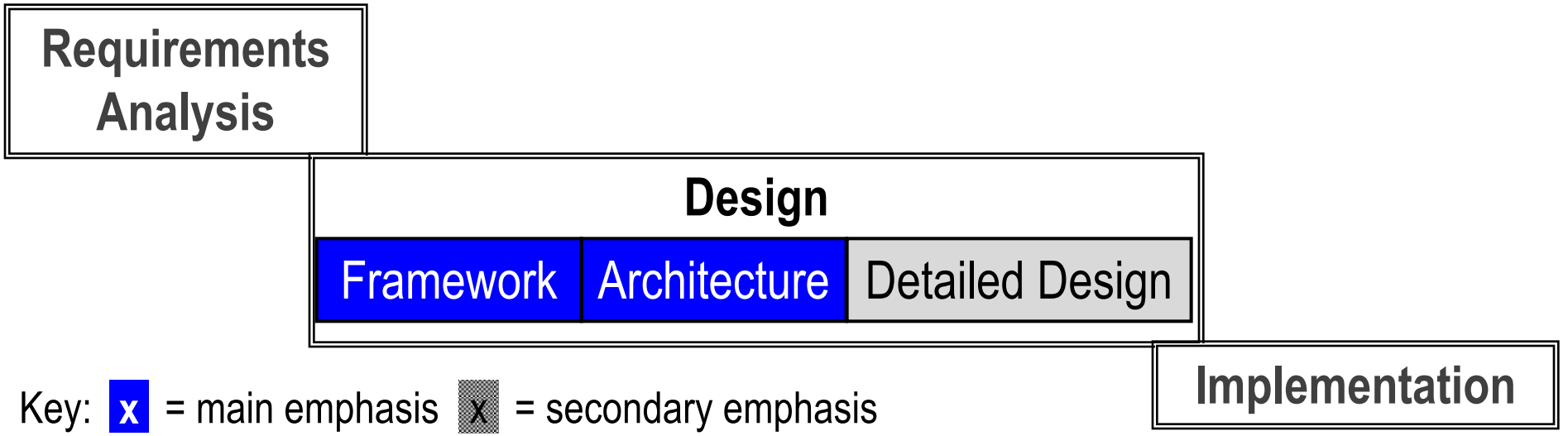
Architecture and Frameworks

1. **Context of Architecture**
2. Models
3. Classifying Architectures
4. Frameworks

A ***framework*** is a collection of **models**, typically a class model, that forms a template for a category of **architectures**.



Software designs at the high level are referred to as "architectures."

Relating Use Cases, Architecture, & Detailed Design

1. Use case (part of requirements)

"Cars should be able to travel from the top of Smith Hill at 65 mph, travel in a straight line, and arrive at Jones Hollow within 3 minutes."

*

**

2. Domain classes

Auto

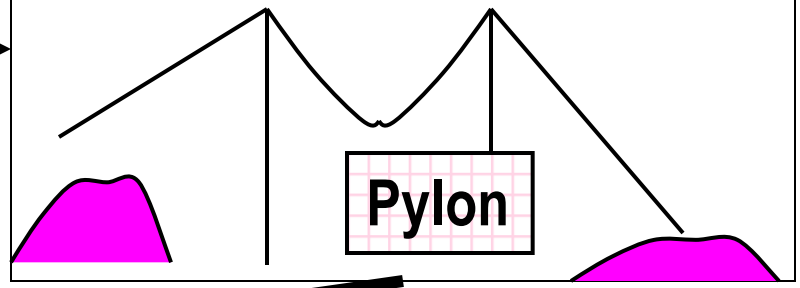
Road

3. Architecure

(not specifically *required*)

