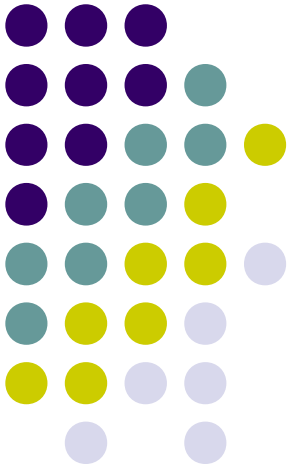
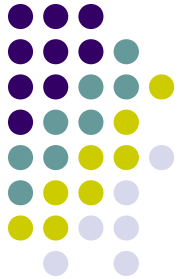
Understanding Source Package Organization Using the Hybrid Model

Xinyi Dong and Michael W. Godfrey
Software Architecture Group (SWAG)
David R. Cheriton School of Computer Science
University of Waterloo



University of Waterloo

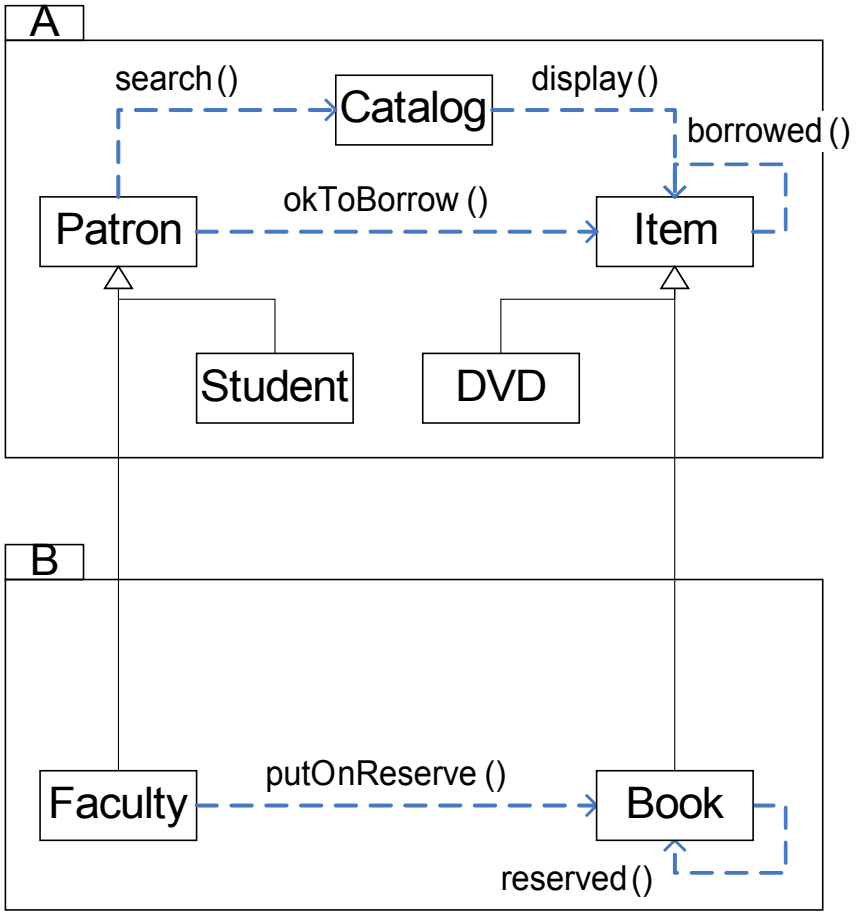
Introduction



- Objects are
  - the basic units of abstraction in OOD.
  - defined by classes.
  - too fine-grained to model large systems.
- Packages are
  - groups of classes.
  - coarse-grained logical design units.
- Understanding package organization is key to understanding high-level design.

