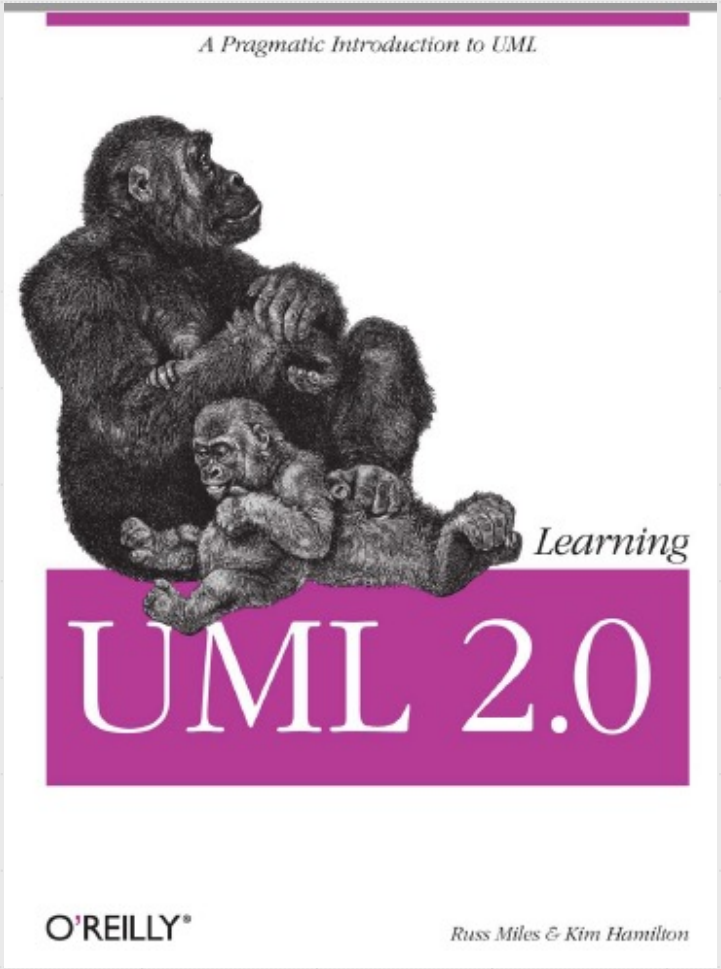
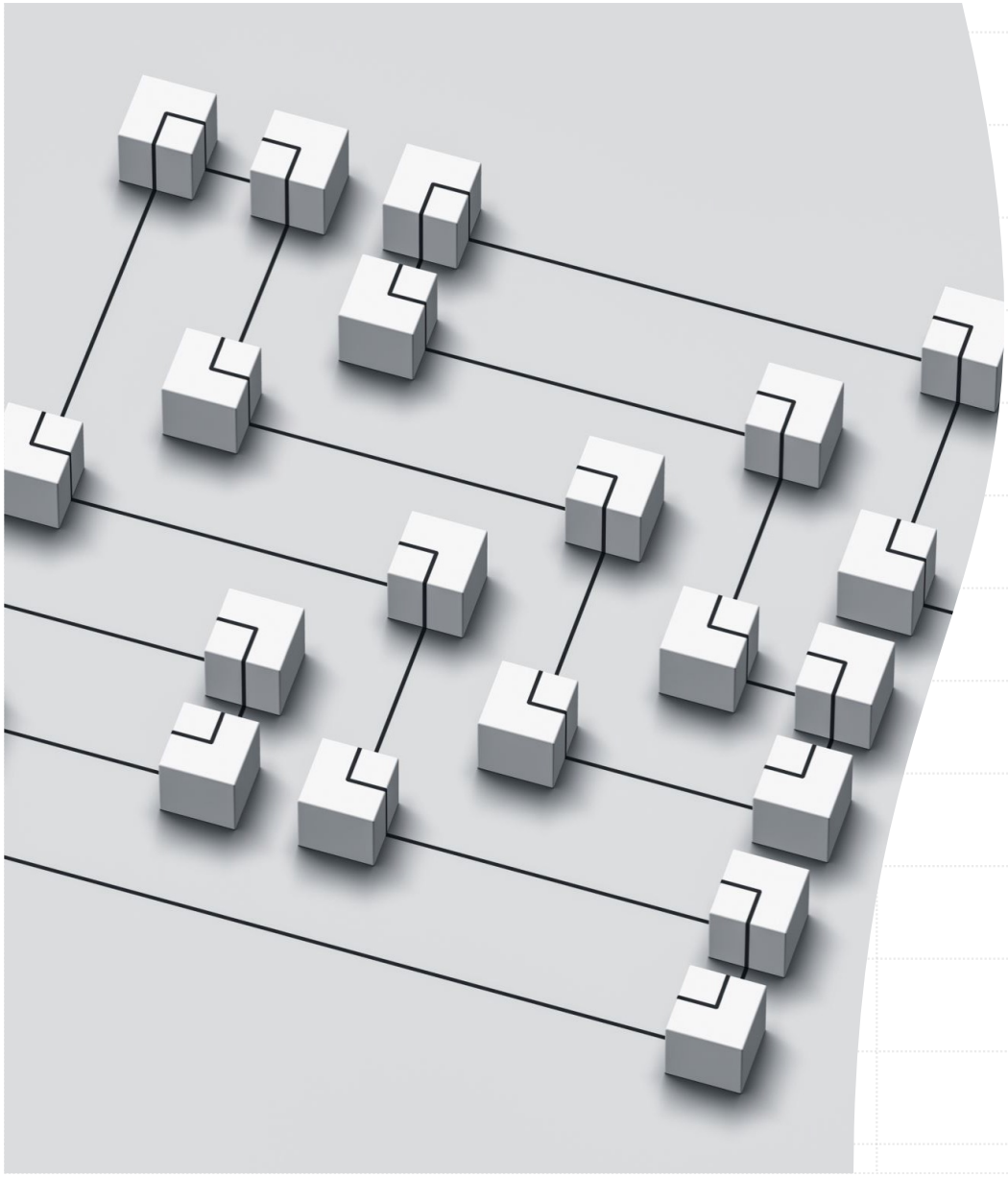
UML diagram and Epic mapping

Software Engineering II

UML diagram

- UML and System Modeling
- UML 2.0 revisit
  - Class diagram
  - Object diagram
  - Sequence diagram
  - Component diagram
  - Subsystem diagram (depreciated)
    - Included in Component diagram (UML 2.5)
  - Deployment diagram



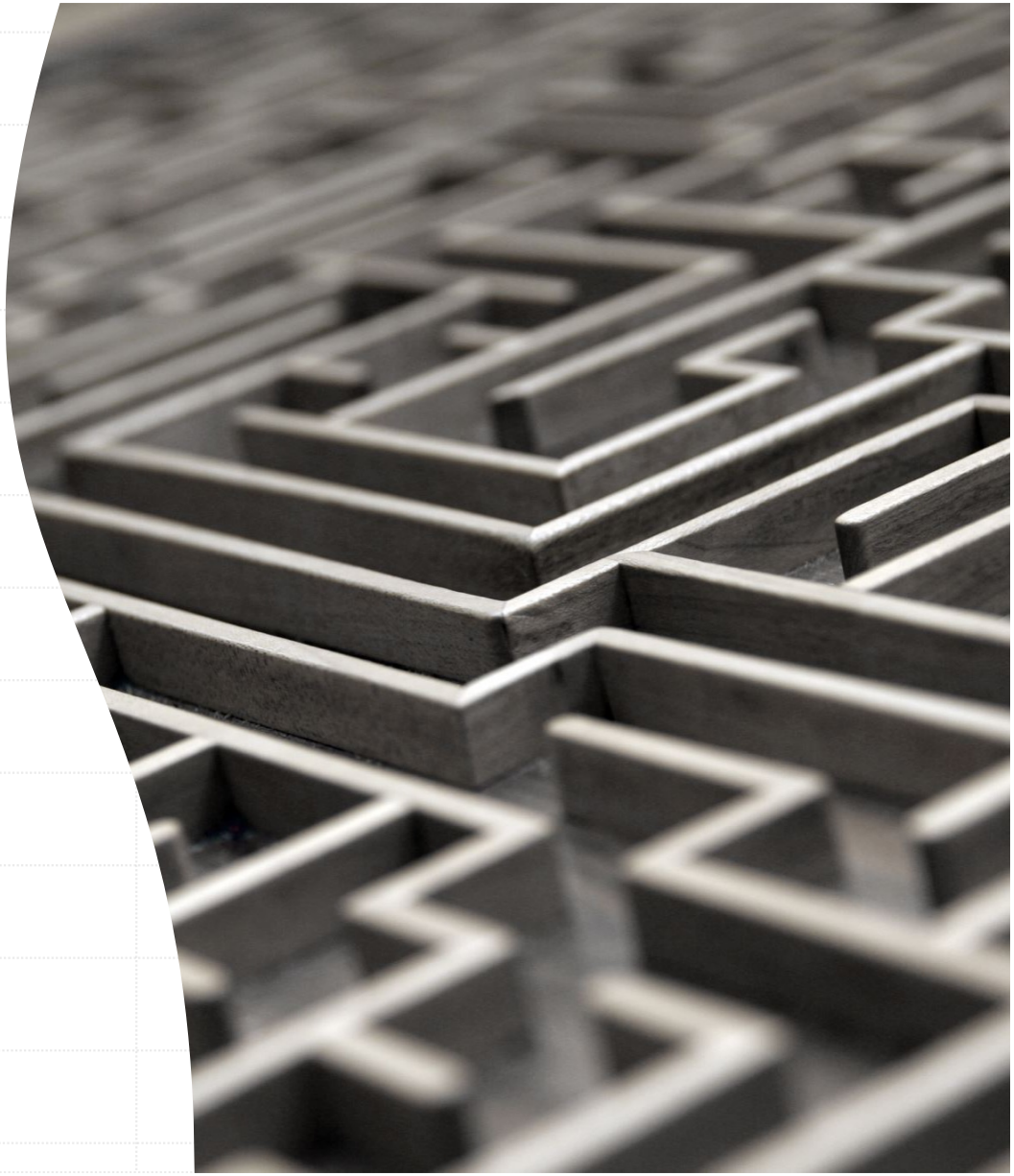


UML and System Modeling

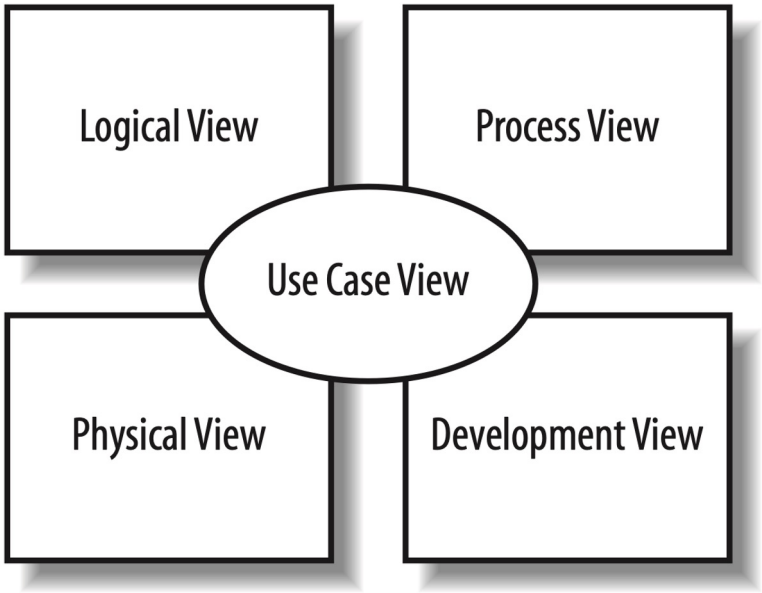
- The Unified Modeling Language (UML) is the standard modeling language for software and systems development.
- In systems design, you model for one important reason: to manage complexity.
  - Modeling helps you see the forest for the trees
  - allowing you to focus on, capture, document, and communicate the important aspects of your system's design.

Why UML?

- Every approach to modeling has different advantages and disadvantages, but UML has six main advantages
  1. It's a formal language
  2. It's concise (กระชับ)
  3. It's comprehensive (ครอบคลุม)
  4. It's scalable (ปรับขนาดได้)
  5. It's built on lessons learned
  6. It's the standard (มาตรฐาน)



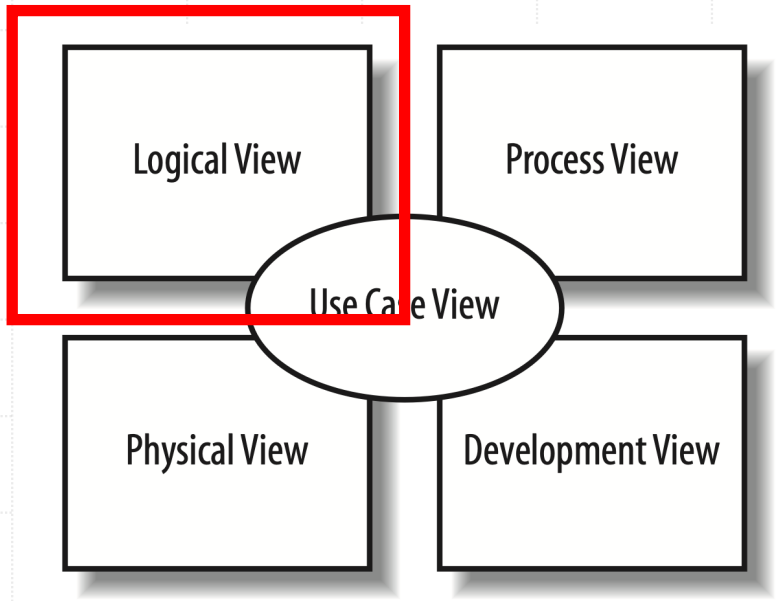
Philippe Kruchten's 4+1 view model



- Philippe Kruchten (born 1952) is a Canadian software engineer, and Professor of Software Engineering at University of British Columbia in Vancouver, Canada.

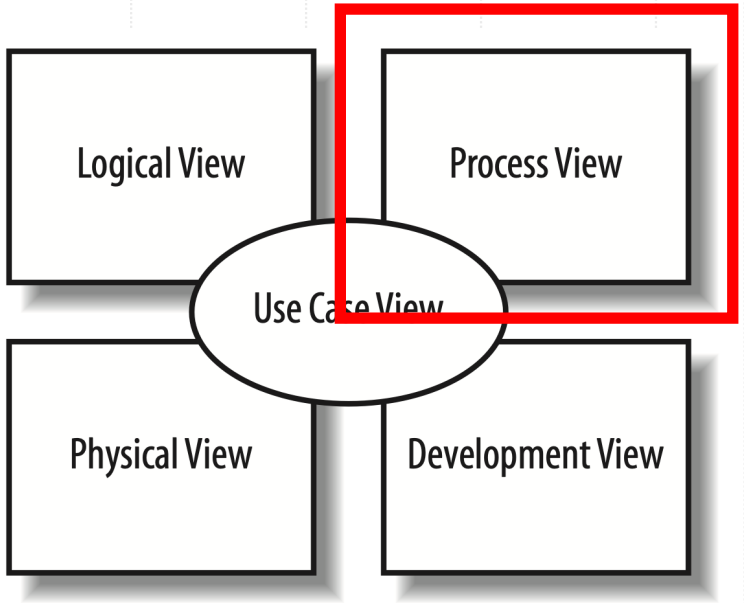
Logical view

- Describes the abstract descriptions of a system's parts.
- Used to model what a system is made up of and how the parts interact with each other.
- The types of UML diagrams that typically make up this view include class, object, state machine, and interaction diagrams.



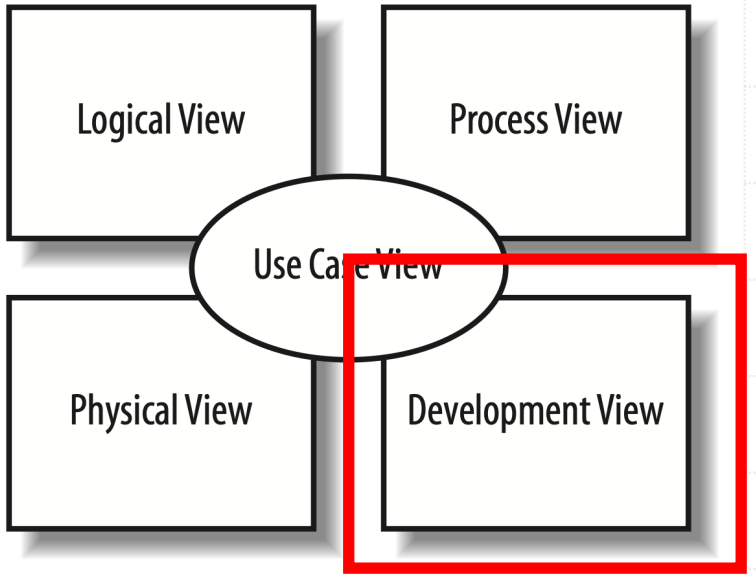
Process view

- Describes the processes within your system.
- It is particularly helpful when visualizing what must happen within your system.
- This view typically contains activity diagrams.



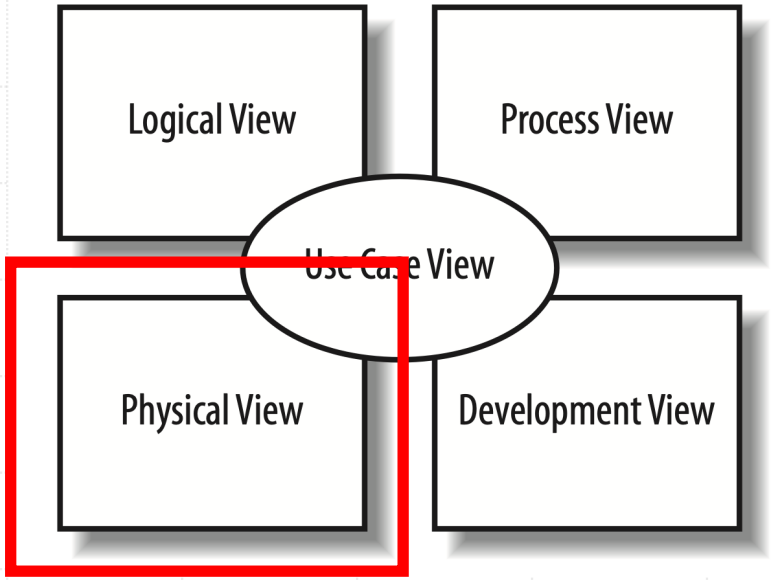
Development view

- Describes how your system's parts are organized into modules and components.
- It is very useful to manage layers within your system's architecture.
- This view typically contains package and component diagrams.



