SWEN 223

Software Engineering Analysis

Object-Oriented Design  
with the UML

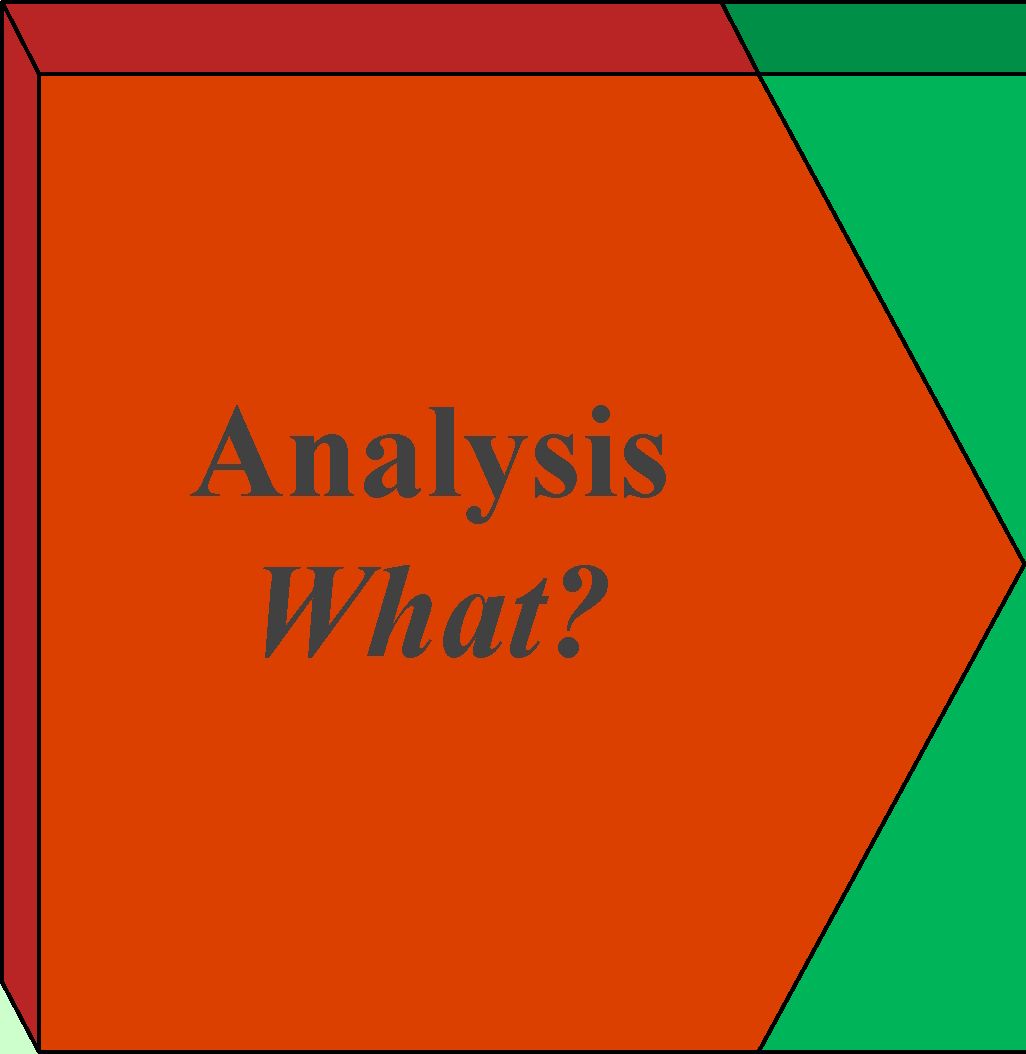
Thomas Kuhne

Victoria University of Wellington  
[Thomas.Kuehne@ecs.vuw.ac.nz](mailto:Thomas.Kuehne@ecs.vuw.ac.nz), Ext. 5443, Room Cotton 233



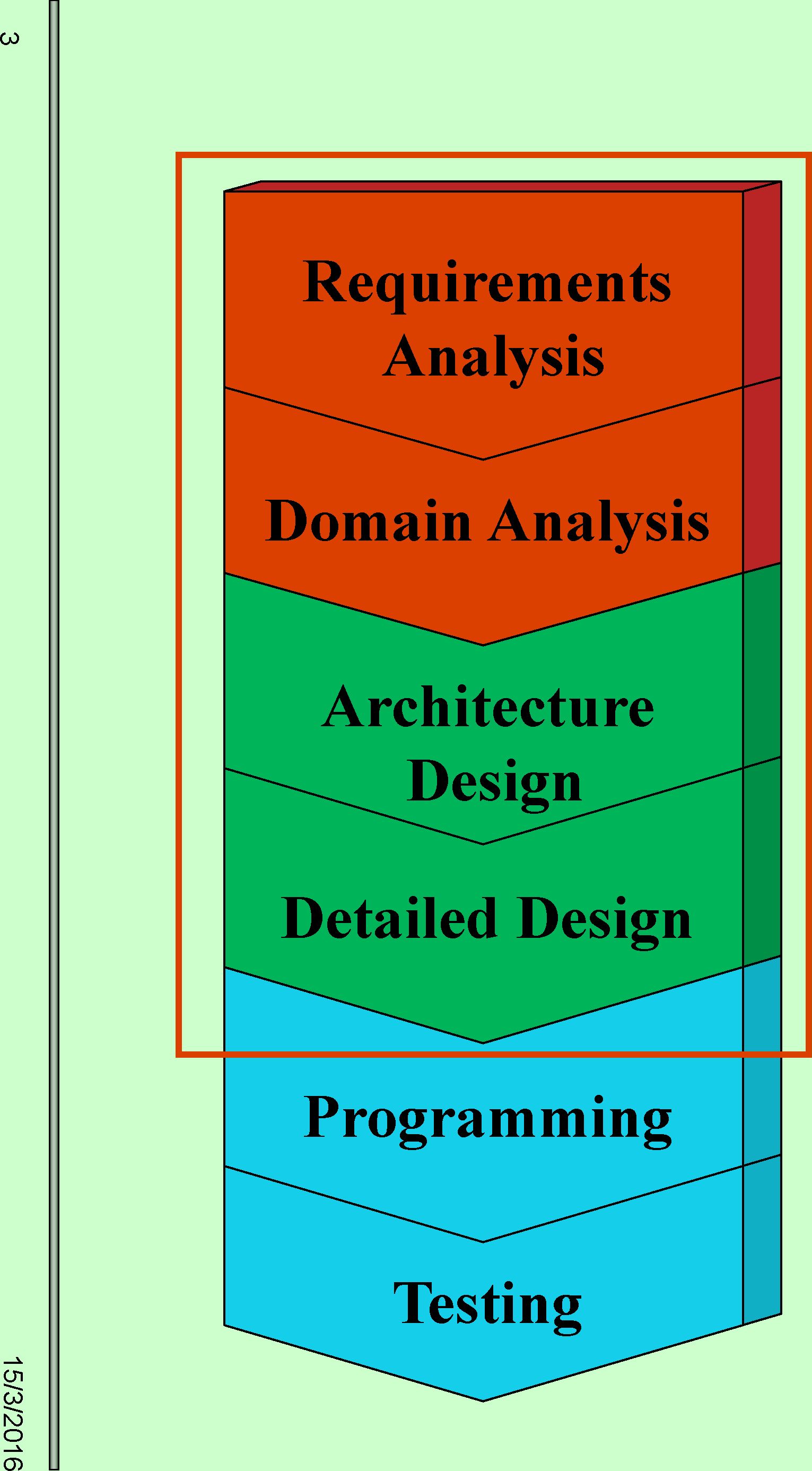
Imple- Design menta-

**How?** tion



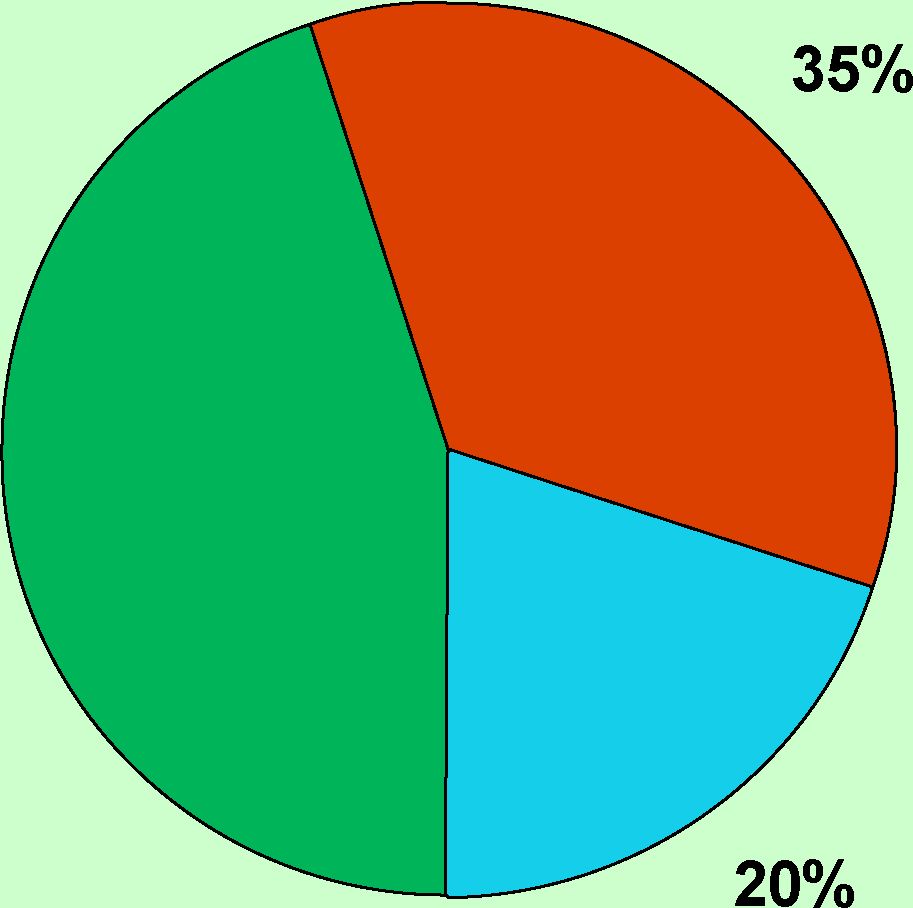
Product!

I



Proportion of Phases

45%



□ Analysis □ Design □Implementation

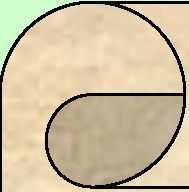
* Most time spent in understanding the requirements / domain
* 80% spent on planning the system
* 20% spent on actually implementing the system



\*



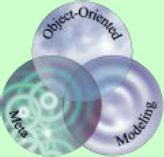
***Analysis means “understand the problem",***



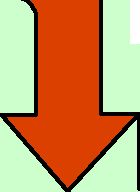
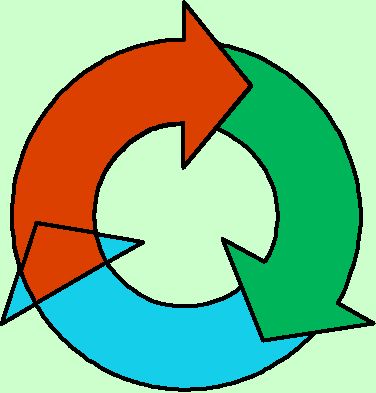
***Design means “plan the solution".***

***Are you saying that you work faster when you don't understand the problem, and have no particular solution in mind?***

John DiCamillo



Why not Hack Away?



▼

* Systematic approach

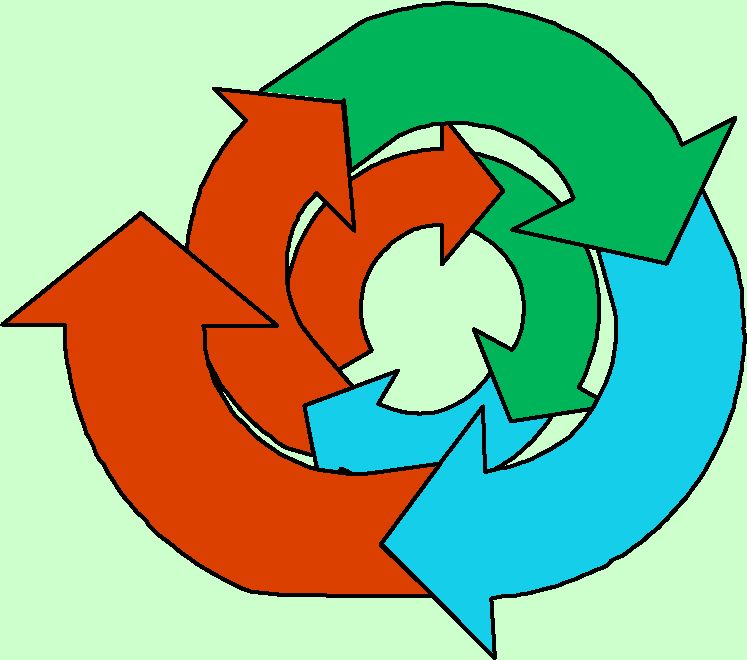
» process » documentation » communication

* Planning is cheaper than mending

» the earlier an error can be caught, the easier and cheaper it is to remove it

Waterfall

Iterative



Incremental

I ,ru\.