Cable

Pylon



4. Detailed Design

Roadbed

Guardrail

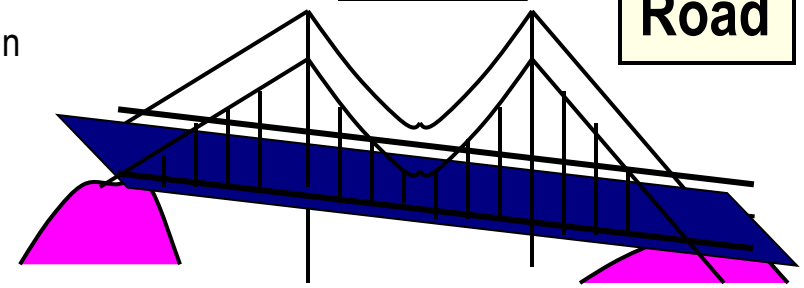
added for detailed design

Cable

Pylon

Auto

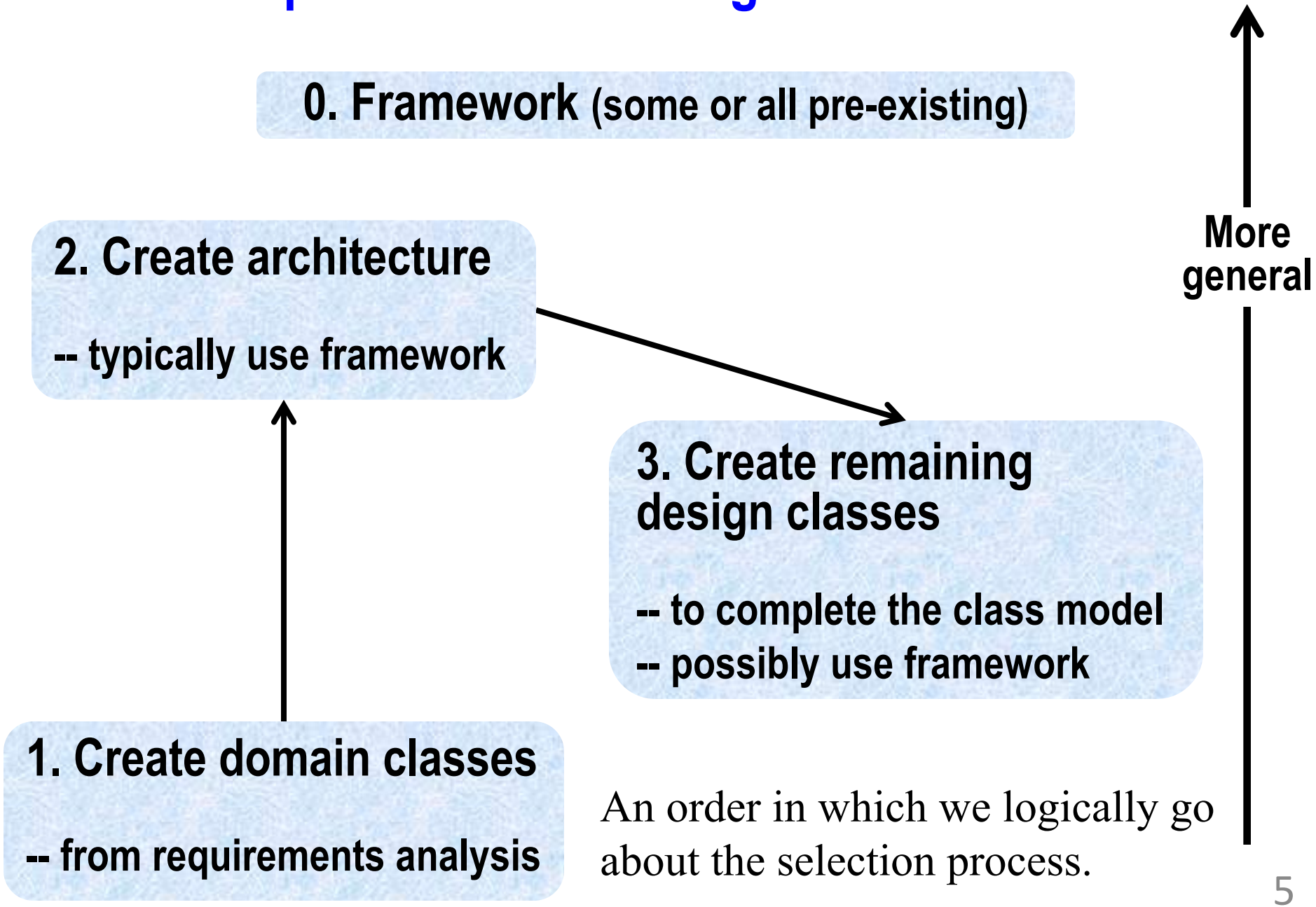
Road



* Use cases used to obtain domain classes.