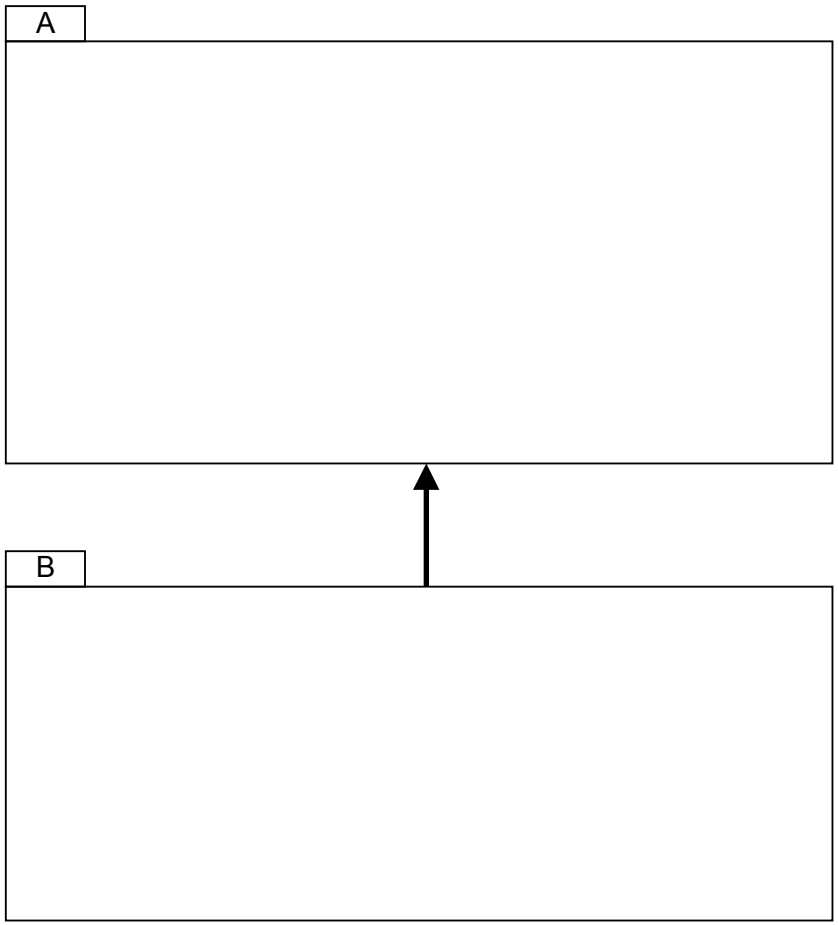
Package Diagram

- A Package Diagram
  - Captures structural dependencies.
  - Contains little semantic information.

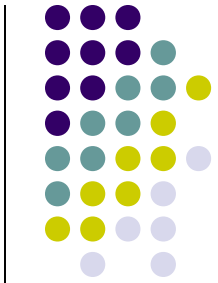


Package Diagram

- How is the functionality implemented in package *A* used by package *B*?
- Why the system's classes are divided into this set of packages?
- Is package *B* a design abstraction or just a contain?
- Is package *B* a part of a bigger abstraction?

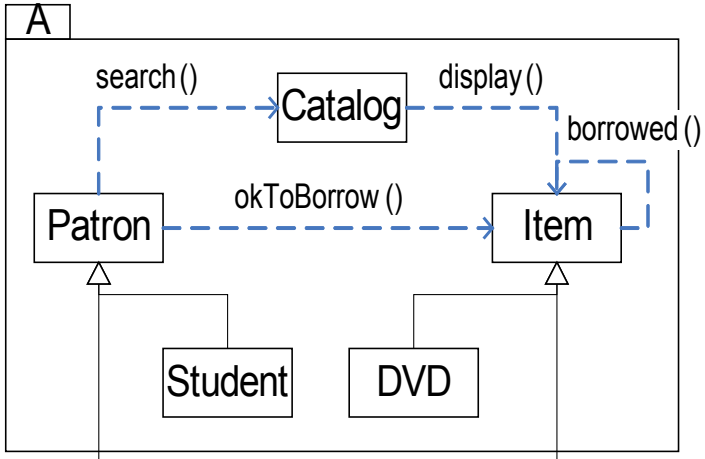


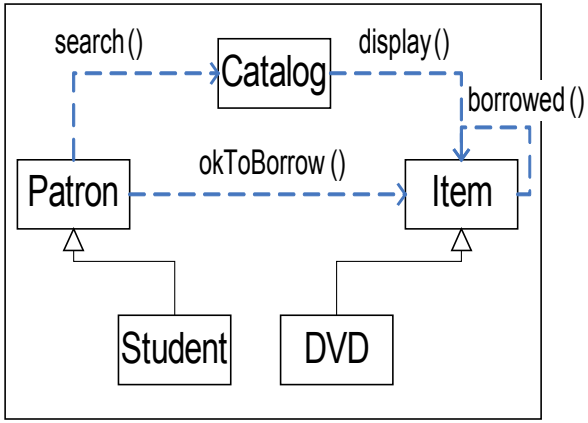
The Hybrid Model

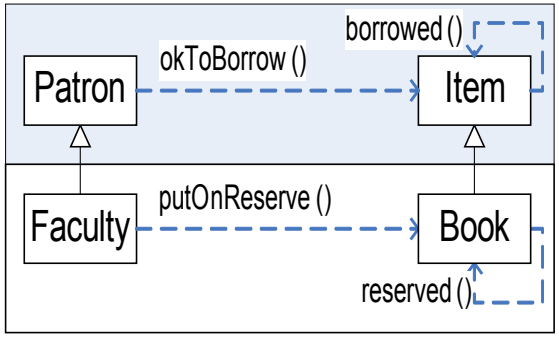


- A Hybrid Model enriches a package diagram with semantic information.
  - A package represents the objects that can be instantiated from concrete classes it contains.
  - Ports of a package capture the interface through which objects interact with its environment.