Rational Unified Process

„Validate early, validate often“

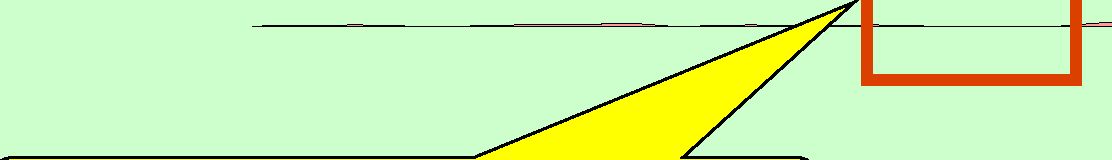
Phases

Activities

|  |  |  |  |
| --- | --- | --- | --- |
| Inception | Elaboration | Construction | Transition |

Business Modeling Requirements Analysis & Design

Implementation Test Deployment



one iteration



**Problem Statement**

Analysis t

**Formal System Model**

Architecture Designi

**System Architecture**

Detailed Design >

**Implementable Design**

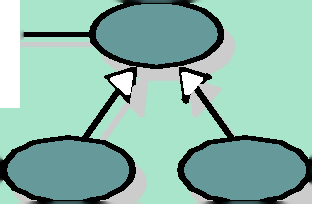
|  |  |  |
| --- | --- | --- |
| Requirements Analysis | 4 | Use Cases |
| Domain Analysis | 4 | Conceptual Model |
| Architecture Design | 4 | Package Diagram |
| Detailed Design | 4 | Class Diagrams |
|  |  | State / Activity / |
|  |  | Interaction Diagrams |
| Programming | 4 | Code |
| Testing | 4 | Bug Report |



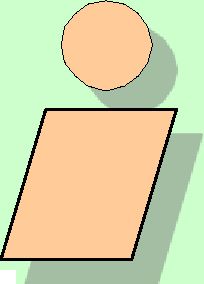
sqpi Requirements Analysis

Use Case Model

\*



Analyst



->

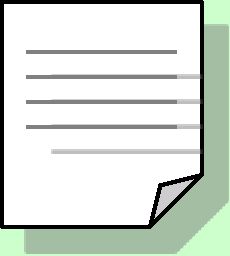
\*



N 1 ^

Feature list

\ ■\* / N



Supplementary

Requirements

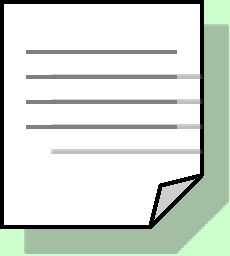
✓

\/

s

s

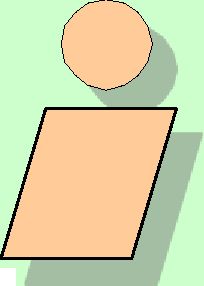
x



✓

N \

User



/ '

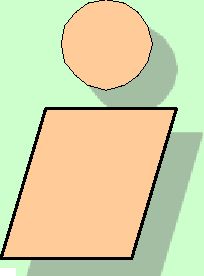
/ Vw

Glossary , ^ \ /

\ /

-- \* /

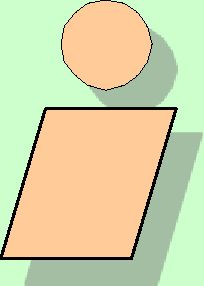
*&*



**/** Customer

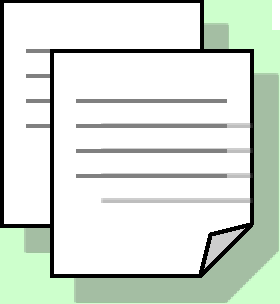
|  |  |
| --- | --- |
| \ |  |
|  | 11 □  0 |

Prototype



User Interface Designer

Domain/Business Model

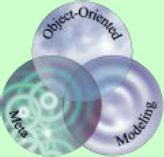


After the “Method Wars”

* OOA/OOD (Coad/Yourdon, 1991)
* OMT (Rumbaugh, 1991)
* Booch (Booch, 1994)
* Objectory (Jacobson, 1994)
* Fusion (Hewlett-Packard)

^ UML

» <http://www.omg.org>



UML Diagram Types

• Functionality & Structure

» use case diagram system usages

» structure diagram static class/object relationships

* Behaviour

» state diagram » activity diagram » sequence diagram » communication diagram

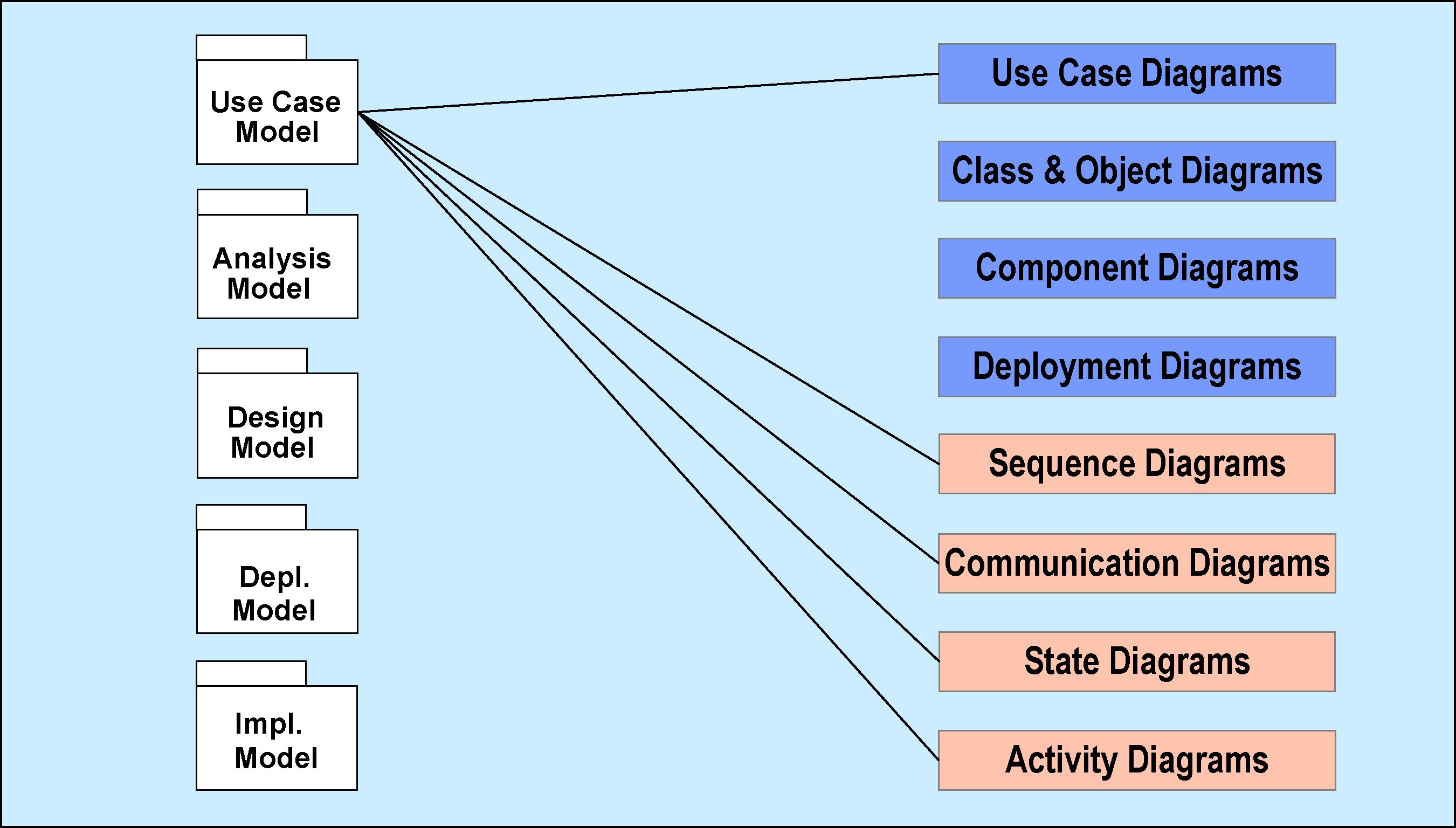
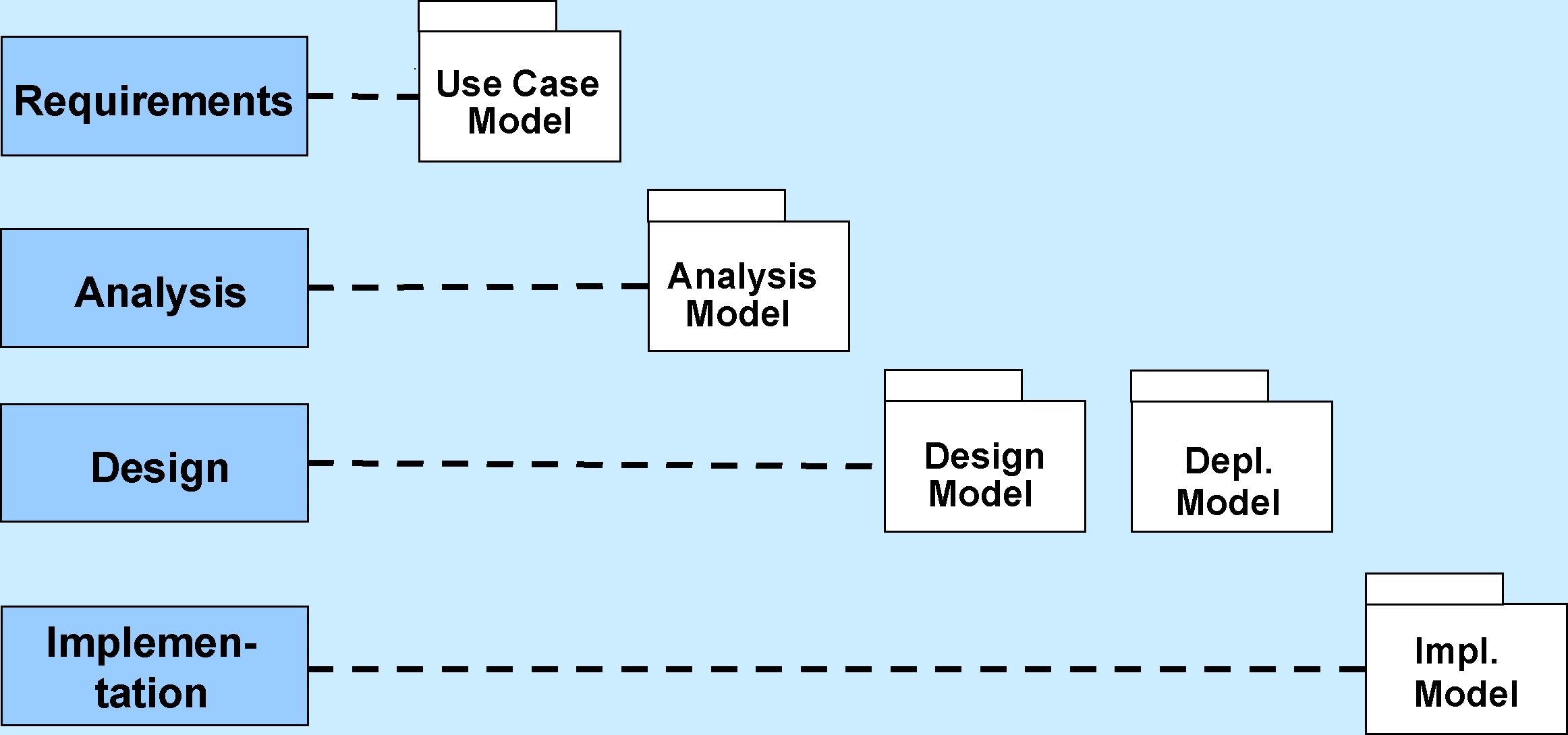
* Implementation

» component diagram » deployment diagram

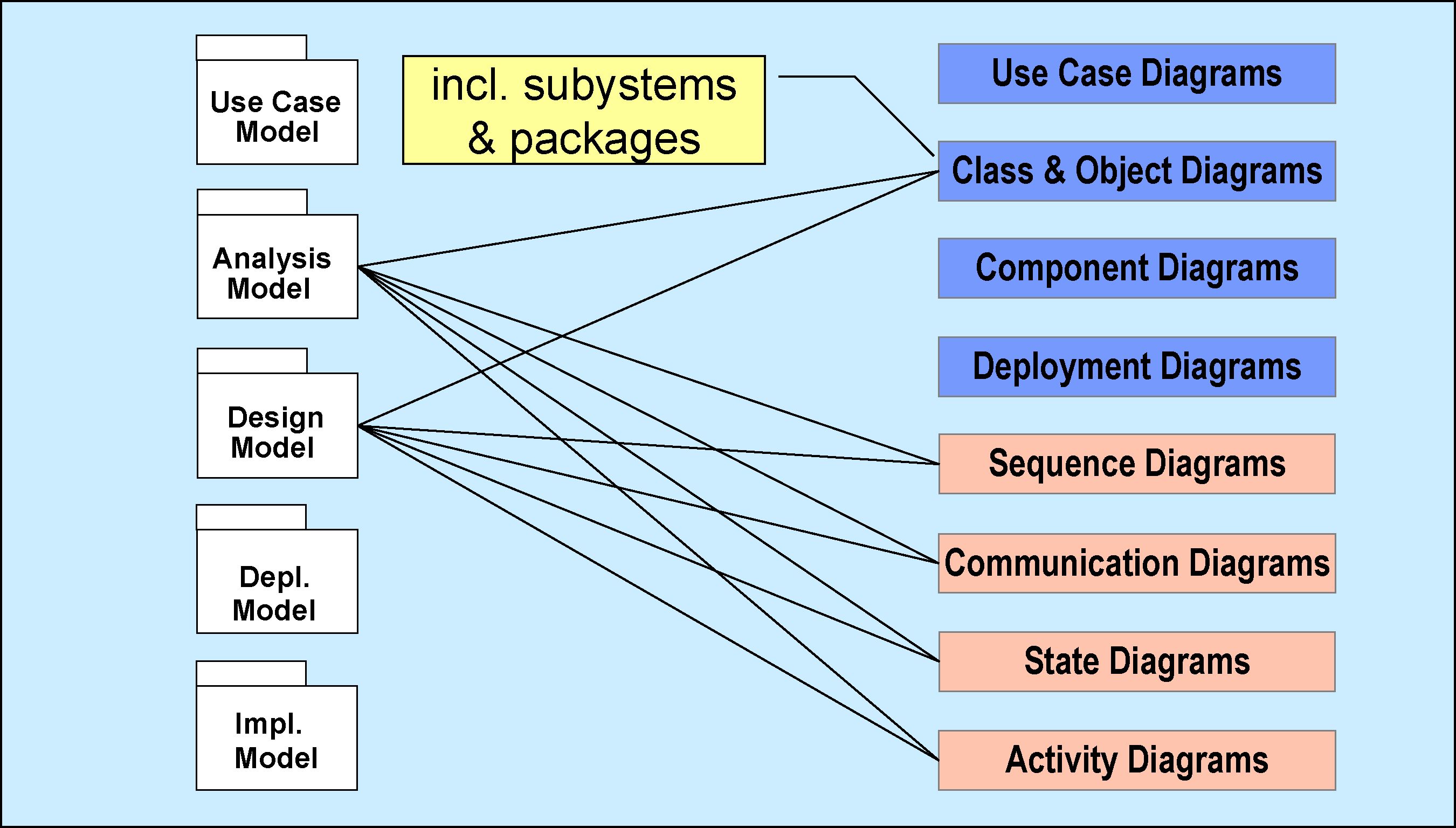
reactive behaviour control flow interactions (— time) interactions (— structure)