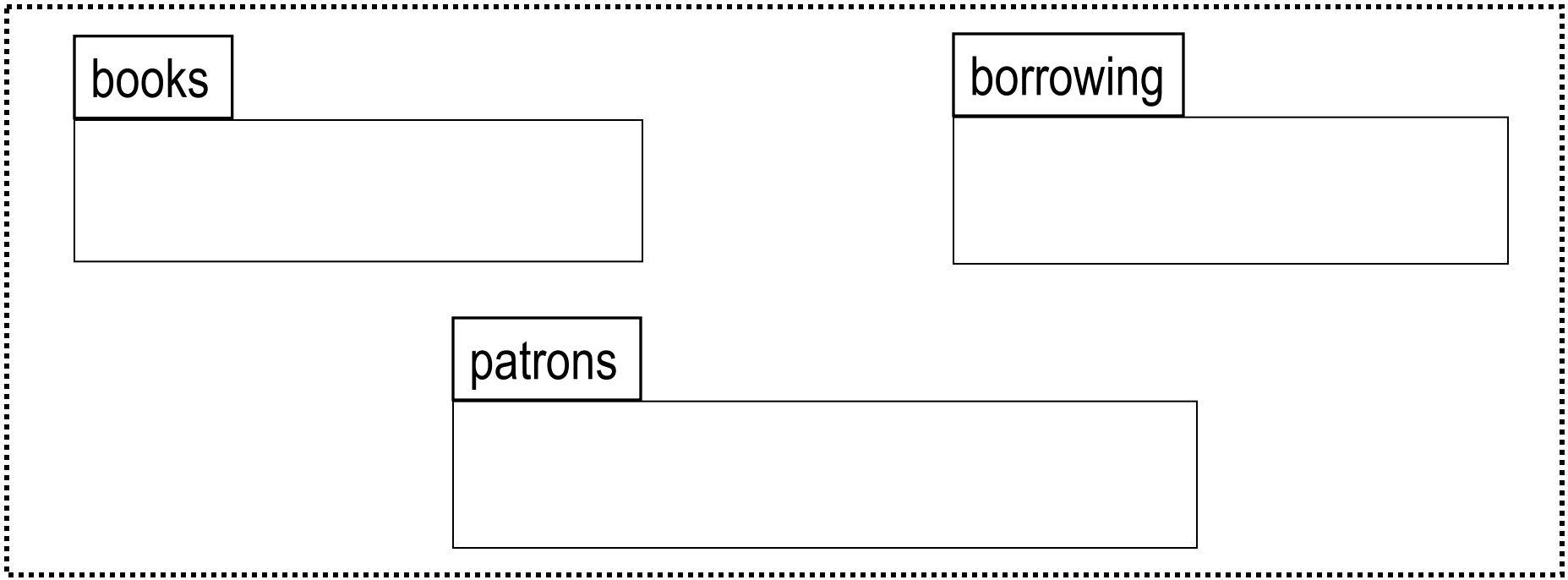
** Use cases used to verify and test design.

A Sequence for Obtaining The Class Model



An order in which we logically go about the selection process.