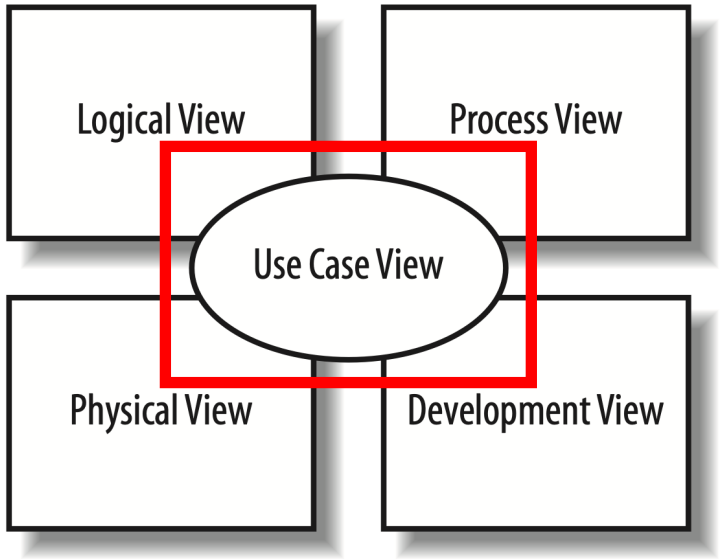
Physical view

- Describes how the system's design, as described in the three previous views, is then brought to life as a set of real-world entities.
- The diagrams in this view show how the abstract parts map into the final deployed system.
- This view typically contains deployment diagrams.



Use case view

- Describes the functionality of the system being modeled from the perspective of the outside world.
- This view is needed to describe what the system is supposed to do.
- All of the other views rely on the use case view to guide them—that's why the model is called 4+1.
- This view typically contains use case diagrams, descriptions, and overview diagrams.



Class diagram

- Class diagram is a blueprint for an object.

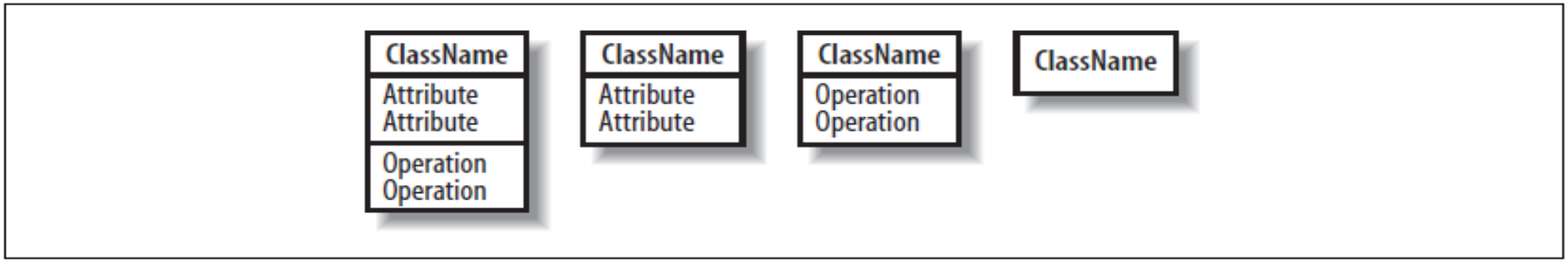


Figure 4-4. Four different ways of showing a class using UML notation

Visibility

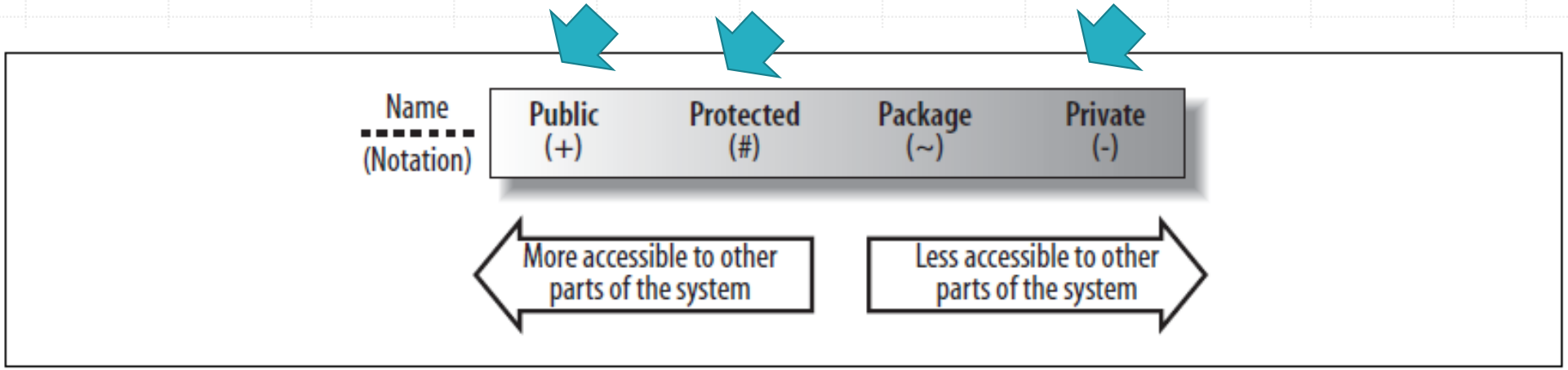


Figure 4-6. UML's four different visibility classifications

Class Relationship

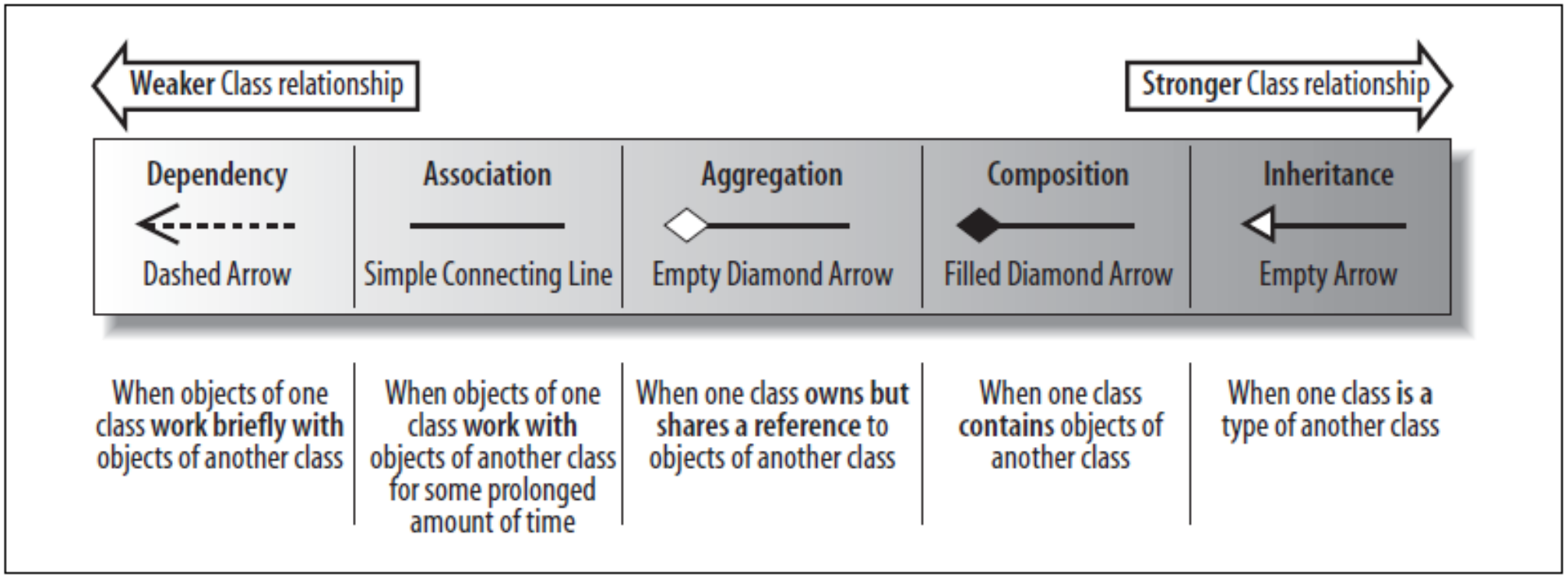


Figure 5-1. UML offers five different types of class relationship

A Class

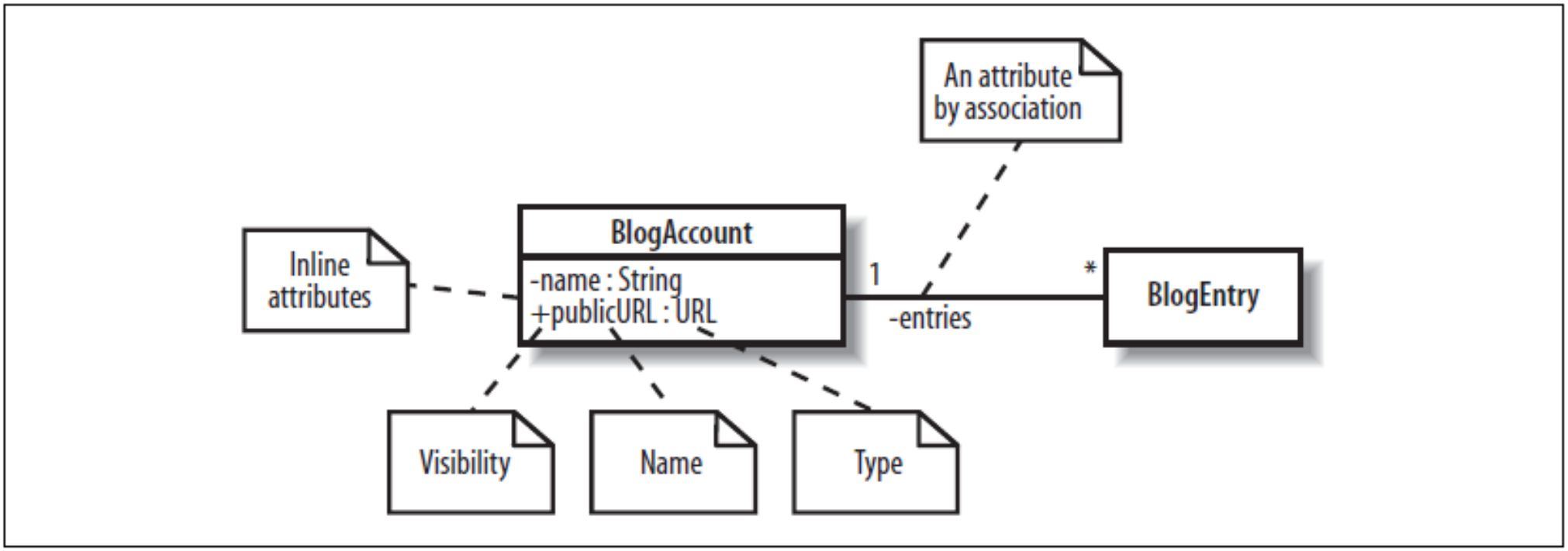


Figure 4-11. The BlogAccount class contains two inlined attributes, name and publicURL, as well as an attribute that is introduced by the association between the BlogAccount and BlogEntry classes

Abstract Class and Inheritance

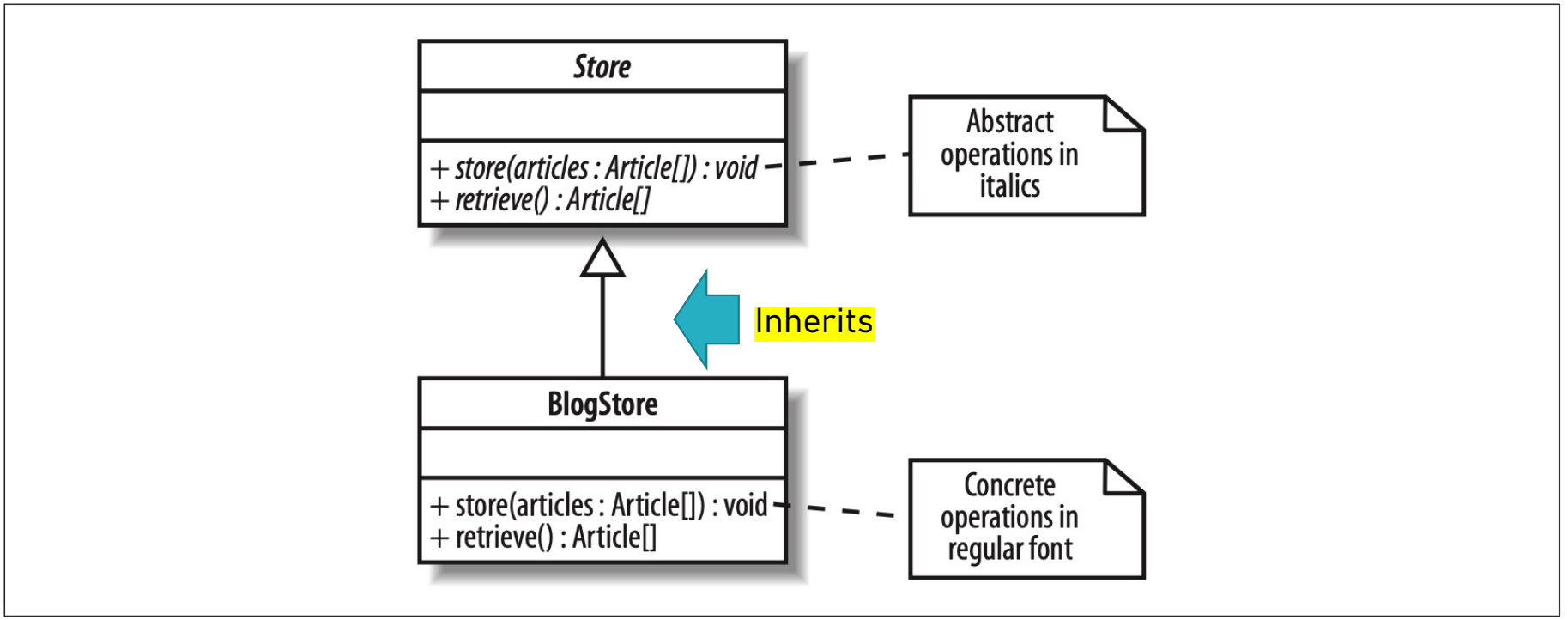


Figure 5-14. The BlogStore class inherits from the abstract Store class and implements the store(..) and retrieve(..) operations; classes that completely implement all of the abstract operations inherited from their parents are sometimes referred to as "concrete"

Interface

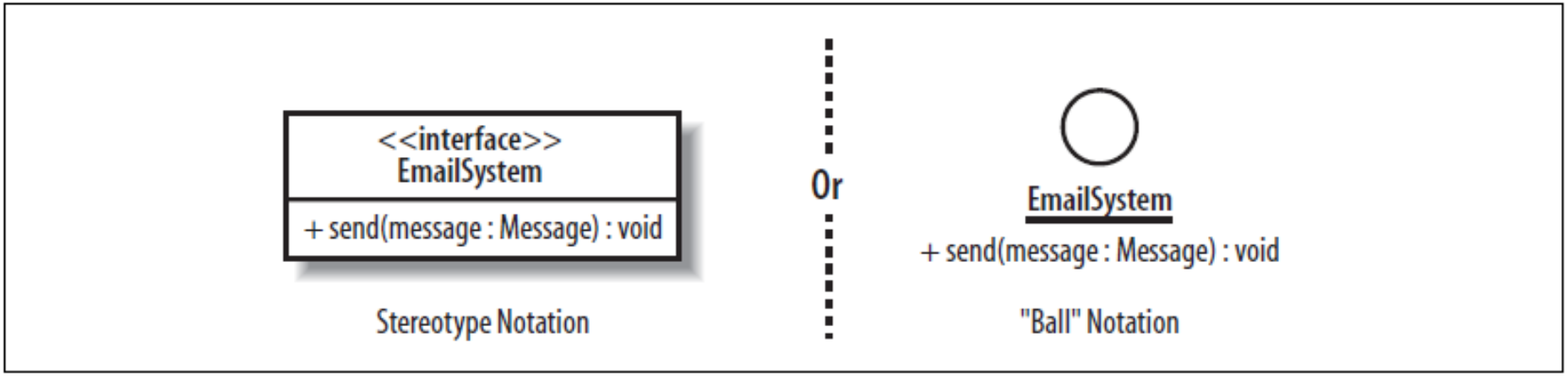


Figure 5-15. Capturing an interface to an EmailSystem using the stereotype and "ball " UML notation; unlike abstract classes, an interface does not have to show that its operations are not implemented, so it doesn't have to use italics

Interface notation

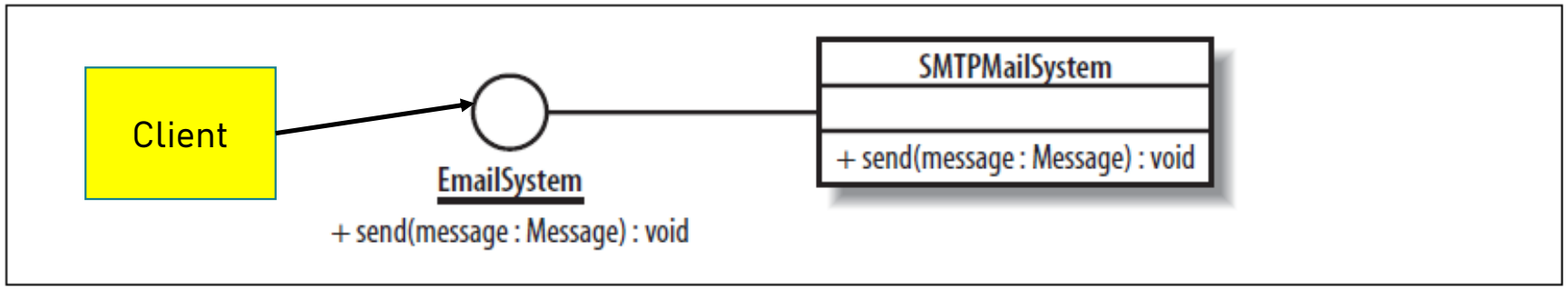


Figure 5-16. The SMTPMailSystem class implements, or realizes, all of the operations specified on the EmailSystem interface

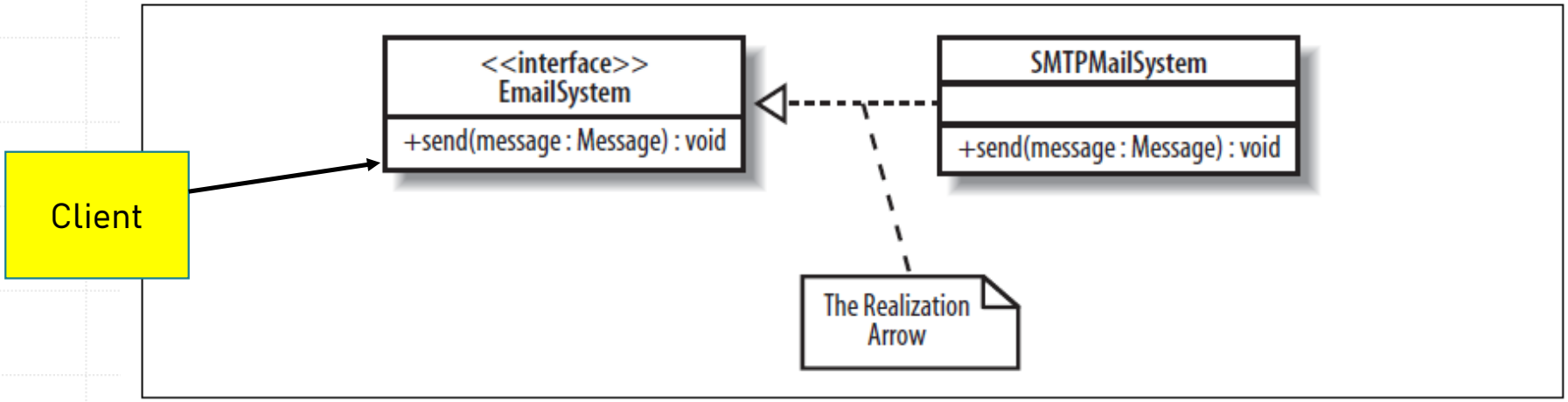


Figure 5-17. The realization arrow specifies that the SMTPMailSystem realizes the EmailSystem