execution scheduling installation plan

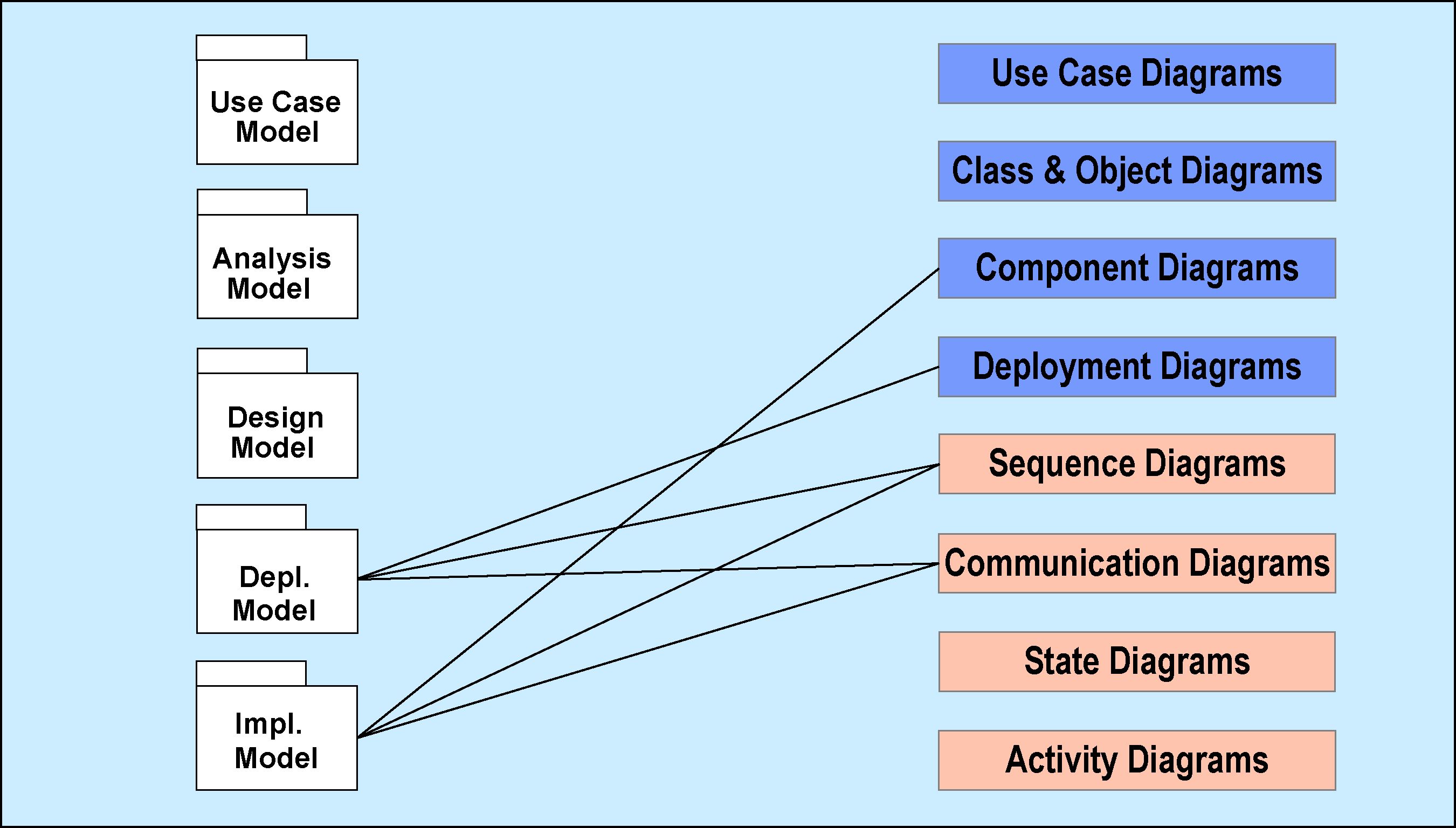
Activities and Models



i ,ru\.



I ,ru\.



What is Analysis?

* Requirements analysis

» enquire about user needs » explore possible services

* Domain analysis

» develop domain model » validate system understanding » validate requirements



Why Analysis?

Identifying the system requirements is hard, because of

* “Tech talk” & domain “slang”
* expert knowledge is difficult to access
* implicit assumptions are often wrong
* future system users have partial ideas only
* never ending appetite for features

—i

**dMS**

15/3/2016 \ **’f j**

A Campus Library, nicely located at the big Plaza, has university staff and students as its primary users.

For up to 4 weeks a member of staff can borrow up to 20 books. Students may borrow up to 10 books for 1 week. Magazines are available for at most 3 days.

Any user may search for books using an on-line catalogue and make reservations for books on issued to a user in case a book is ov with lending, reservations, and addi

Identification of  
domain concepts  
and activities

The University is interested in an electronic library system to support the above functions.



\*



A Campus Library, nicely located at the big Plaza, has university staff and students as its primary users.

For up to 4 weeks a member of staff can borrow up to 20 books. Students may borrow up to 10 books for 1 week. Magazines are available for at most 3 days.

Any user may search for books using an on-line catalogue and make reservations for books on loan. A reminder will be issued to a user in case a book is overdue. Librarians deal with lending, reservations, and adding and removing books.

The University is interested in an electronic library system to support the above functions.

A Campus Library, nicely located at the big Plaza, has university staff and students as its primary users.

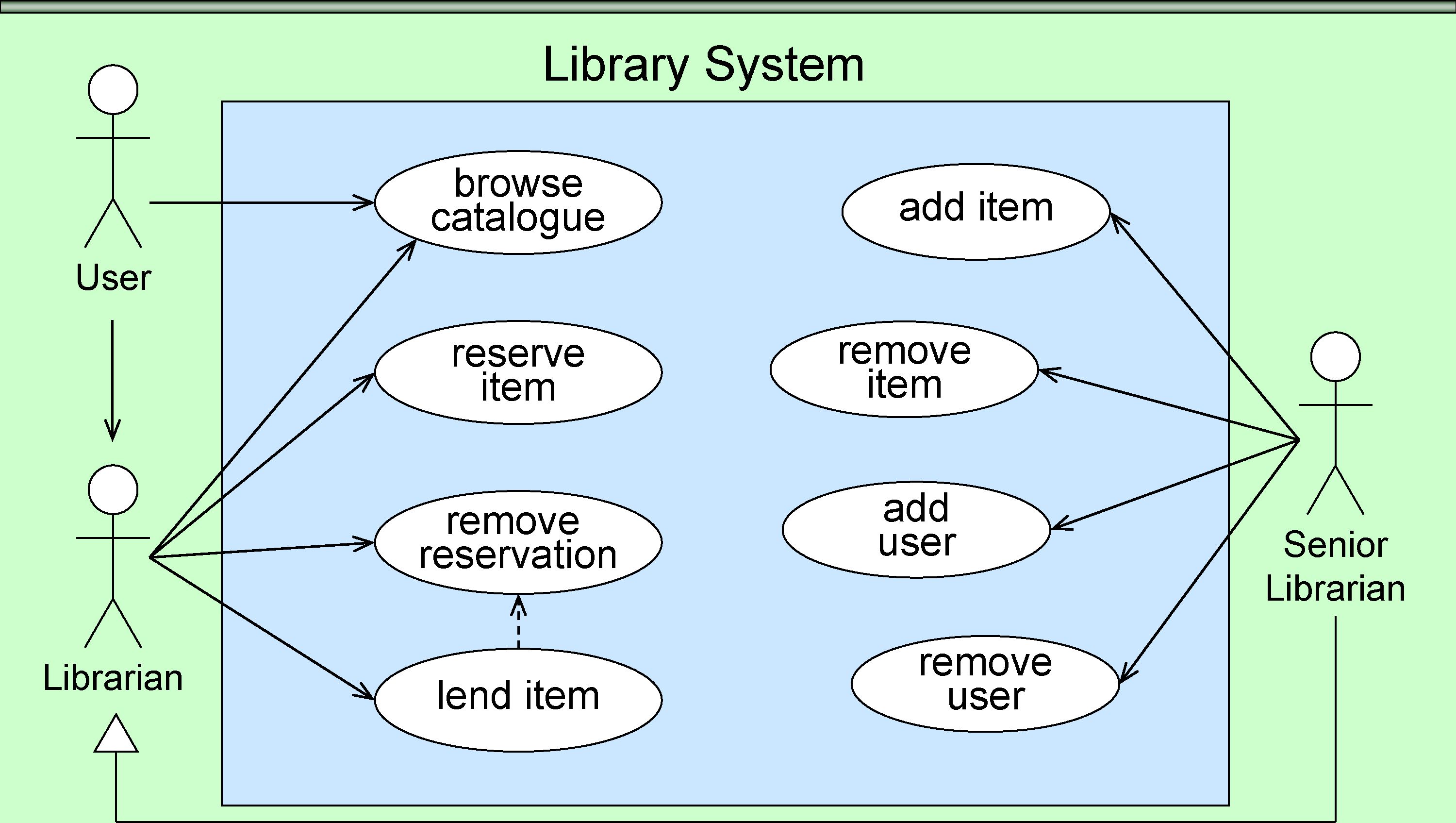
For up to 4 weeks a member of staff can borrc concept books. Students may borrow up to 10 books f or Magazines are available for at most 3 days. property?

Any user may search for books using an^n-HnS'catalogue and make reservations for books on loan. A reminder will be issued to a user in case a book is overdue. Librarians deal with lending, reservations, and adding and removing books.

The University is interested in an electronic library system to support the above functions.



Library Use Cases



* Using (include)