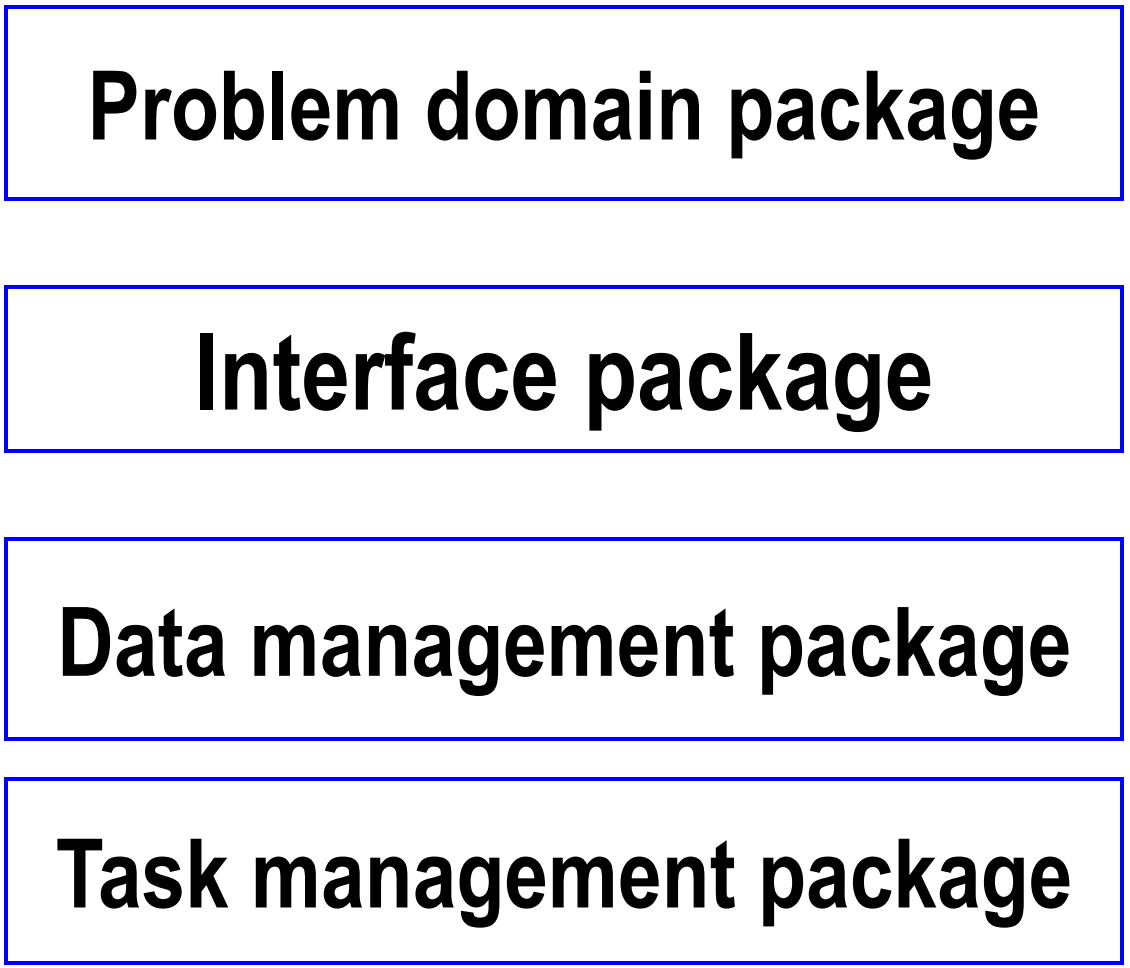
An Architecture for a Library Application



At a practical level, architecture selection amounts to the identification of packages. For example, we might decompose the classes of a library application into the books, the patrons, and the borrowing records that tie these two together.

Coad-Yourdon Use of *Packages*

One common architecture, expressed in terms of four generic packages.