Interface and its realization /Implementation

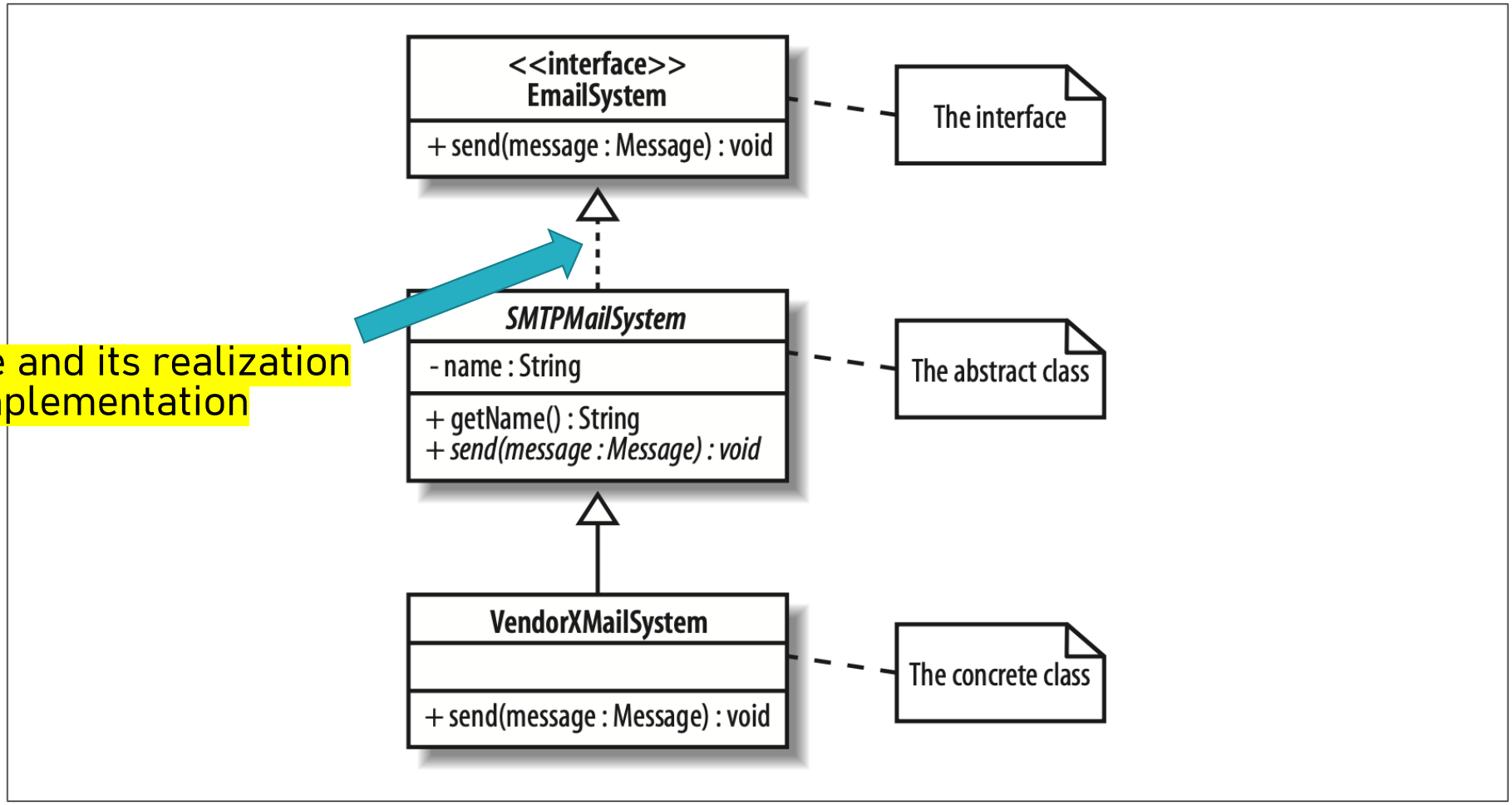


Figure 5-18. Because the SMTPMailSystem class does not implement the send(..) operation as specified by the EmailSystem interface, it needs to be declared abstract; the VendorXMailSystem class completes the picture by implementing all of its operations

UML representation of classes/objects:

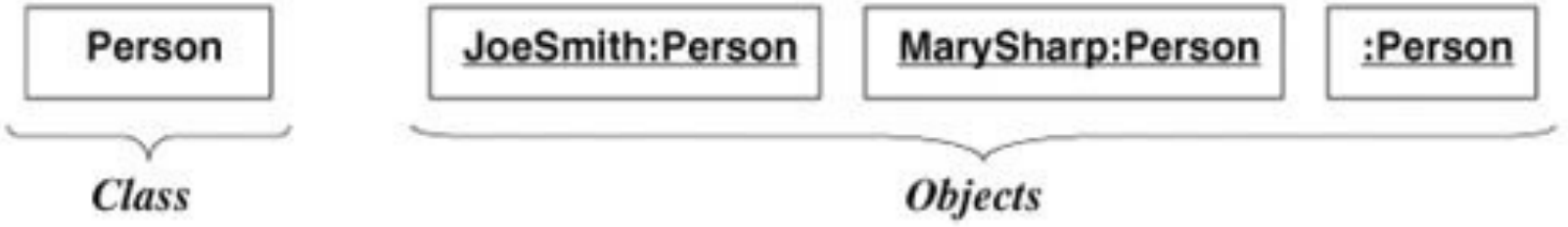


Figure 3.1 A class and objects. Objects and classes are the focus of class modeling.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

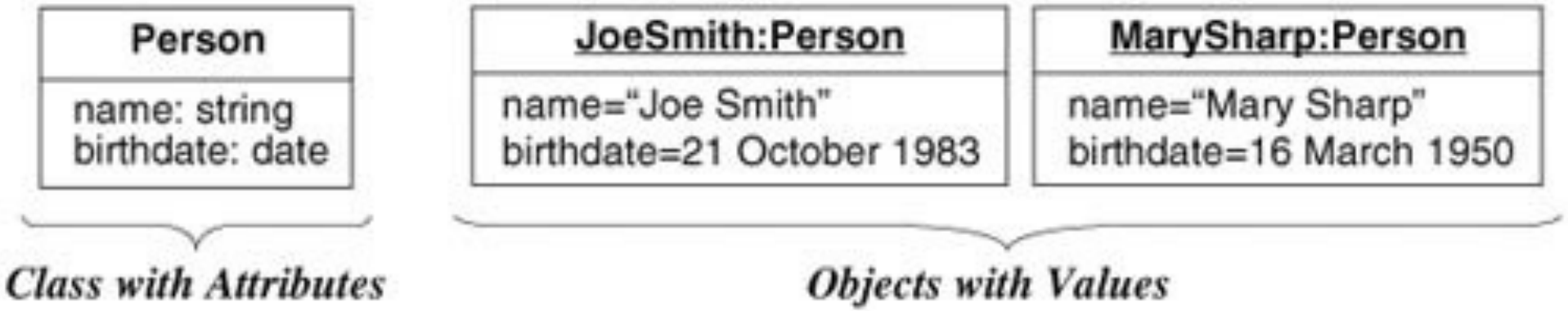


Figure 3.2 Attributes and values. Attributes elaborate classes.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Object ID is Implicit

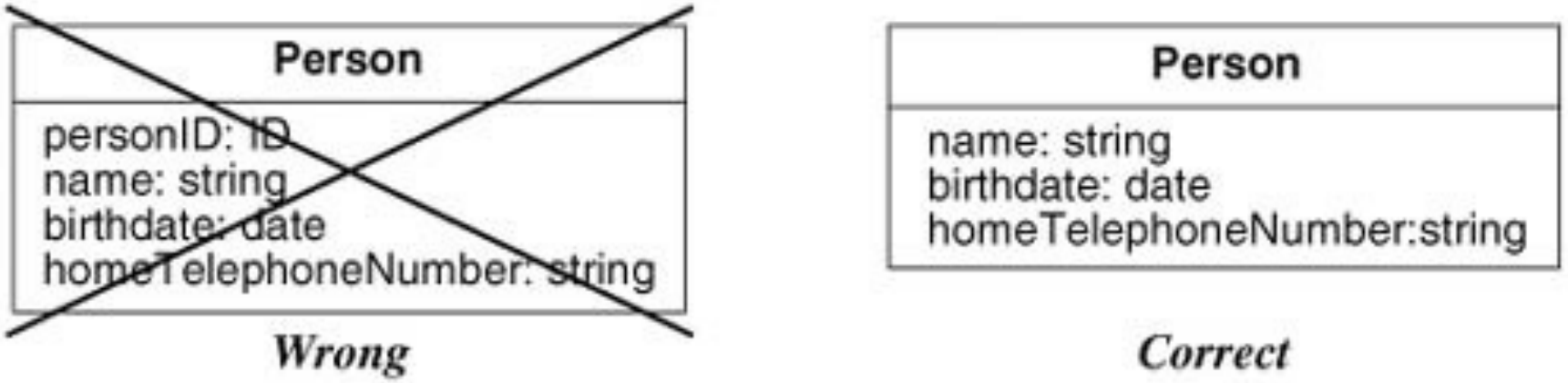


Figure 3.3 Object identifiers. Do not list object identifiers; they are implicit in models.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Examples

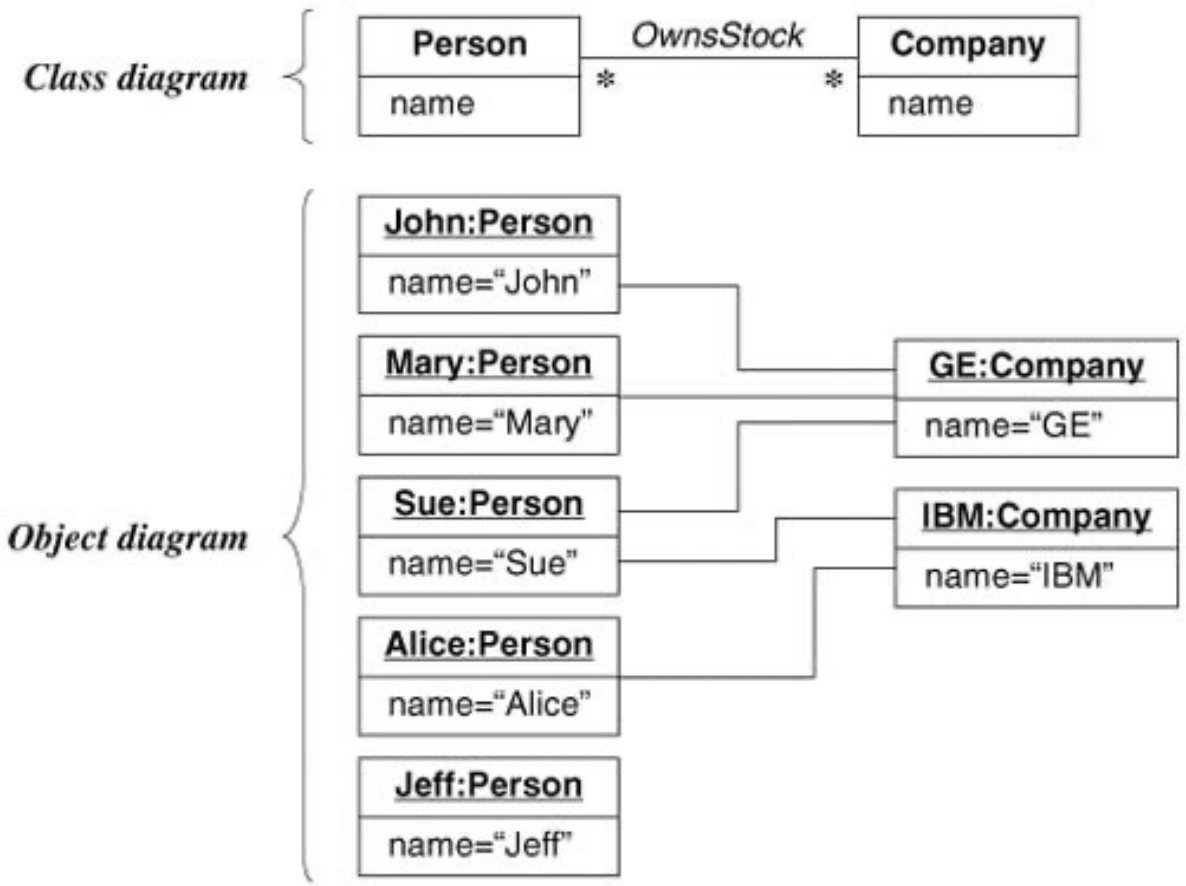


Figure 3.7 Many-to-many association. An association describes a set of potential links in the same way that a class describes a set of potential objects.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Sequence Diagram