Library Management System(1)

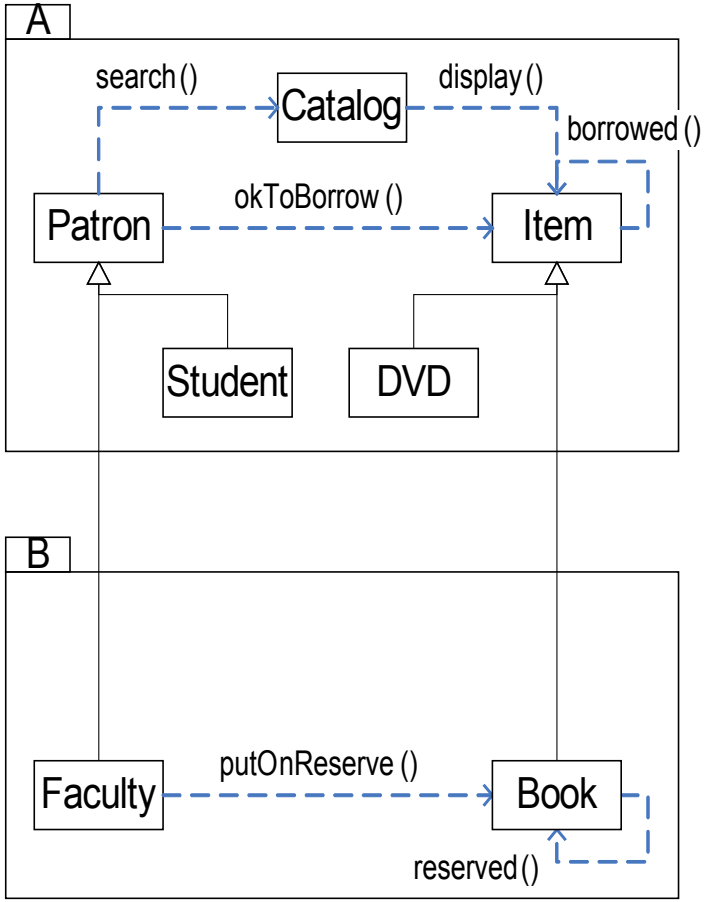




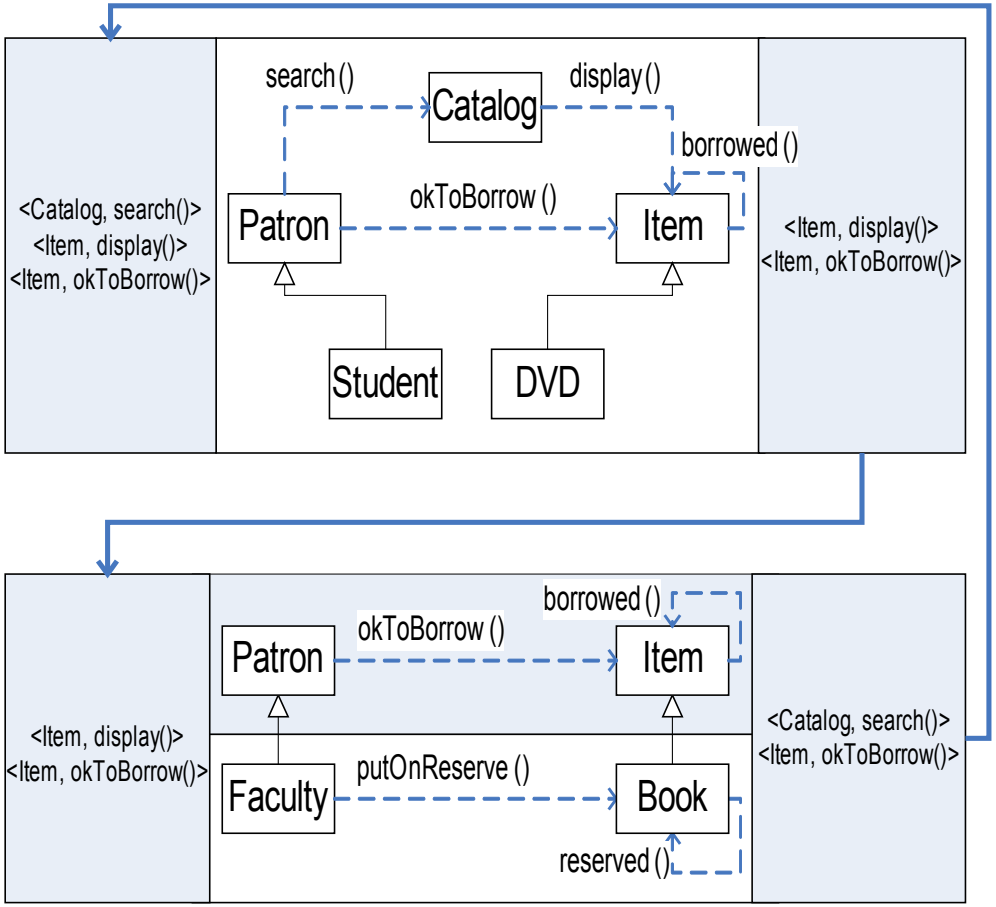


UML Package Diagram

Library Management System(2)



