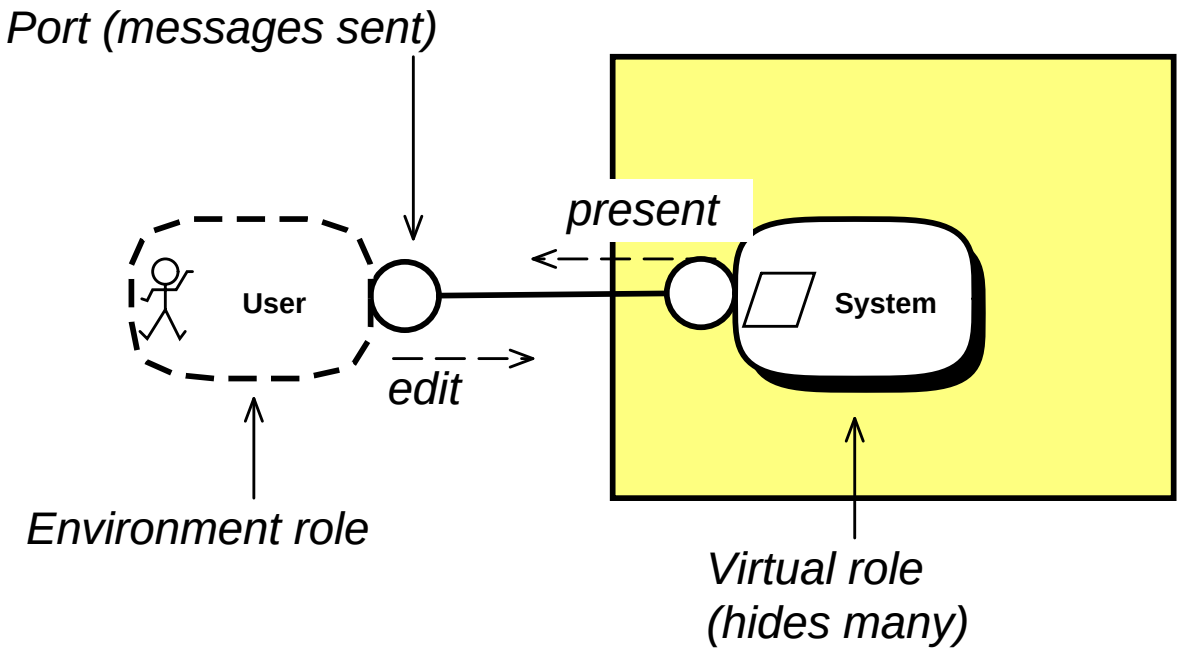
2.4.2
Compare with class hierarchy
(VisualWorks class library)

Controller
- ControllerWithMenu
- - ListController
- StandardSystemController
. . .

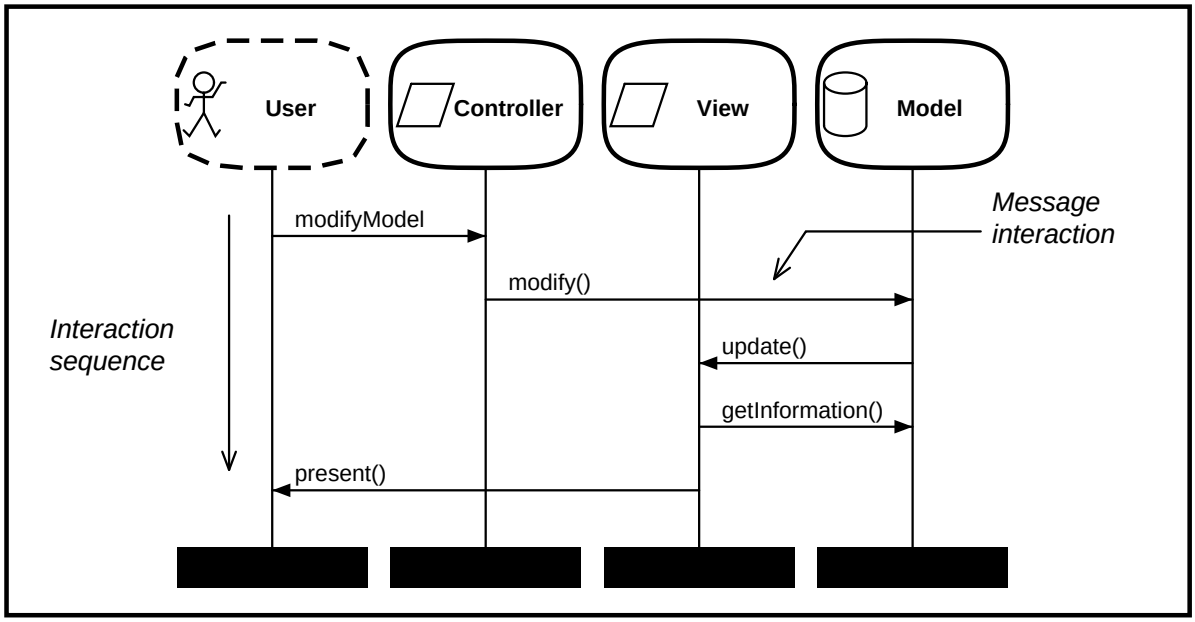
View
- AutoScrollingView
- - ListView
- FractionalWidgetView
- LauncherView
- Scrollbar
...

Model
- Browser
- - Debugger
- Inspector
- - ContextInspector
- - DictionaryInspector
- PopUpMenu
. . .

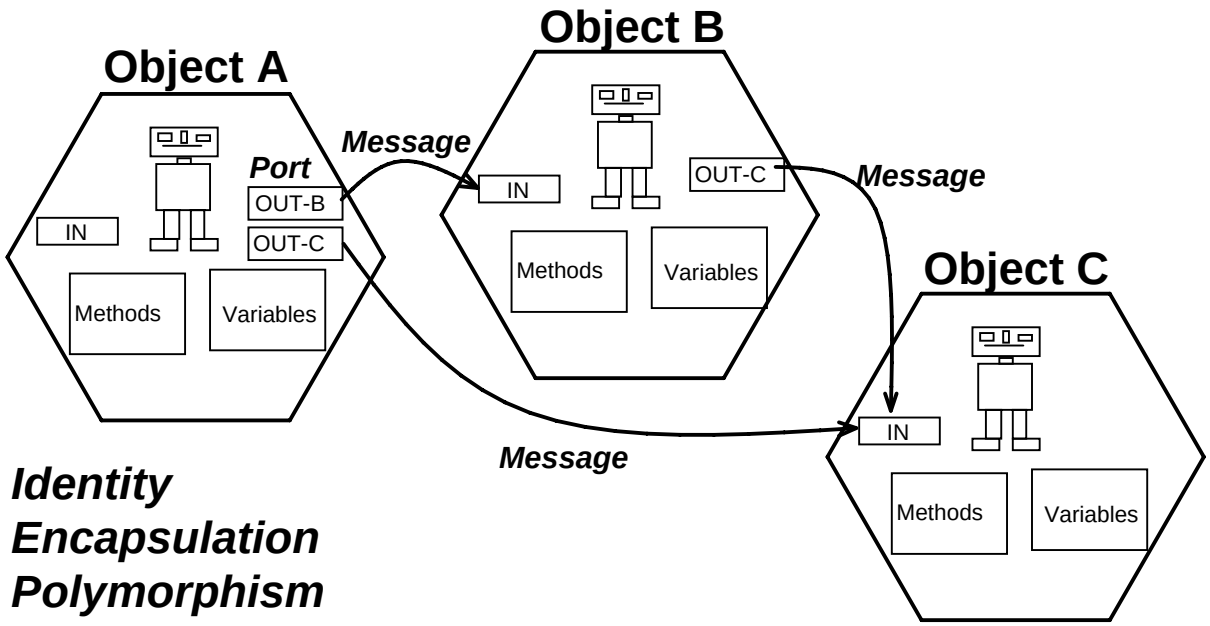
2.4.3
Environment Collaboration view



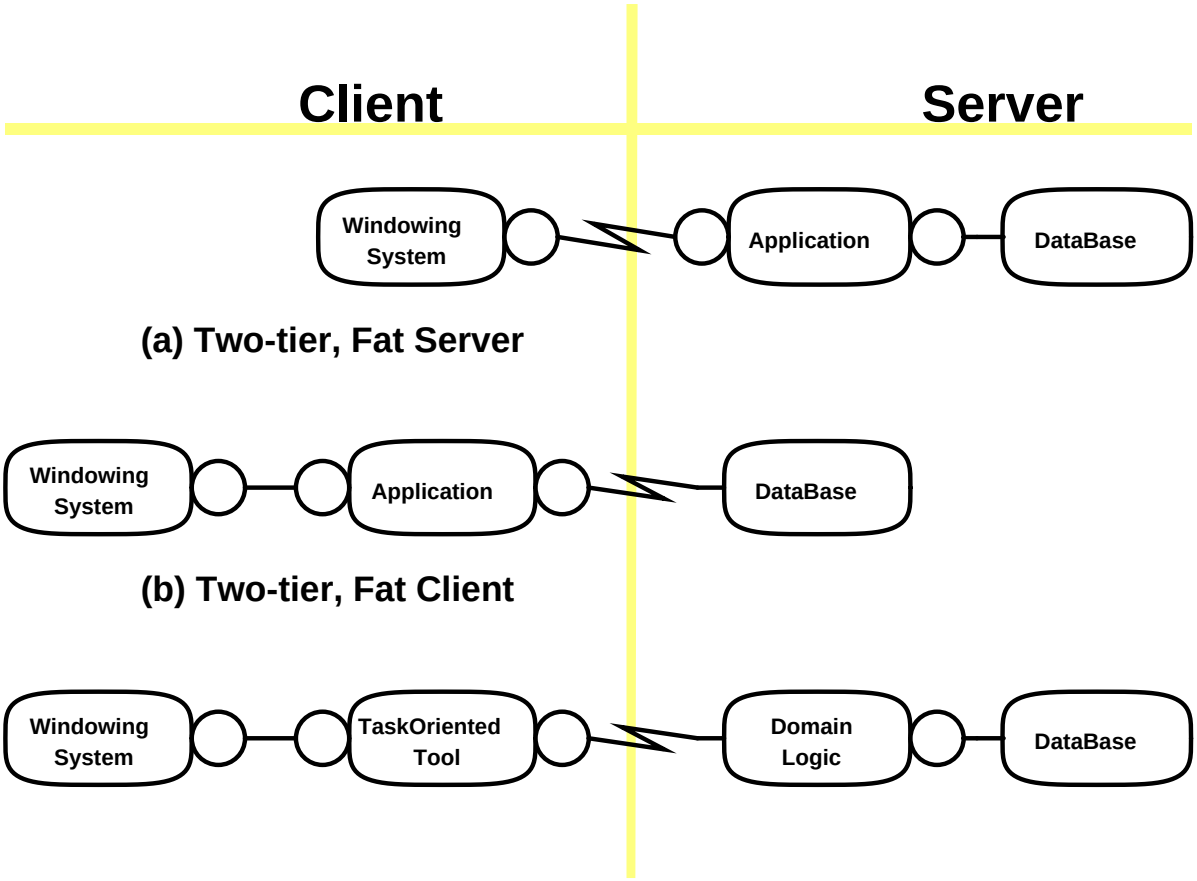
2.4.4
Scenario view



2.4.5
Role modeling
focuses on object patterns



2.5
Modeling distribution



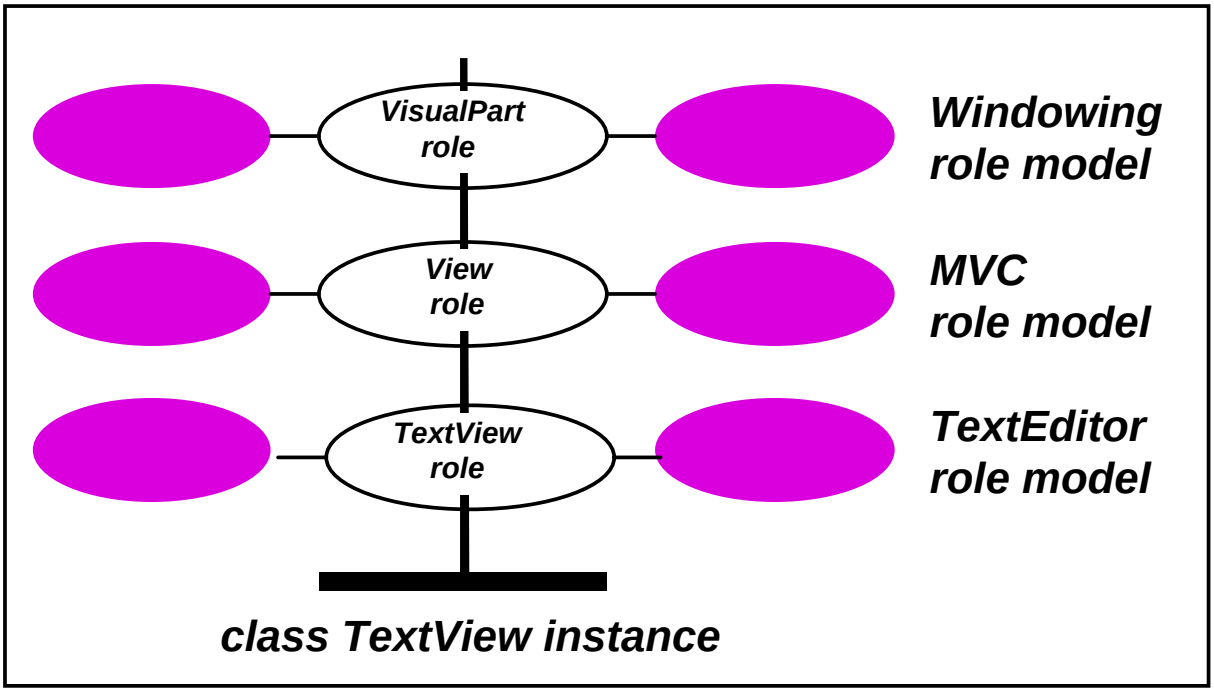
2.5.1
Determine distribution architecture

Iterate

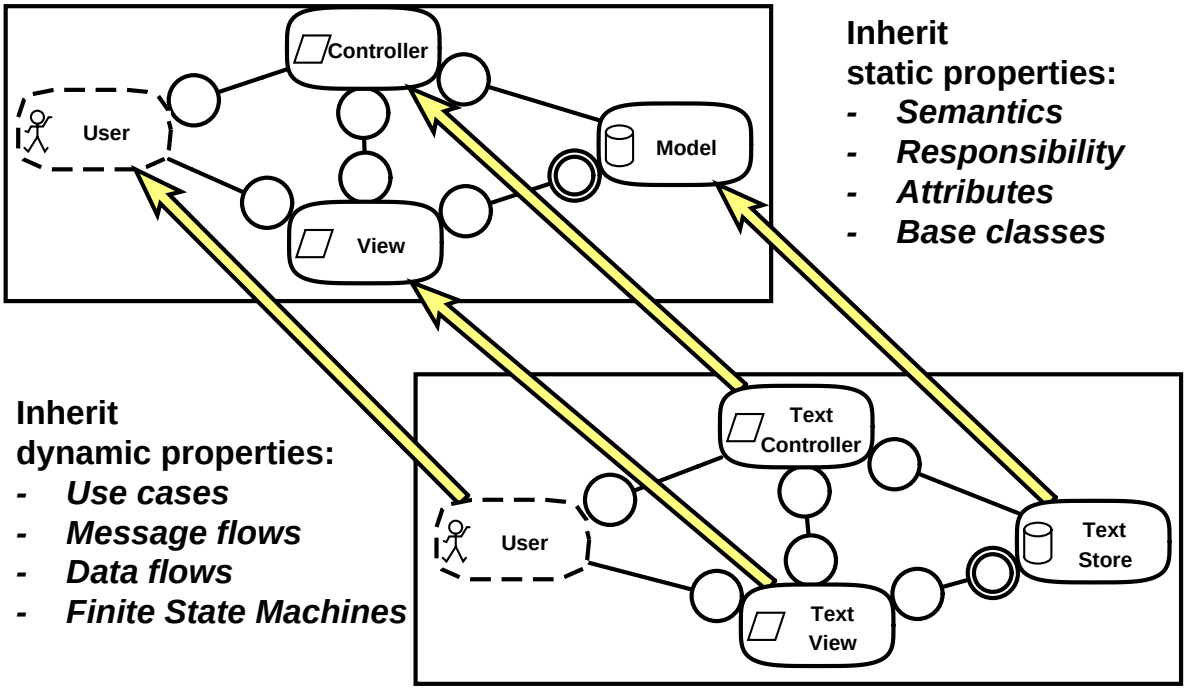
- **Select possible communication architecture**
  - ¤ Evaluate communication requirements
  - ¤ Choose communication method
    - e.g., on demand
    - e.g., caching
  - ¤ Estimate communication costs (time, money)