» a common subpart is used within a containing use case

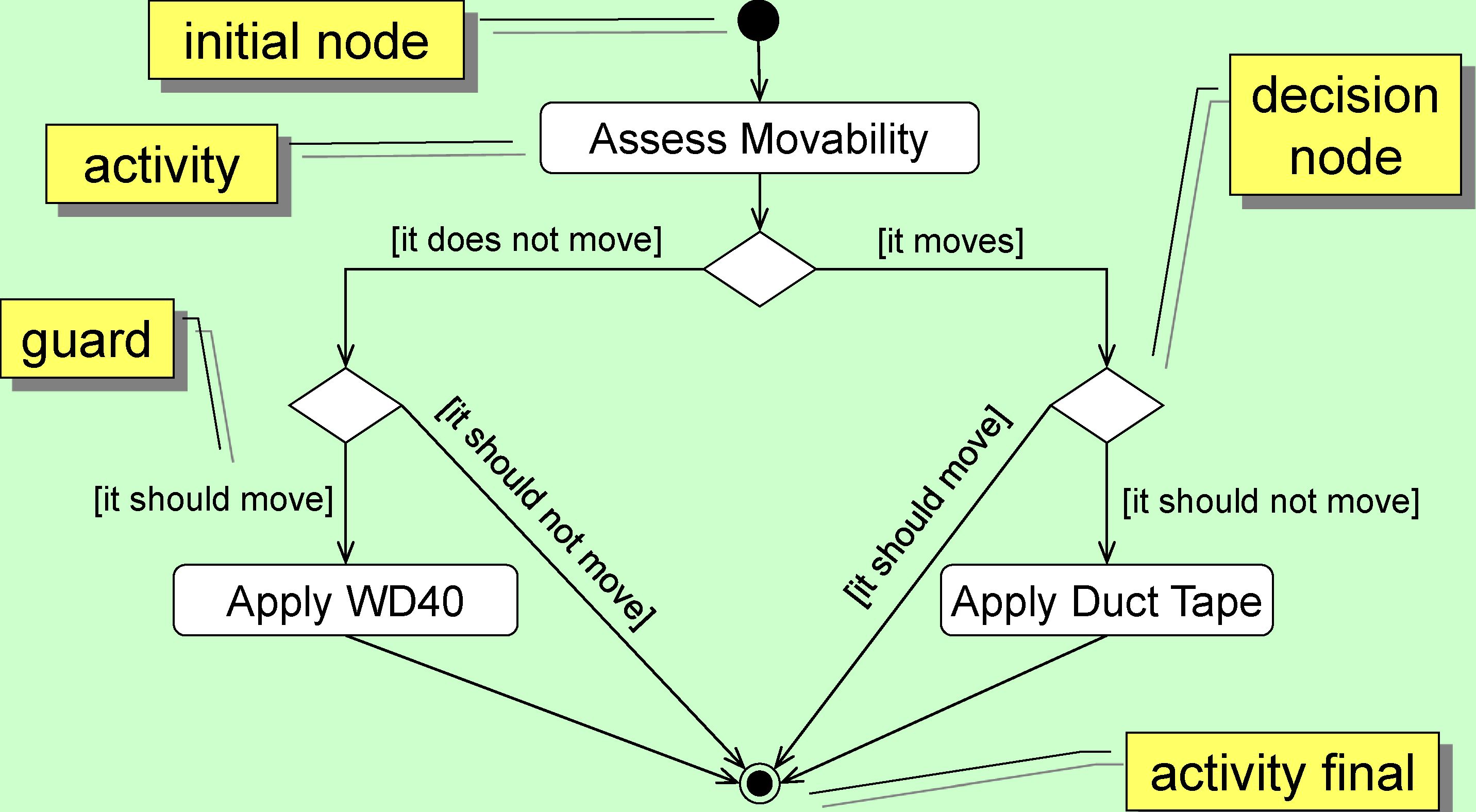
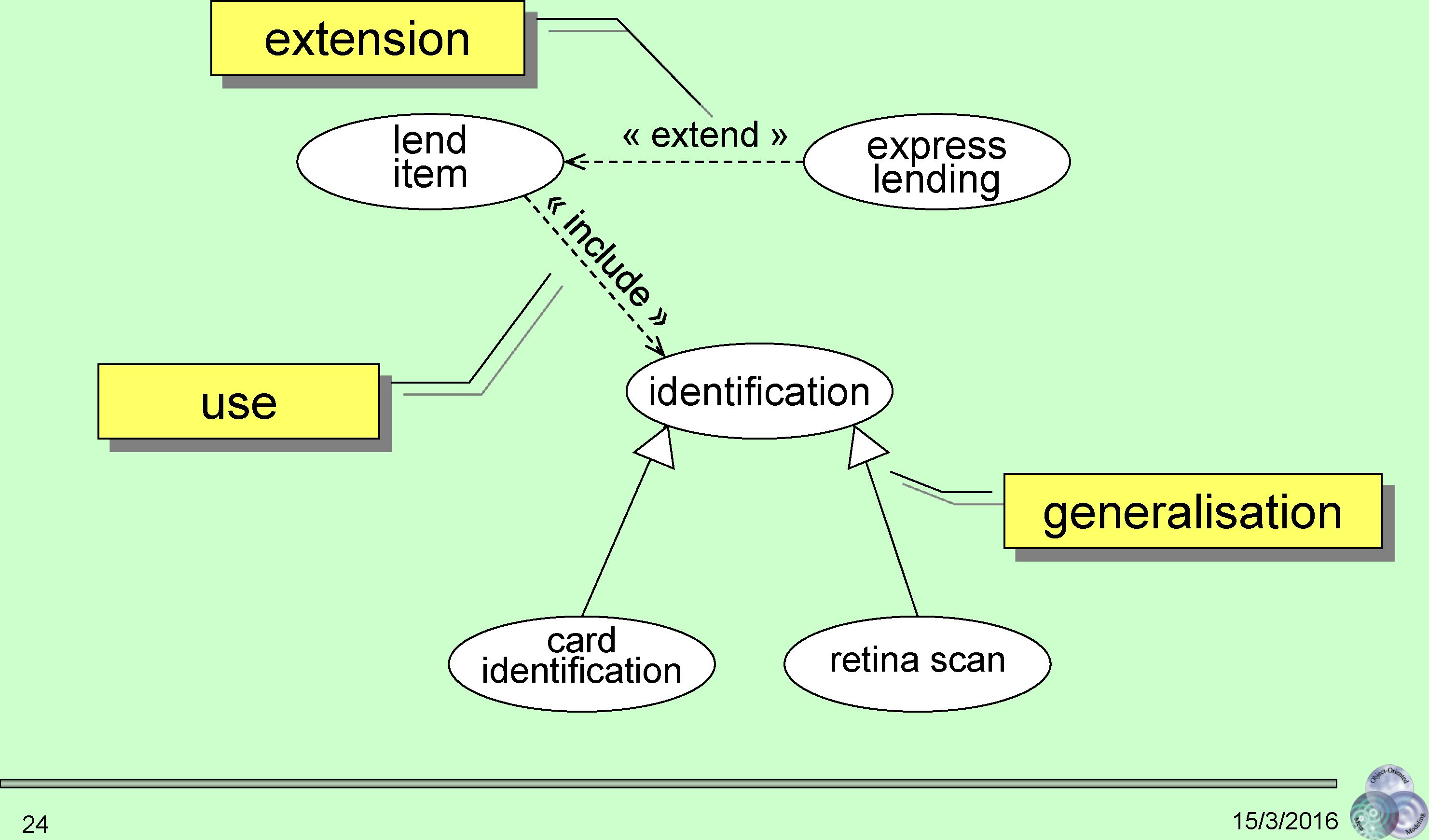
* Extending (extend)

» a variant or exceptional situation extends the normal case

* Generalisation

» several special use cases are generalised to a common general description

—i *<r°\*



Lend Item

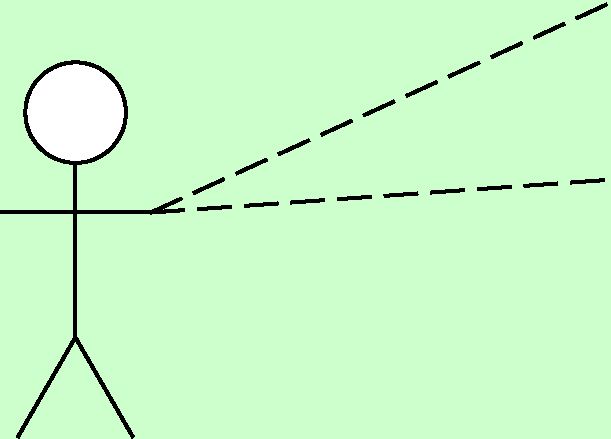
* Identify user

realization details  
of this functionality  
are unimportant at  
this stage

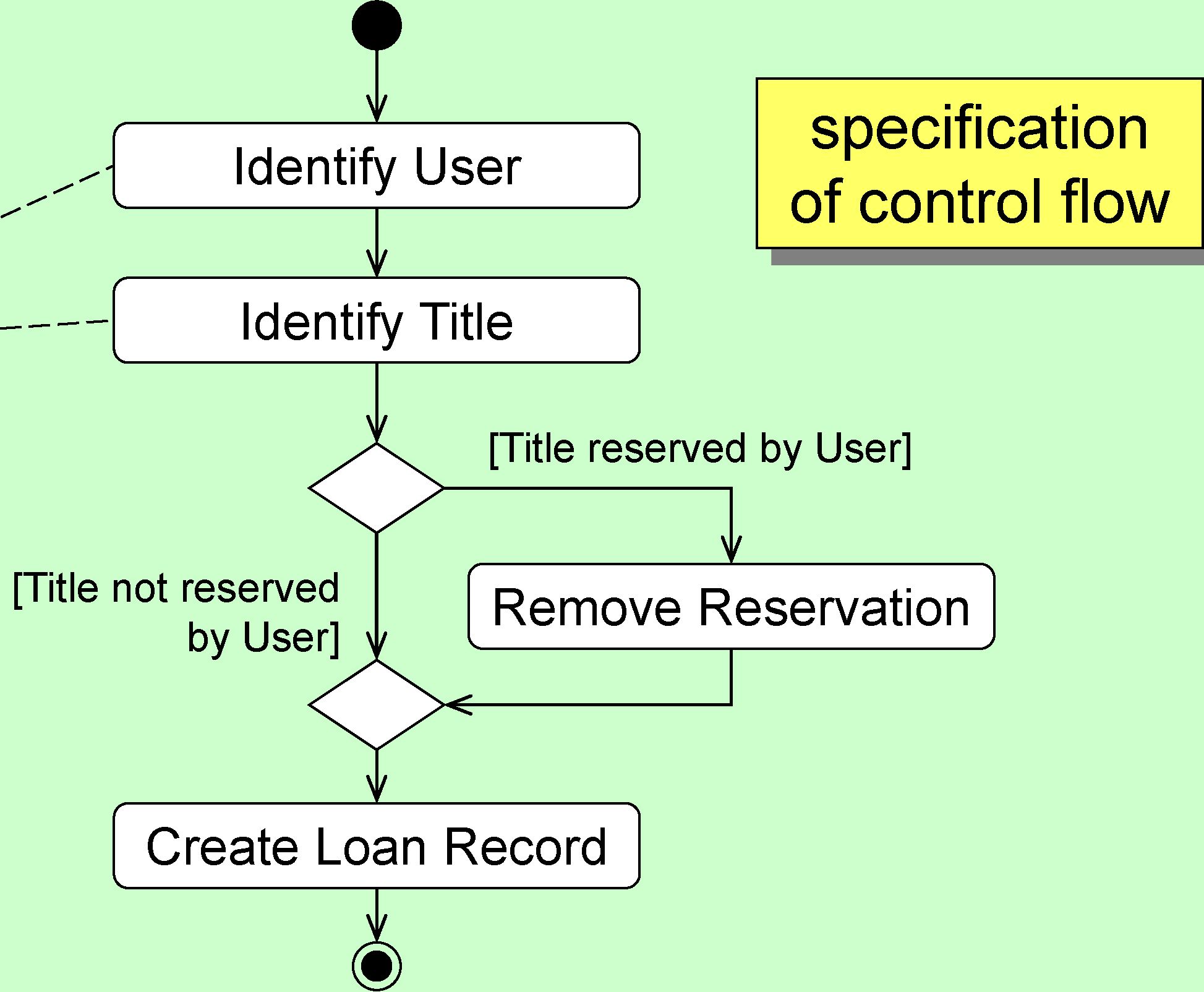
* Identify title
* Has title been reserved by user?

» yes remove reservation

* Create a loan record for lent item
* Lend item to user



Librarian



* System Use Case

» describes user

and

system

at a technical level

* Essential Use Case

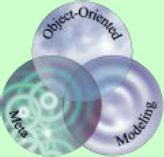
» describes user

and system

Essential Use Case

*“a simplified and generalized form of use case ... intended to capture the essence of problems through technology-free, idealized, and abstract descriptions”*

*- Constantine & Lockwood*



Comparison

System Use Case

System Response

Essential Use Case

System Responsibility

display account menu prompt for amount display amount return card

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| insert card | read magnetic stripe |  | identify self | verify identity |
| enter PIN | request PIN |  | choose WD | offer choices |
|  | verify PIN |  |  | dispense cash |
|  | display transaction menu |  | take cash |  |