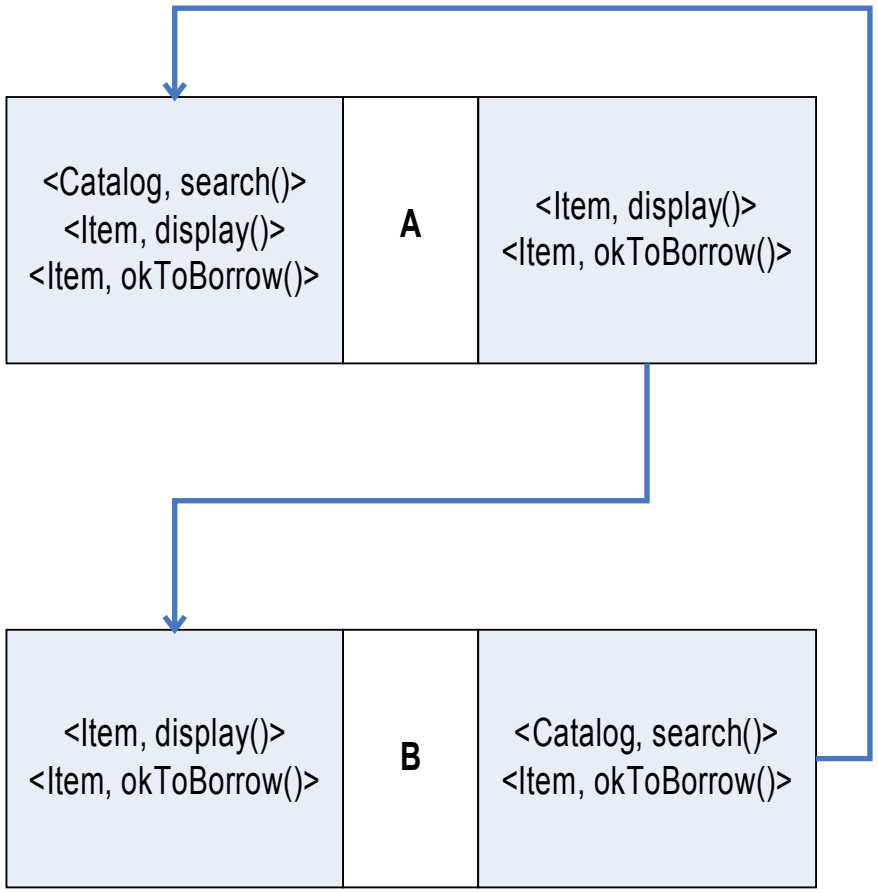
UML Package Diagram



Hybrid Model

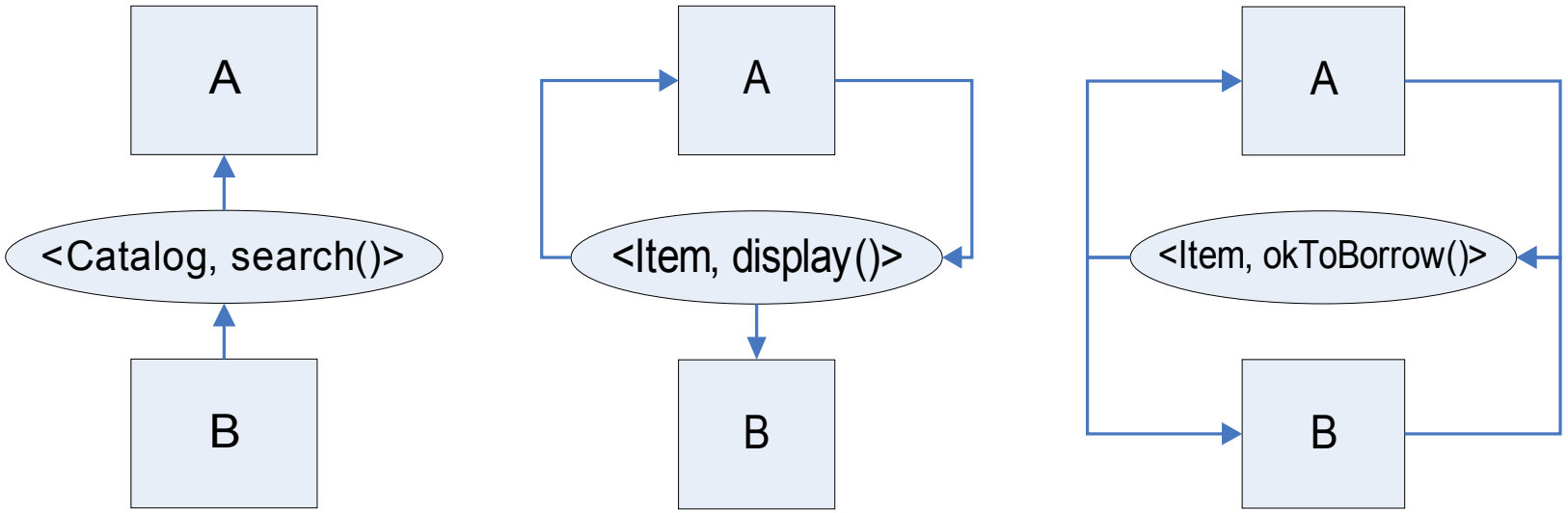
Ports

- An ***inport*** is the interface through which a package provides resources to others.
- An ***outport*** is the interface through which a package requires resources from others.

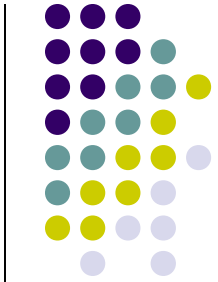


Connectors