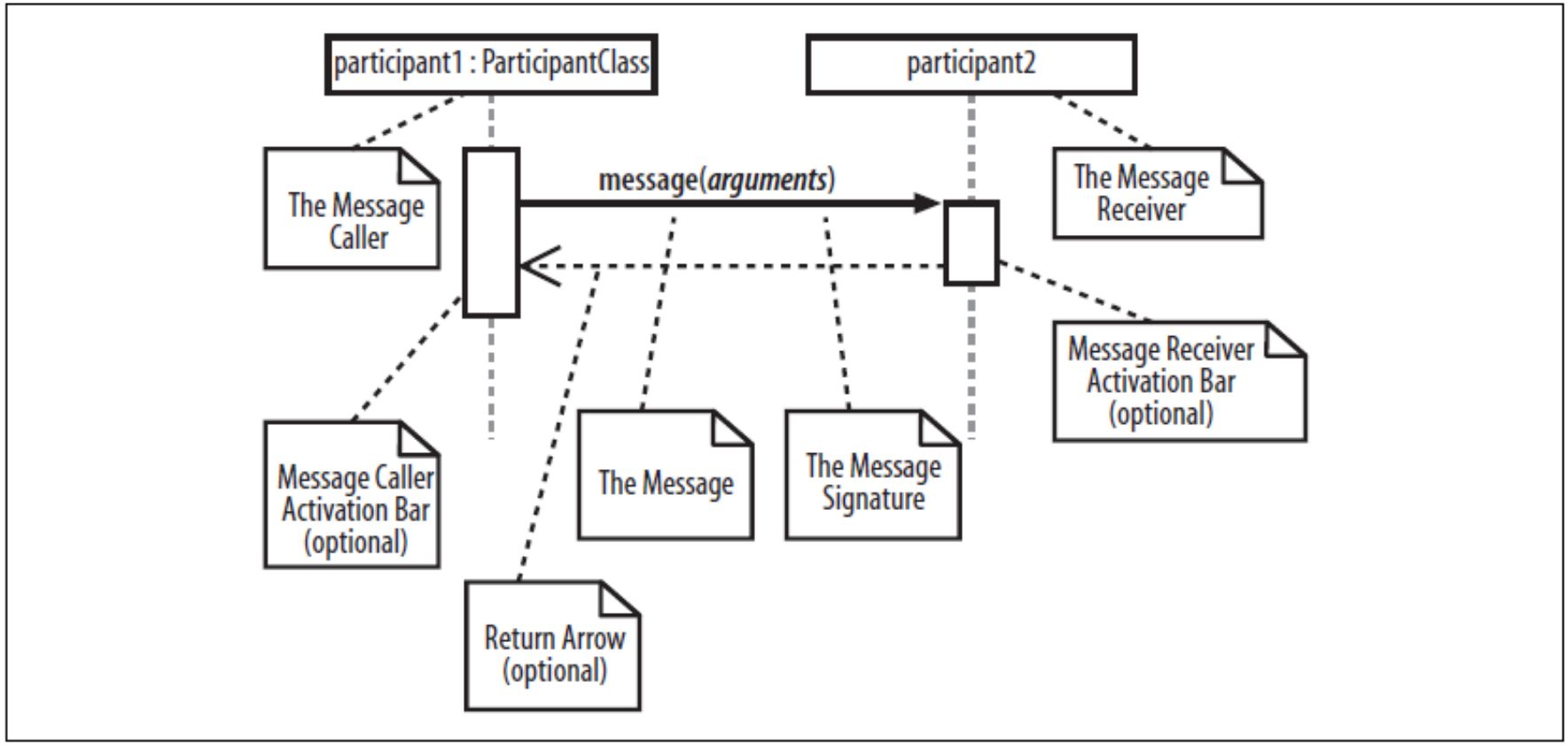
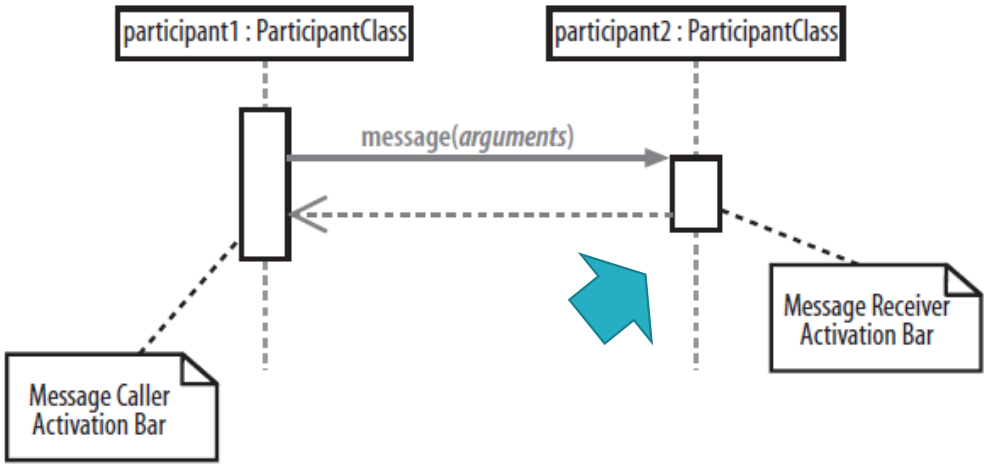
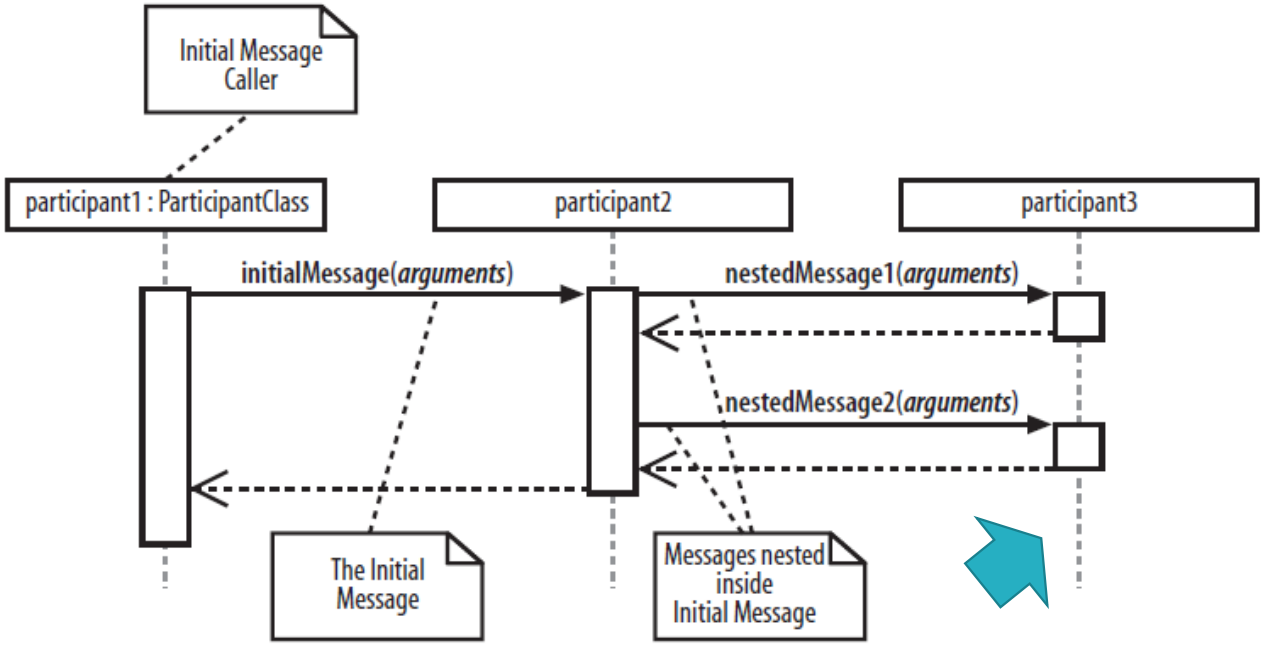
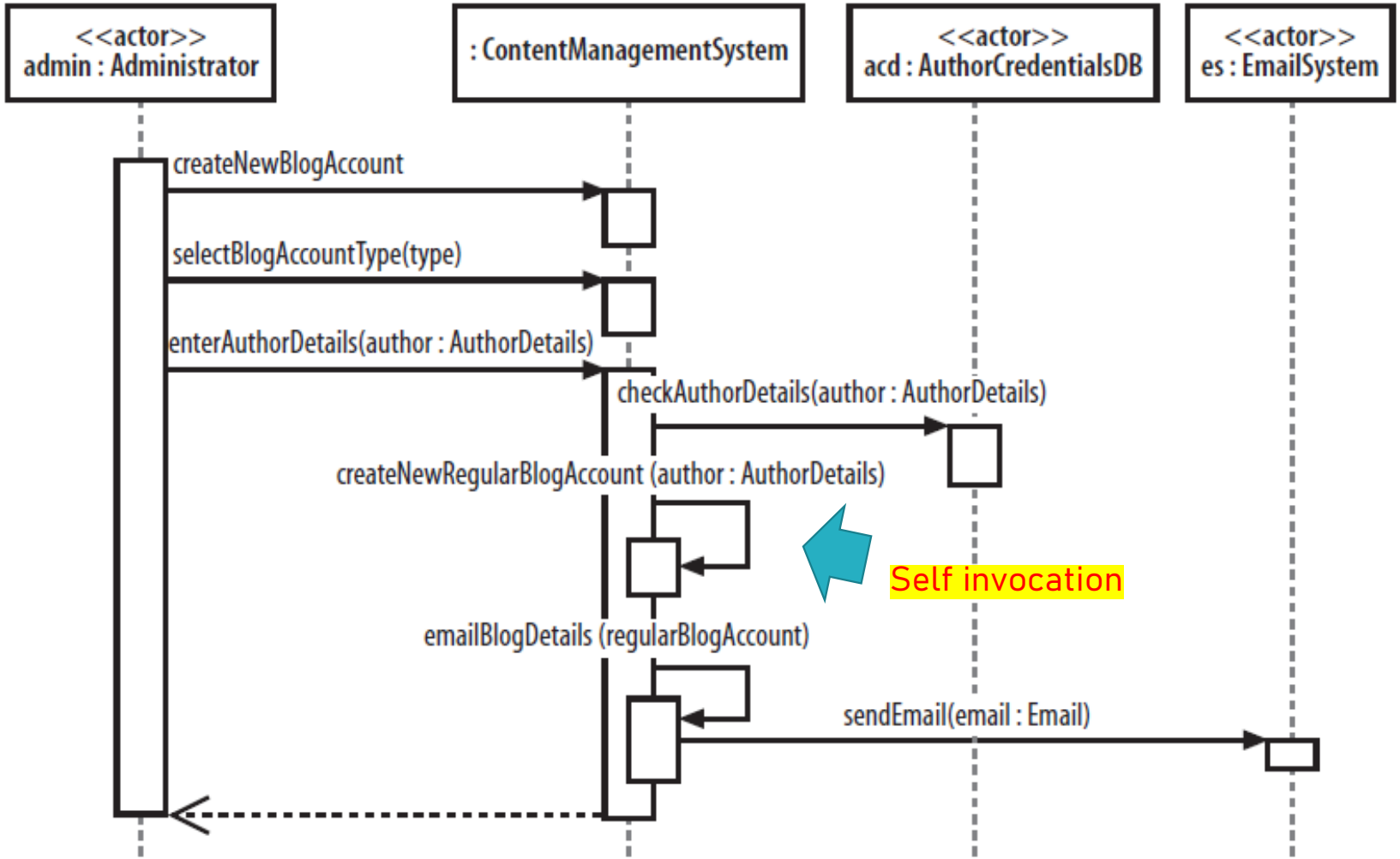


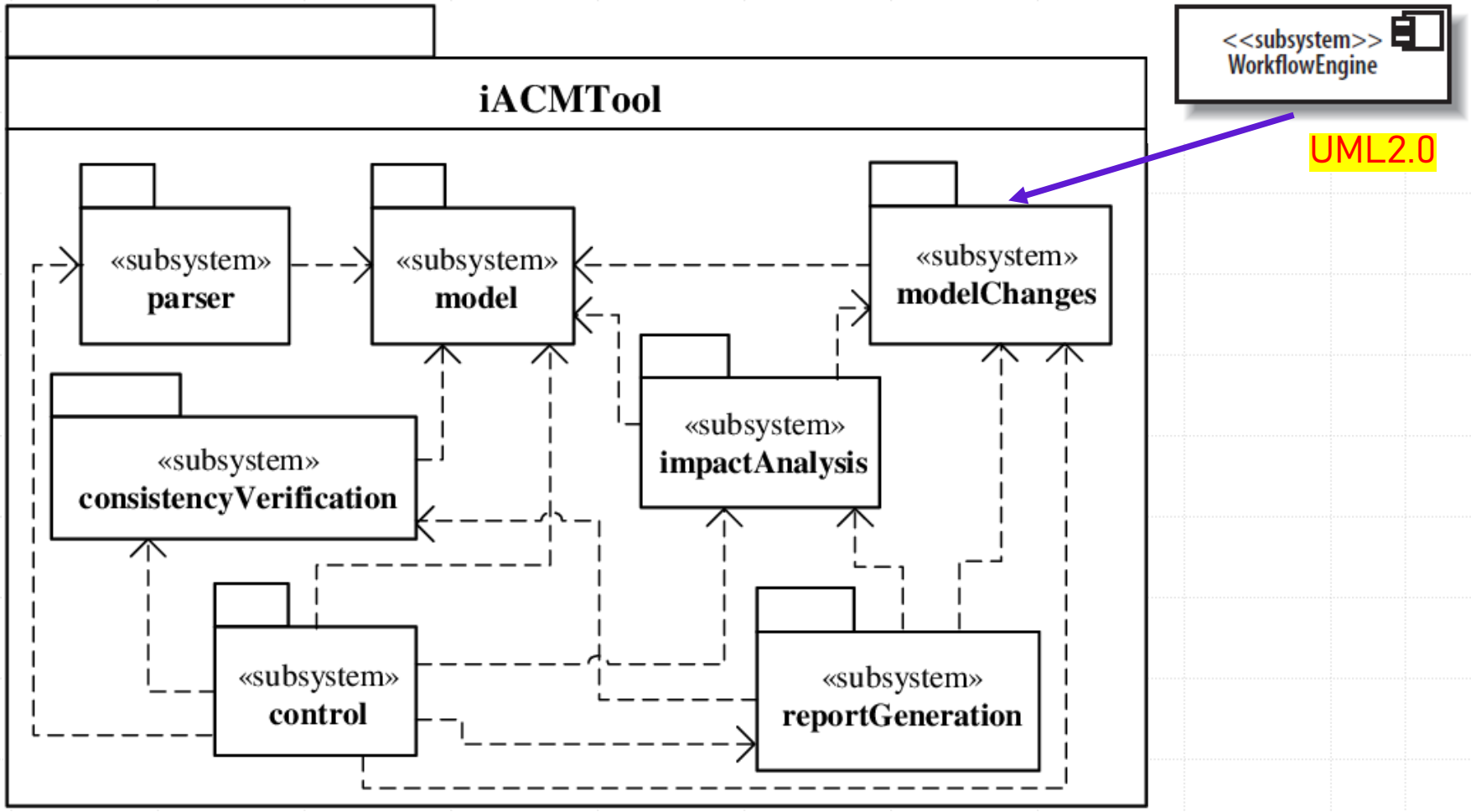
Figure 7-5. Interactions on a sequence diagram are shown as messages between participants


participant1 : ParticipantClass
participant2 : ParticipantClass
message(arguments)
Message Receiver Activation Bar
Message Caller Activation Bar

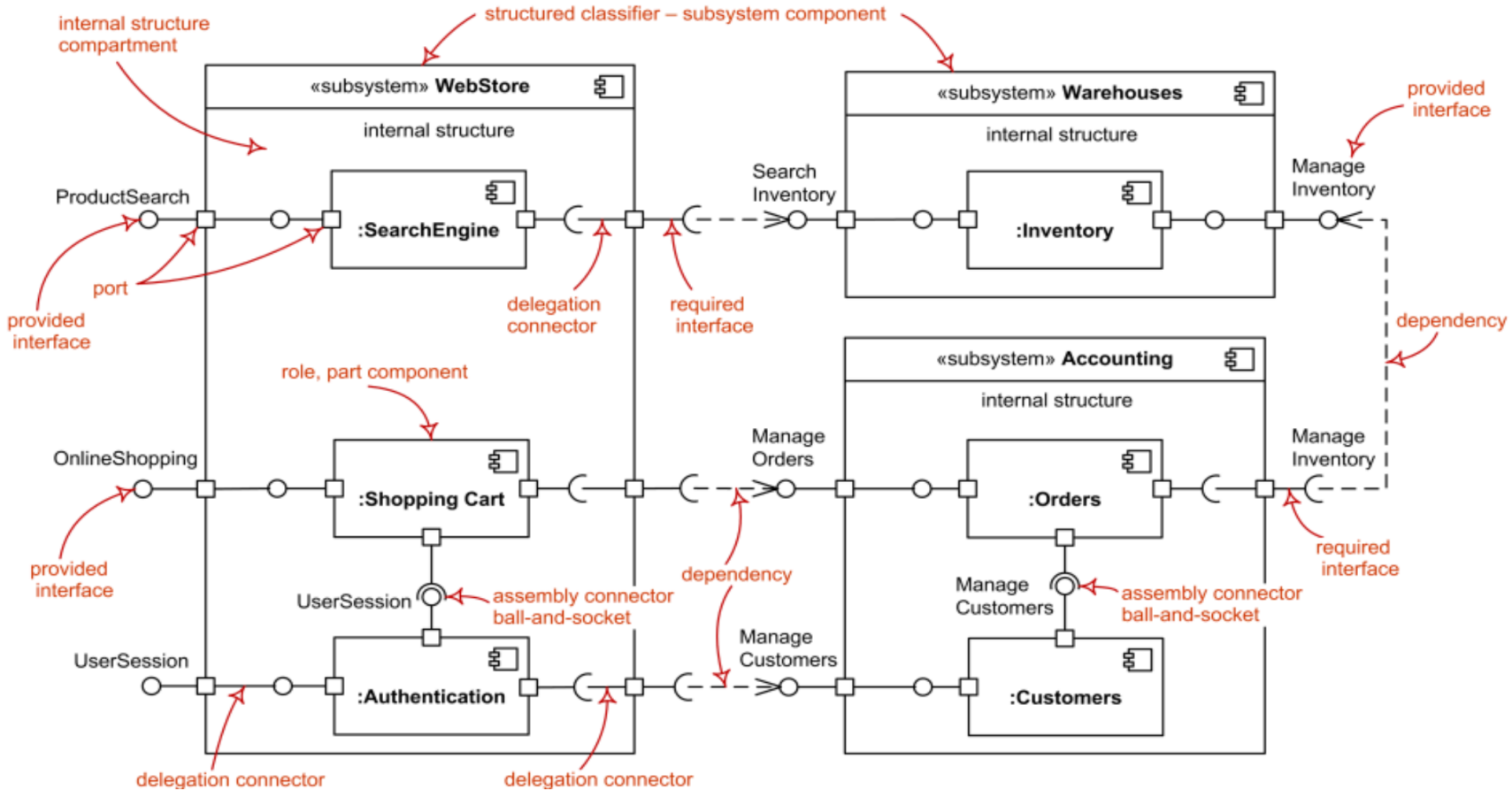

Initial Message Caller
participant1 : ParticipantClass
participant2
participant3
initialMessage(arguments)
nestedMessage1(arguments)
nestedMessage2(arguments)
The Initial Message
Messages nested inside Initial Message


<<actor>>
admin : Administrator
: ContentManagementSystem
<<actor>>
acd : AuthorCredentialsDB
<<actor>>
es : EmailSystem
createNewBlogAccount
selectBlogAccountType(type)
enterAuthorDetails(author : AuthorDetails)
checkAuthorDetails(author : AuthorDetails)
createNewRegularBlogAccount (author : AuthorDetails)
Self invocation
emailBlogDetails (regularBlogAccount)
sendEmail(email : Email)

Subsystem diagram (depreciated – UML 2.5)



(Example) Component Diagram with Subsystem classifier



https://www.uml-diagrams.org/component-diagrams.html

What Is a Component?

- A component is an encapsulated, reusable, and replaceable part of your software.
  - You can think of components as building blocks: you combine them to fit together to form your software.
  - Can range in size from relatively small, up to a large subsystem.
  - Autonomous, self-contained, and substitutable.
- Good candidates for components are items that perform a key functionality and will be used frequently throughout your system.
  - Can be third-party components, or components you create yourself.
  - Ex. loggers, XML parsers, or online shopping carts.

Component Diagram

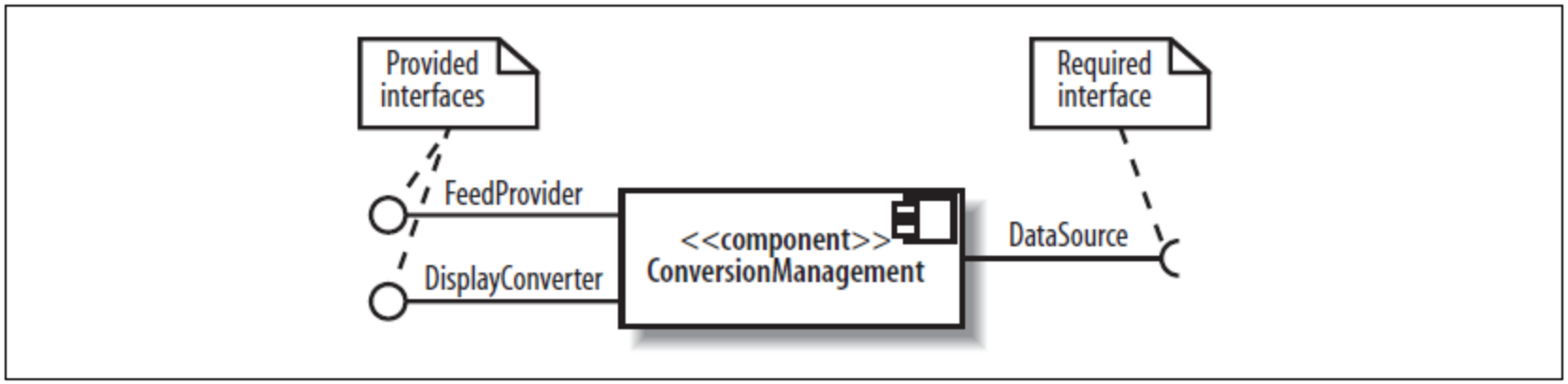
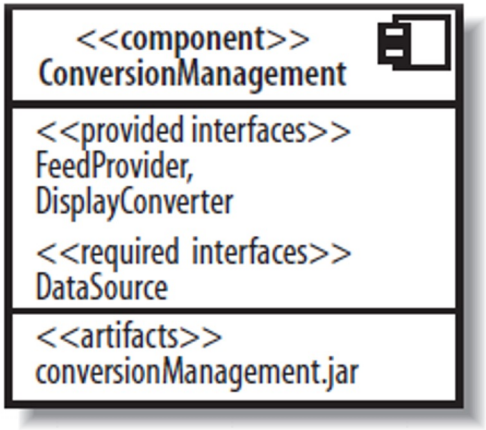


Figure 12-4. The ball and socket notation for showing a component's provided and required interfaces