- **Implement solution**
  - ¤ Subclass frameworks
  - ¤ Specialize code
  - ¤ Optimize code

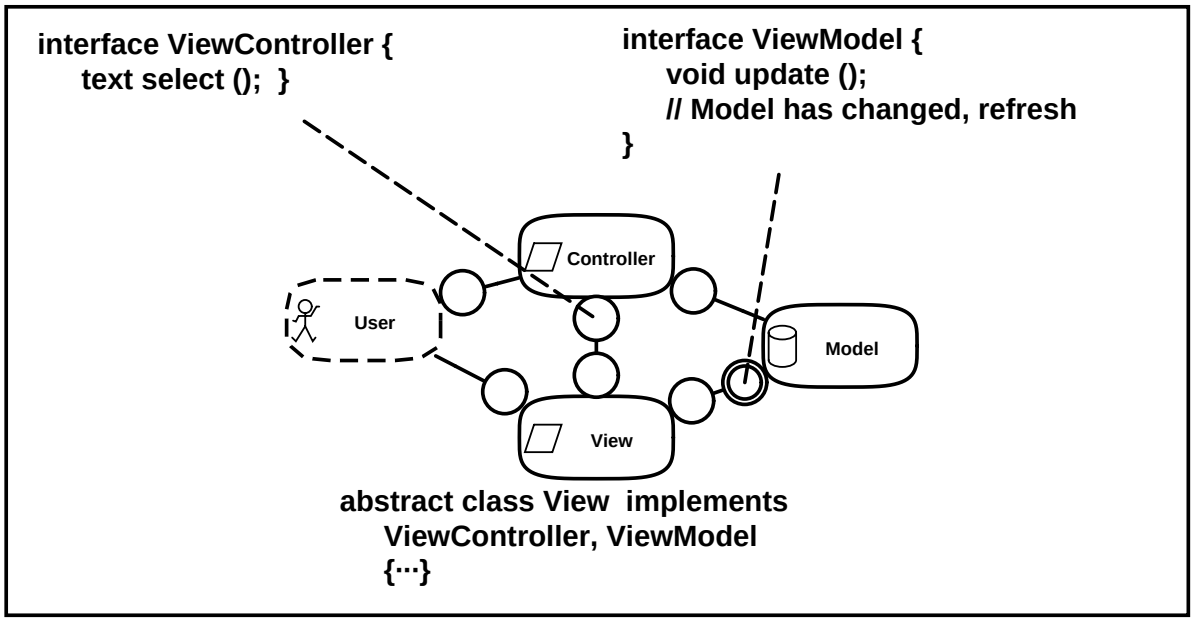
2.6
Separation of Concern
A TextView object plays many roles



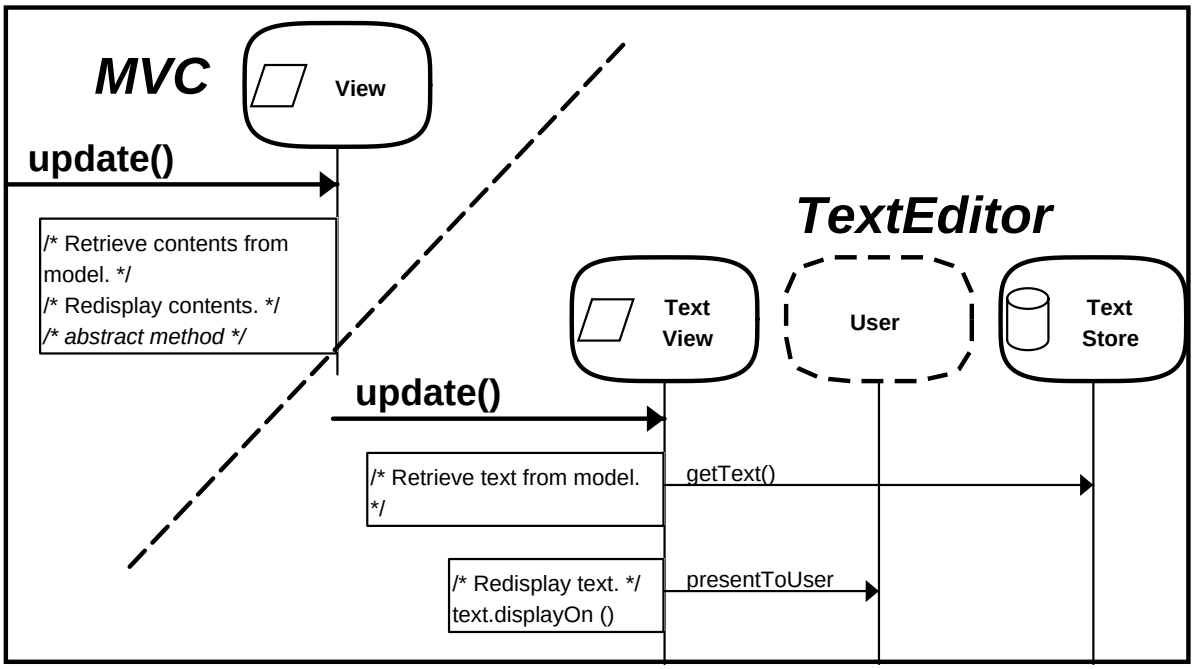
2.7
Reuse through model inheritance



2.8
Seamless bridge to implementation



2.8.1
Method views
Inside object perspective



2.8.2
Mapping concepts from OOram to implementation

Roles
- --> Interfaces
  - --> Classes

Ports
- --> Member variables & other references
  - --> Interfaces

Messages
- --> *Methods*

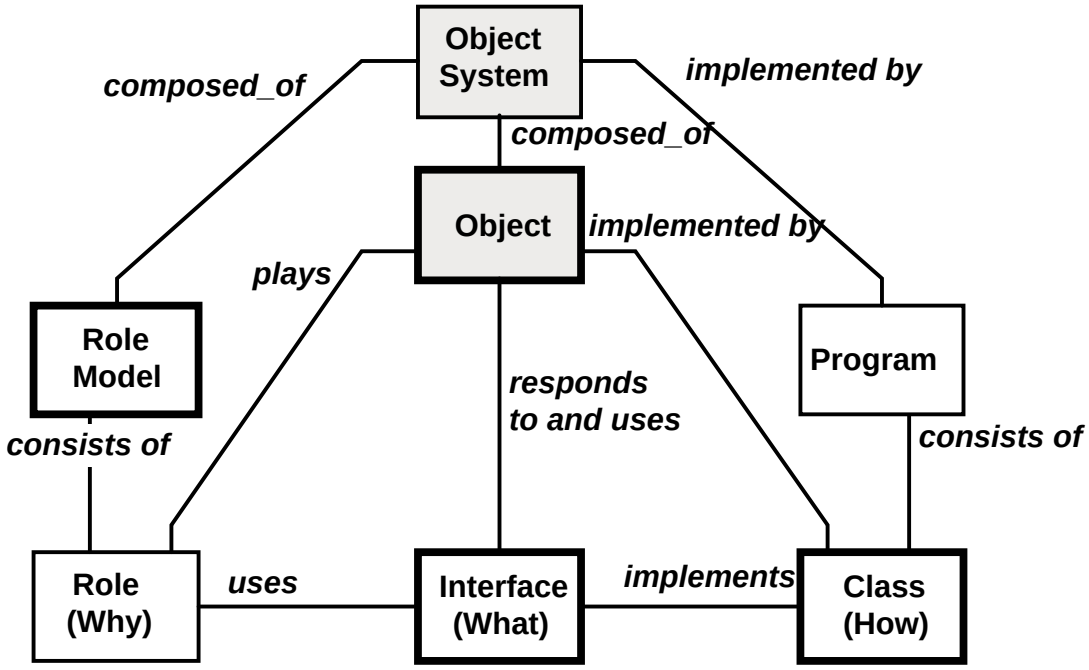
Model inheritance
- --> Class ensemble inheritance

2.8.3
Program code
Class perspective

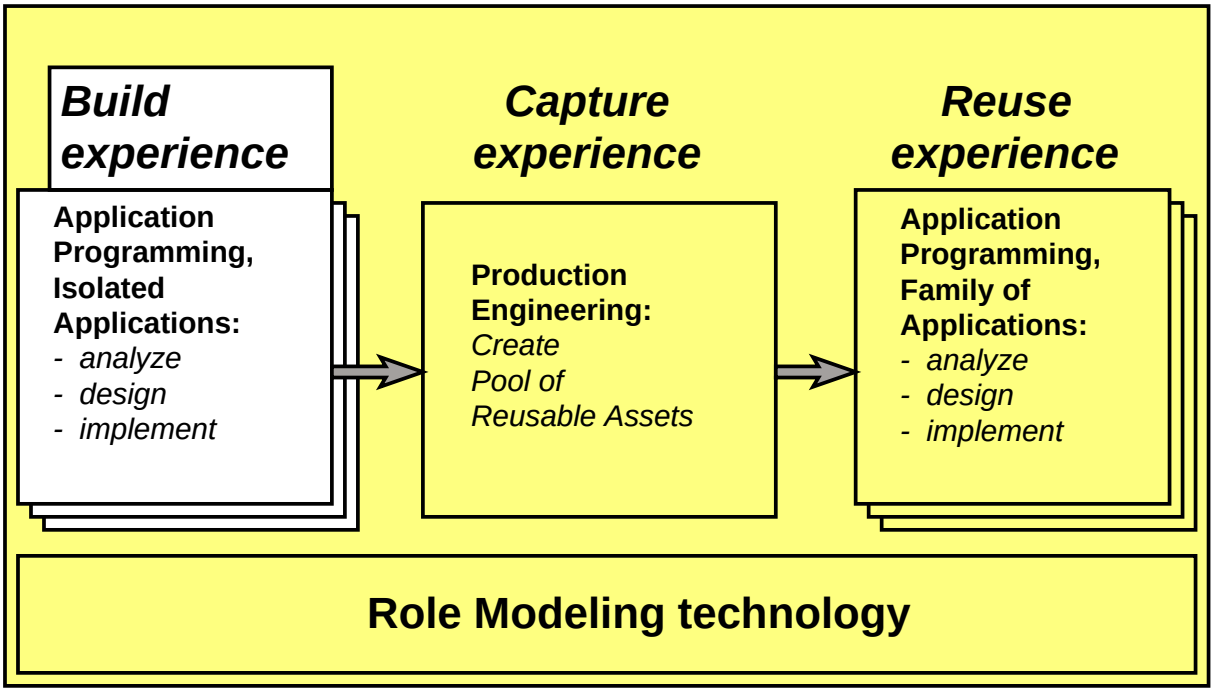
```
abstract class View  implements ViewController, ViewModel {
    Model model;
    ControllerView controller;
    abstract void select (···) ;
    abstract void update (···) ;
    // Model has changed, refresh
    ...
}

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                    class TextView extends View
                        implements TextViewController TextViewText {
                        Text textCache;
                        void update () {
                            //Retrieve text from model.
                            textCache = model.getText();
                            //Redisplay text.
                            this.display();
                        }
                    }
```

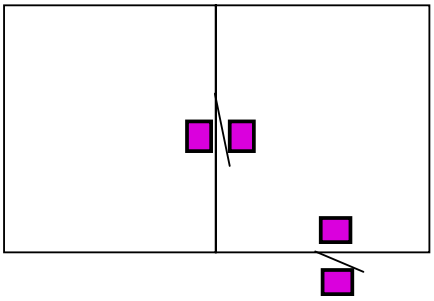
2.8.4
Semantic model - Object System



3.
The first isolated application
Card access control system (CAC)



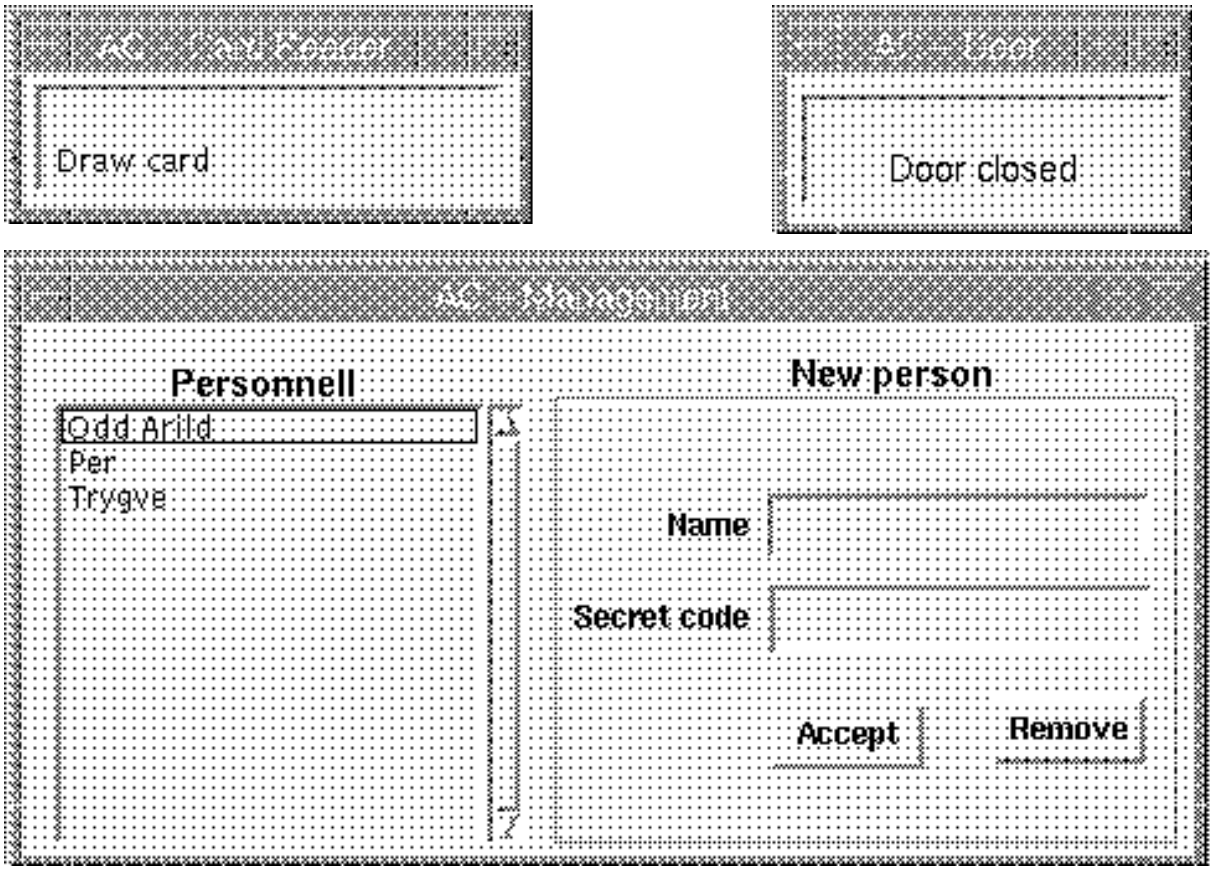
3.1
An Access Control problem
Area of concern



An office complex has a number of office areas separated by automatic doors.

We want to limit the access of different persons through a system of machine-readable identity cards.