- A connector specifies the client-server relationship between two packages.



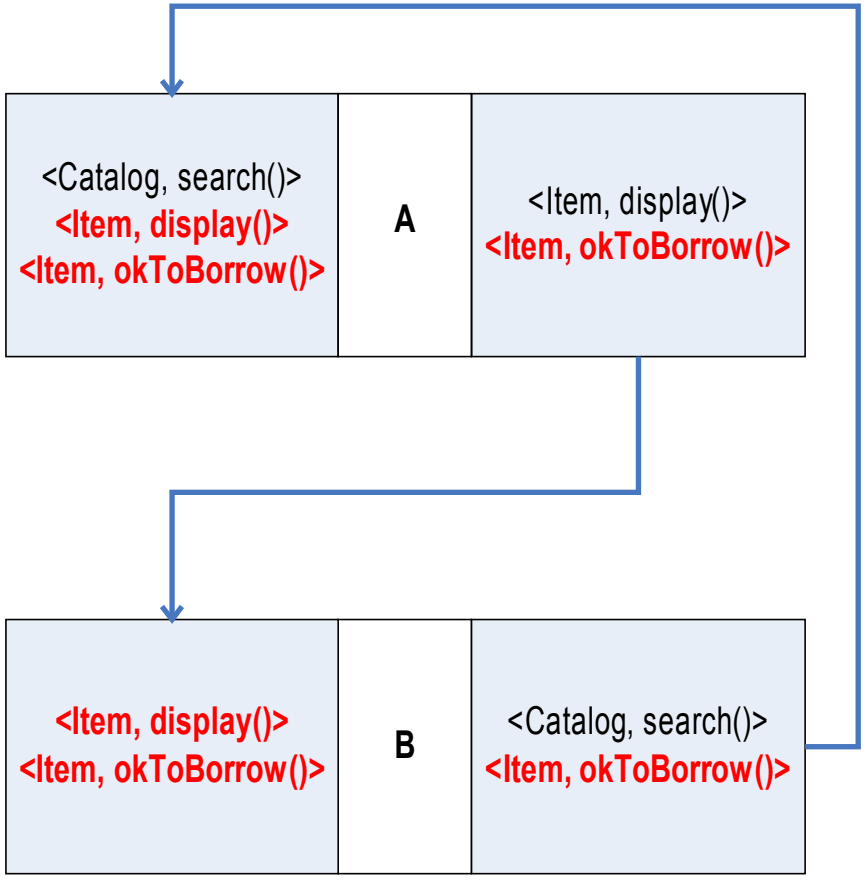
Analyzing Package Design Using the Hybrid Model



- We examine package relations from two perspectives:
  - Resources associated with a port
    - Similarity in ports
  - Resources associated with a connector
    - Collaboration patterns