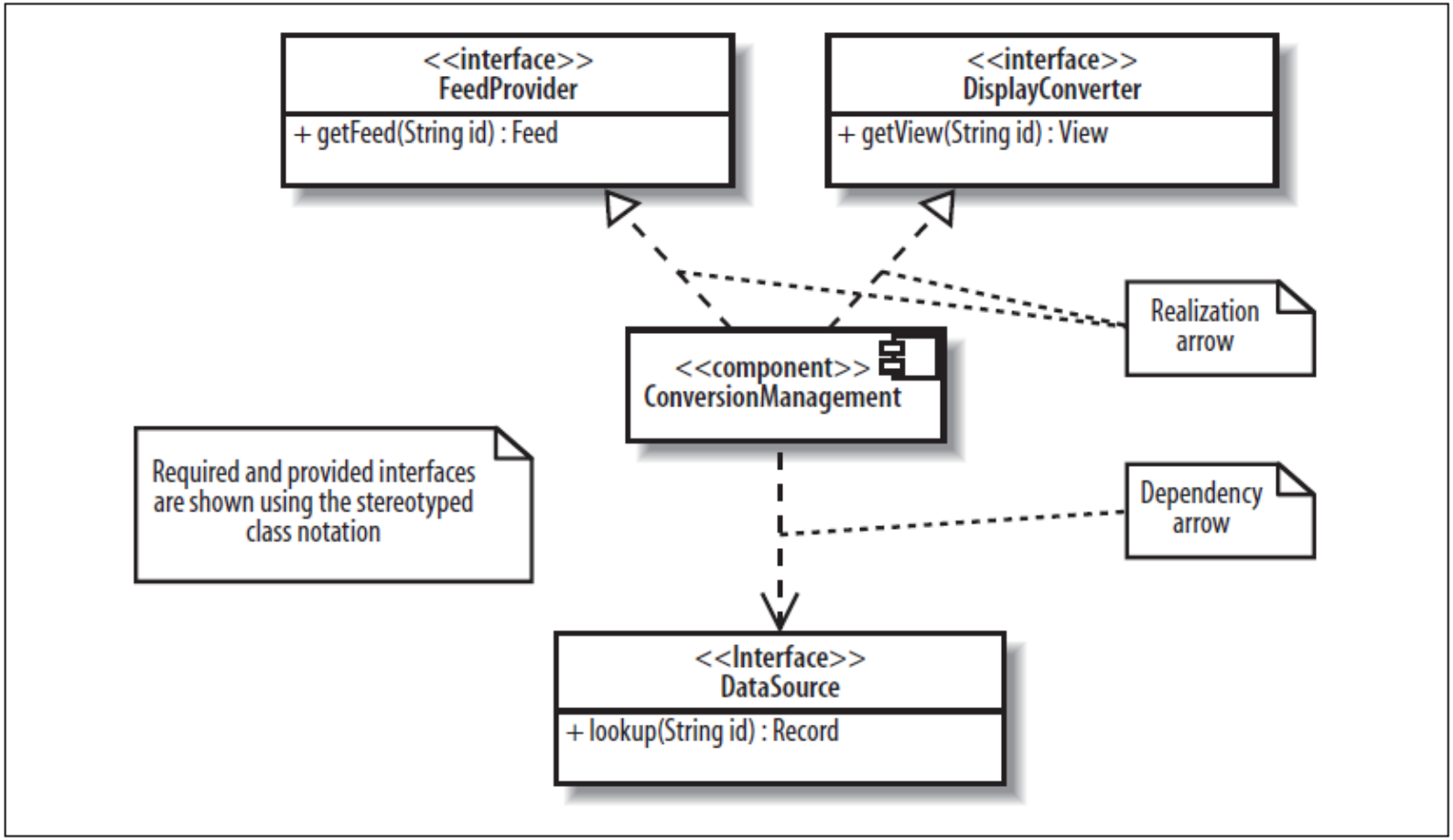


Figure 12-5. The stereotyped class notation, showing operations of the required and provided interfaces

Two Components Working Together

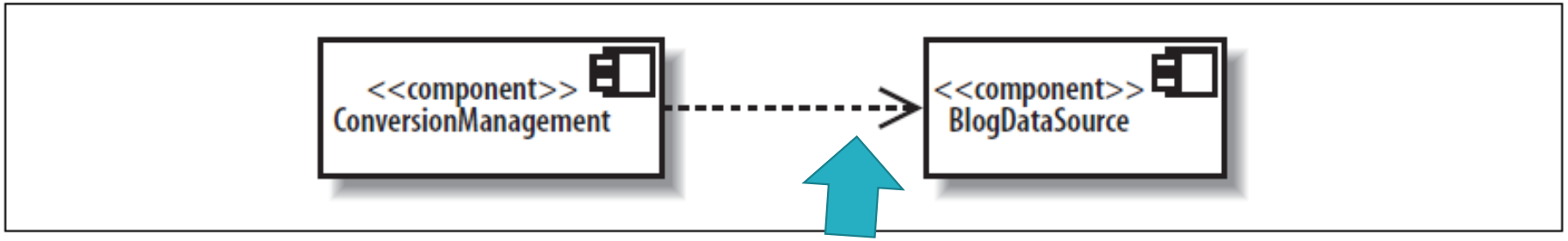


Figure 12-9. You can draw dependency arrows directly between components to show a higher level view

No interface ….
What does it mean?
Why?

สังเกตทิศของ Association หมายถึงอะไร

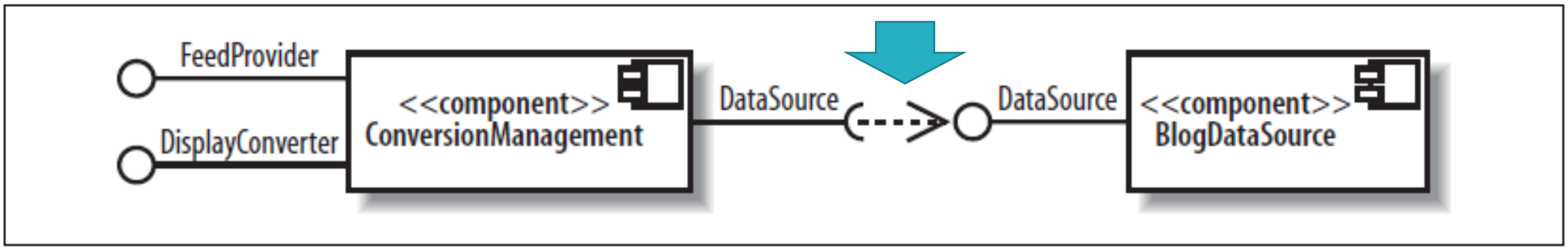


Figure 12-7. The ConversionManagement component requires the DataSource interface, and the BlogDataSource component provides that interface

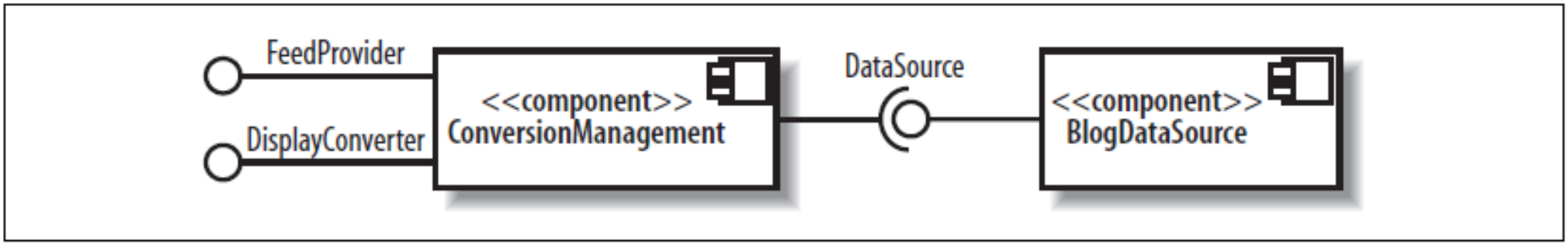
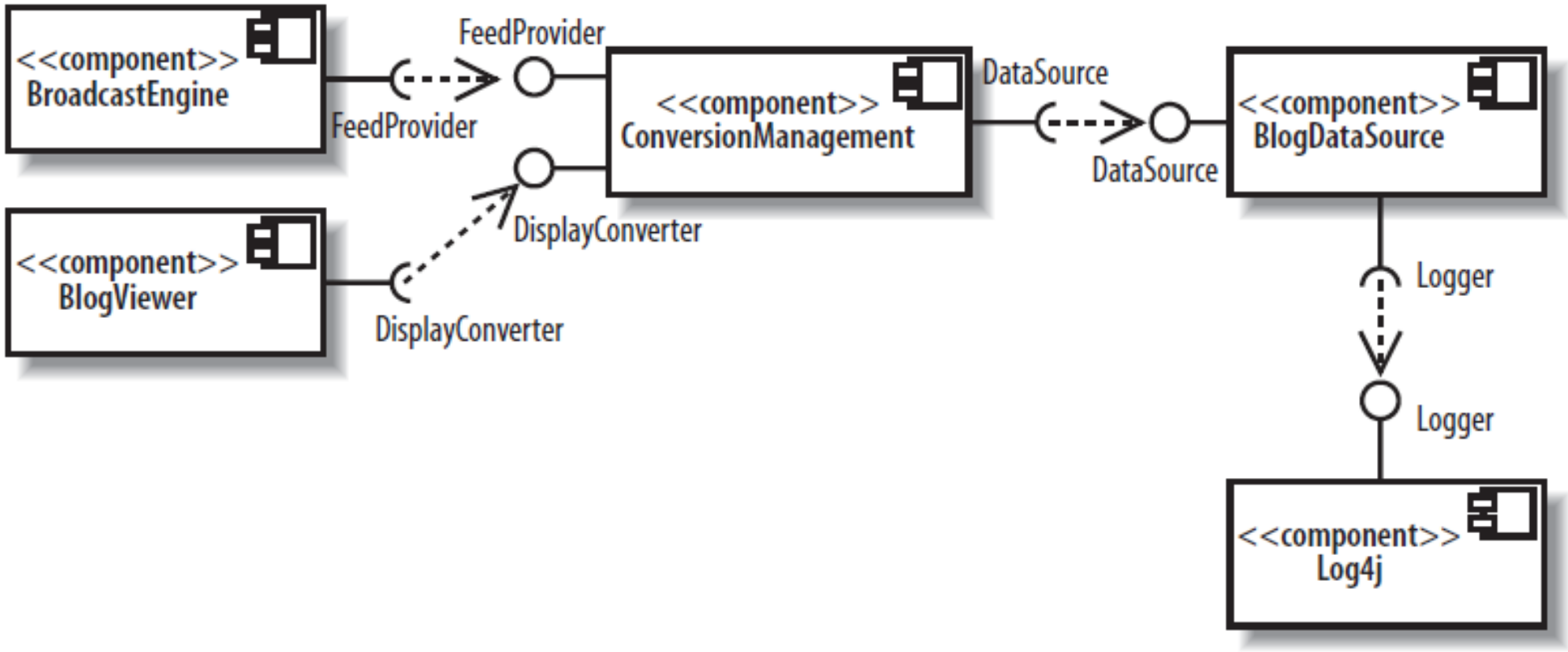


Figure 12-8. Presentation option that snaps the ball and socket together

Sample


<<component>>
BroadcastEngine
FeedProvider
FeedProvider
<<component>>
ConversionManagement
DataSource
DataSource
<<component>>
BlogDataSource
<<component>>
BlogViewer
DisplayConverter
DisplayConverter
Logger
Logger
<<component>>
Log4j

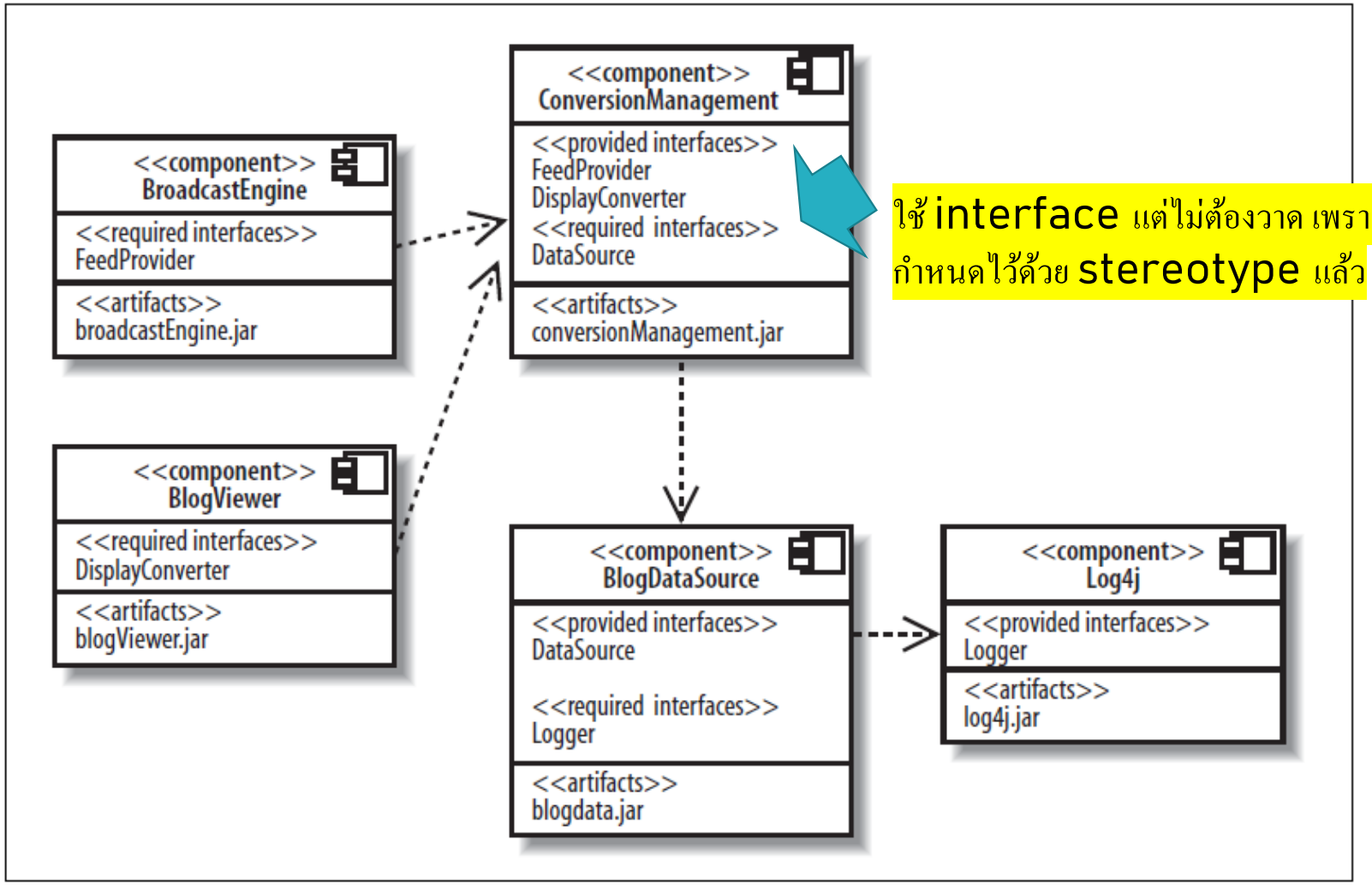


Figure 12-11. Focusing on component dependencies and the manifesting artifacts is useful when you are trying control the configuration or deployment of your system

Realization within Component (nested components)

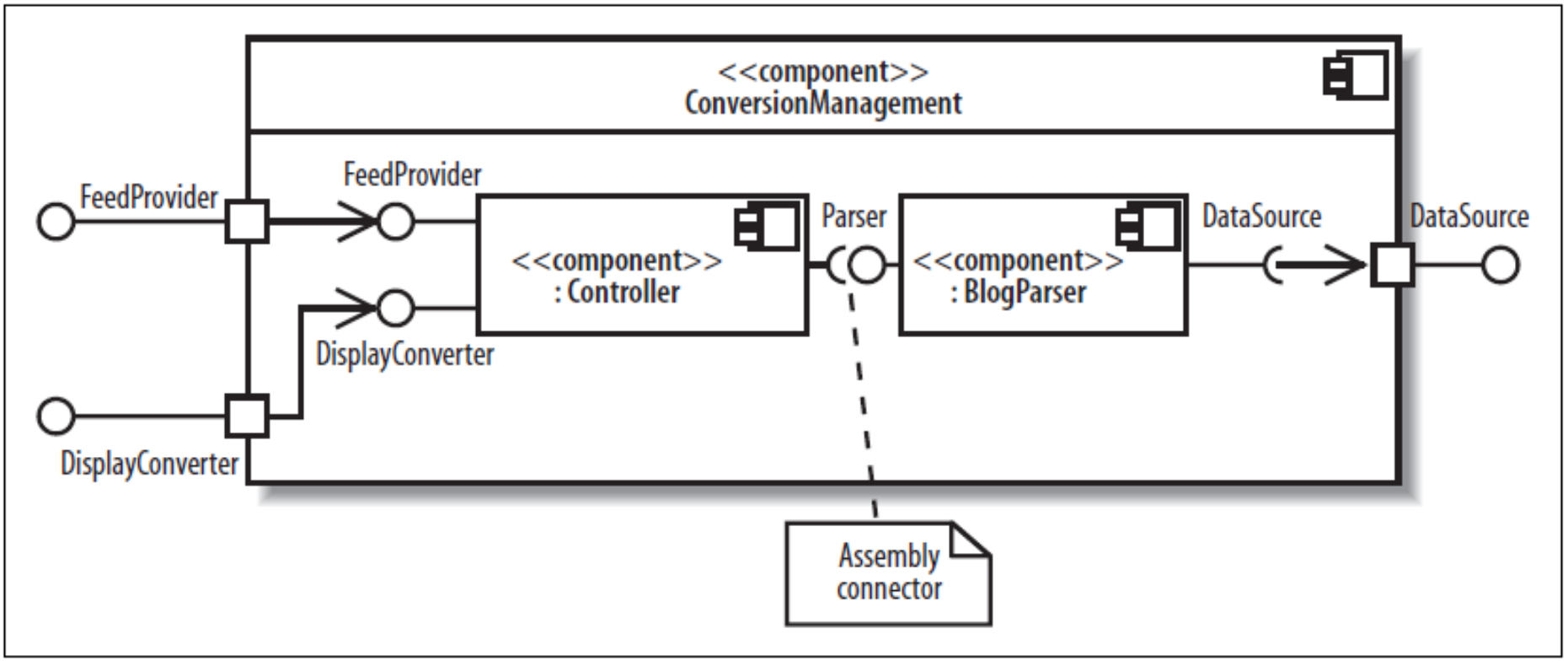


Figure 12-19. Assembly connectors show components working together through interfaces