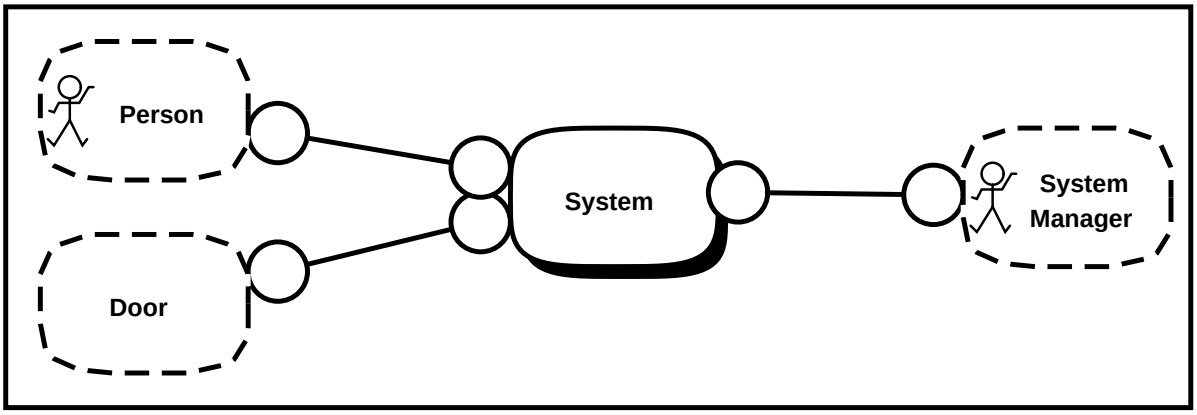
3.2
User interfaces
Prototype Program



3.3
Process Steps

- Define Area Of Concern
- Describe system seen from environment
- Consider system architecture
- Consider separation of concern
  For each sub-system:
  - Describe system seen from environment
  - Describe system roles and collaborations
- Synthesize system from sub-systems

3.3.1
System seen from environment
Environment collaboration view



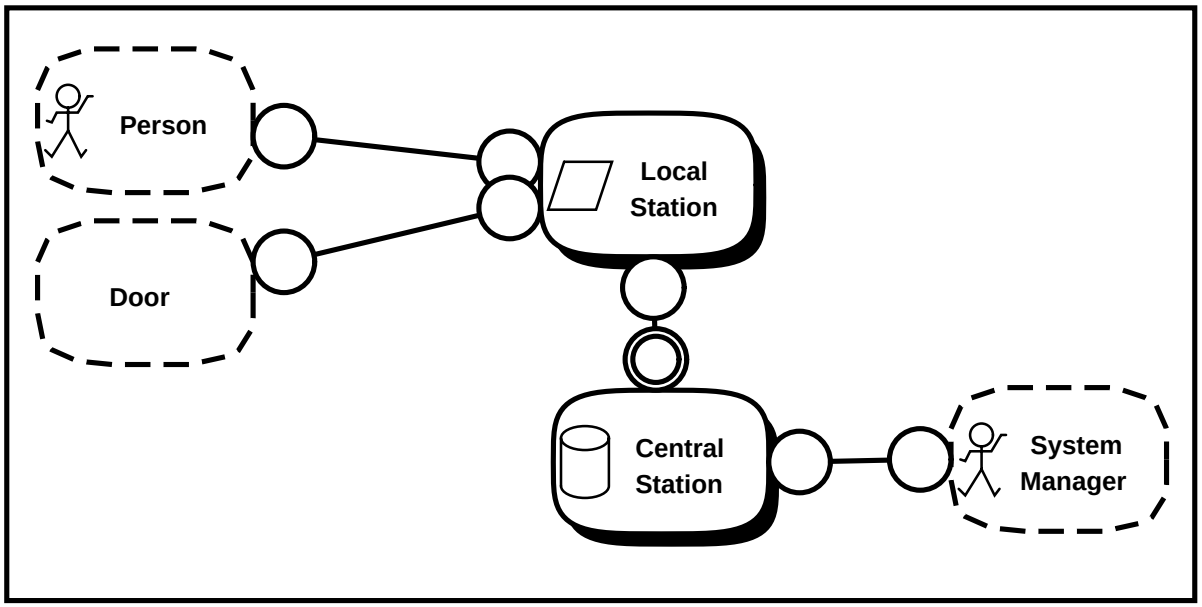
3.3.2
System seen from environment
Stimulus-Response view

Stimulus message	Response	Comment

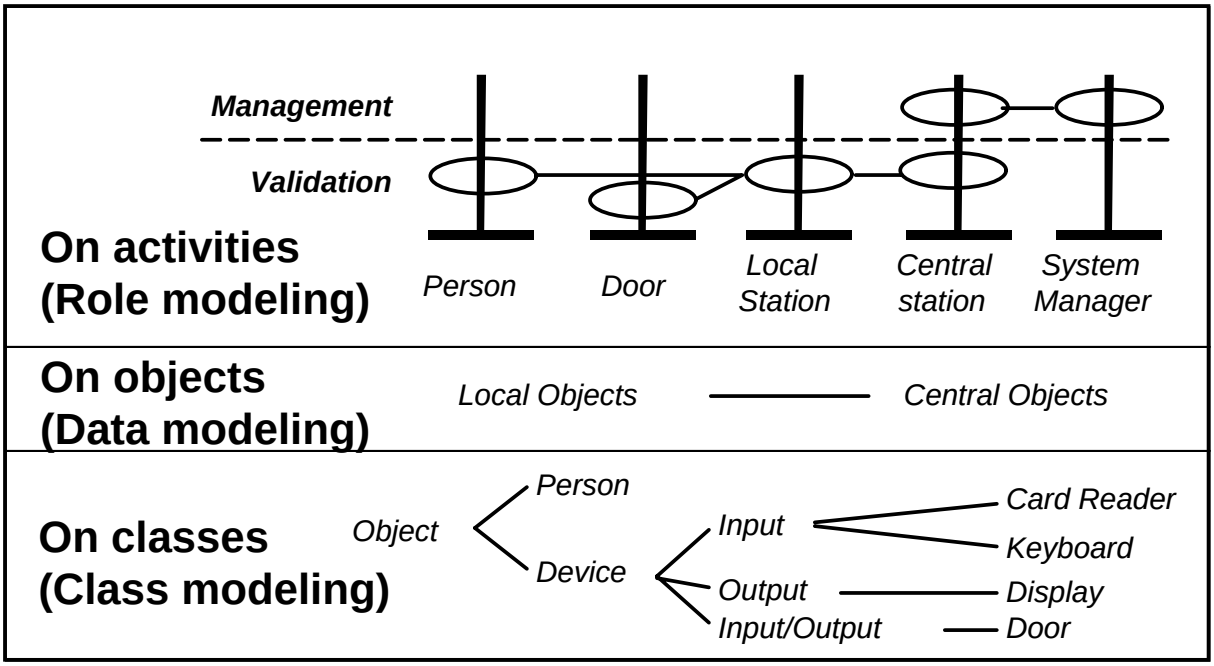
Person.identification (from);	Door.open ();
	// time delay
	Door.close();
		Person accepted

	Person.reject ();	Person rejected
	AlarmHandler .doorOpenAlarm();	Door left open
SystemManager .addPersonWithAccess ();	Access data base updated	

3.3.3
System Architecture
A Choice



3.3.4
Separation of Concern



3.3.5
Consider Separation of concern

Hints for separation of concern:

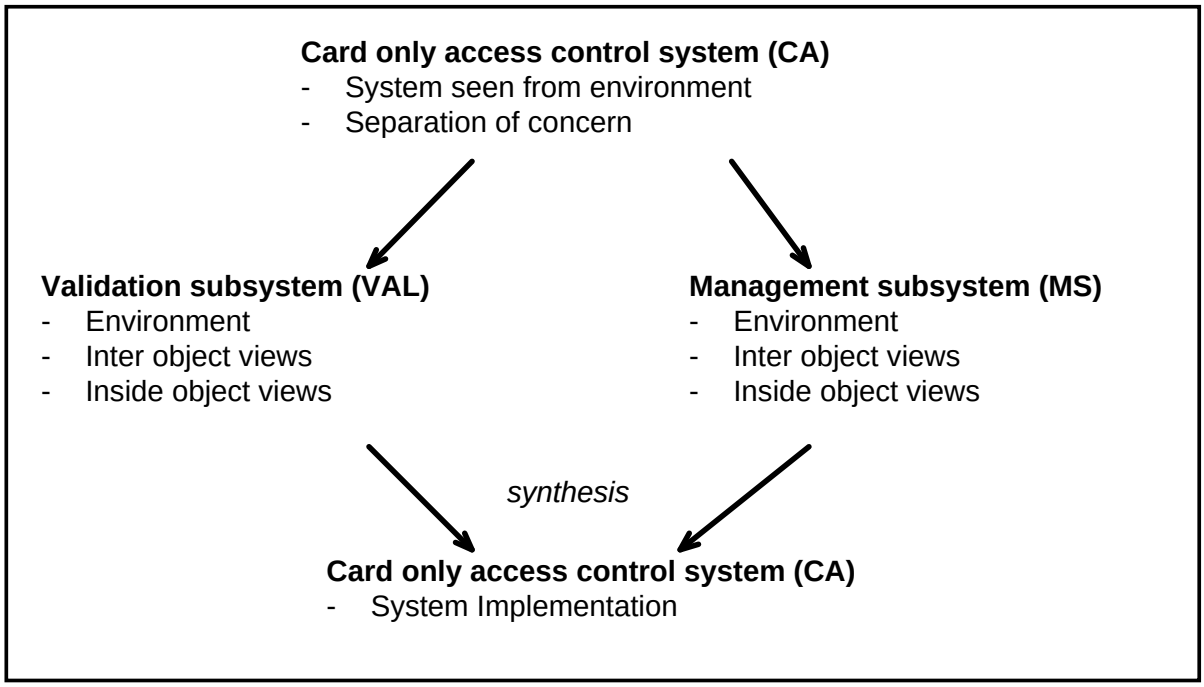
- ***Separate on stimulus objects***
- ***Separate on stimulus messages***

	Stimulus message	Response	Comment

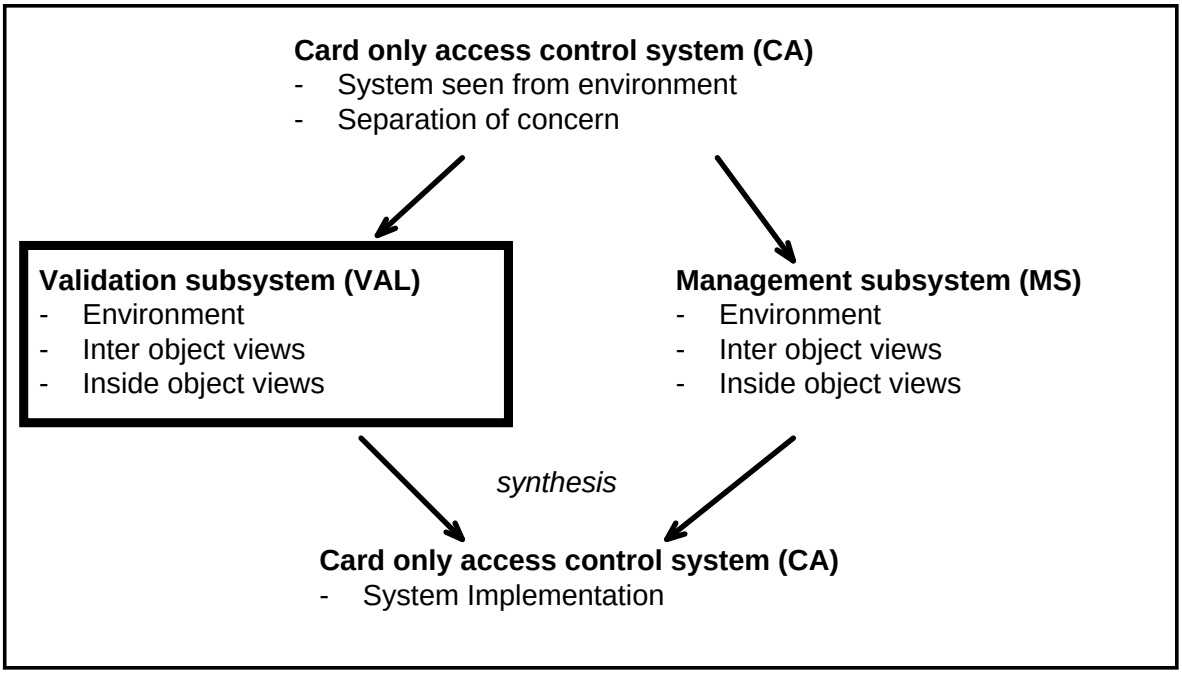
(VAL)	Person.identification (from);	Door.open ();
		// time delay
		Door.close();
			Person accepted

		Person.reject ();	Person rejected
		AlarmHandler .doorOpenAlarm();	Door left open
(MS)	SystemManager .addPersonWithAccess ();	Access data base updated	

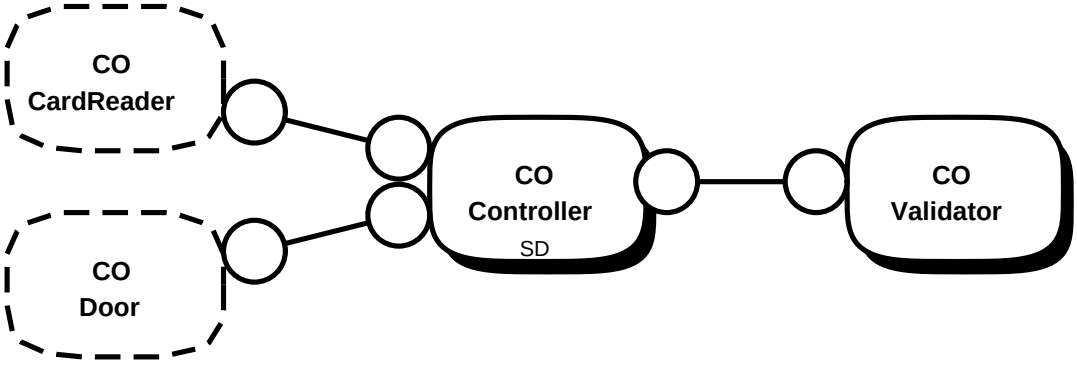
3.3.6
Revised plan for system development



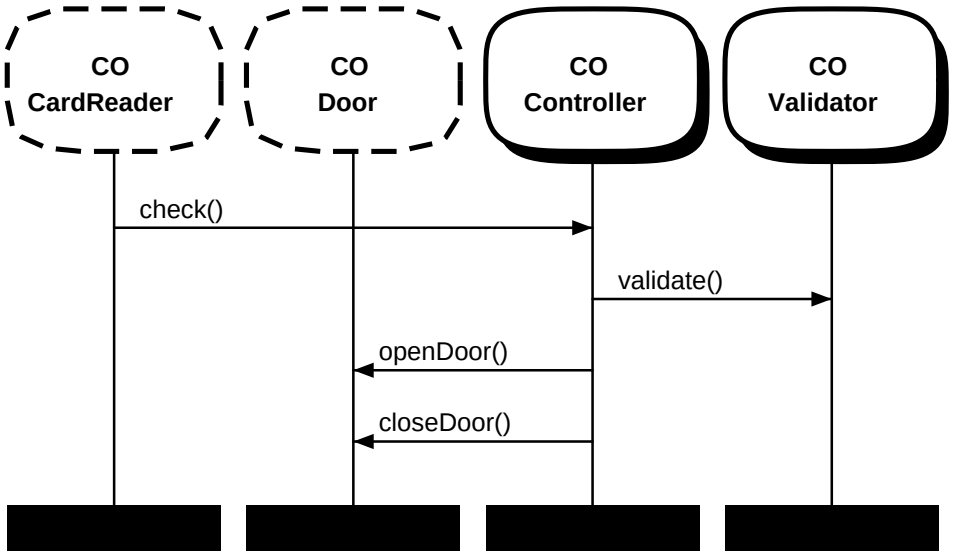
3.4
Validation subsystem (VAL)