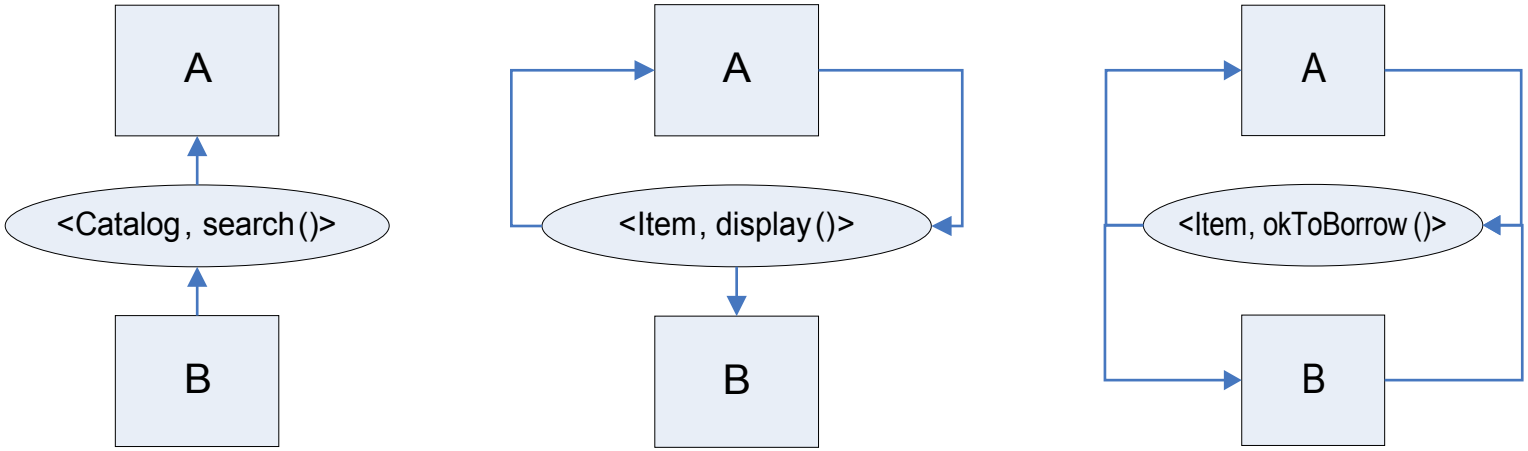
Similarity in Ports

- High similarity in inports
  - Two packages contain implementations of the same set of concepts.
  - E.g. 9 sub-packages in *org.opencms.db*
- High Similarity in outports
  - Two packages may have similar behaviors.
  - E.g. 3 sub-packages in *org.opencms.xml*



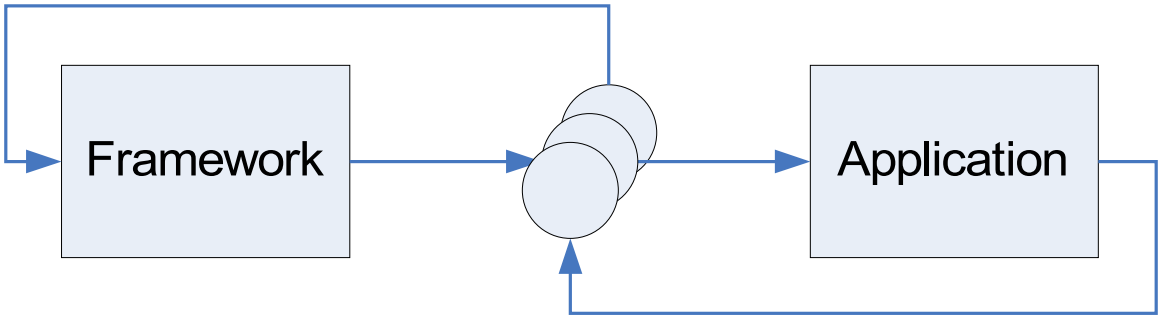
Collaboration Patterns

- How is a resource shared among packages?
  - 1→1 : One consumer one provider
  - m→1 : Many consumers one provider
  - 1→n : One consumer many providers
  - m→n : Many consumers many providers