User Action

User Intention

press A-key enter amount press D-key

take card



dispense cash

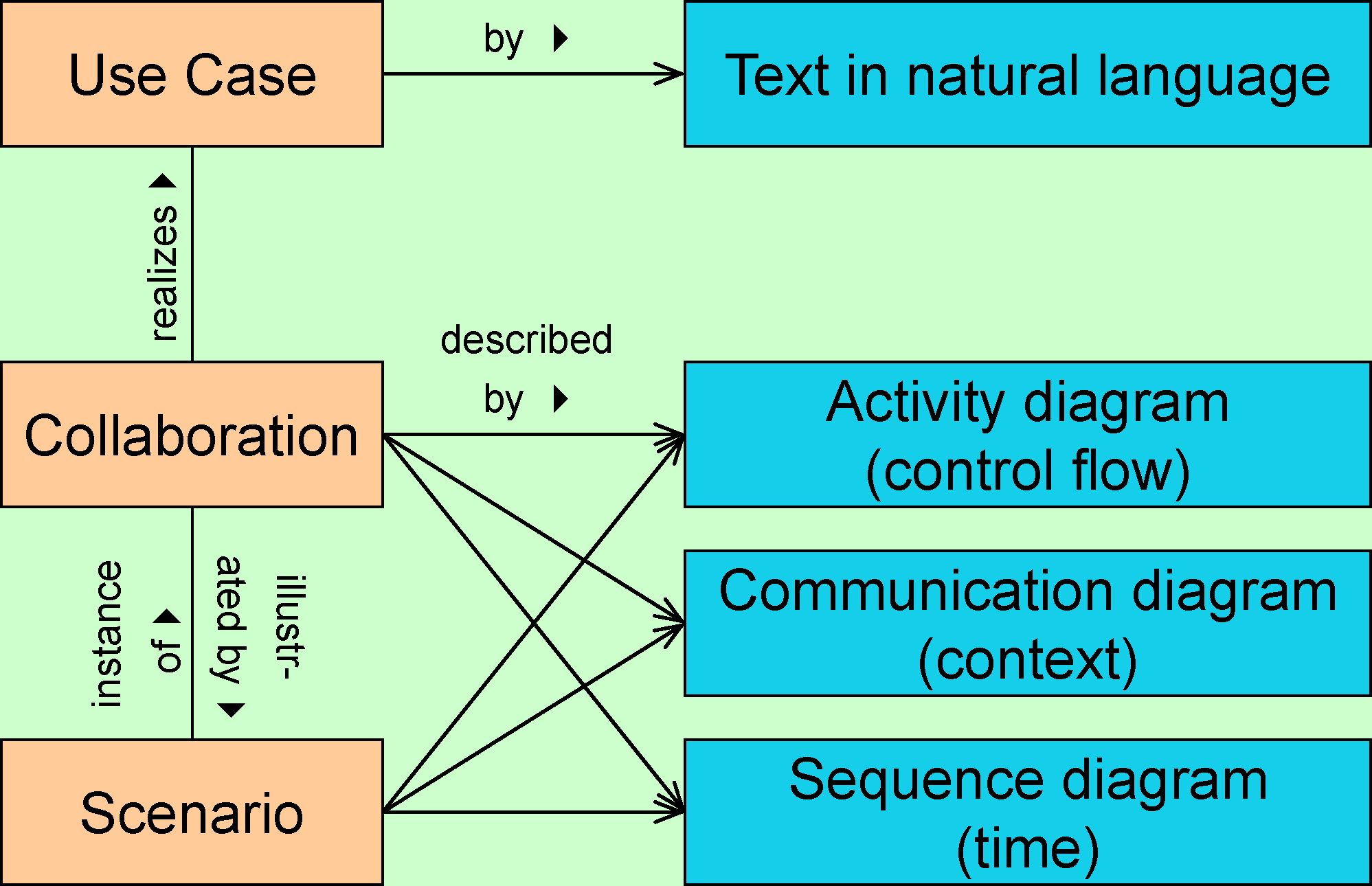


Use Cases: The Big Picture

**Perspective**

**Model type Modeling Notation**

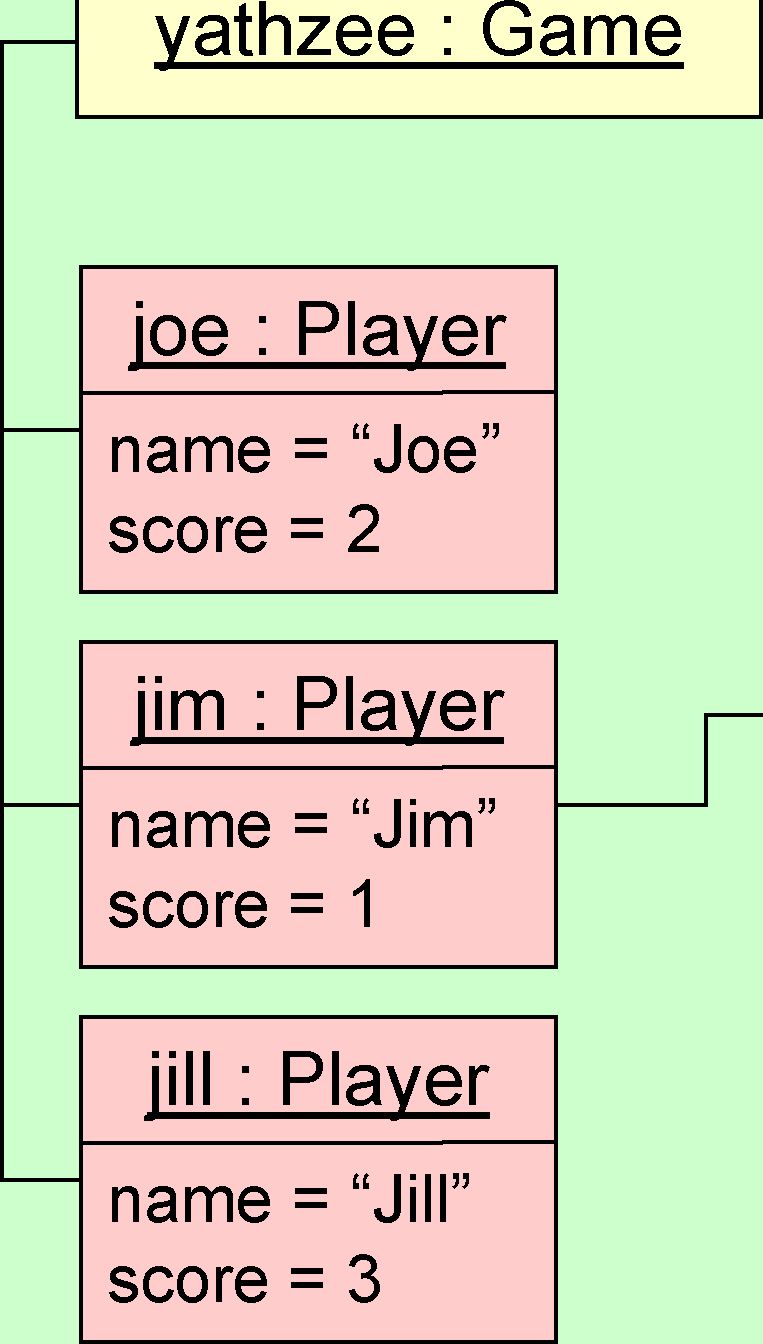
described



Outside the system (Actor’s view)

Inside the system (realization)

Execution path through the system

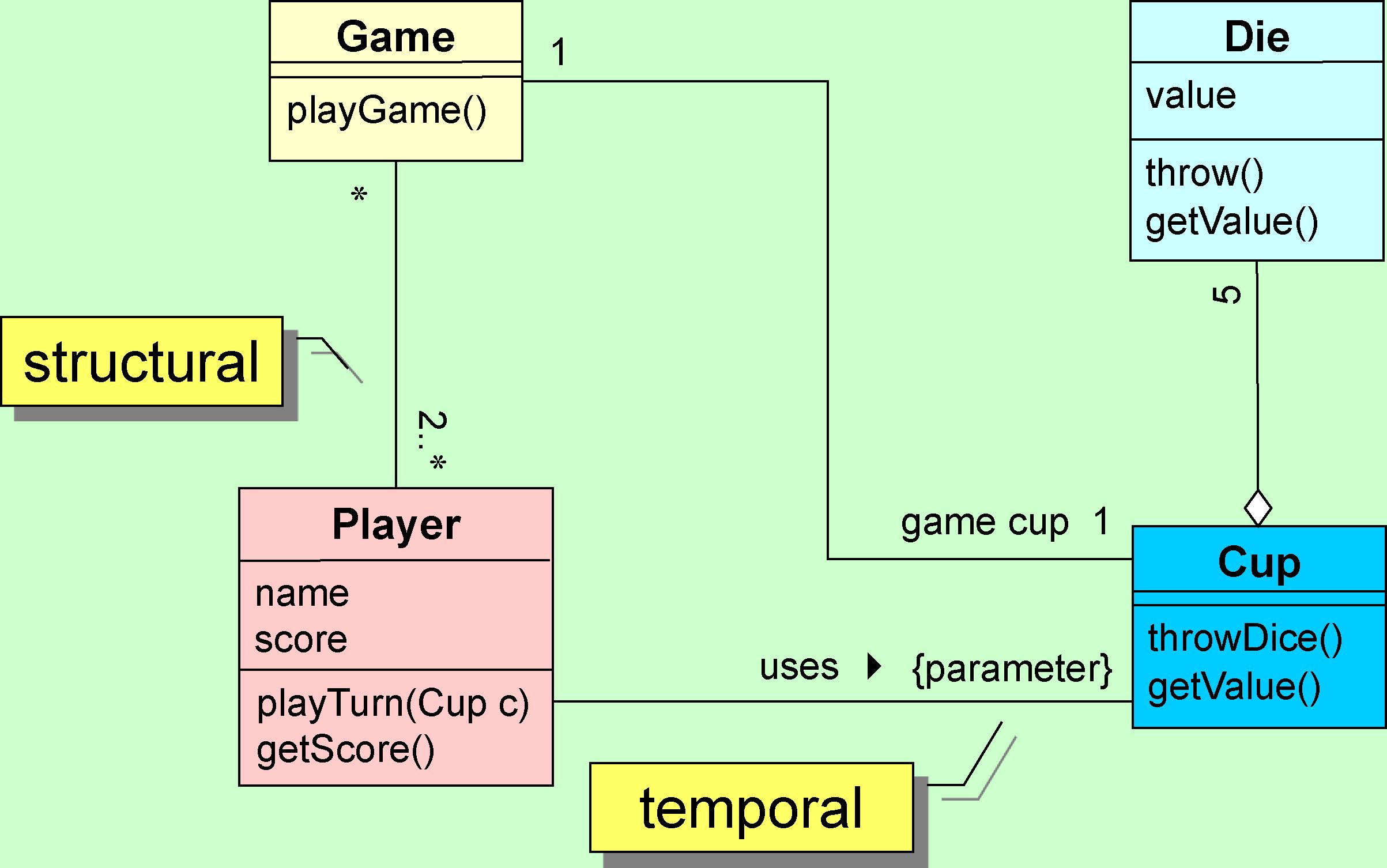


|  |  |  |
| --- | --- | --- |
|  | |  |
|  | teacup : Cup | |
|  |  | |

|  |  |
| --- | --- |
|  | diel : Die |
|  | value = 4 |
|  | |
|  | die2 : Die |
|  | value = 1 |
|  | |
|  | die3 : Die |
|  | value = 4 |
|  | |
|  | die4 : Die |
|  | value = 3 |
|  | |
|  | die5 : Die |
|  | value = 6 |



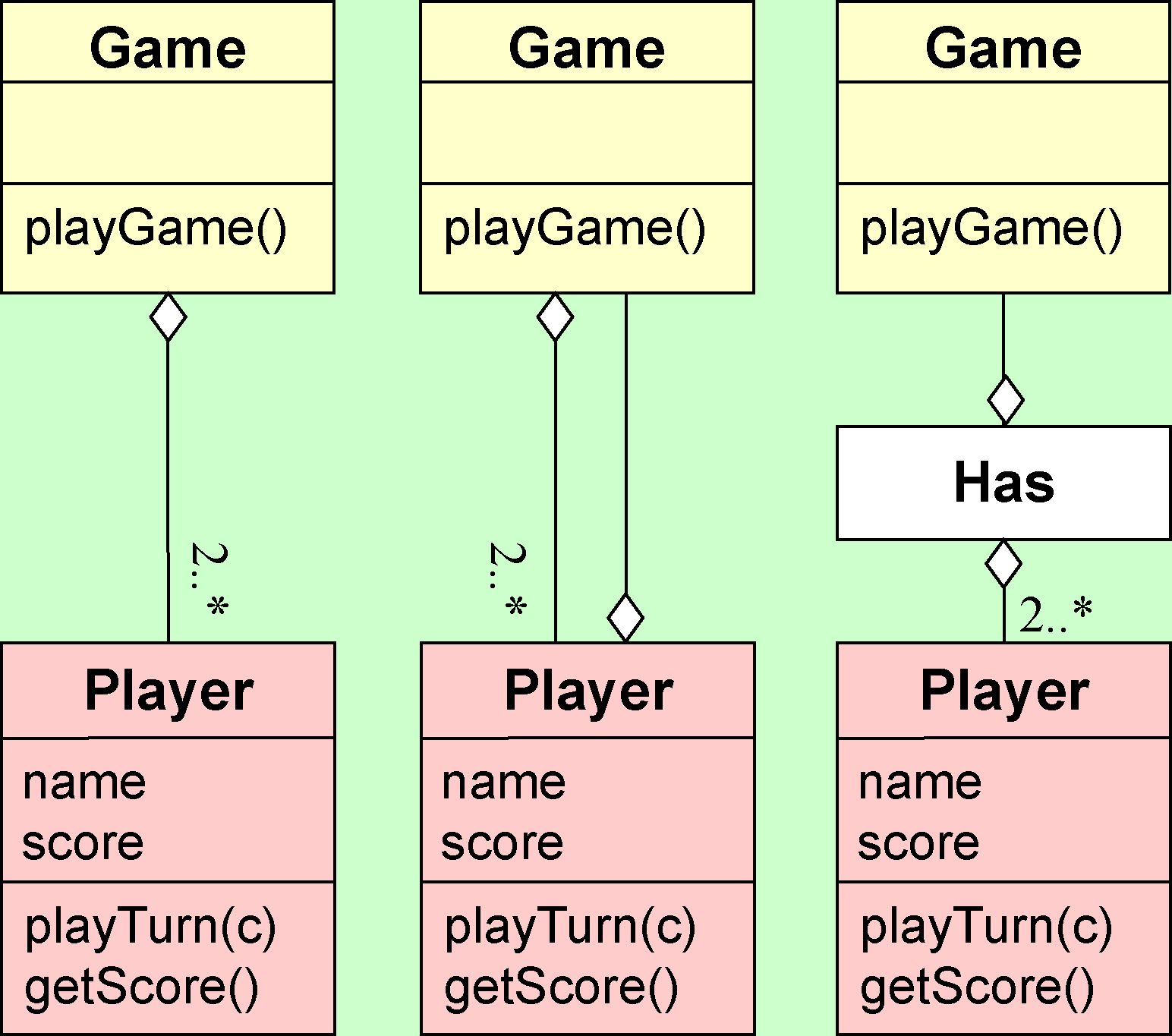
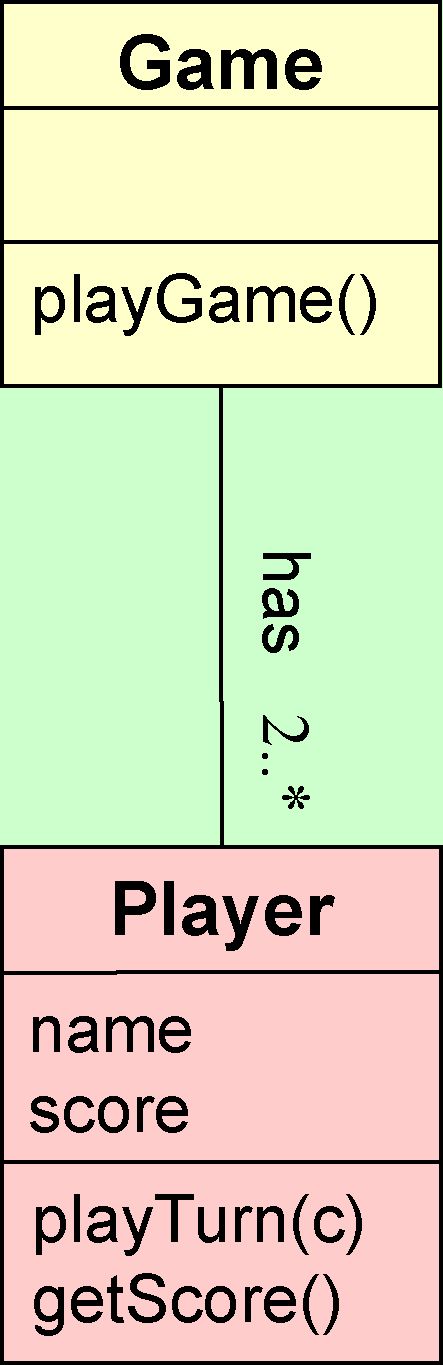
\*