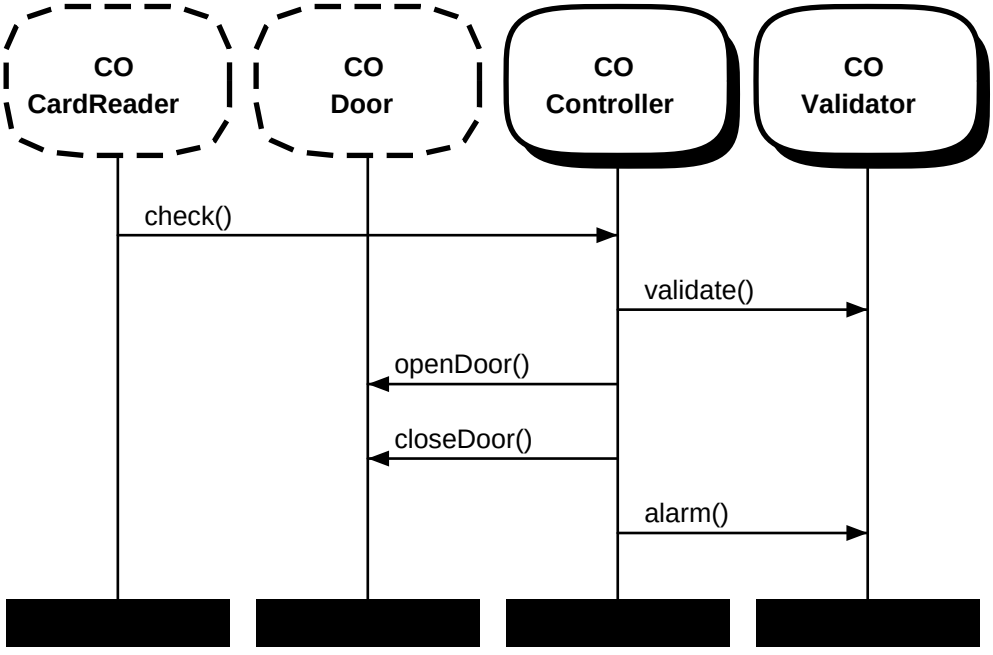
3.4.1
Validation Environment



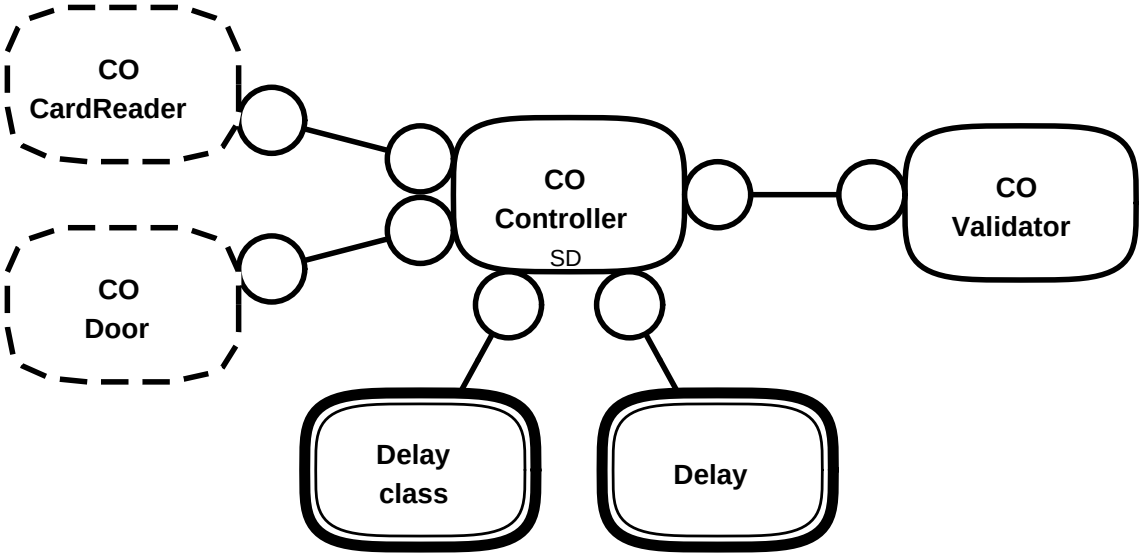
3.4.2
VAL Normal access scenario



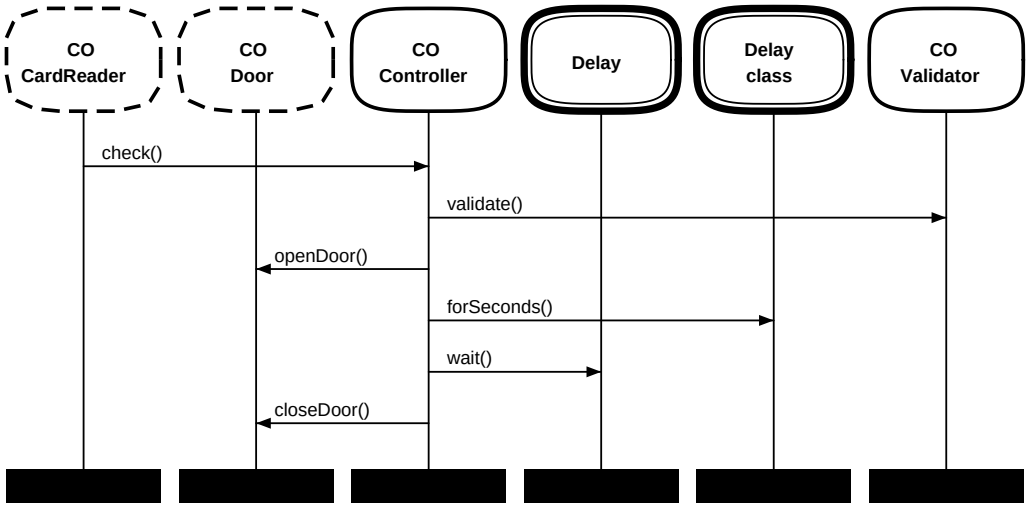
3.4.3
VAL Open door alarm scenario



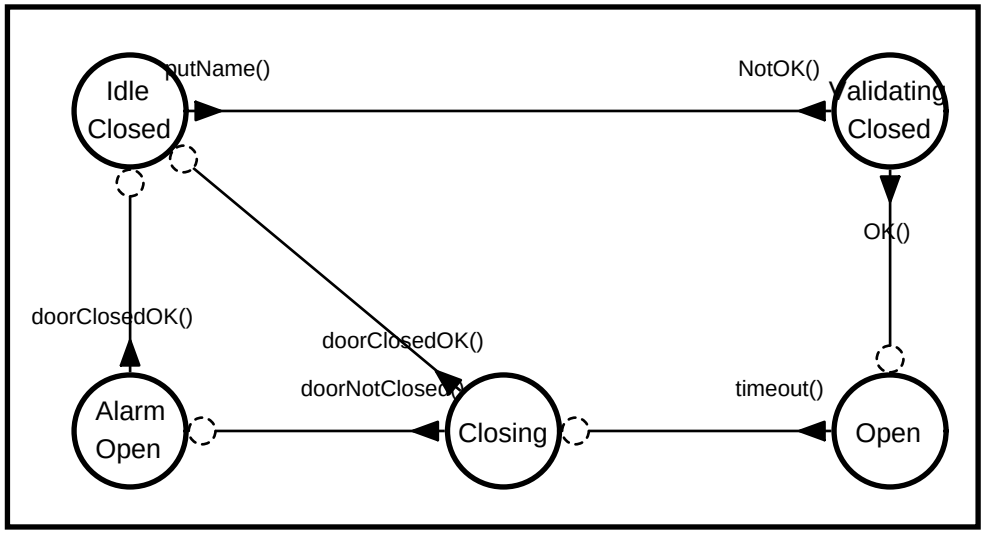
3.4.4
Validation sub-system
Controller details



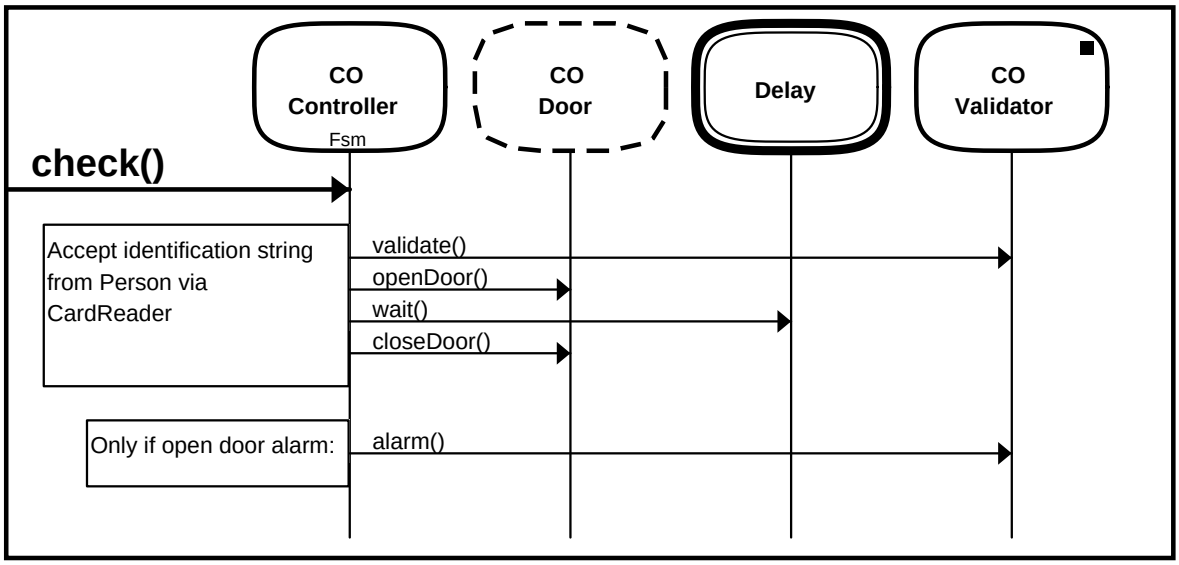
3.4.5
Validation
Normal access scenario
Controller details



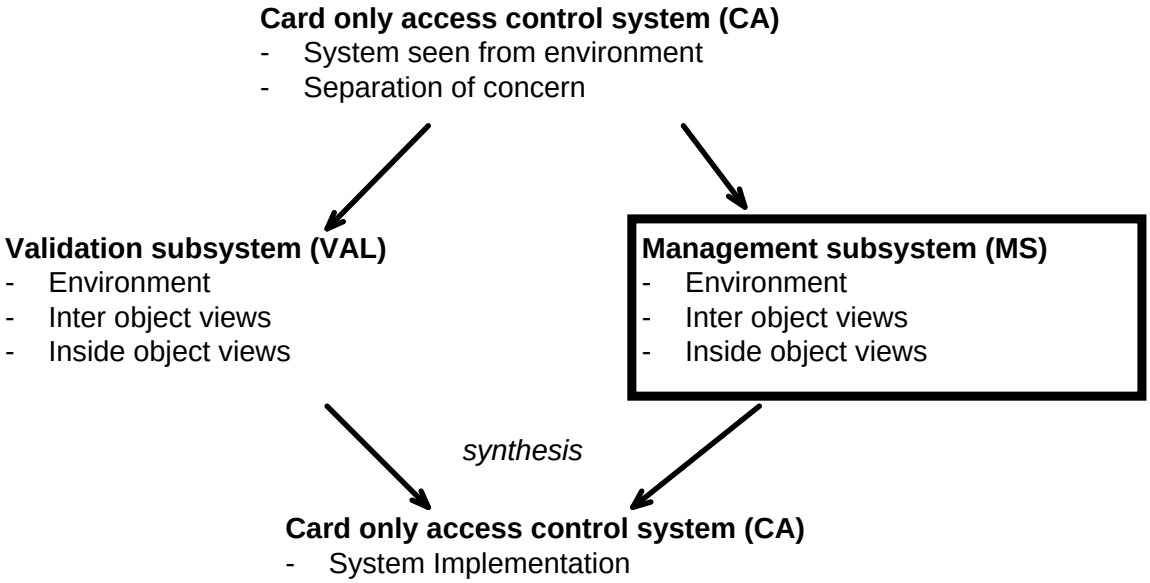
3.4.6
Inside object perspective
Controller state diagram



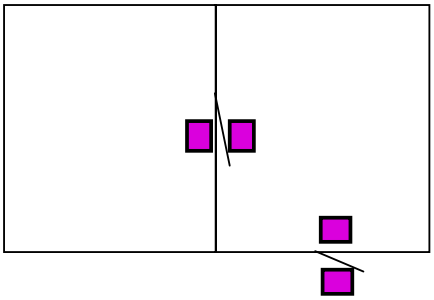
3.4.7
Inside object perspective
A Controller Method



3.5
Management subsystem (MS)



3.5.1
Area of Concern
MS Design Model

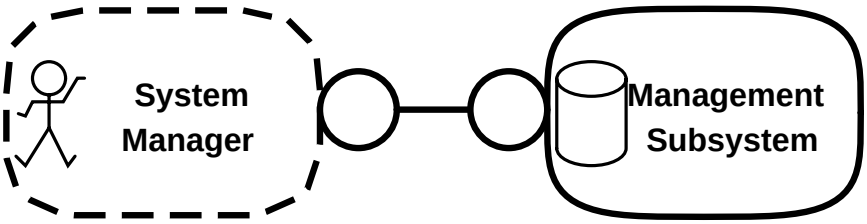


An office complex has a number of office areas separated by automatic doors.

We shall design a subsystem that permits a SystemManager to manage people's access privileges

3.5.2
Describe object structure
MS Design Model

Stimulus message	Response
SystemManager . addPersonWithAccess	Access data base updated



3.6
Card access control system (CA)

Card only access control system (CA)
- System seen from environment
- Separation of concern

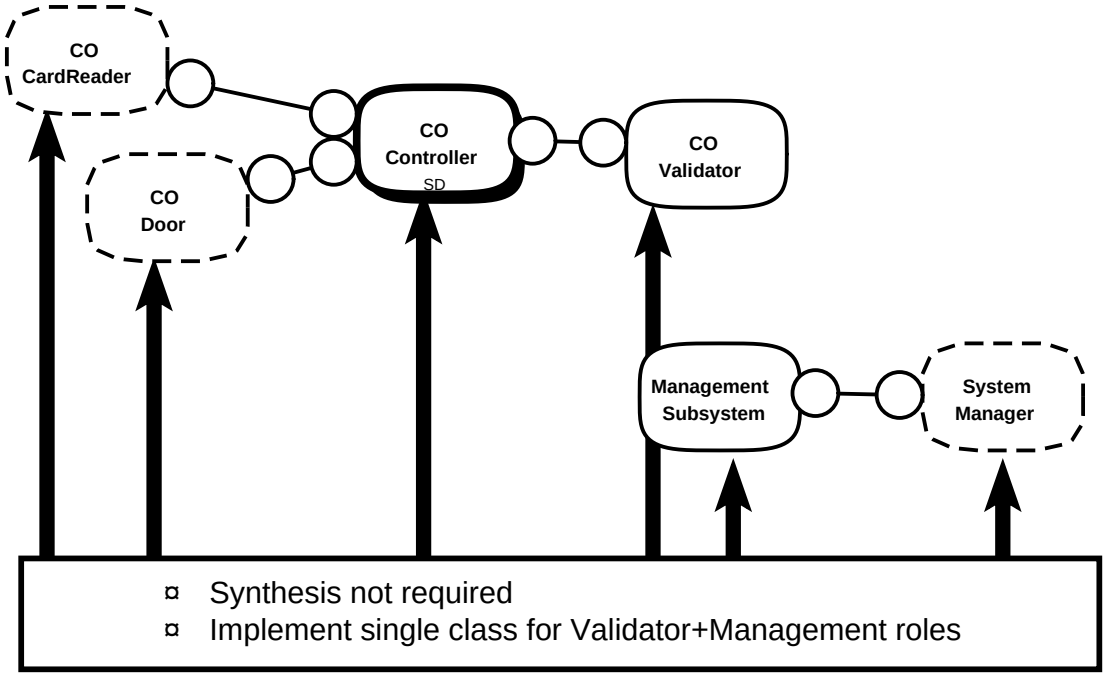
Validation subsystem (VAL)
- Environment
- Inter object views
- Inside object views

Management subsystem (MS)
- Environment
- Inter object views
- Inside object views