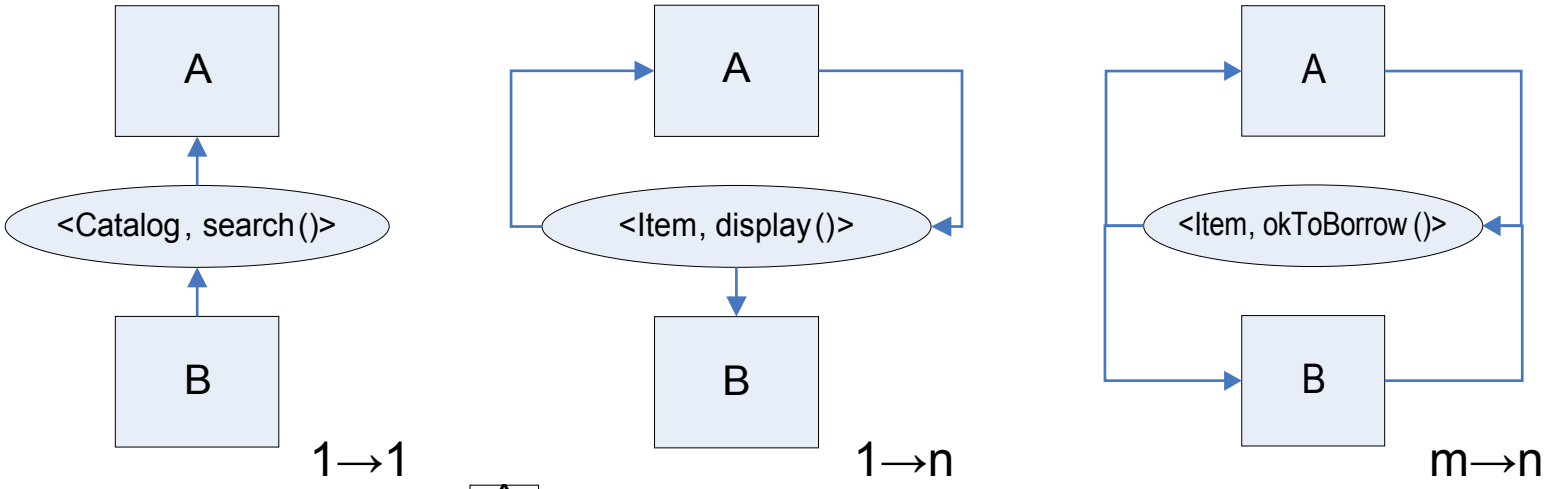


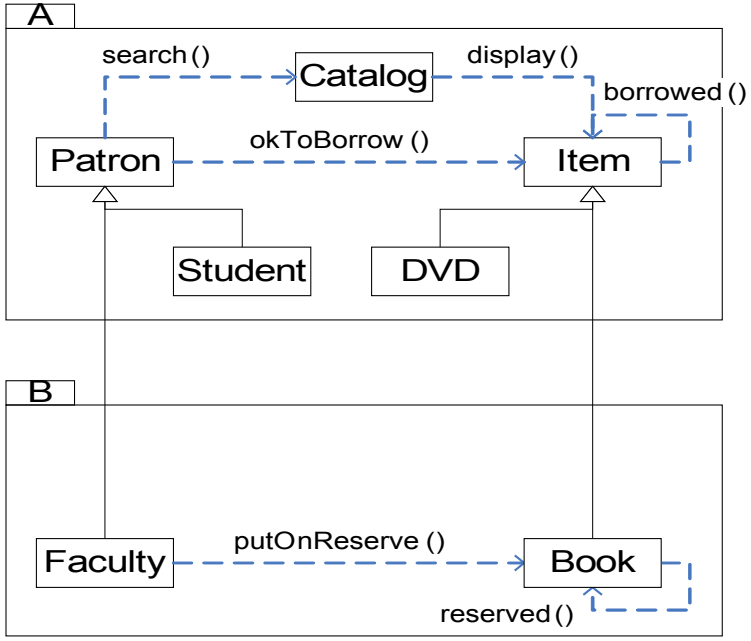
Package Collaboration

- Check the composition of the set of resources shared between two packages.