**iKN**

Analysis vs Design

• The association... ...could be implemented as

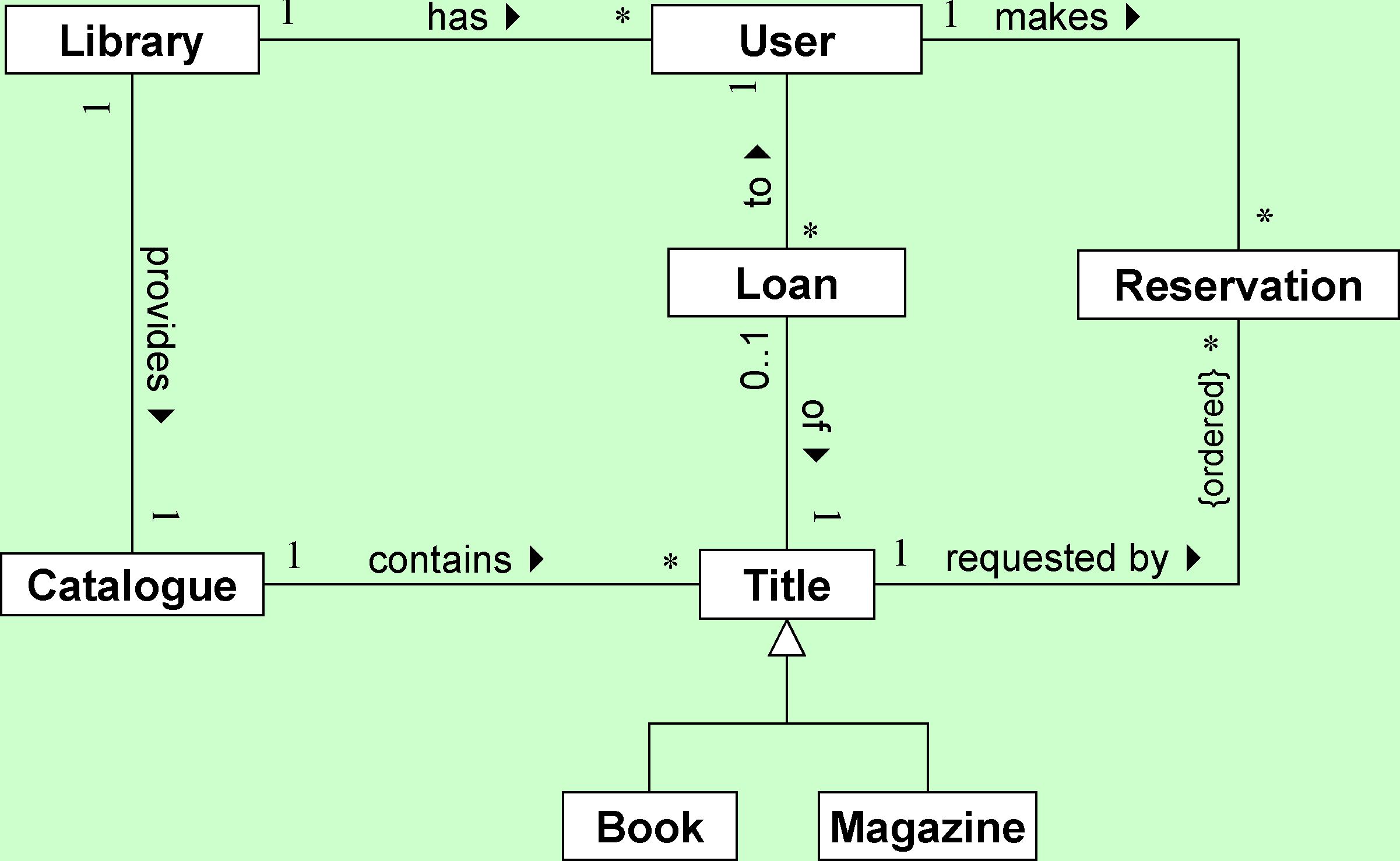


sqpi Finding Concepts

CRC Cards

|  |  |
| --- | --- |
| Title | |
| **record basic publication information** |  |
| **know the max. tending period** | **Book, Magazine** |
| **record reservations** | **Rese**r**v**a**ti**on |
| **record loans** | L**o**an |

Analysis: Library System



emphasises  
the structural  
aspect of  
interactions

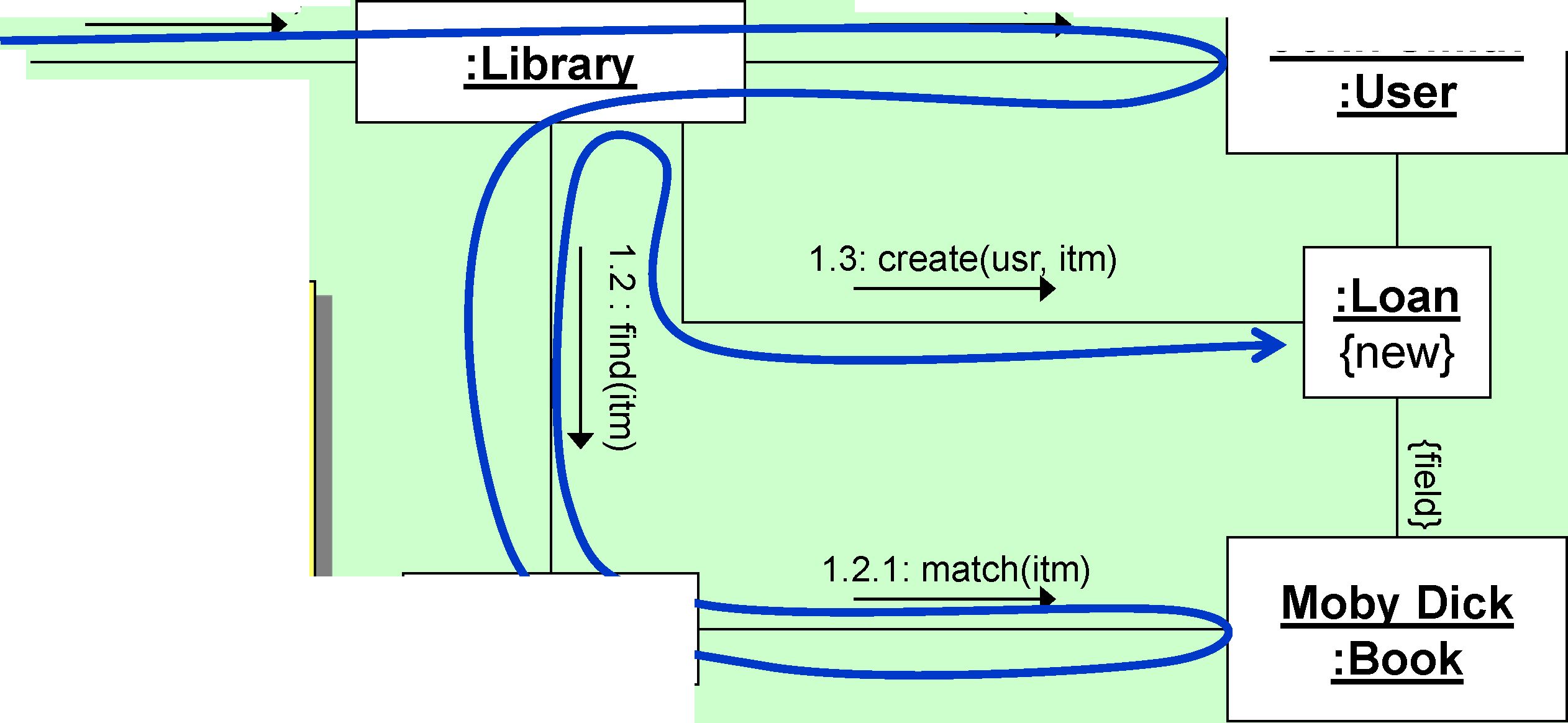
1: lend(usr, itm)

1.1: match(usr)

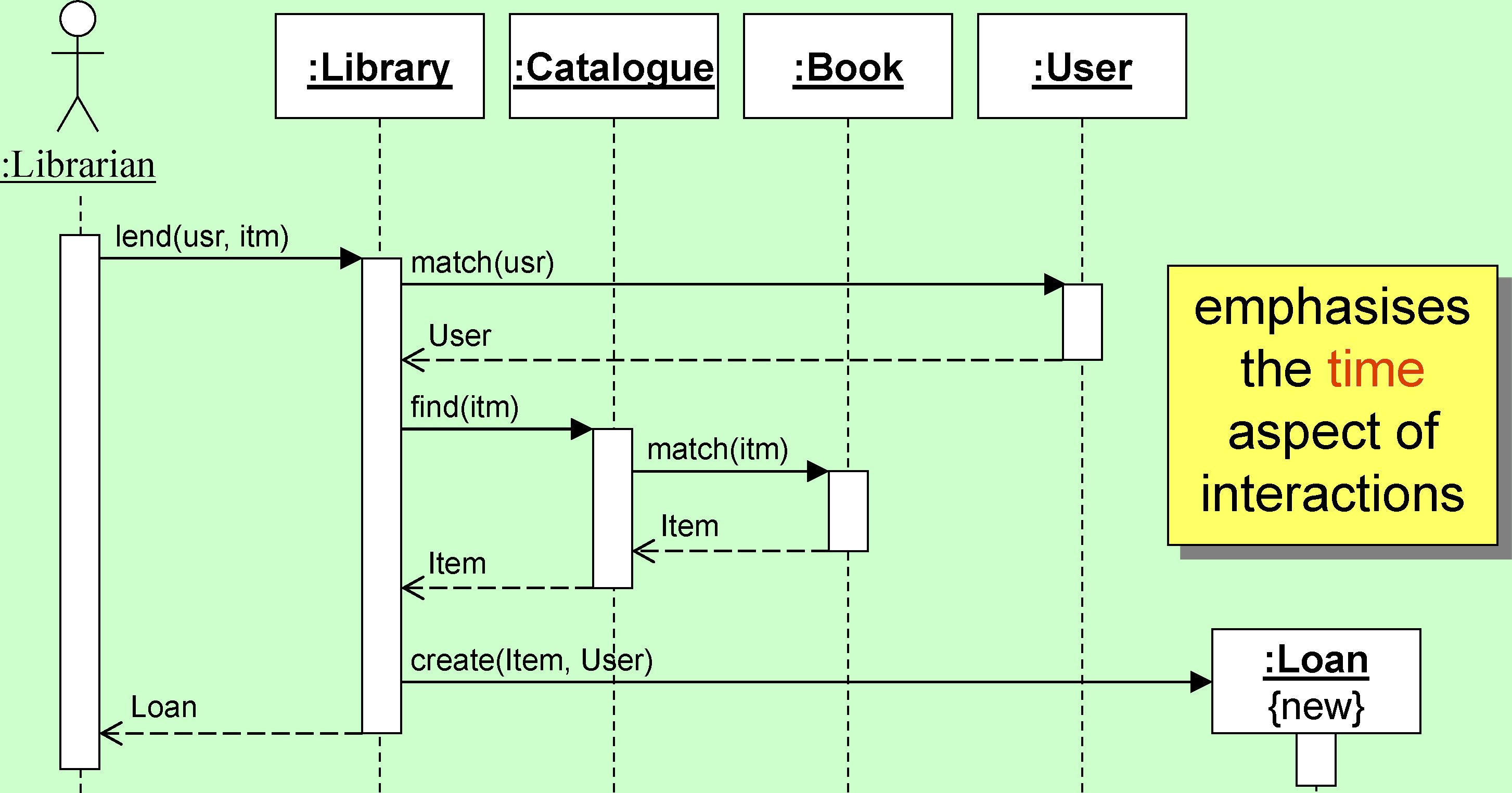
A

[Librarian

John Smith



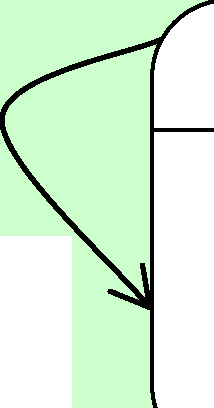
|  |  |
| --- | --- |
| :Cata |  |
|  | |