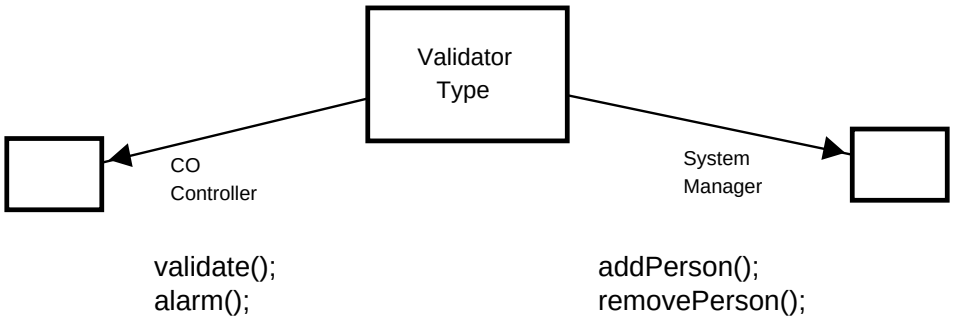
synthesis

Card only access control system (CA)
- System Implementation

3.6.1
Synthesize from subsystems
CA Design model

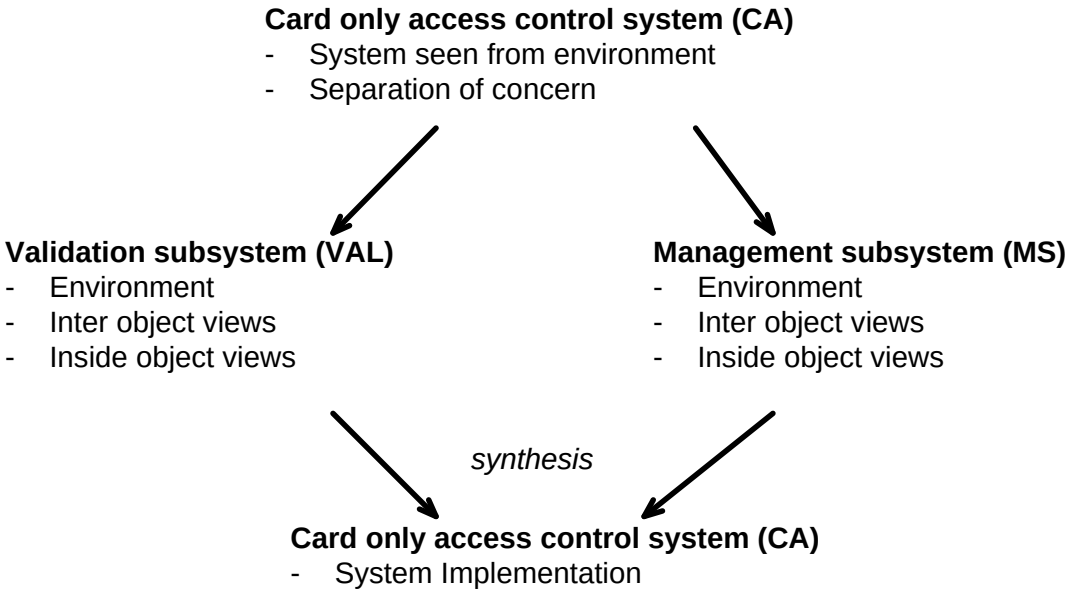


3.6.2
The Validator type satisfies COValidator and ManagementSubsystem Roles

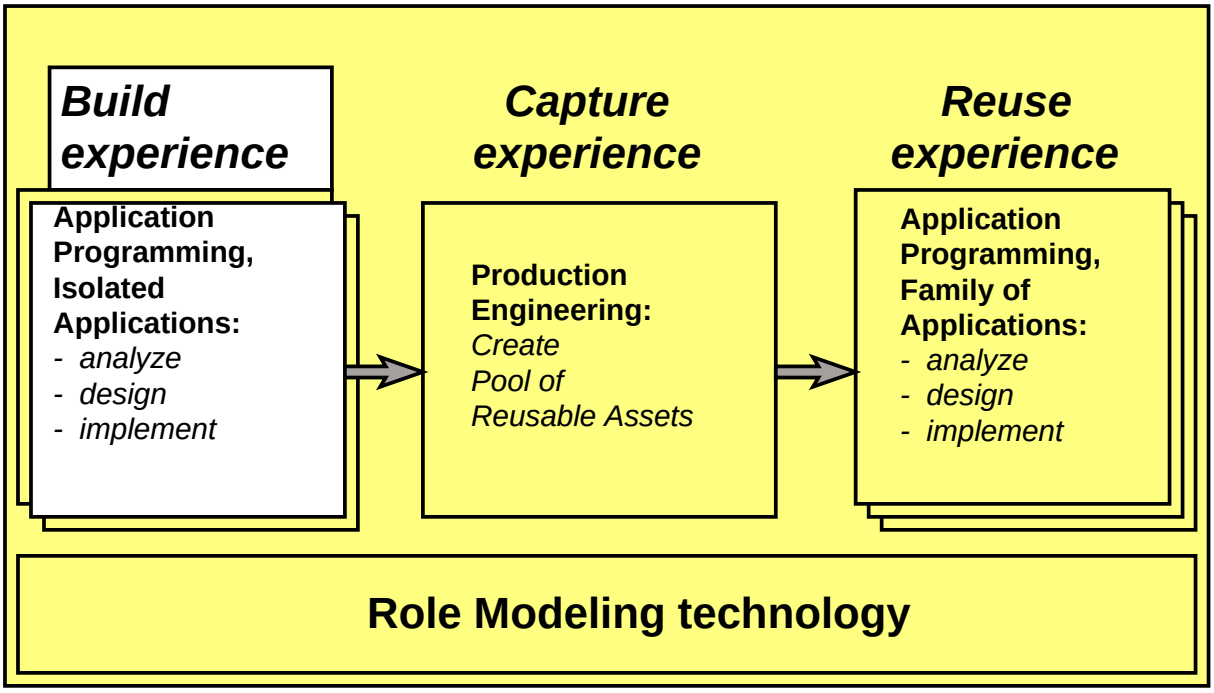


- ¤ Synthesis not required
- ¤ Implement single class for Validator+Management roles

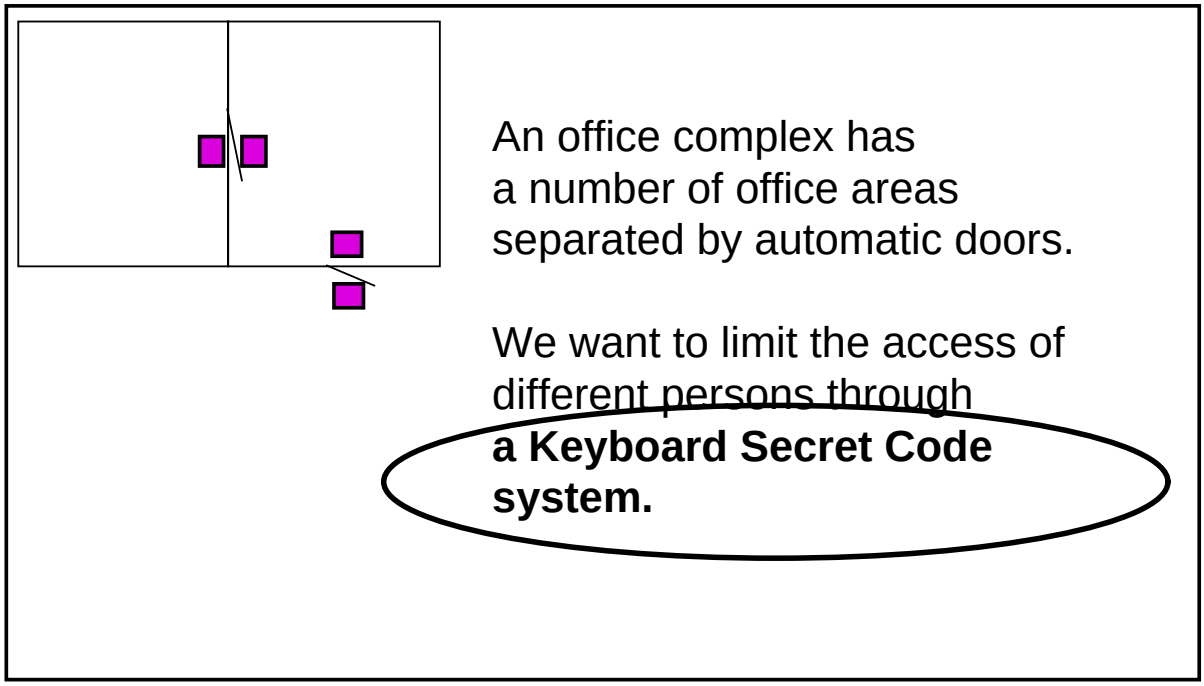
3.7
Summary
First isolated application



4.
The second isolated application
Keyboard only access control (KO)



4.1
Area of concern
Keyboard only access control (KO)



4.2
Incidental reuse

search existing solutions for applicable ideas, models, code, ...

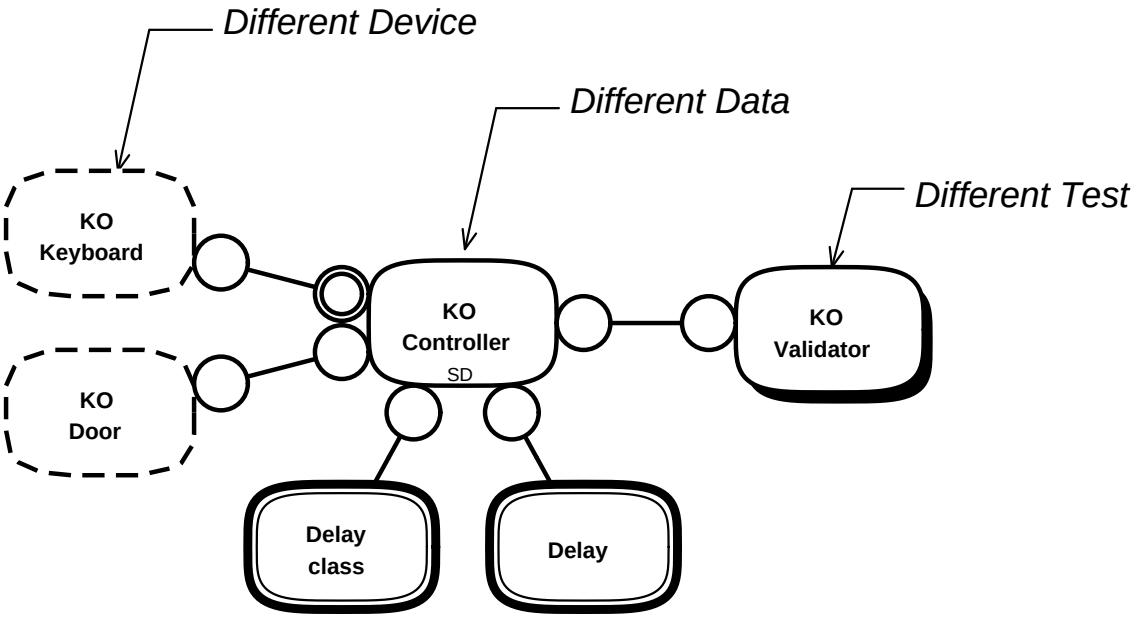
4.3
Stimulus-Response view
Keyboard only access control (KO)

Stimulus message	Response	Comment

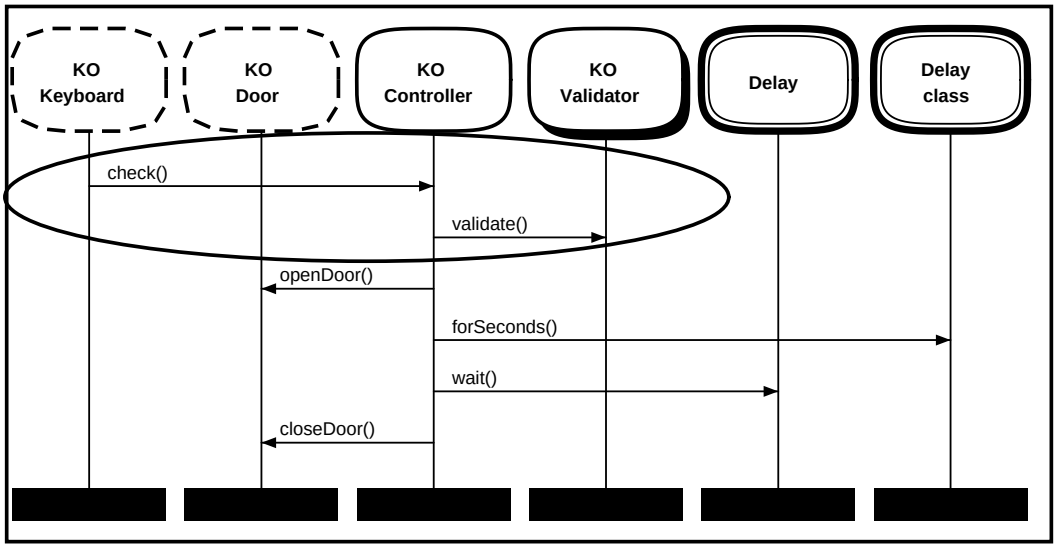
Person.identification (from);	Door.open ();
	// time delay
	Door.close();
		Person accepted

	Person.reject ();	Person rejected
	AlarmHandler .doorOpenAlarm();	Door left open
SystemManager .addPersonWithAccess ();	Access data base updated	

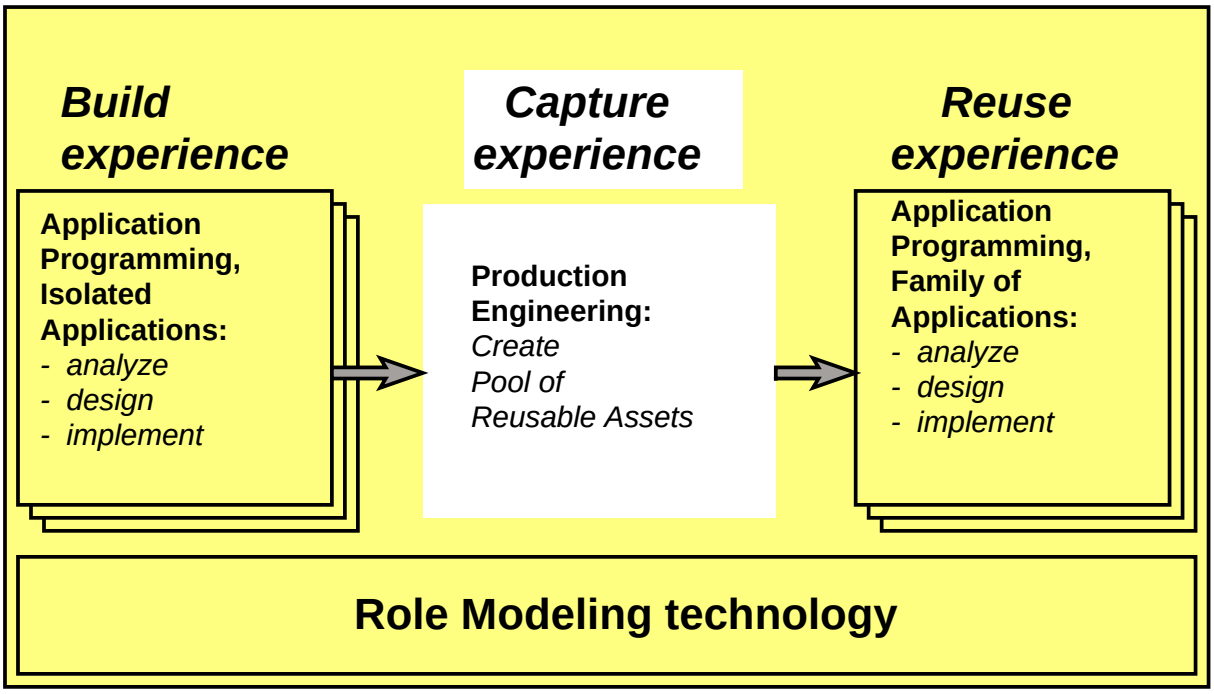
4.4
Collaboration View
Keyboard only access control (KO)