Black Box vs. White Box Component

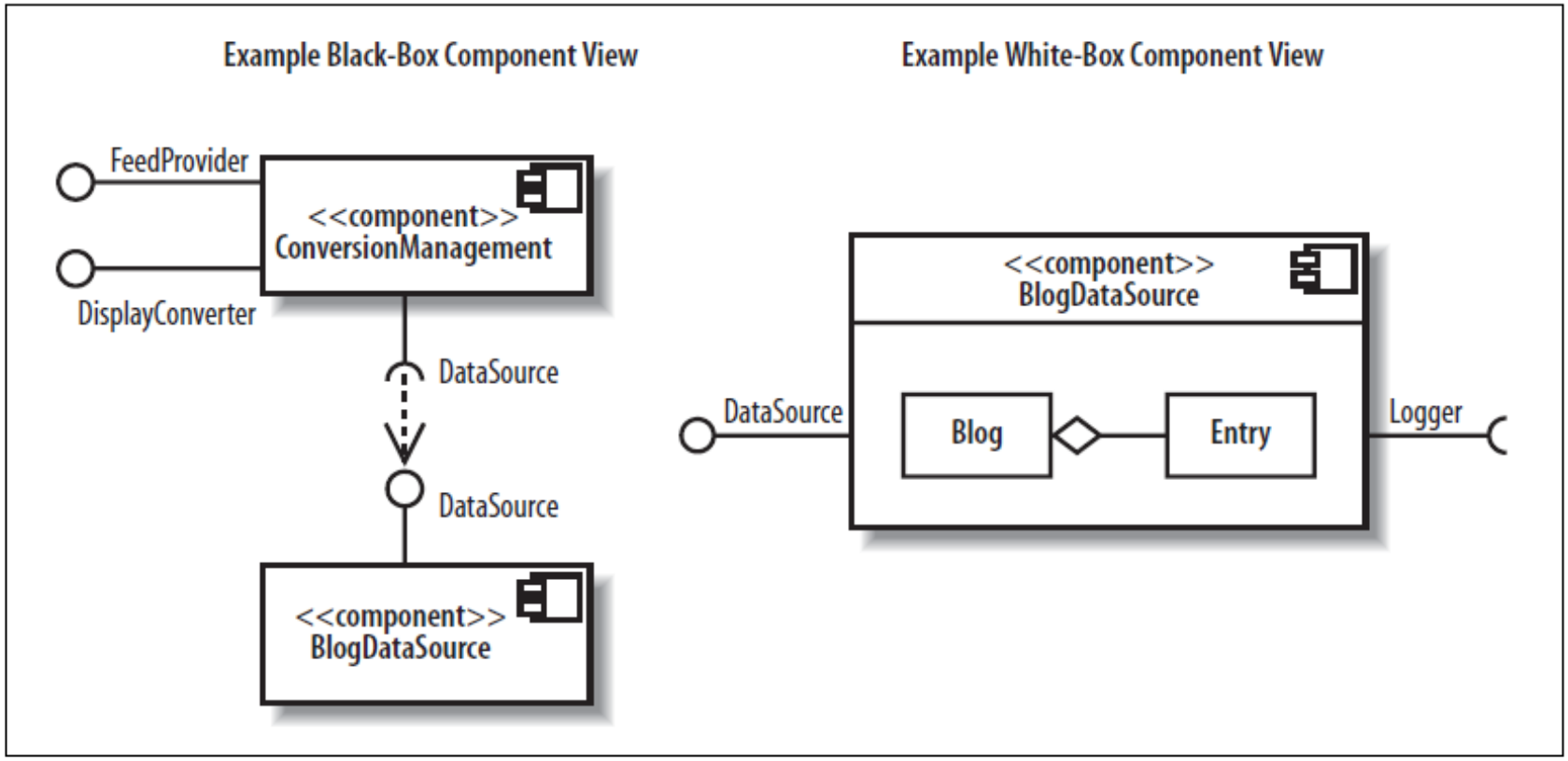
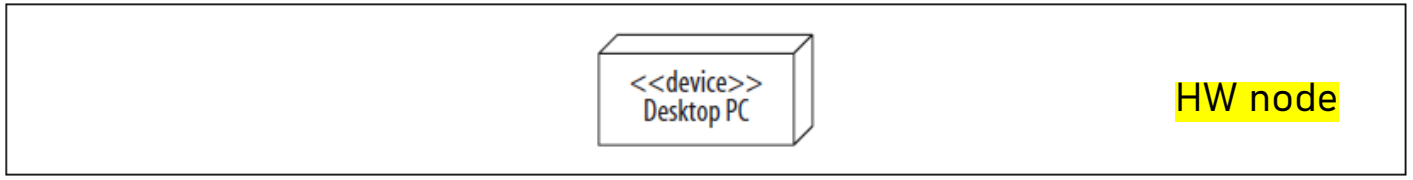


Figure 12-20. Black-box component views are useful for showing the big picture of the components in your system, whereas white-box views focus on the inner workings of a component

Deployment



HW node

Figure 15-2. Use nodes to represent hardware in your system

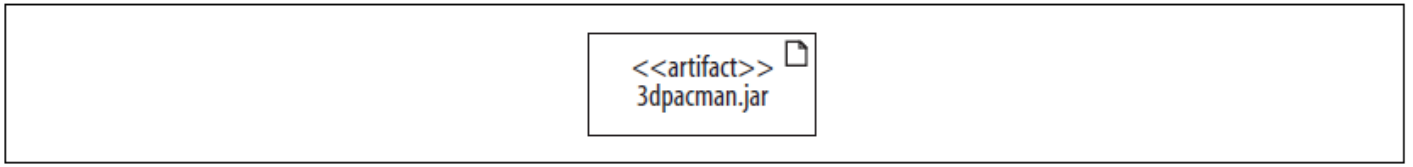
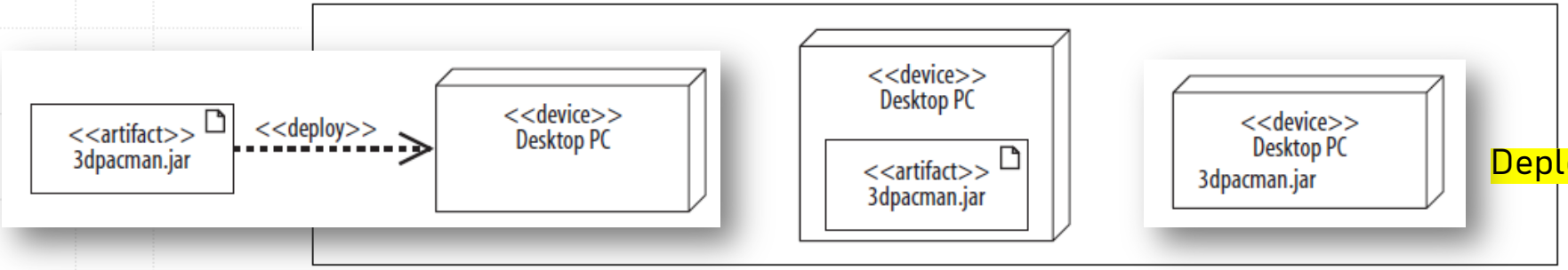


Figure 15-3. A physical software file such as a jar file is modeled with an artifact



Deploy Artifact in a Node

Figure 15-4. Drawing an artifact inside a node shows that the artifact is deployed to the node

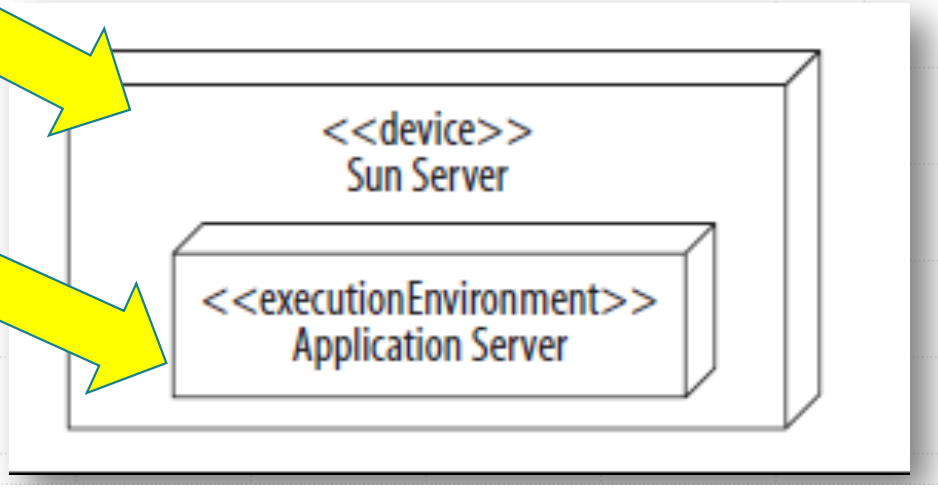
A *node* is a hardware or software resource that can host software or related files. You can think of a software node as an application context; generally not part of the software you developed, but a third-party environment that provides services to your software.

The following items are reasonably common examples of hardware nodes:

- Server
- Desktop PC
- Disk drives

The following items are examples of e...ion environment nodes:

- Operating system
- J2EE container
- Web server
- Application server



Communication Associations between Nodes

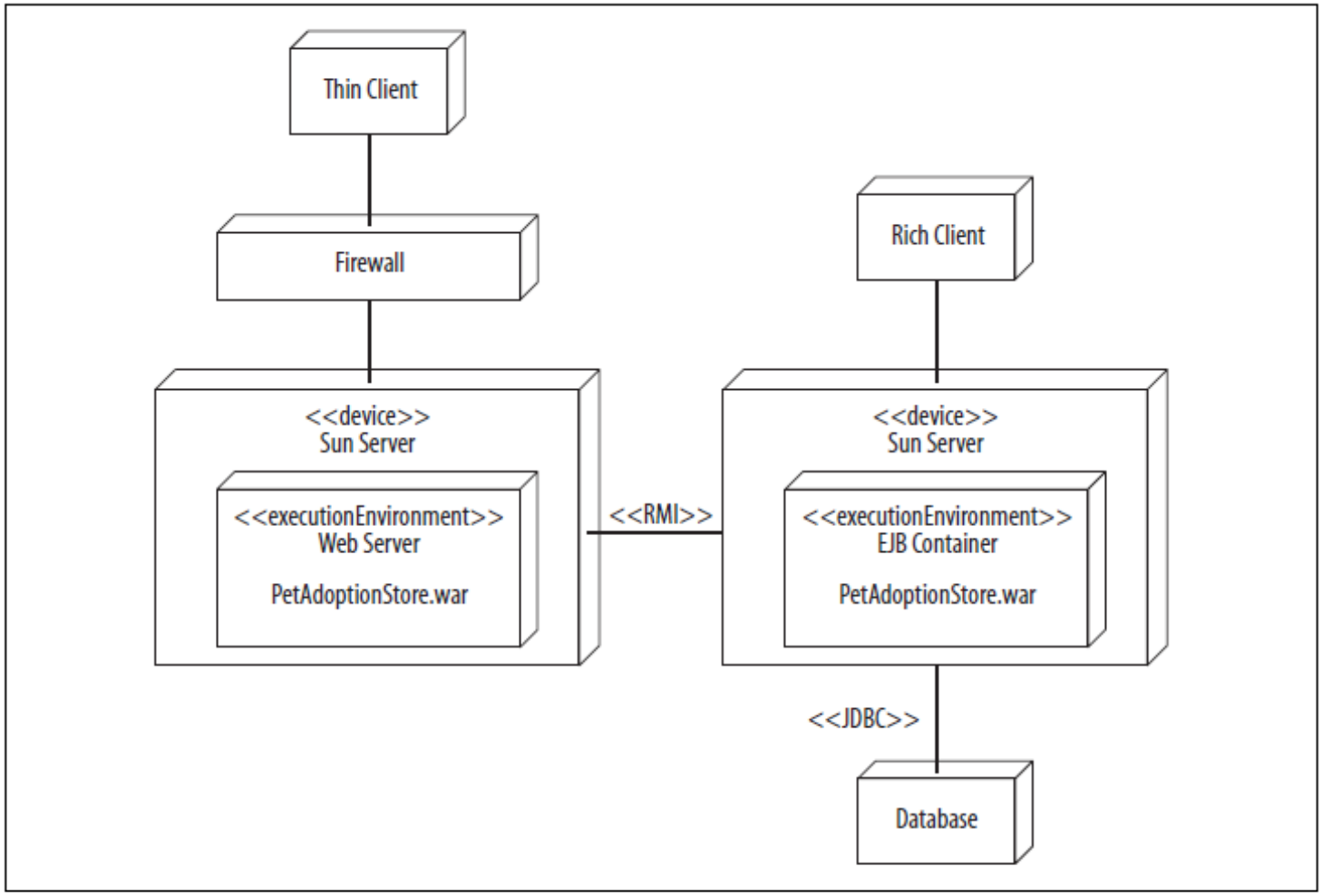
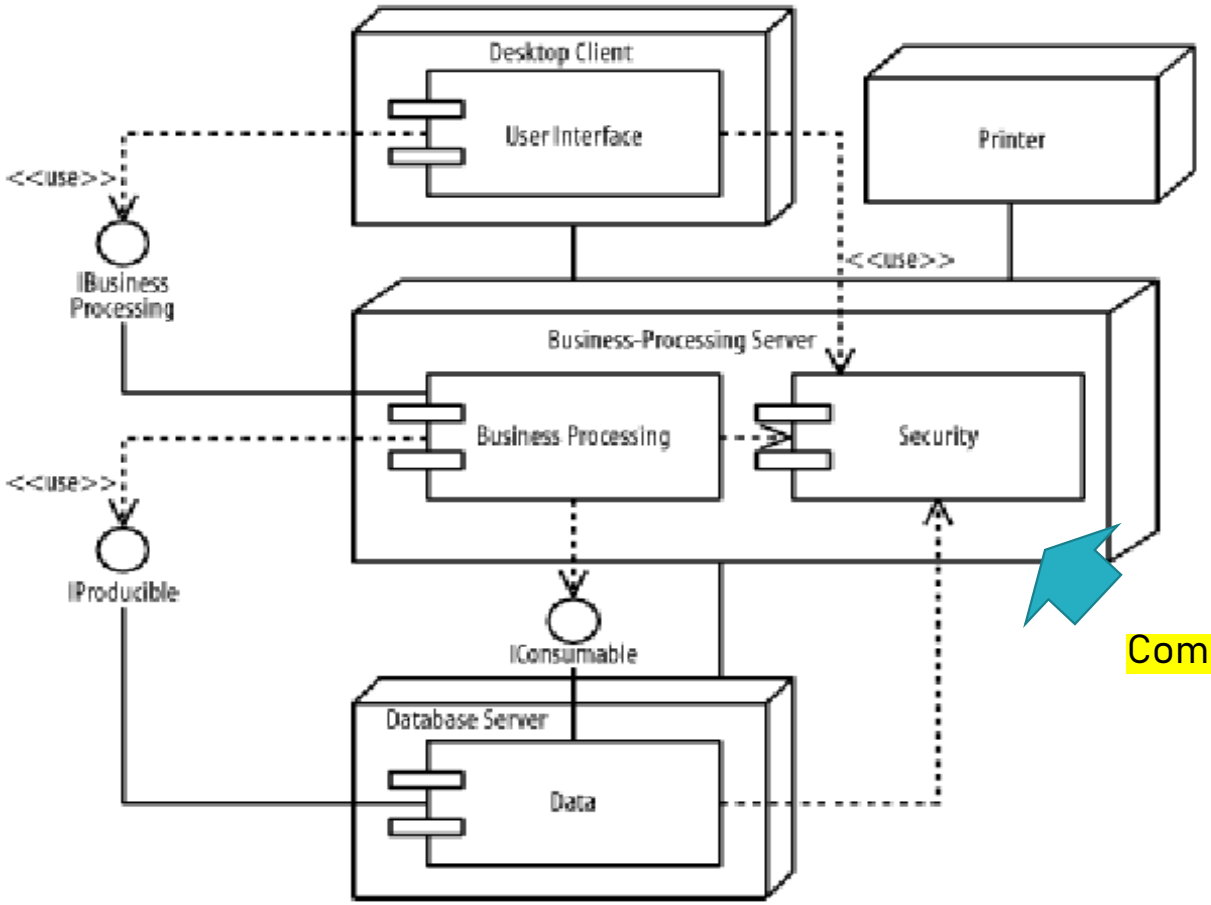
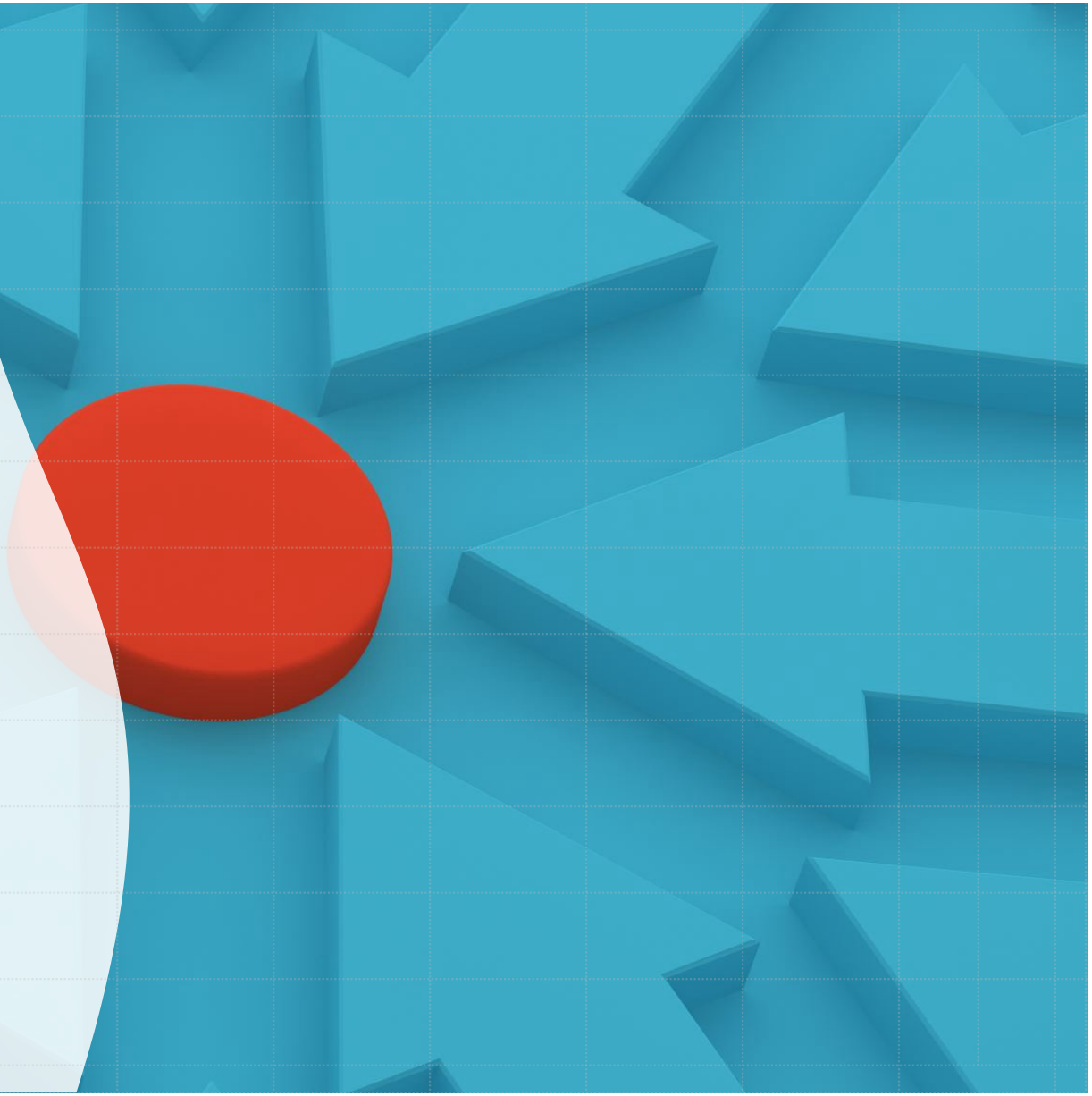


Figure 15-22. You can provide any amount of detail about the physical design of your system