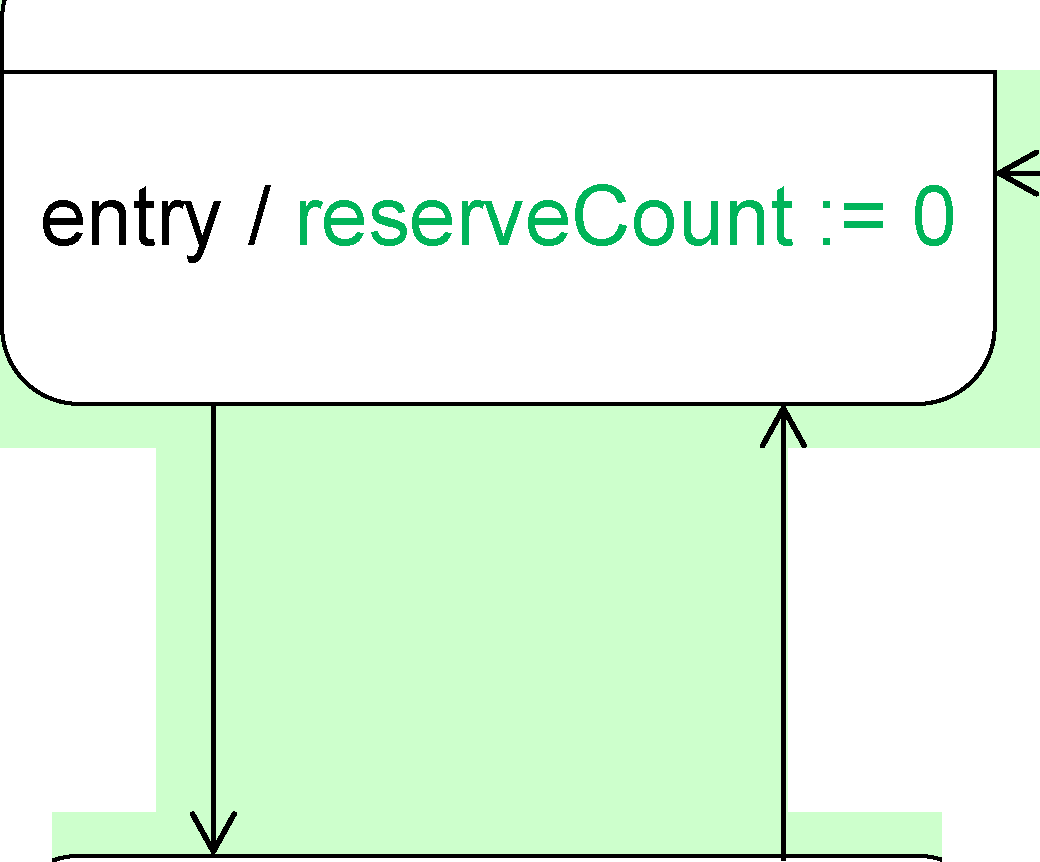
useful for  
specifying  
reactive  
behaviour

title reservation  
/ increment reserveCount

title reservation  
/ increment reserveCount



NotReserved

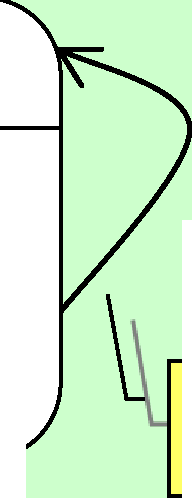


reservation removed  
[reserveCount = 1]

/ decrement reserveCount

Reserved

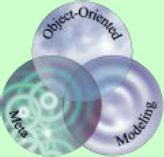
entry / “keep at counter” exit / “item back on shelf”



reservation removed  
[reserveCount > 1]

/ decrement reserveCount

should be internal!

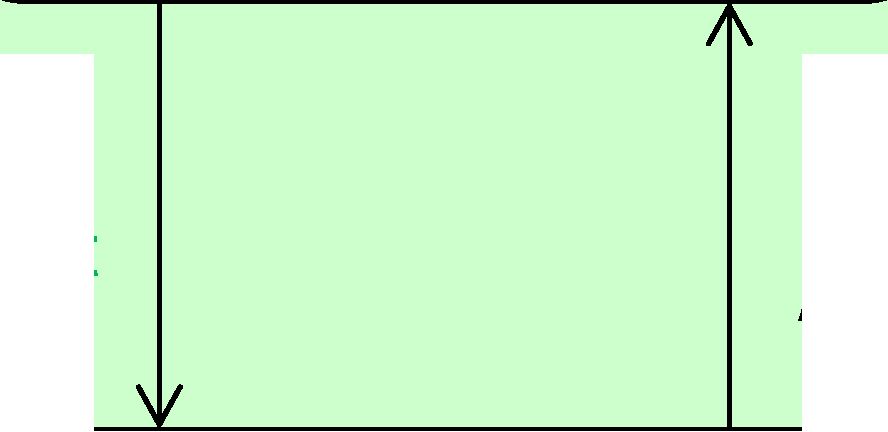


NotReserved

internal  
transitions

<-

entry / reserveCount := 0



title reservation

l **\**

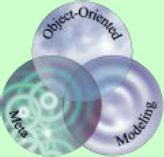
/ increment reserveCount

reservation removed  
[reserveCount = 1]

/ decrement reserveCount

Reserved  
entry / “keep at counter”  
title reservation / increment reserveCount  
reservation removed [reserveCount > 1] / decrement reserveCount

exit / “item back on shelf”



**DigitalWatch**

+ modeButton()

^—a-

inc/sec := sec + 1

SetTime

inc/day := day + 1

SetDate

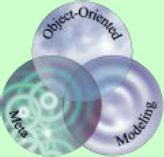
modeButton

do / display time

do / display date

modeButton

**a**

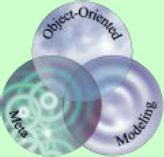


What is Design?

* Architectural design

» determine overall architecture » subsystem identification » tentative efficiency considerations

* Detailed design



System Design

* Architecture (system topology)

» MVC, Pipes & Filters, Blackboard

* Subsystems (system breakdown)

» layers & partitions

* Concurrency (system threads)

» independent subsystems » task identification



Software Architecture

*A software architecture is a description of the subsystems and components of a software system*

*and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a*

*software system.*

*The software architecture of a system is an artifact. It is the result of the software design activity.*

Buschman et al. (1996)

**4KN**

Two Architecture Views

* Logical Architecture

» system functionality » components & their relationships » collaborations

* Physical Architecture

» location of classes (objects) in modules (processes)

» location of processes (servers, etc.)

» network design

i

15/3/2016

Components & Nodes

\

\

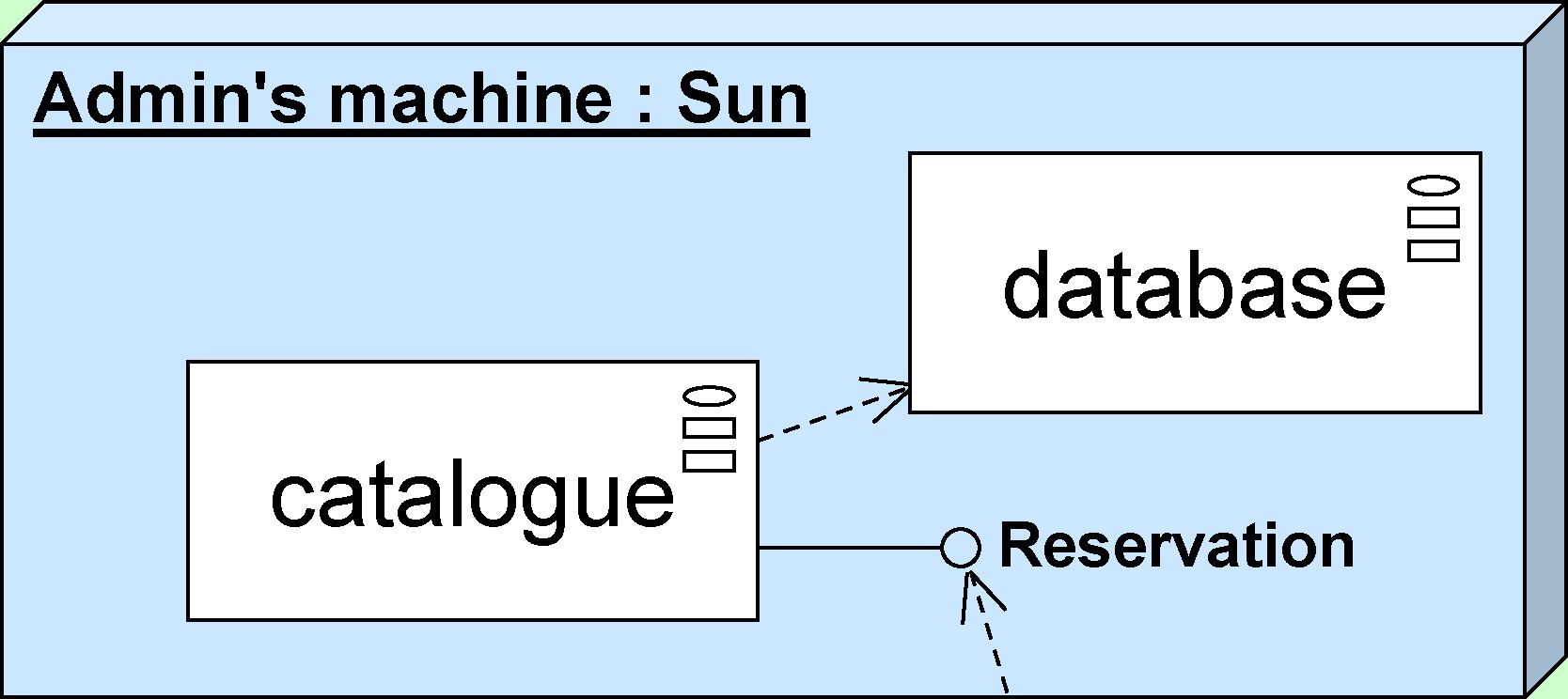
\

\

\

\*—

\



/

**Joe’s machine : PC**

71

\

\

\

\

\

\

browser

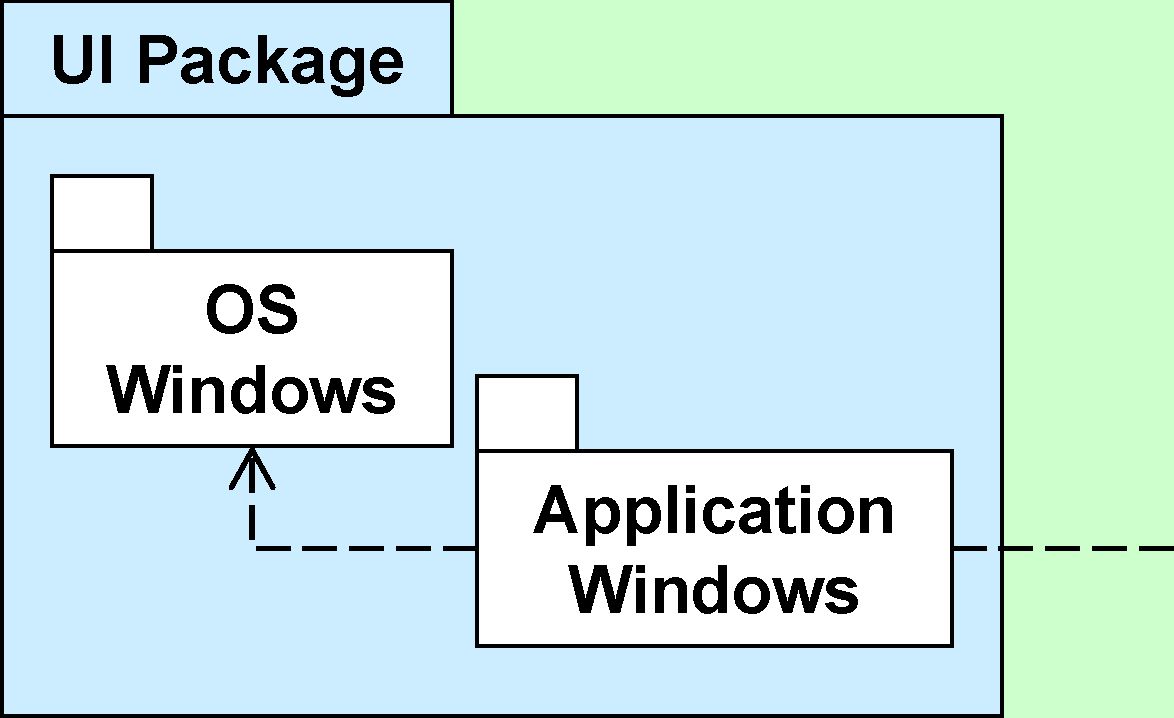
J/



\*



qpi "Three-Tier" Architecture



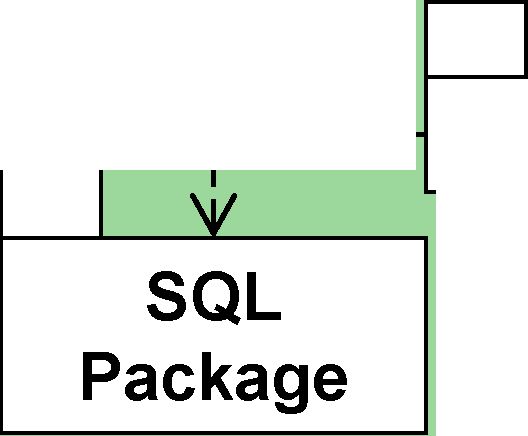
Business Objects  
Package

> Interface Objects

\_\_i

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | i  y |  |  |  |
| Entity  Objects | | <— | Control  Objects | |

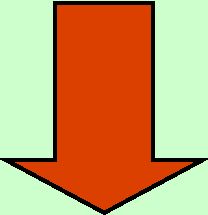
Database Package



Adapter



Layers & Partitions



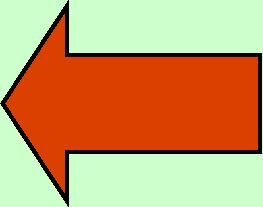
layers depend on each other

Layers

application

|  |  |  |
| --- | --- | --- |
| user  dialog  control | window graphics | simulation |
| pixel graphics | package |
| operating system | | |

computer hardware



partitions do not depend on each other

►

Partitions



OSI Model Layers



|  |  |
| --- | --- |
| Data | Layer |
| **CData 3** | r Application >  **Network Process to**  V **Application** j |
| Data ) | ( Presentation ^  **Data Representation**  V **and Encryption** j |
| S ( Data ^ | ( Session x  **Inlerhost Communication**  L J |
| ( Segments | ( Transport ^  **End-to-End Connections ^ and Reliability** J |
| Packets^) | ( Network ^  **Path Determination ^ and IP (Logical Addressing)** J |
| **J** fFrames '  5 V J | f Data Link  **MAC and LLC (Phyiscal addressing)** J |
| \* (BitS J | ( Physical \  **Media, Signal, and**  **Binary Transmission** J |

* Architectural design

» determine overall architecture » subsystem identification » tentative efficiency considerations

* Detailed design

» adding implementation details to analysis » ensure performance » cater for reusability and maintenance



Planning for extensions...