Library Management System



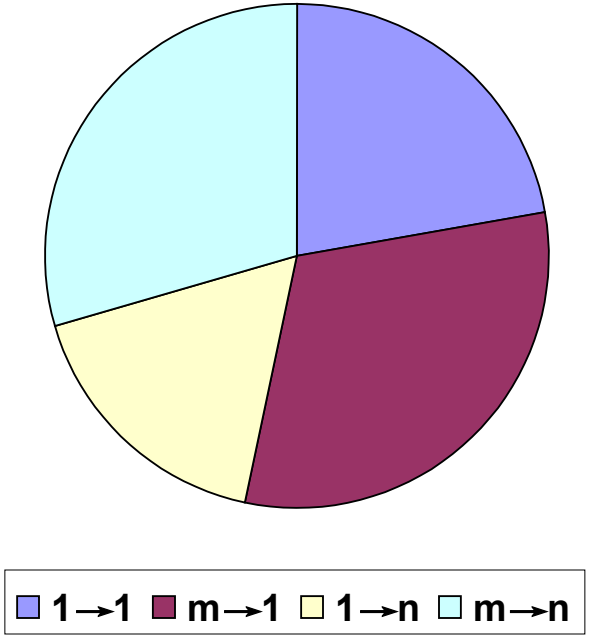


Case Study

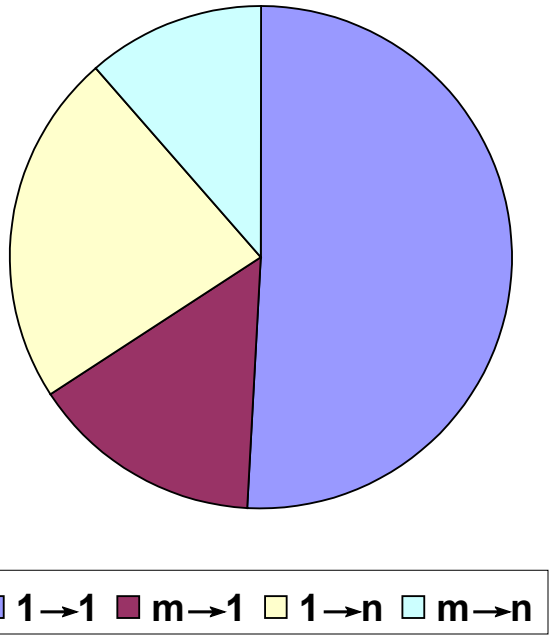
- Subject System: JHotDraw
  - Version 6.0 beta 1
    - 71,000 LOC, 17 packages, 396 classes
  - Version 7.1
    - 91,000 LOC, 32 packages, 561 classes
- Goal:
  - Did the package partition criteria change as the architecture changed ?

Shared Resources

- JHotDraw 6.0b1
  - # top-level pkg. = 8
  - # shared res. = 648
  - Avg. inport size = 157
  - Avg. outport size = 181



- JHotDraw 7.1
  - # top-level pkg. = 10
  - # shared res. = 838
  - Avg. inport size = 107
  - Avg. outport size = 107



Shared Resources in V6.0

	1→1	m→1	1→n	m→n
#res	*144*	*201*	*112*	***191***
applet	*3*		*7*	*8*
application	*3*	*12*	*22*	*16*
contrib	*34*	*2*	*74*	*151*
figures	*30*	*9*	*57*	*105*
framework	*11*	*17*		
samples			*71*	*125*
standard	*32*	*93*	*61*	*139*
util	*31*	*68*	*35*	*36*
total	*144*	*201*	*327*	***580***

res associated with inports

	1→1	m→1	1→n	m→n
#res	*144*	*201*	*112*	***191***
applet		*40*	*4*	*9*
application	*3*	*73*	*12*	*29*
contrib	*41*	*163*	*10*	*164*
figures	*18*	*107*	*21*	*88*
framework				
samples	*64*	*191*	*5*	*106*
standard	*15*	*29*	*35*	*131*
util	*3*	*23*	*25*	*42*
total	*144*	*626*	*112*	***569***

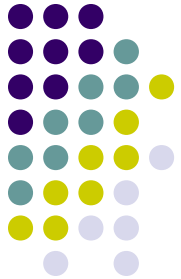
res associated with outports

Shared Resources in V7.1

	1→1	m→1	1→n	m→n
# res	*403*	*119*	*182*	*90*
samples	*32*		*182*	*90*
app	*13*	*6*	*15*	*4*
draw	*255*		*163*	*86*
beans				*1*
gui	*31*	*6*		
undo	*10*	*1*		
xml	*13*	*27*	*4*	*1*
geom	*27*	*63*		
io	*16*	*3*		
util	*6*	*13*		

	1→1	m→1	1→n	m→n
# res	*403*	*119*	*182*	*90*
samples	*298*	*115*	*15*	*89*
app	*58*	*11*	*13*	*4*
draw	*42*	*116*	*146*	*86*
beans				*1*
gui	*5*	*1*	*2*	
undo		*4*		
xml		*2*	*6*	*2*
geom				
io				
util				
Total	*403*	*240*	*182*	*182*

Discussion



- Compared to JHotDraw 6.0, JHotDraw 7.1
  - Has more packages.
  - Shares more resources among packages.
  - Has more 1→1 and 1→n collaboration, but fewer m→1 and m→n collaborations.
  - Exhibits a layered structure.
  - Forms a clear framework-application pattern.

- Reducing coupling, especially inheritance coupling, is the main concern in version 7.0.

Conclusion & Future Work

- Package Diagrams lack of necessary semantic information to understanding package design.
- Hybrid Models are helpful in interpreting package partition criteria.

- Future Work
  - Identify collaboration patterns in hierarchical package organization.
  - Automate the process of analyzing package relations.