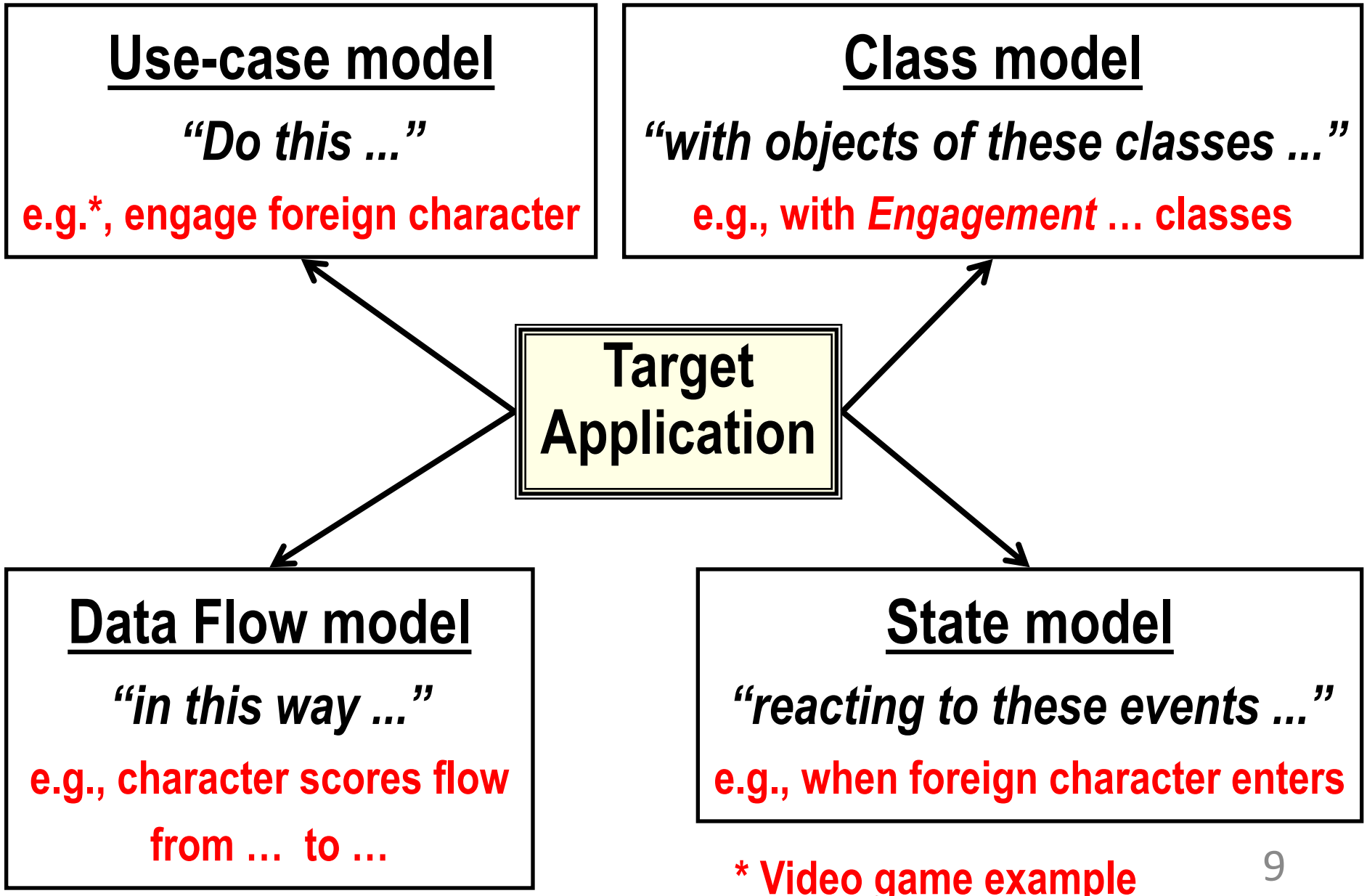


Book:
Coad, P. and E. Yourdon, Object-Oriented Analysis, 2nd ed., Prentice-Hall, 1991