* Associations design

» pointer, double pointer, or explicit object design

* Algorithm design

» find constructive & efficient solutions

* Inheritance

» abstract common behaviour

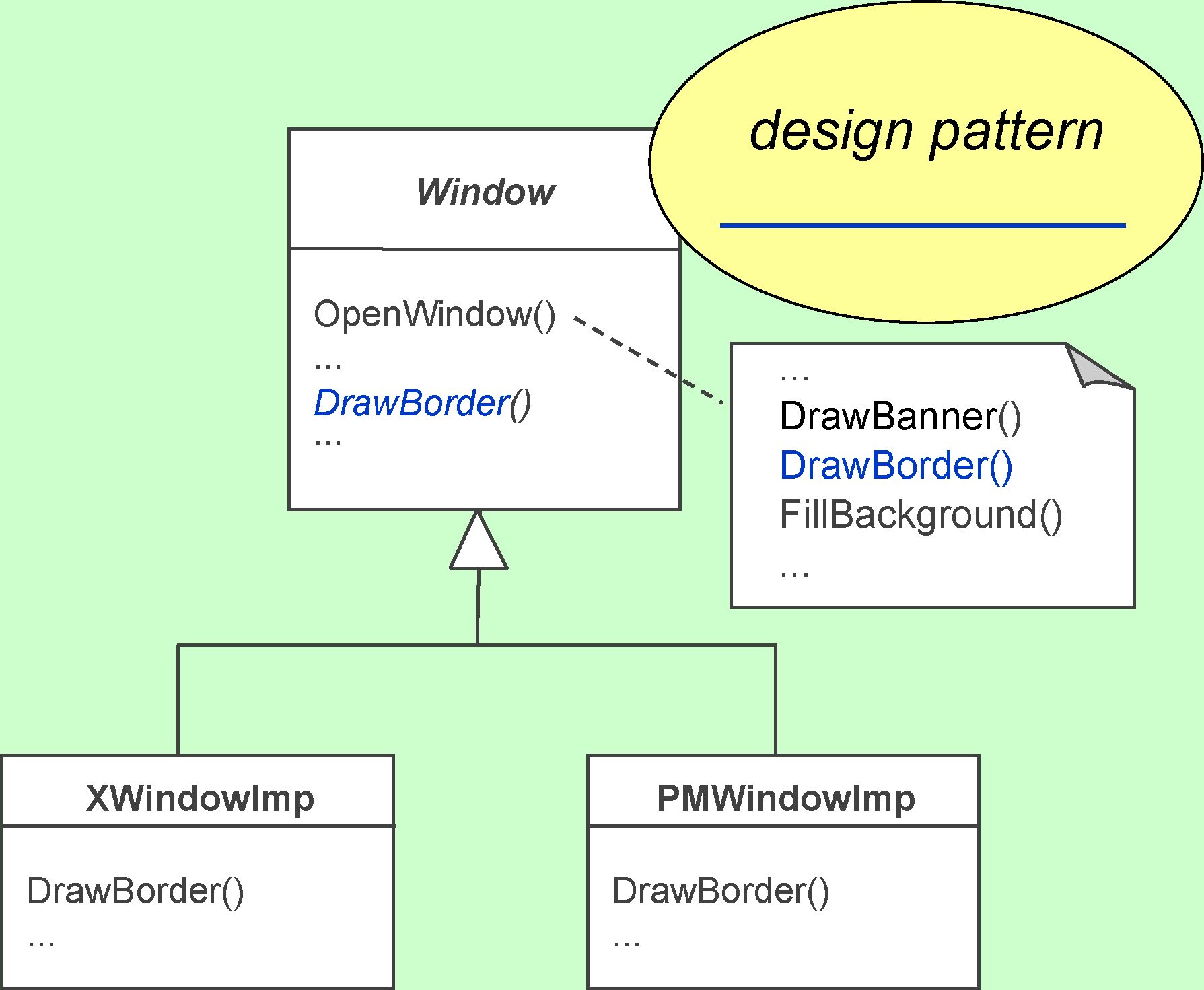
* Module identification

» information hiding

—i *<r°\*

Behaviour Abstraction

* concentrate common behaviour



* defer

responsibilities

* cater for extensions

» unify method names



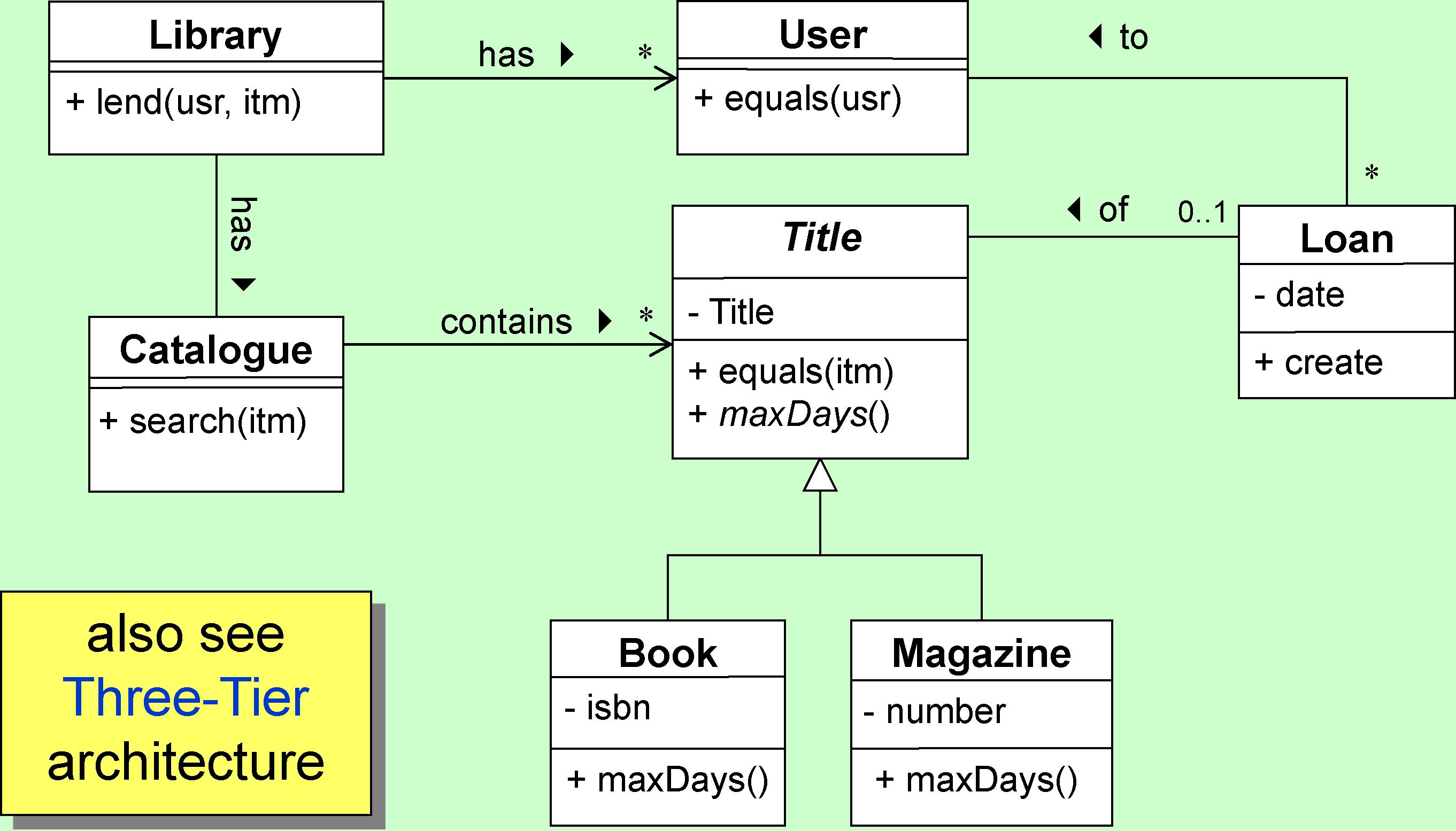
Information Hiding

* Make attributes & operations private
* Limit association traversal
* Decrease coupling

» program to an interface, not to an implementation

* Increase cohesion

» do one thing well, no Swiss Army knives » divide policy and implementation



Algorithm Design

* Find constructive descriptions

» model will only specify/constrain the result

* Provide efficient solutions

» appropriate data structures » reduce computational complexity » efficient strategies

» consider “caching” and other optimisations techniques



(Don’t) Optimise

* An efficient system is to be preferred over an inefficient one
* However,

Optimisations Endangers

» 1. Law: Don’t optimise

» 2. Law: If you must, do it late

» 3. Law: Only where it really pays off (profiling)



Summary

|  |  |
| --- | --- |
| Analysis | Design |
| • Requirements  » what shall the application do? | • Architecture Design  » plan of attacking the solution |
| • Modeling  » what are the domain concepts? | • Detailed Design  » detailed basis for implementation |
|  |  |

Summary

Static View

* Use Cases

» system functionality

* Conceptual Model

» entities » relationships

Dynamic View

* Interaction Diagrams

» sequence » communication

* Activity Diagrams
* State Diagrams



Semantic Gap

* as hard as one may try to check implementations against watertight specifications...
* ... there is no way to formally win against the

Requirements Specification

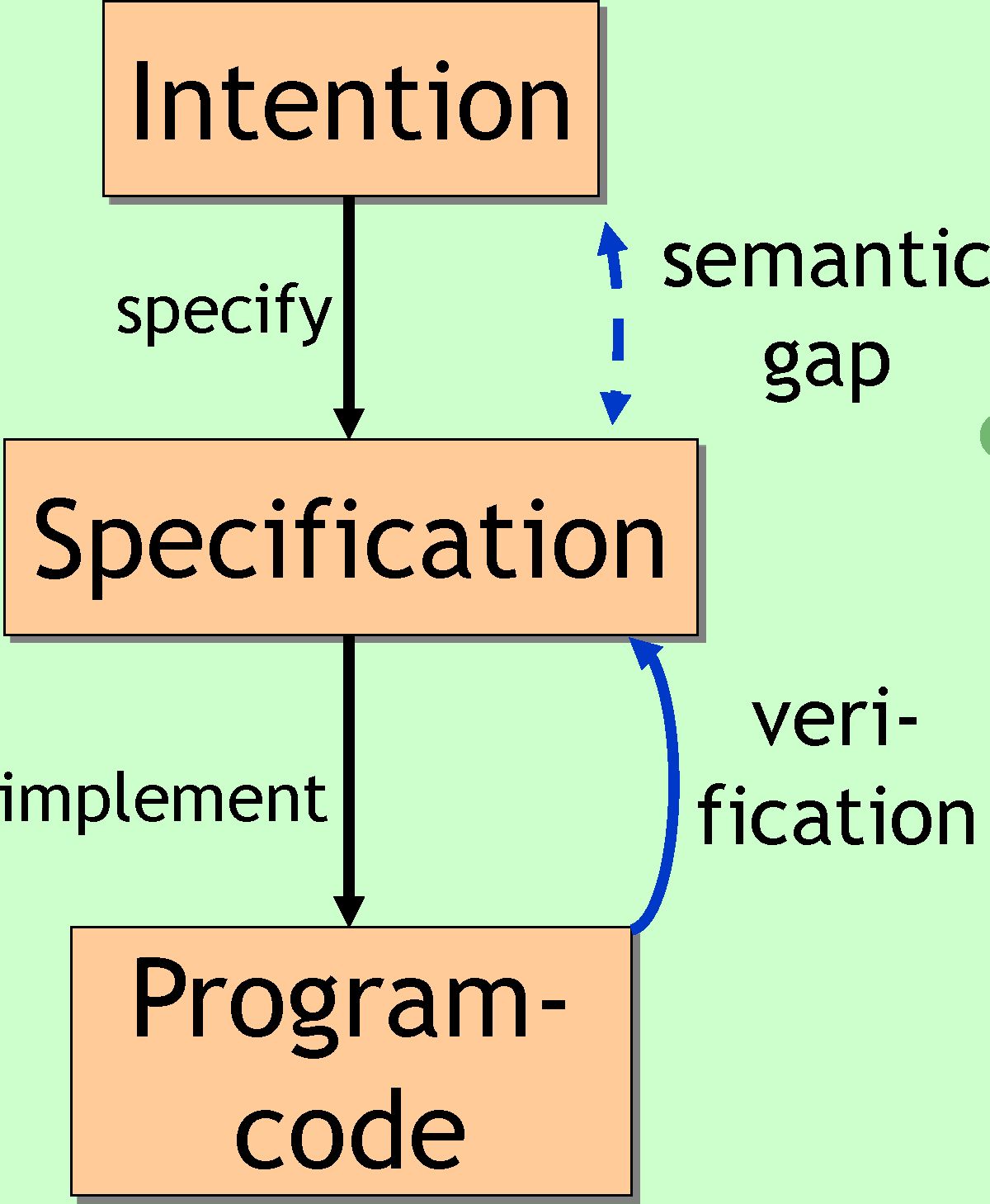
* it is a very challenging task to find out whether a specification really expresses the requirements
* a correct implementation can't help this problem

—i

Intention versus Specification

• Intention

» what is desired?



» not necessarily expressed by specification

» check if program is a

realization of the specification

» cannot compare against original intention

» specification may contain errors as well, but is simpler than code -> easier to check