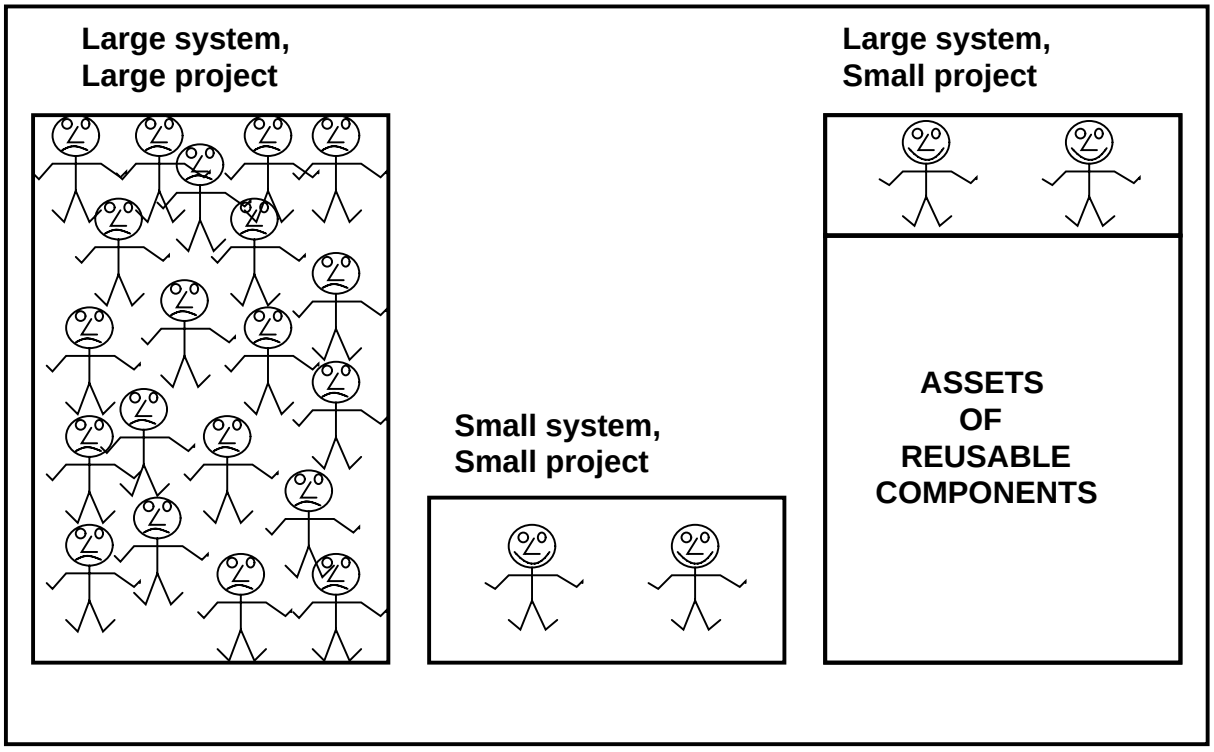
4.5
Normal Access Scenario View
Keyboard only access control (KO)



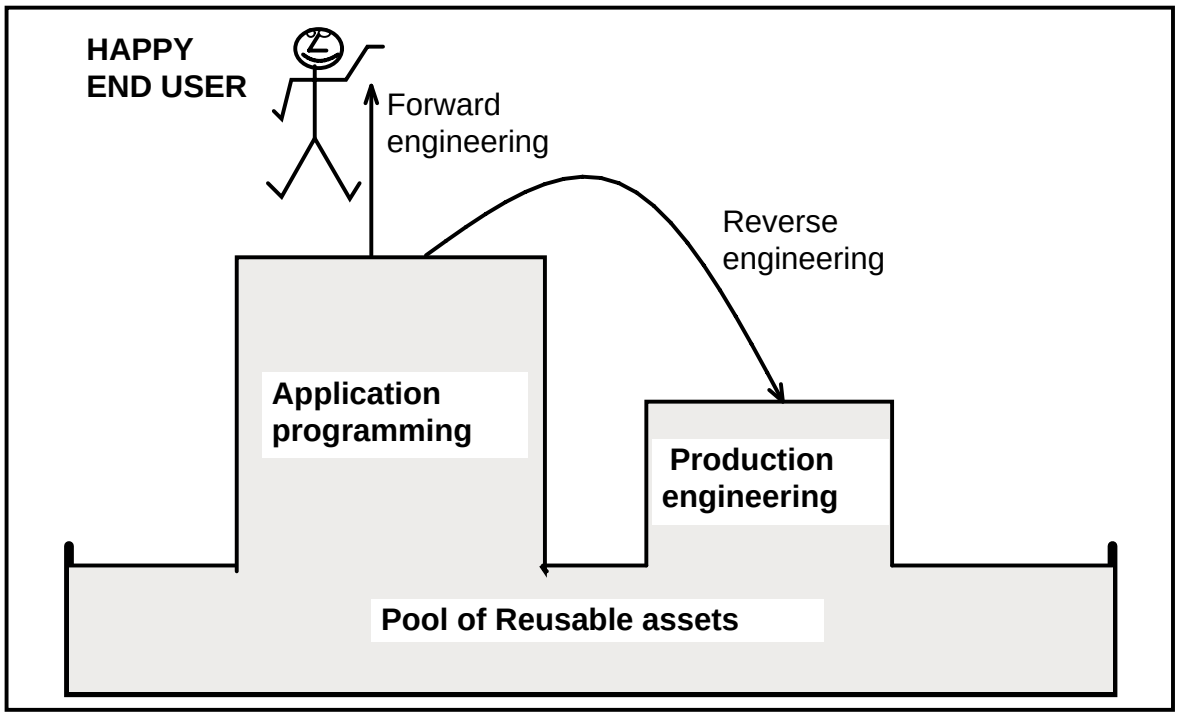
5.
Creating reusable components



5.1
Large systems from small projects



5.2
The Taskon Fountain Model for Planned Reuse



5.3
Some reuse terminology - 1

Alexander:

A pattern is a description of a problem with hints for its solution ensuring coherence and humanity

Some OO methodologists:

An object pattern is a structure of interacting objects

5.4
Some reuse terminology - 2

OOram Pattern:

An OOram pattern is a description of a problem with a Role Model describing its solution

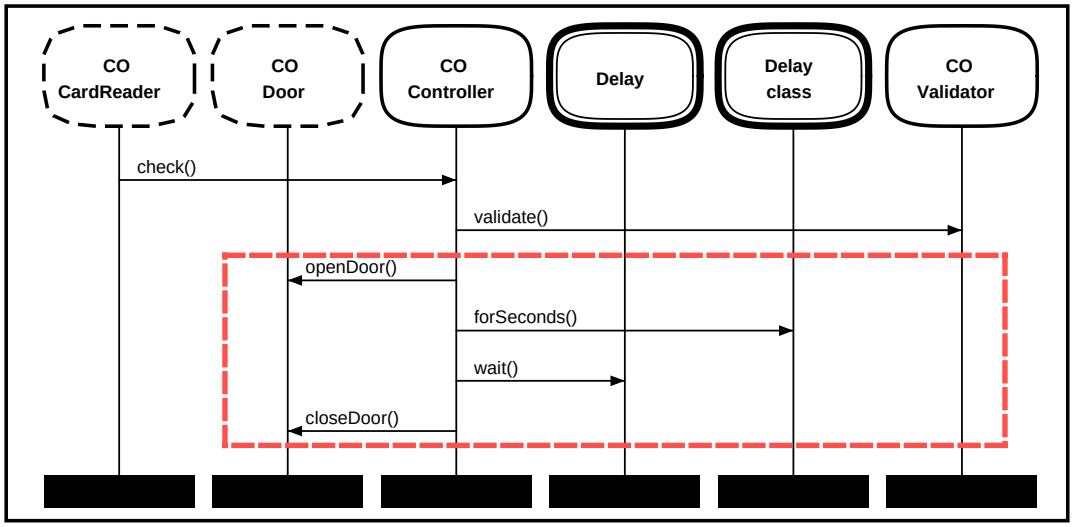
OOram Framework:

A canned solution with role model + base classes for subclassing

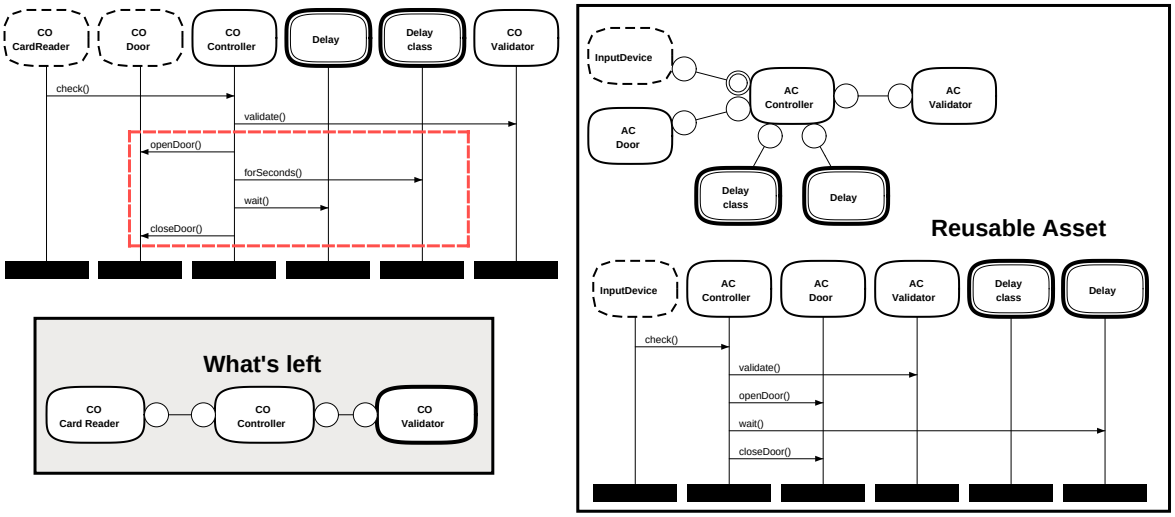
5.5
Creating an OOram framework

- Identify consumers and consumer needs
- Perform cost-benefit analysis
- Perform reverse engineering on existing applications
- Create new framework(s)
- Inform user community

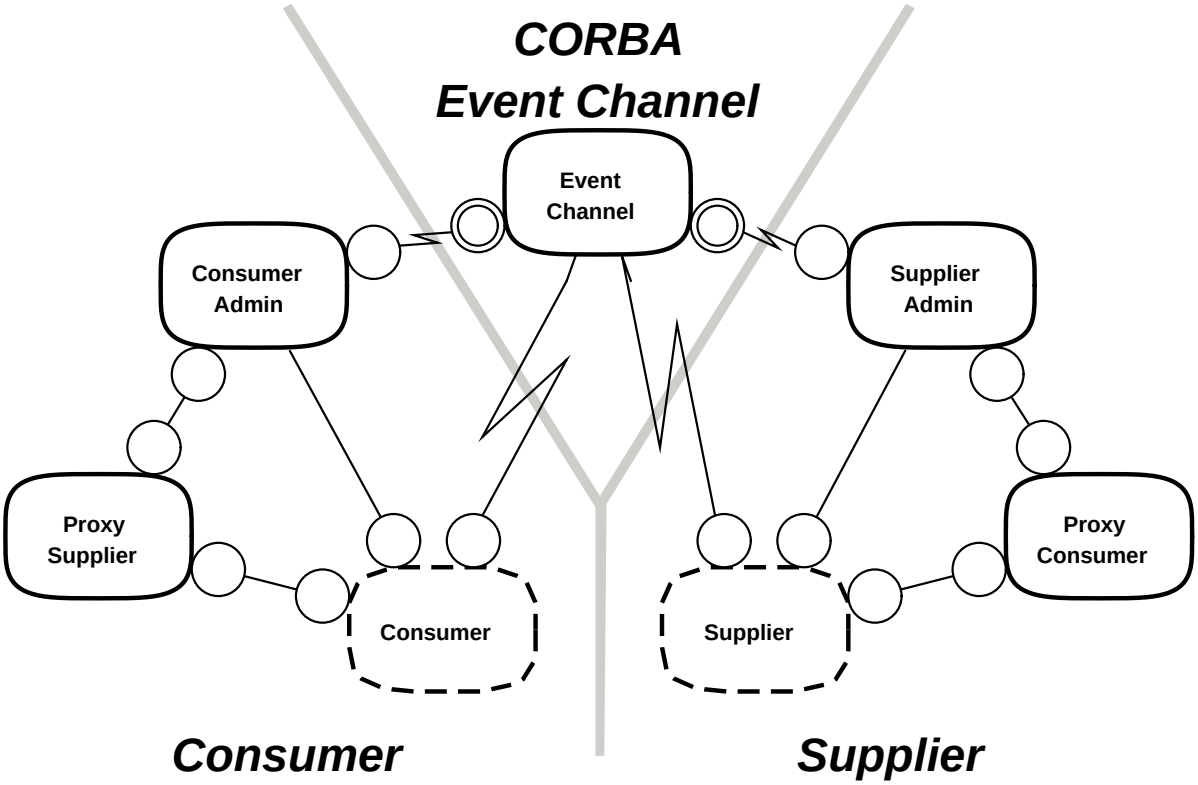
5.6
Identify reusable activities:
CO Scenario view



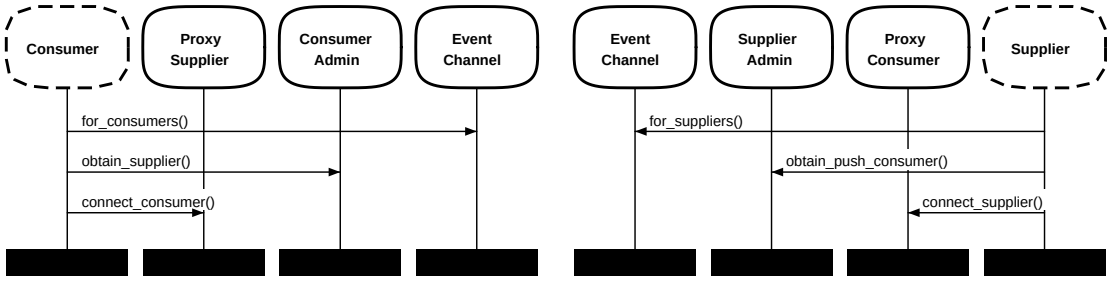
5.7
Isolate reusable parts
Inverse synthesis