Component in UML1.5

Epics to Subsystem Mapping

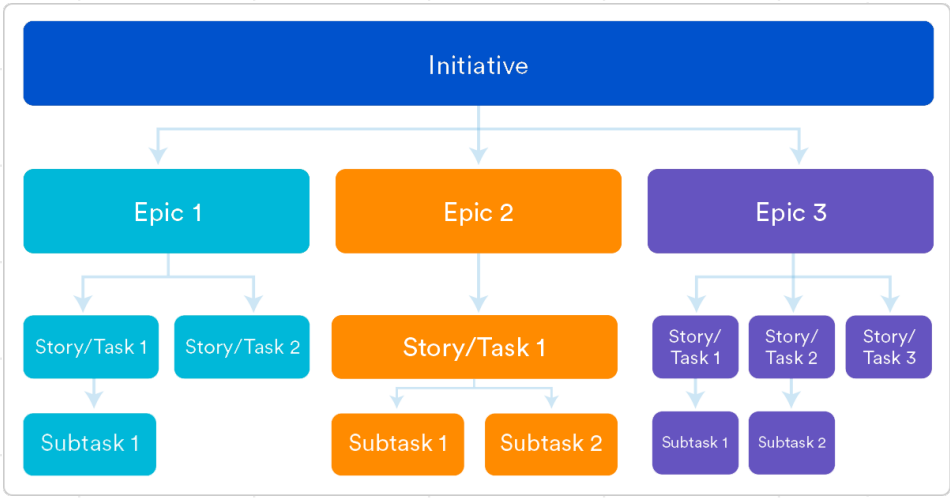


Product Backlog

EPIC ID	EPIC Name	User Story ID	User Story	Acceptance criteria	Estimate (User Story Points)
EPIC 2	Registration and Login system	US2-5	As a patient I want to edit my profile So that my profile is up-to-date	**Given** the patient is logged in into the system **When** the patient edit their profile **Then** ensure edited profile is recorded	5
		US2-6	As a physician I want to edit my profile So that my profile is up-to-date	**Given** the physician is logged in into the system **When** the physician edit their profile **Then** ensure edited profile is recorded	5
		US2-7	As a patient I want to log in to the system so that I can consult with a physician	**Given** the patient is not logged in into the system **When** the patient log in to the system **Then** display the log in form and authenticate the user	21
		US2-8	As an admin I want to verify the physicians' profiles so that I can ensure the credibility of the system	**Given** the physician profile **When** the admin wants to verify the physician profile **Then display** the physician's profile **And** allow the admin to accept or deny the profile **And** notify that physician about the verifying status.	21
EPIC 3	Video call system	US3-1	As a patient I want to request a video call to a physician so that I can ask about my illness directly	**Given** the physician accept matching request **And** the patient's device can use video call function **And** the physician's device can use video call function **When** the patient send a video call request to the physicain **Then** ensure the request is sent to the physician **And** ensure the patient is informed about the request status	8
		US3-2	As a physician I want to request a video call to a patient so that I can give advice to him/her about his/her illness	**Given** the physician accept matching request **And** the patient's device can use video call function **And** the physician's device can use video call function **When** the physician send a video call request to the patient **Then** ensure the request is sent to the patient **And** ensure the physician is informed about the request status	8
		US3-3	As a patient I want to accept video call requests from physicians so that I can ask about my illness directly	**Given** the patient receive a video call request from the physician **When** the patient accept the video call request **Then** trigger video call function on both patient's and physician's device **And** ensure both patient and physician can turn camera	55

Stories, epics, and initiatives

- **Stories**, also called “user stories,” are short requirements or requests written from the perspective of an end user.
- **Epics** are large bodies of work that can be broken down into a number of smaller tasks (called stories).
- **Initiatives** are collections of epics that drive toward a common goal.



Reference:- https://www.atlassian.com/agile/project-management/

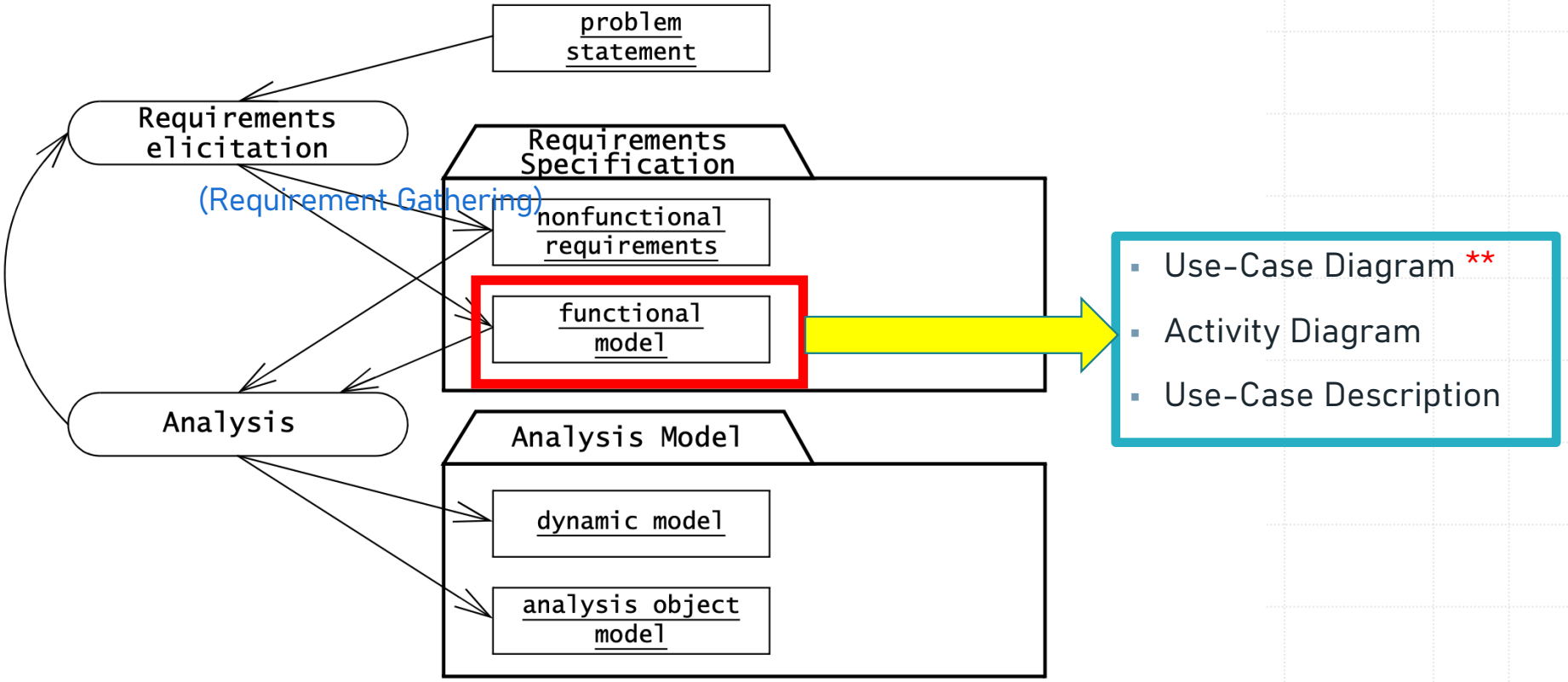
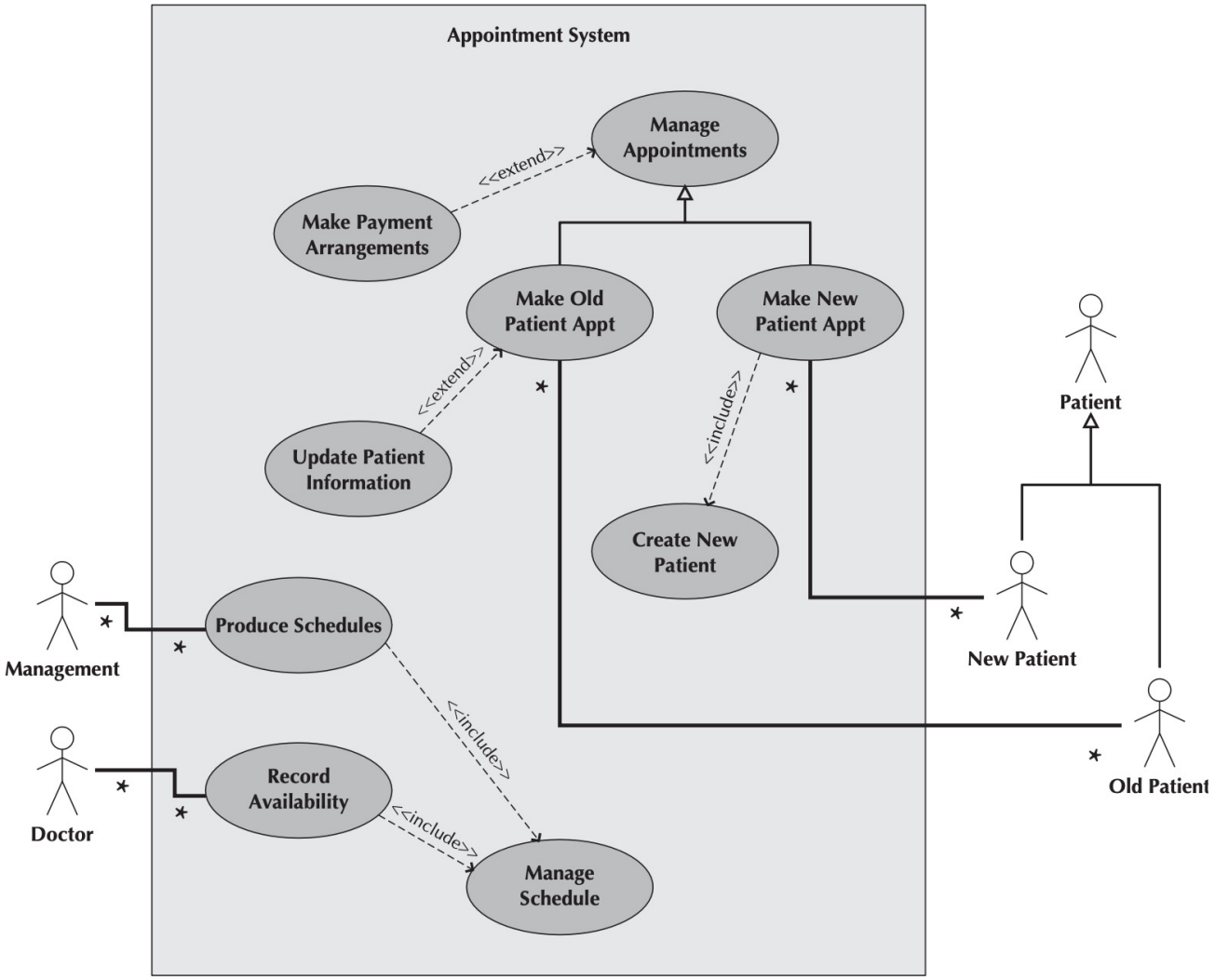


Figure 4-1 Products of requirements elicitation and analysis (UML activity diagram).

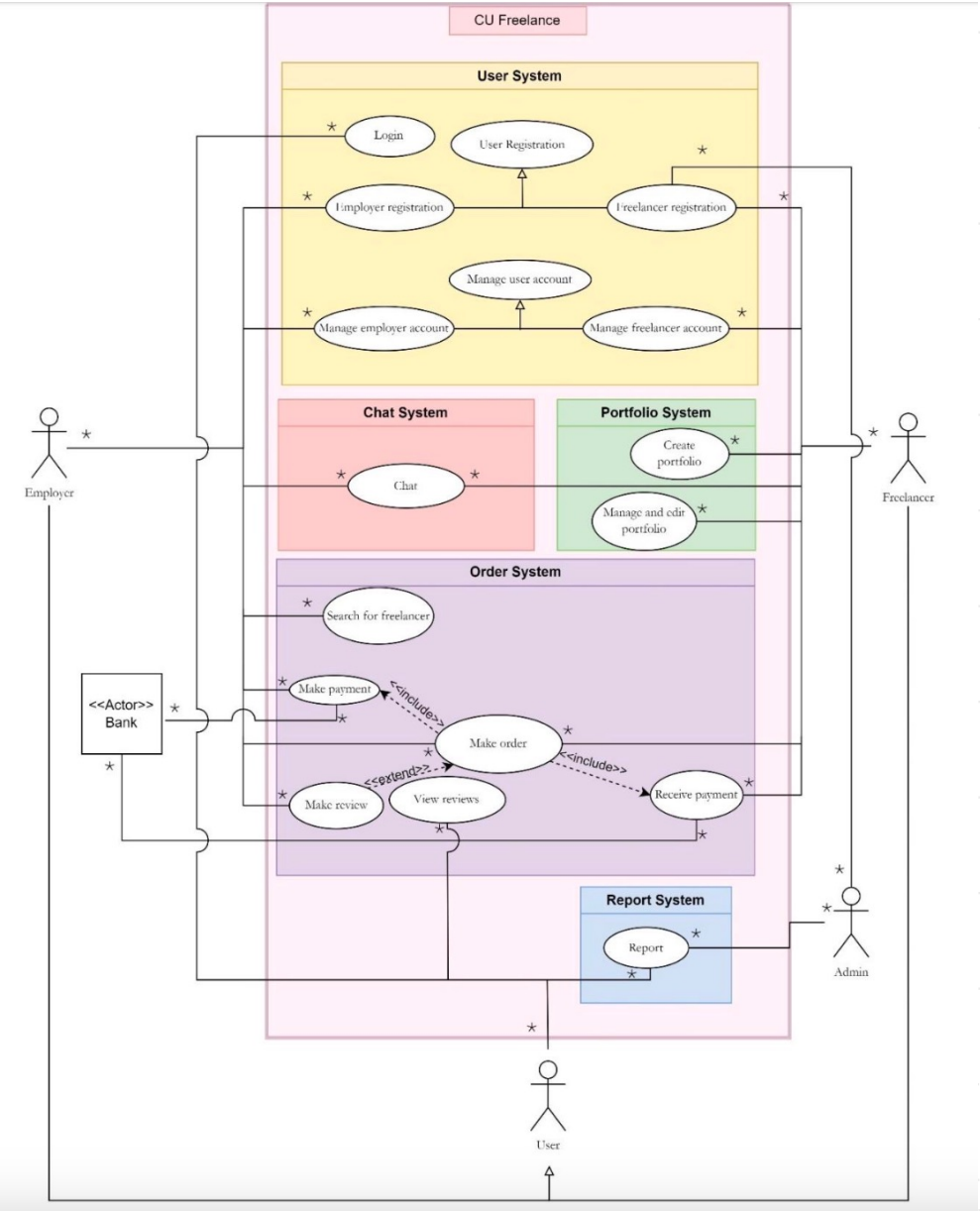
From Bernd Bruegge and Allen Dutoit "Object-Oriented Software Engineering Using UML, Patterns, and Java" 3rd Edition, Pearson.

Example of Use-Case Diagram

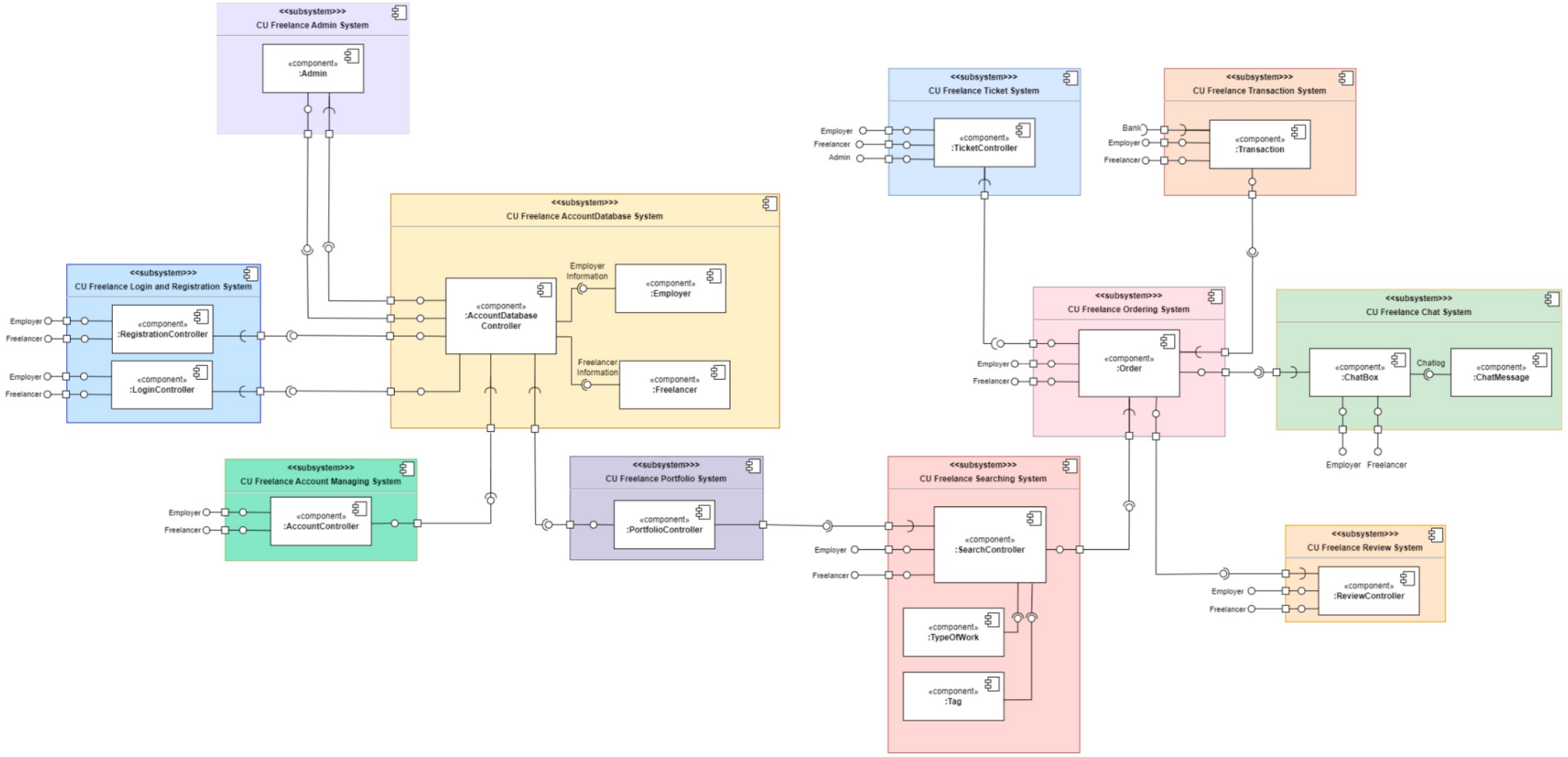


From Alan Dennis, Barbara Haley Wixom, David Tegarden “Systems Analysis and Design An Object-Oriented Approach with UML”, 6th Edition, John Wiley and Sons, Inc. 2020