5.8
Object Management Group (OMG): Event Service (simplified)

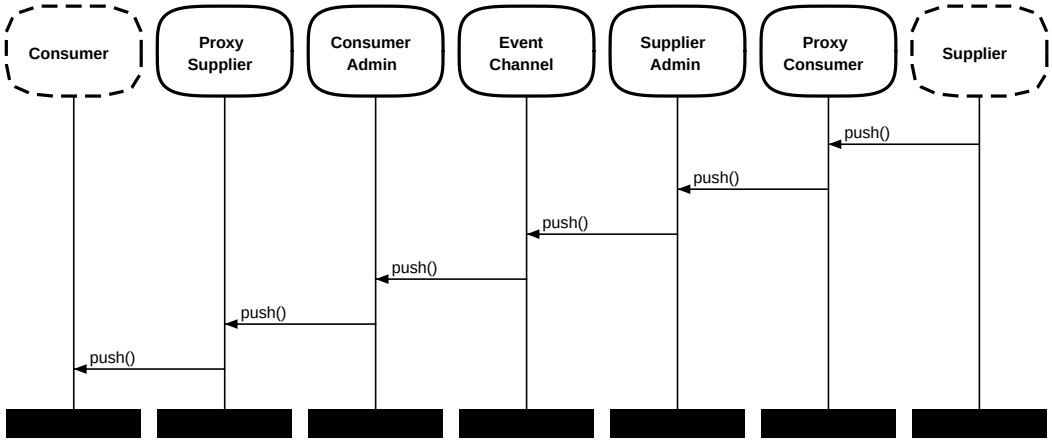


5.8.1
Establish connections

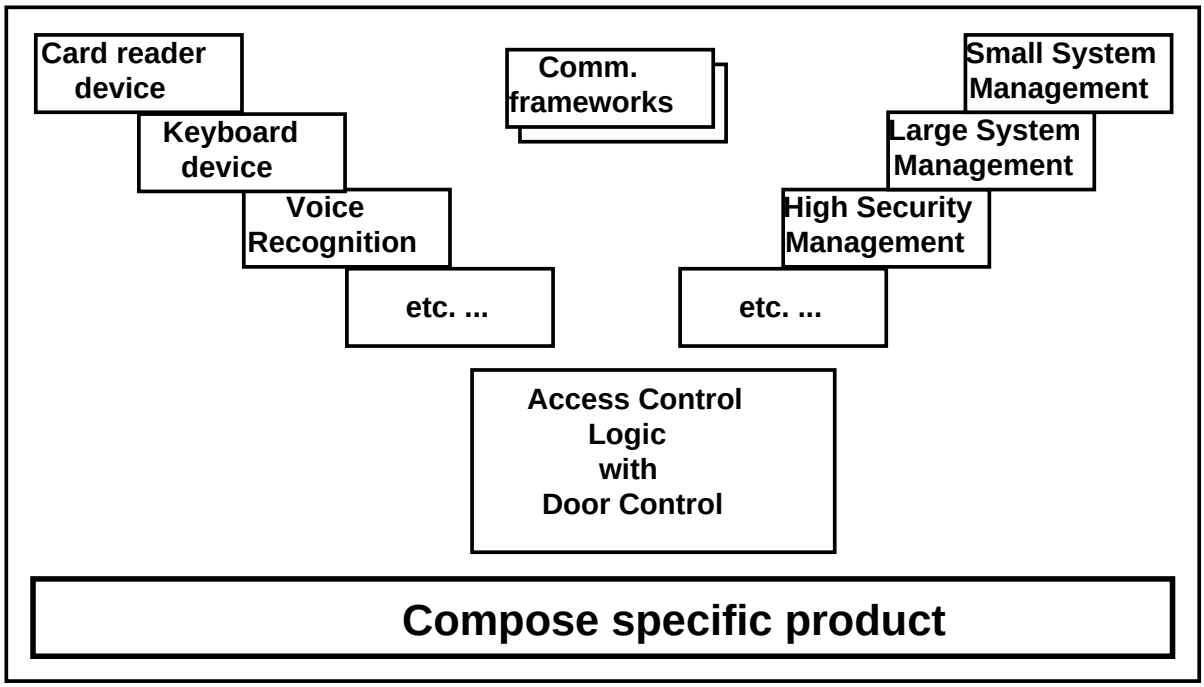


Establish consumer connection **Establish supplier connection**

5.8.2
Supplier Changed



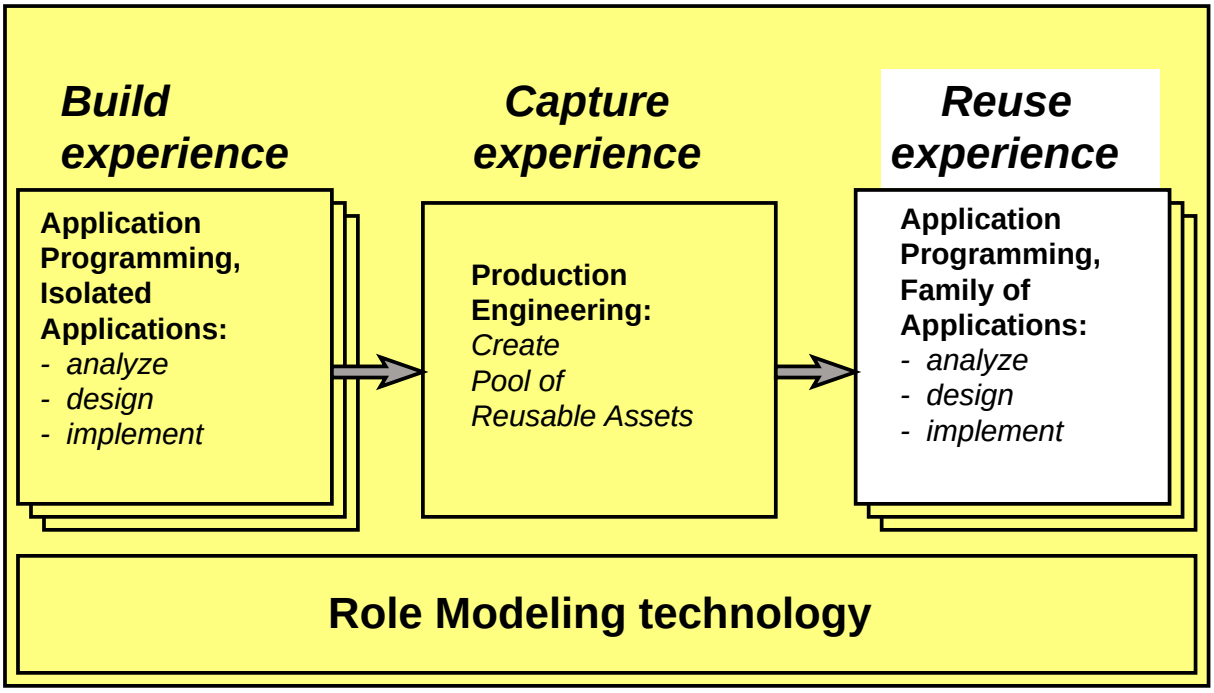
5.9
Access Control Frameworks



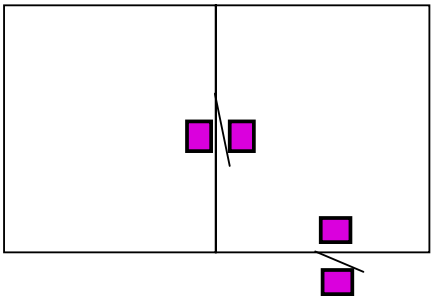
5.10
Key LocalStation control method

```
void identification (String identString, TypeSymbol typeSymbol) {
    /* This is a generic method that collects all identifications
     in Dictionary, passing it on for validation when complete.  */
    Boolean accessOK;
    identDict.add (typeSymbol, identString);
    if ((identDict.size = this.noOfInputs()); == true) {
        accessOK = validator.validate(identDict);
        identDict = new IdentityDictionary(); // Clear
        if (accessOK) this.openAndCloseDoor();
    }
}
```

6.
Card/Key Access Control System (CK)



6.1
Area of Concern
CK Requirements

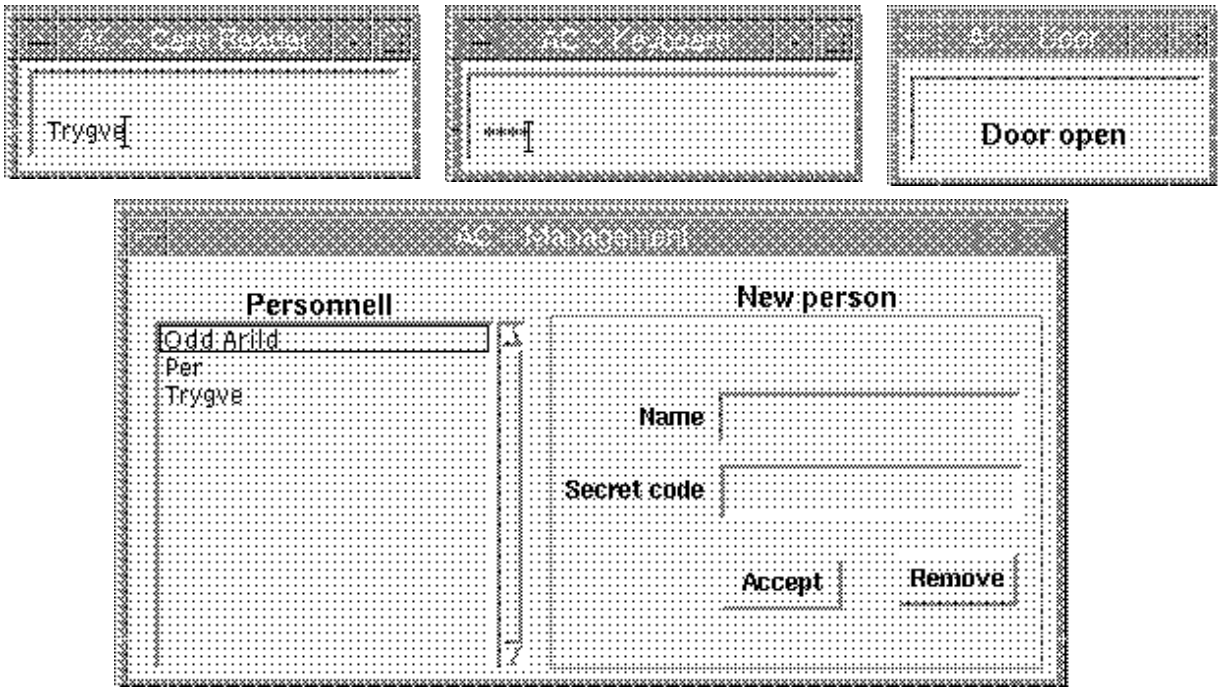


An office complex has a number of office areas separated by automatic doors.

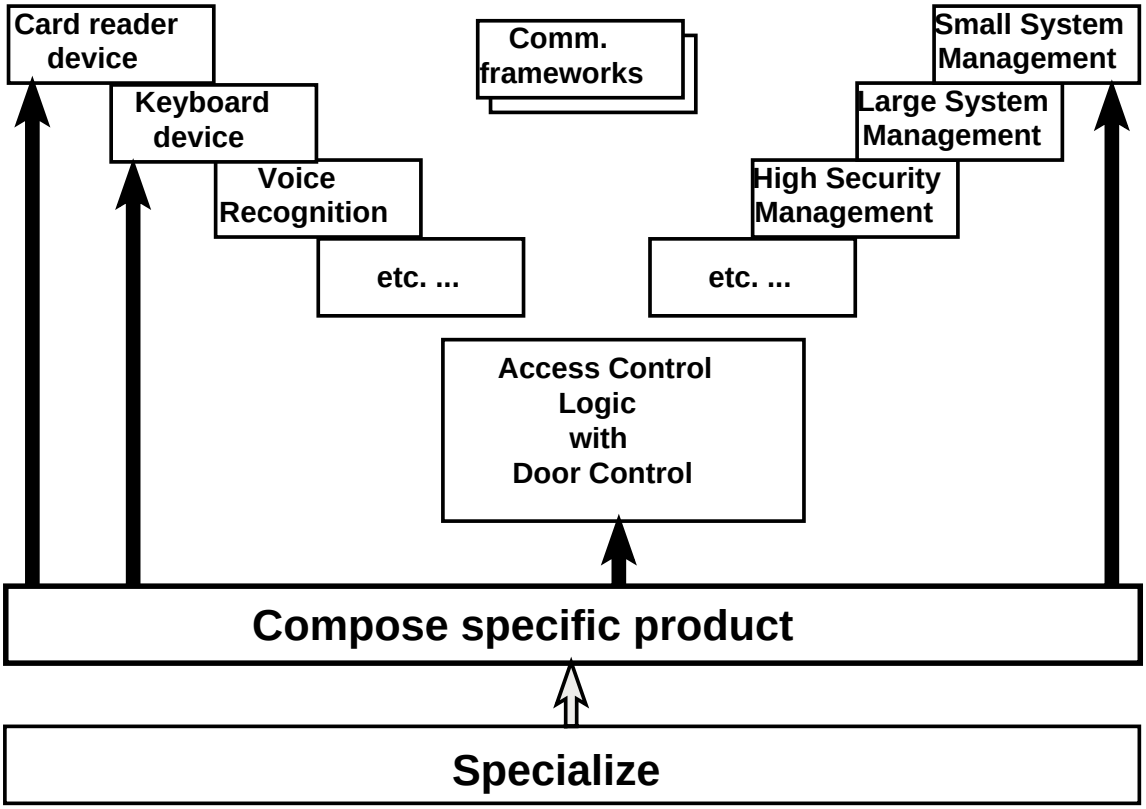
We shall design a System that controls access to the protected Zones

- **identified by card during working hours**
- **identified by card and key code at other times**

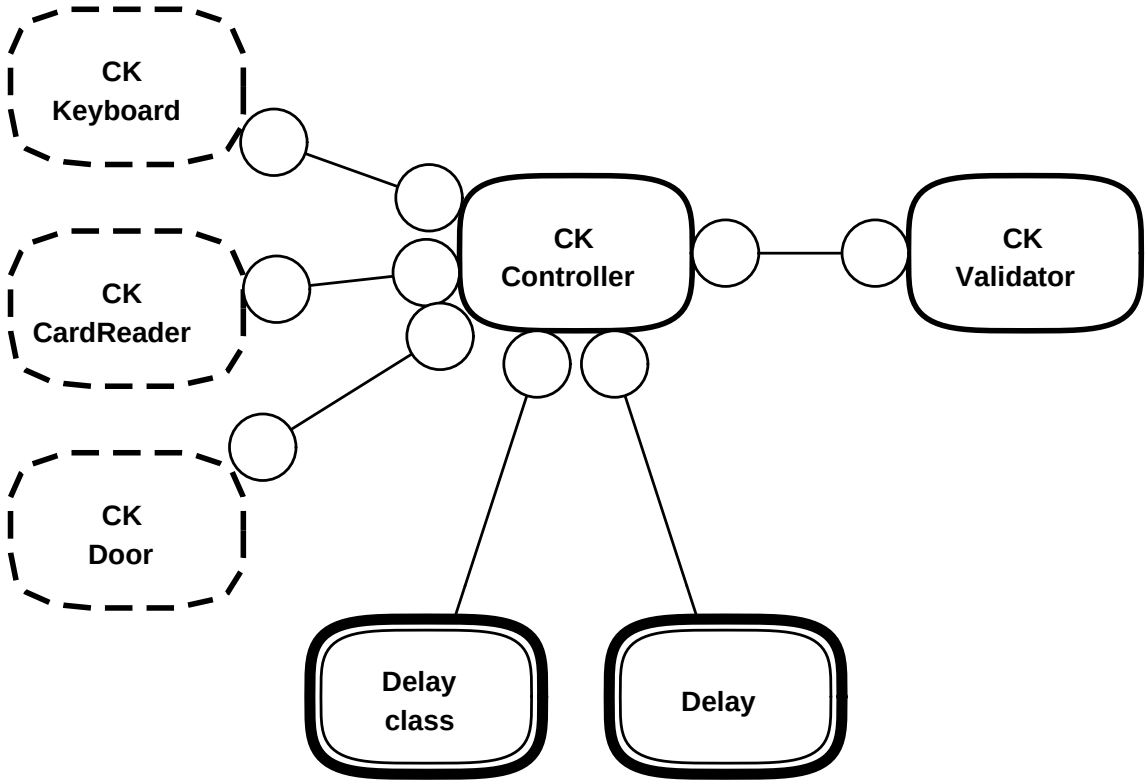
6.2
User interfaces
CK Prototype Program



6.3
Synthesize application
CK Design



6.3.1
Synthesized Card/Key model



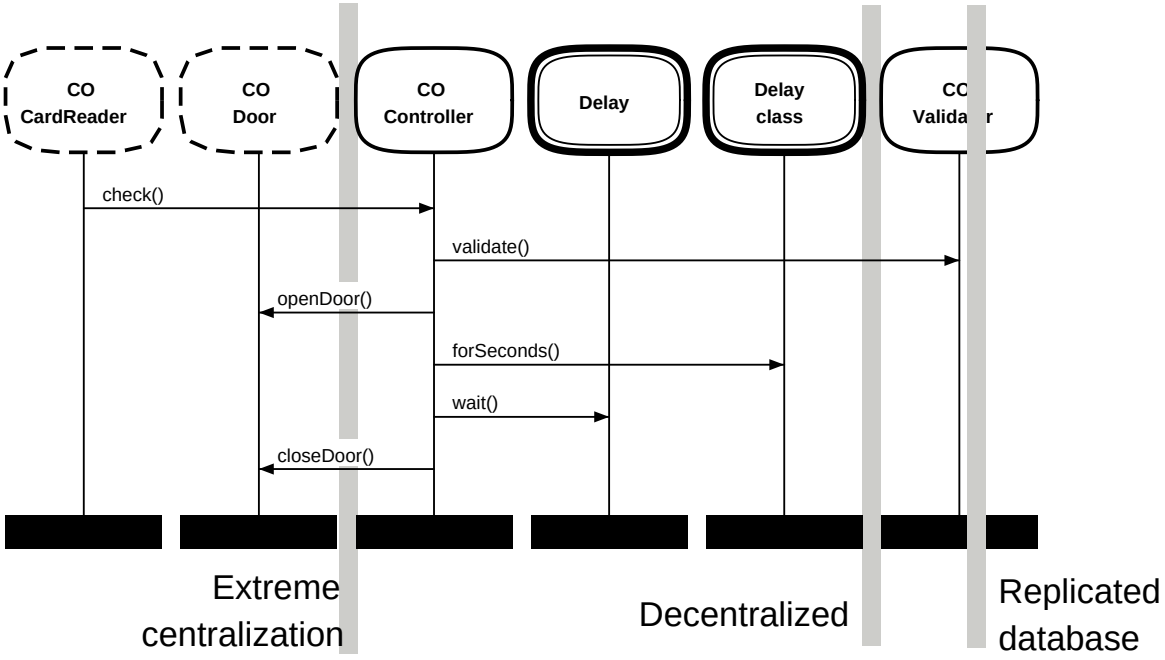
6.3.2
Determine distribution architecture