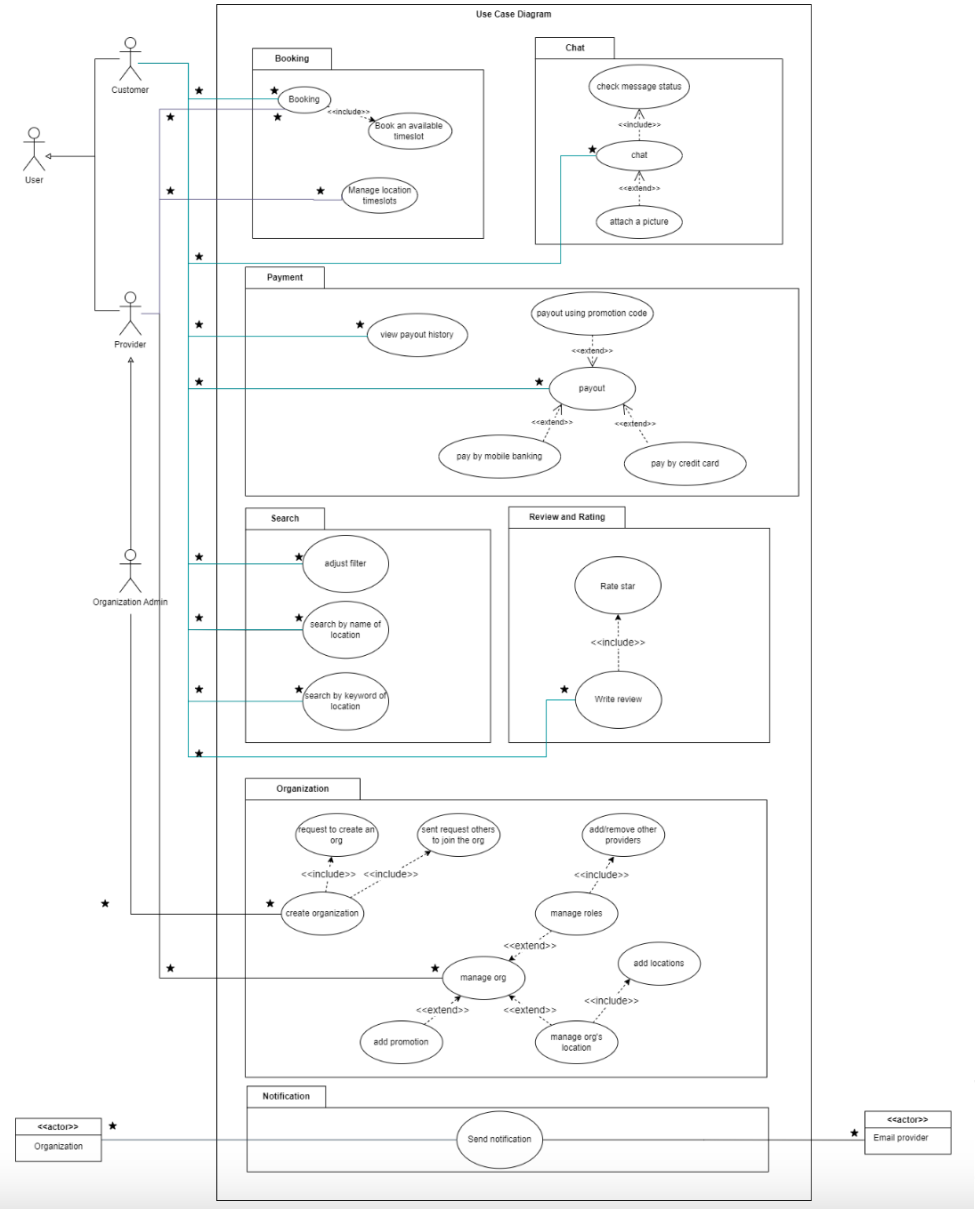
Use-Case Diagram
“CU Freelance” (2022/1)



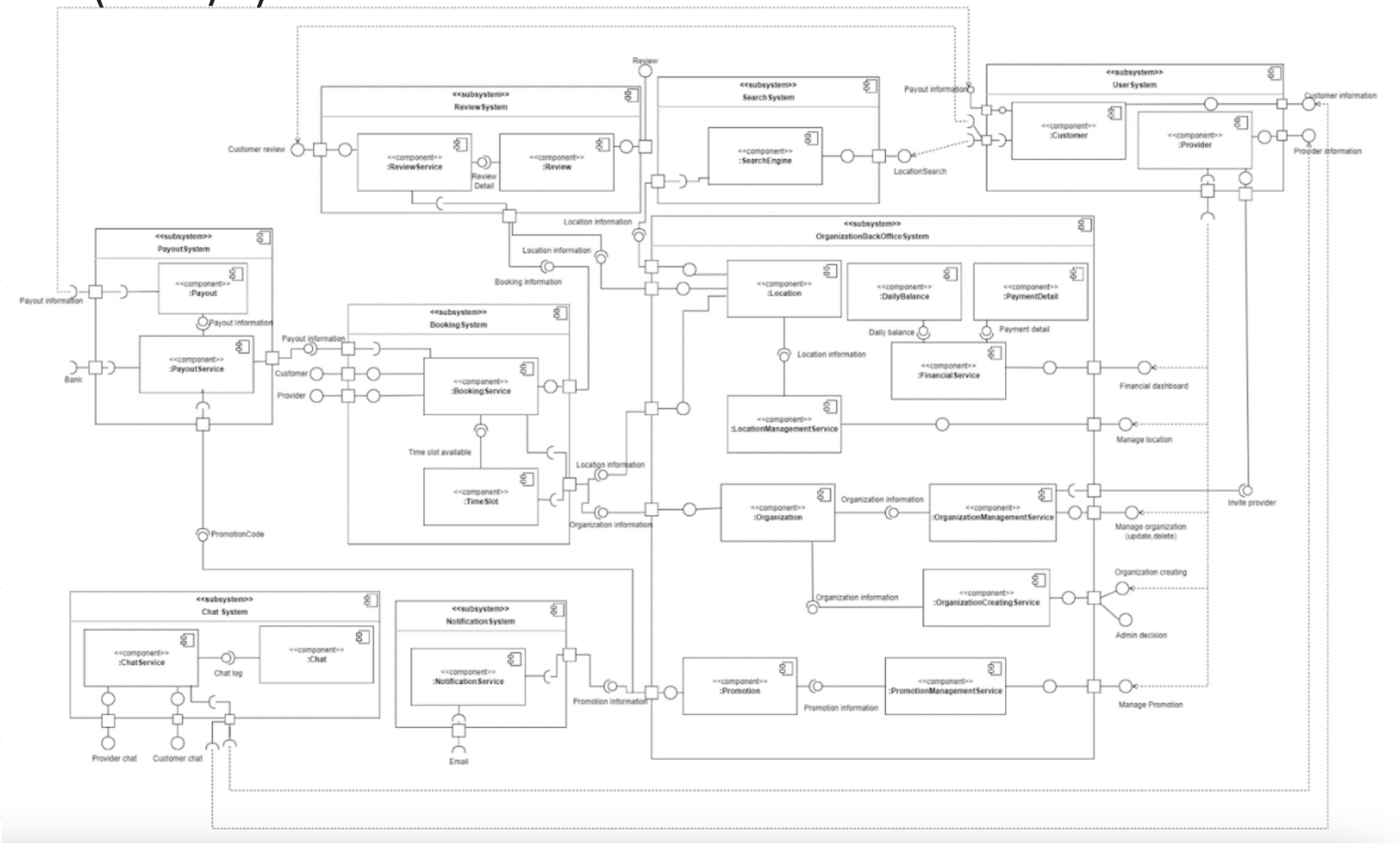
Component Diagram
“CU Freelance” (2022/1)



Use-Case Diagram “BookBix” (2022/1)



Component Diagram
“BookBix” (2022/1)



Describing Entity Classes/Representations in Each Subsystem

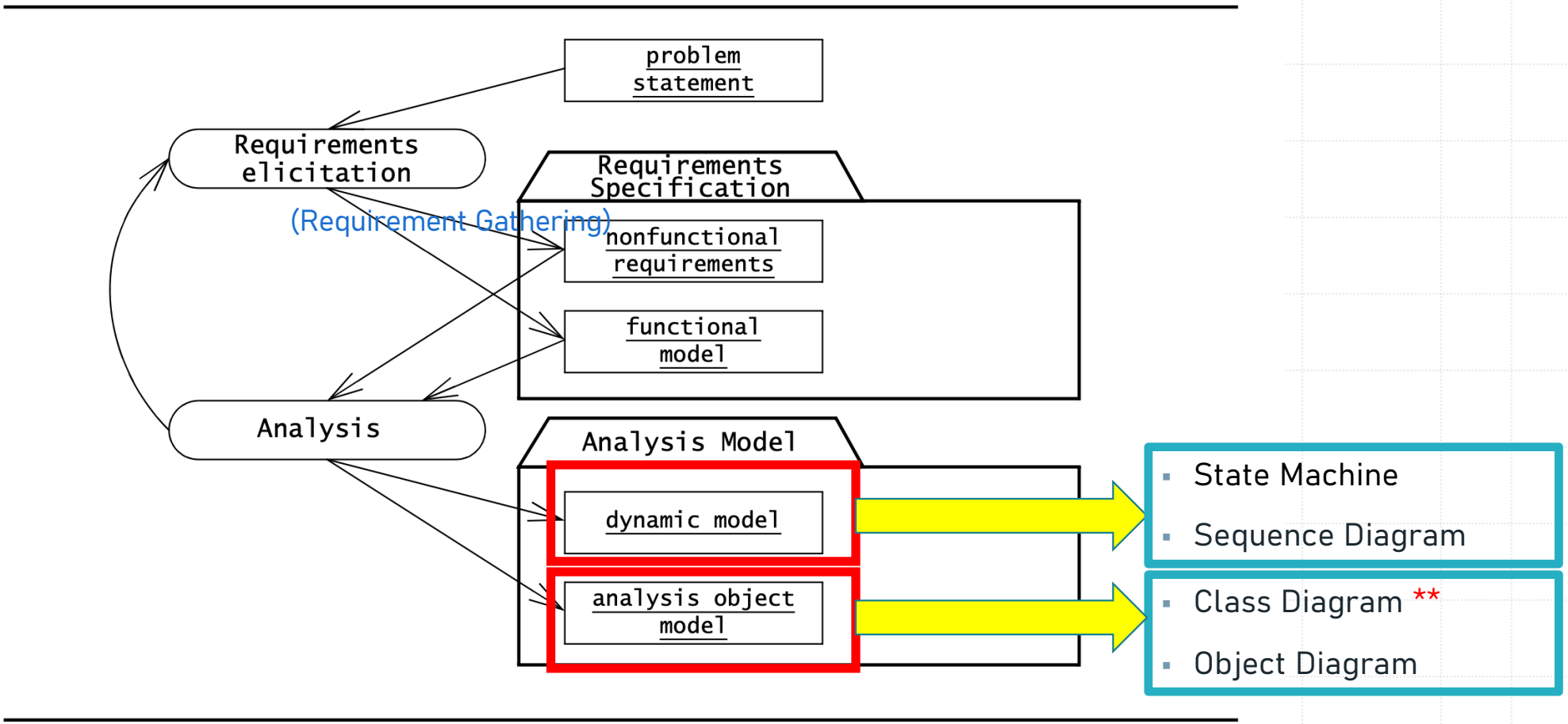
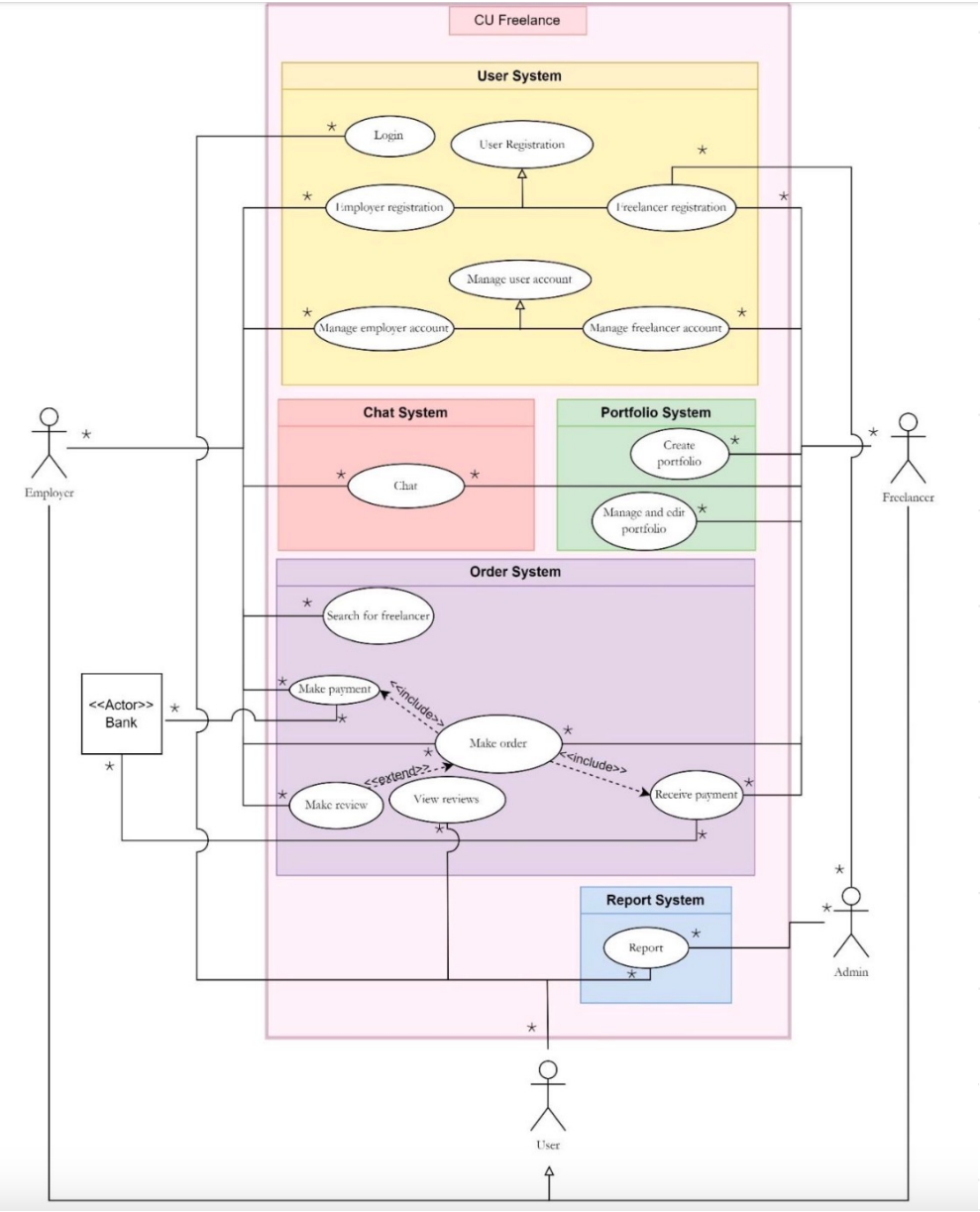


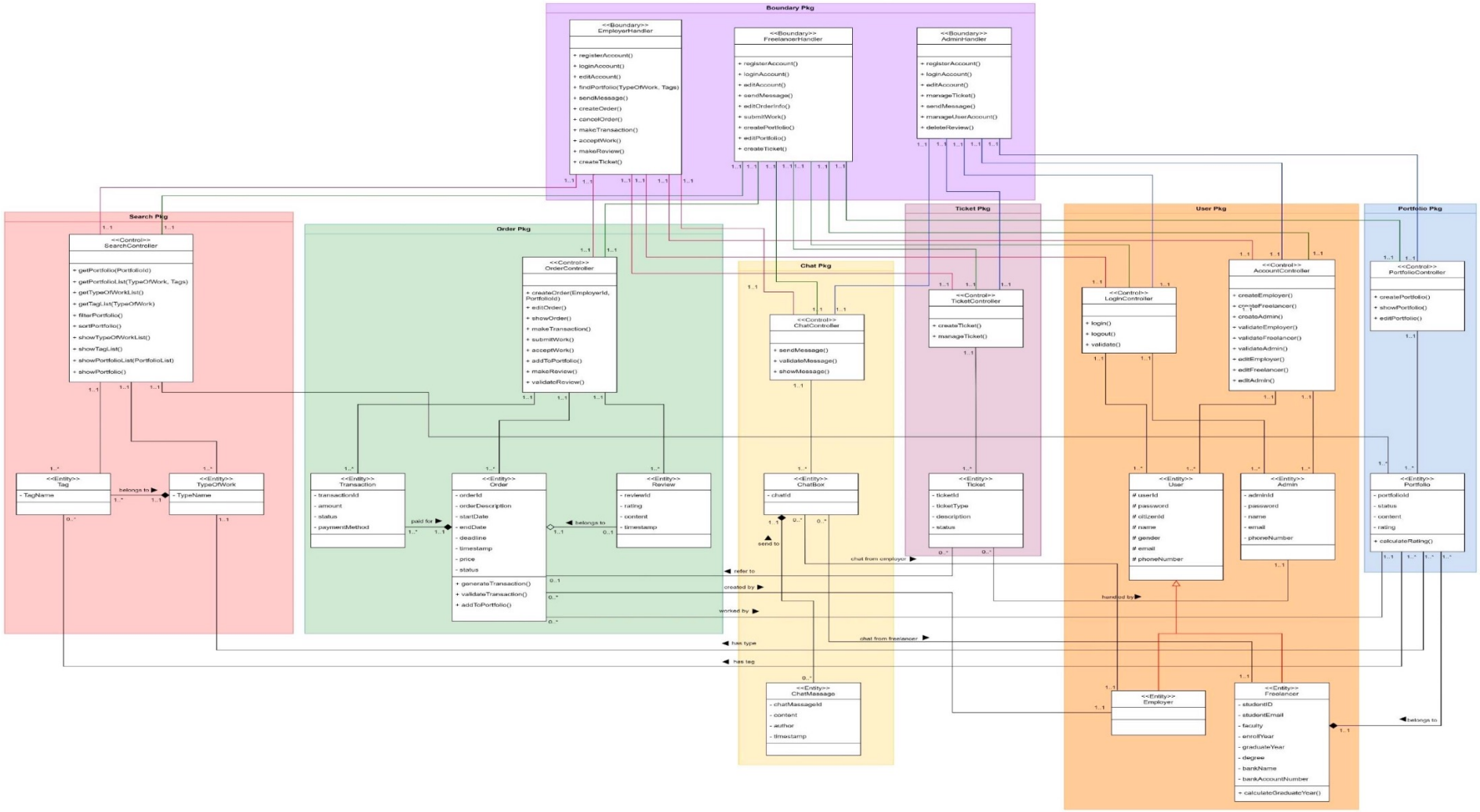
Figure 4-1 Products of requirements elicitation and analysis (UML activity diagram).

From Bernd Bruegge and Allen Dutoit "Object-Oriented Software Engineering Using UML, Patterns, and Java" 3rd Edition, Pearson.

Use-Case Diagram
“CU Freelance” (2022/1)



Class Diagram
“CU Freelance” (2022/1)



UML API Diagram
“CU Freelance” (2022/2)