Iterate

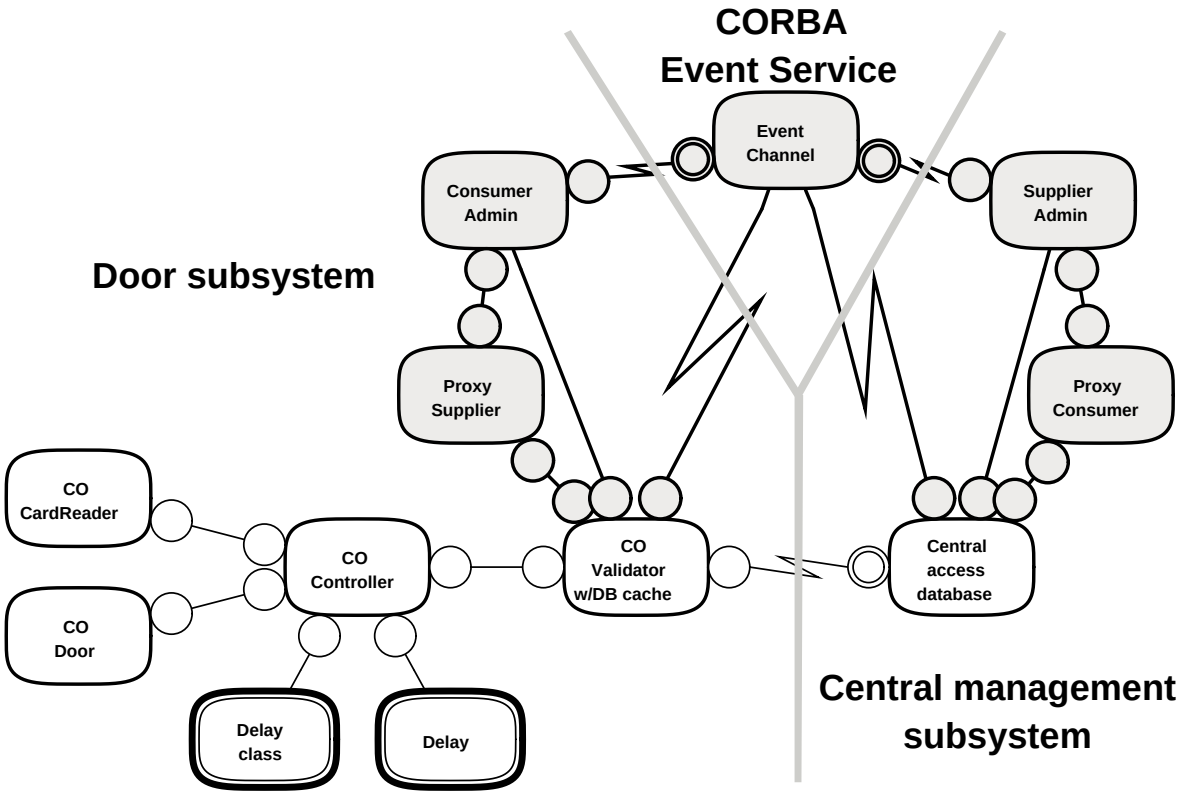
- **Select possible communication architecture**
  - ¤ **Evaluate communication requirements**
  - ¤ **Choose communication method**
    - **e.g., on demand**
    - **e.g., caching**
  - ¤ **Estimate communication costs (time, money)**

- **Implement solution**
  - ¤ **Subclass frameworks**
  - ¤ **Specialize code**
  - ¤ **Optimize code**

6.3.3
Some possible distributions



6.3.4
Example:
Replicated database



6.4
Controller, initialization and checking

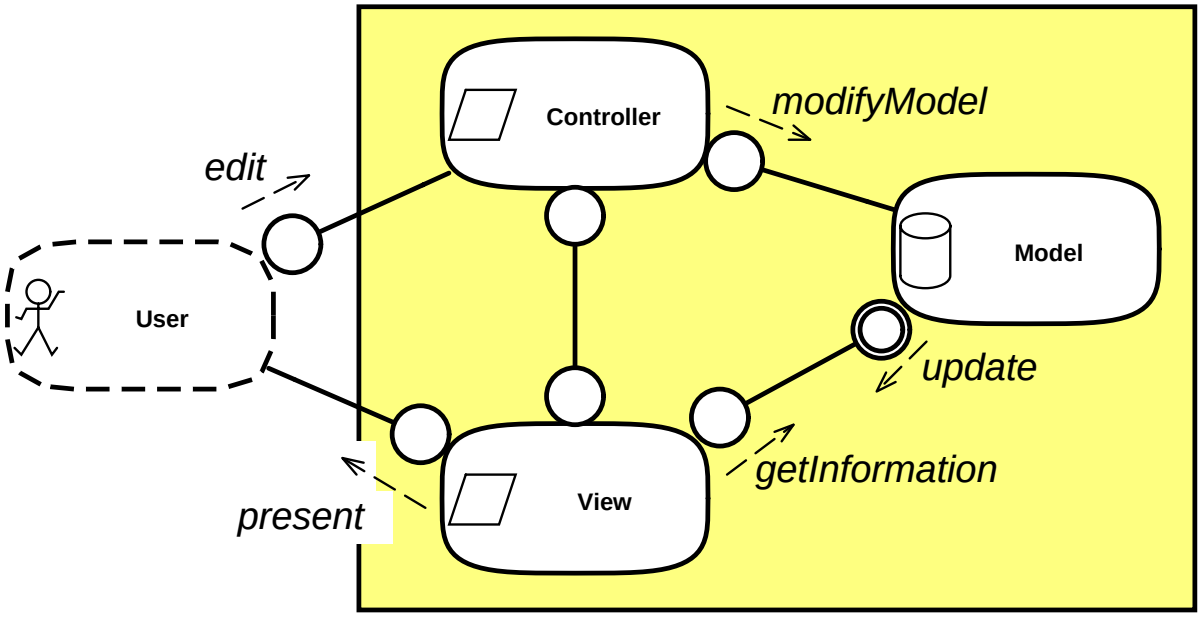
```
class Manager {
    /* Initialize structure. */
    manager (Manager aManager) {
        super.manager (aManager);
        acr = new ACCardReader();
        acr.openInterface("windowSpec", this);
        akr = new ACKeyboardReader();
        akr.openInterface("windowSpec", this);
        inputDevices = new OrderedCollection (acr, akr);
    }
```

```
    void identification (IdentSpec identString, TypeSymbol typeSymbol) {
        /* Collect identifications in Dictionary, validate when complete. */
        Boolean accessOK;
        identDict (typeSymbol, identString);
        if (((Time.hours > 8) and (Time.hours <= 16)
                    and (typeSymbol == "CardReader"))
                or (identDict size() == 2)) {
            accessOK = manager.validate(identDict);
            if (accessOK) {
                this.openAndCloseDoor();
            }
        }
    }
}
```

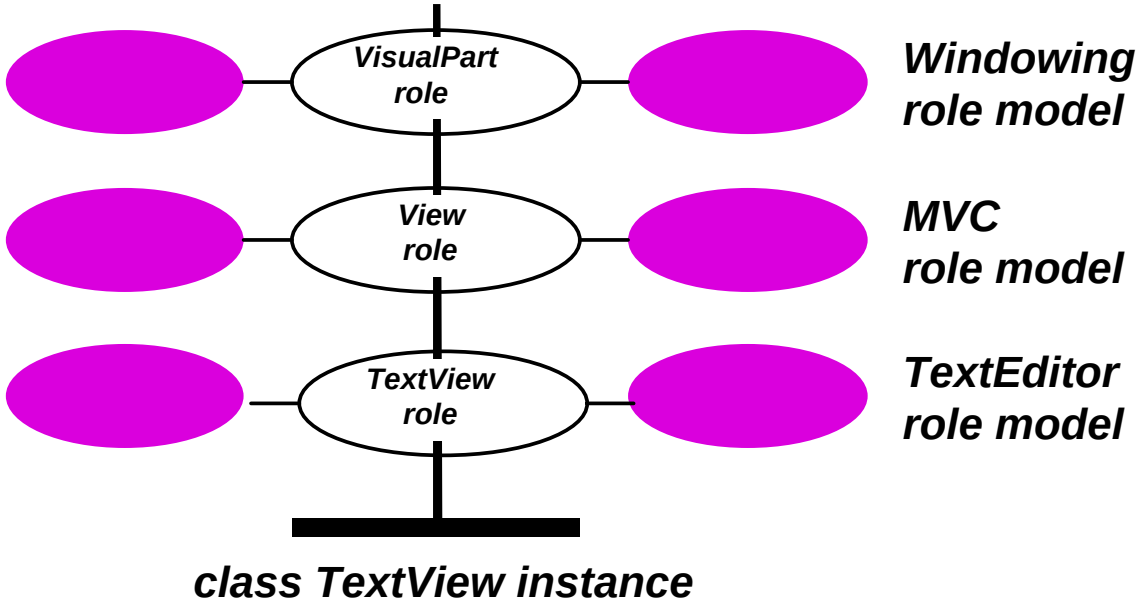
7.
Conclusion

- Role models describe object patterns
- Modeling distribution
- Separation of Concern
- Reuse through model inheritance
- Seamless bridge to implementation

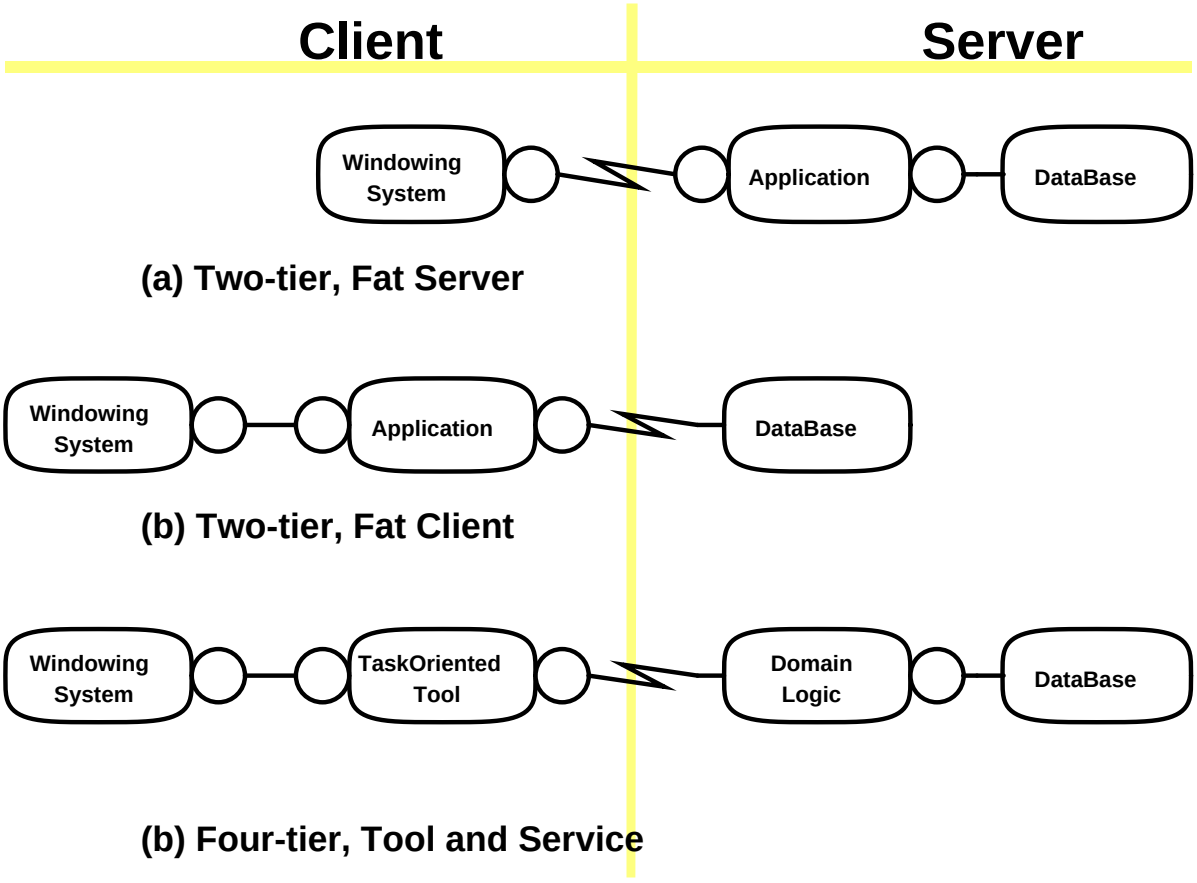
7.1
Role models describe object patterns



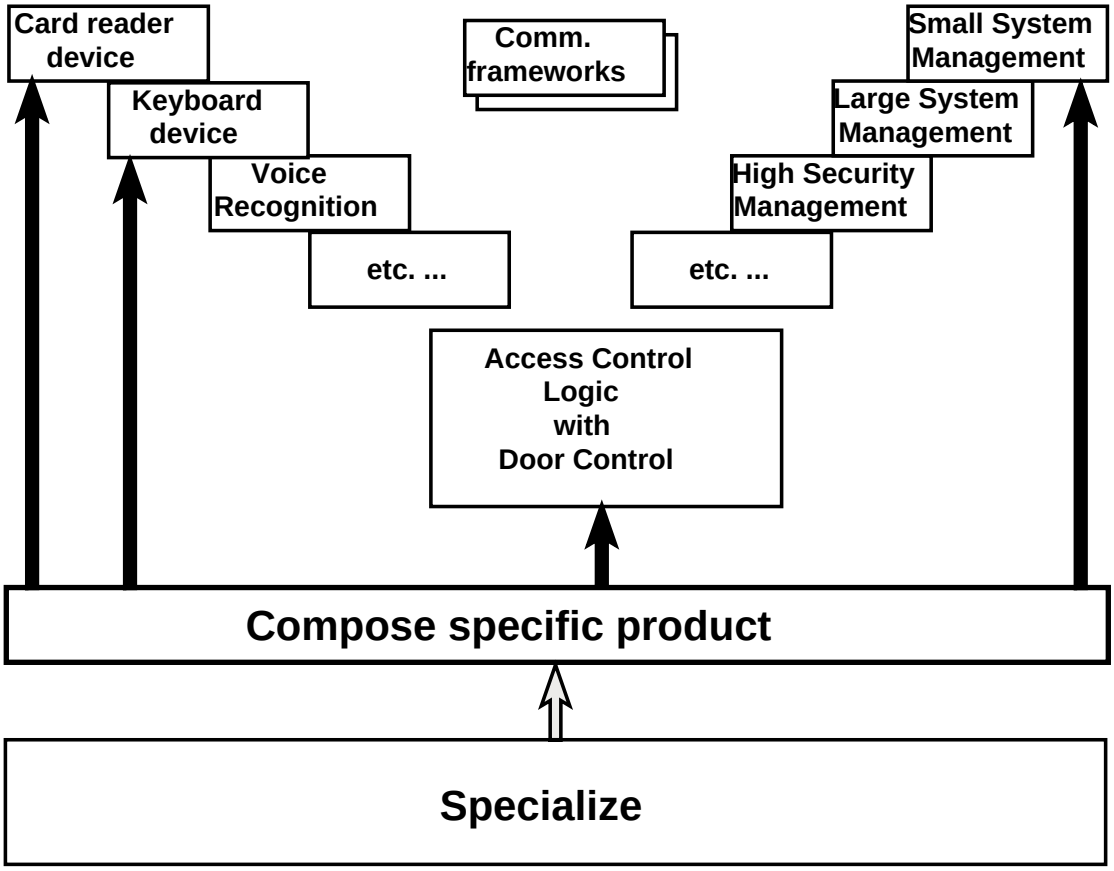
7.2
Separation of Concern
Object pattern composition



7.3
Modeling distribution



7.4
Reuse through model inheritance



7.5
Seamless bridge to implementation

Roles
- --> Interfaces
  - --> Classes

Ports
- --> Member variables & other references
  - --> Interfaces

Messages
- --> *Methods*

Model inheritance
- --> Class ensemble inheritance

7.6
More information

Management processes:
Goldberg and Rubin: "Succeeding with objects".
Addison-Wesley 1995. ISBN 0-201-62878-3
Modeling method:
T. Reenskaug, P. Wold, O.A. Lehne: "Working With Objects"
Manning/Prentice Hall 1996.
ISBN 0-13-452930-8
Patterns:
Gamma, Helm, Johnson, Vlissides: "Design Patterns".
Addison-Wesley 1994. ISBN 0-201-63361-2
Distribution:
ODP: Open Distributed Processing
ISO/IEC DIS 10746-2 Foundations. ISO/IEC DIS 10746-3 Architecture.
ISO 1995, 1996

OOram tools can be downloaded from:
http://www.sn.no/taskon/

TABLE OF CONTENTS