1. Context of Architecture
2. **Models**
3. Classifying Architectures
4. Frameworks

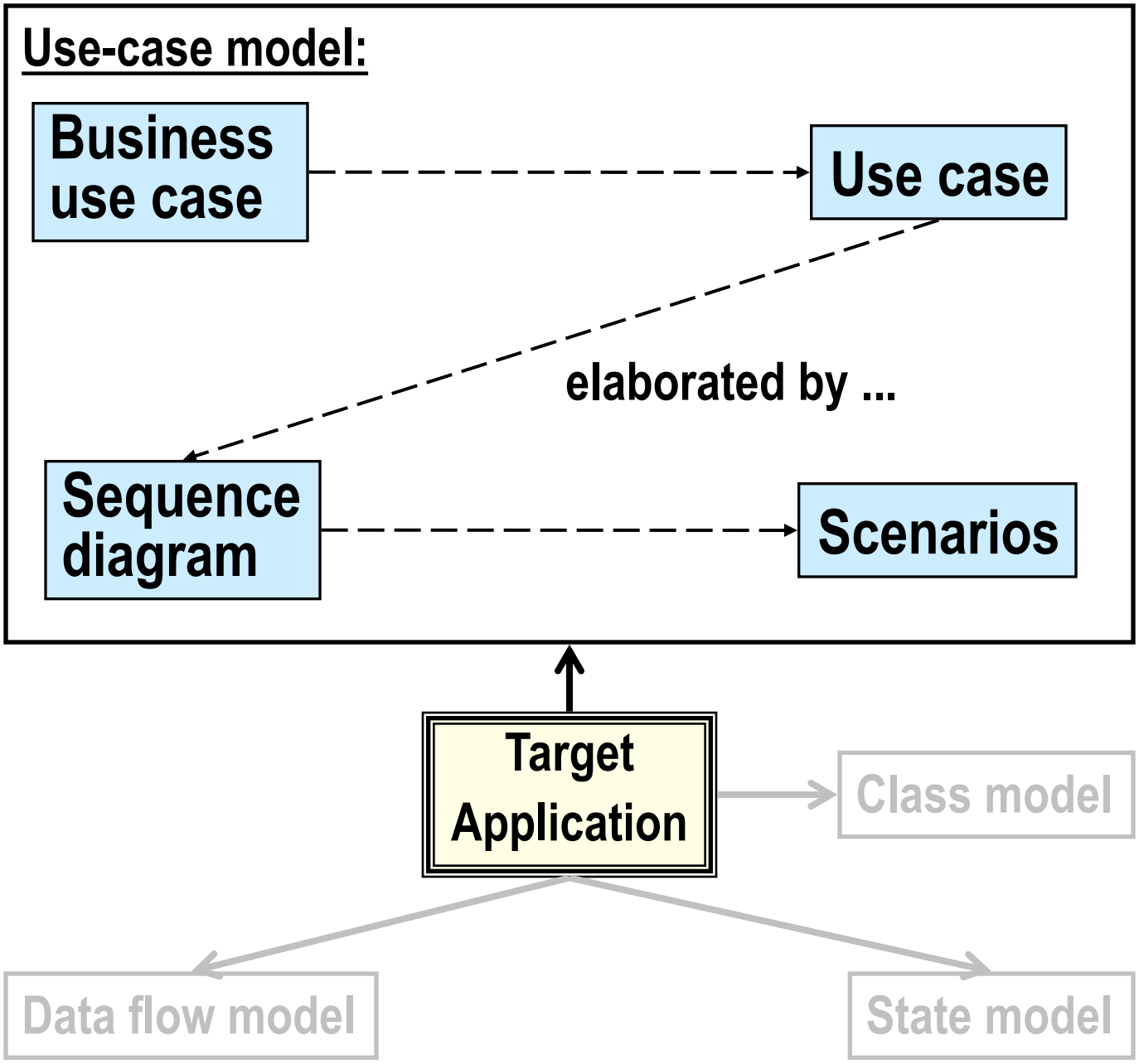
A ***framework*** is a collection of **models**, typically a class model, that forms a template for a category of **architectures**.

Models To express requirements, architecture & detailed design

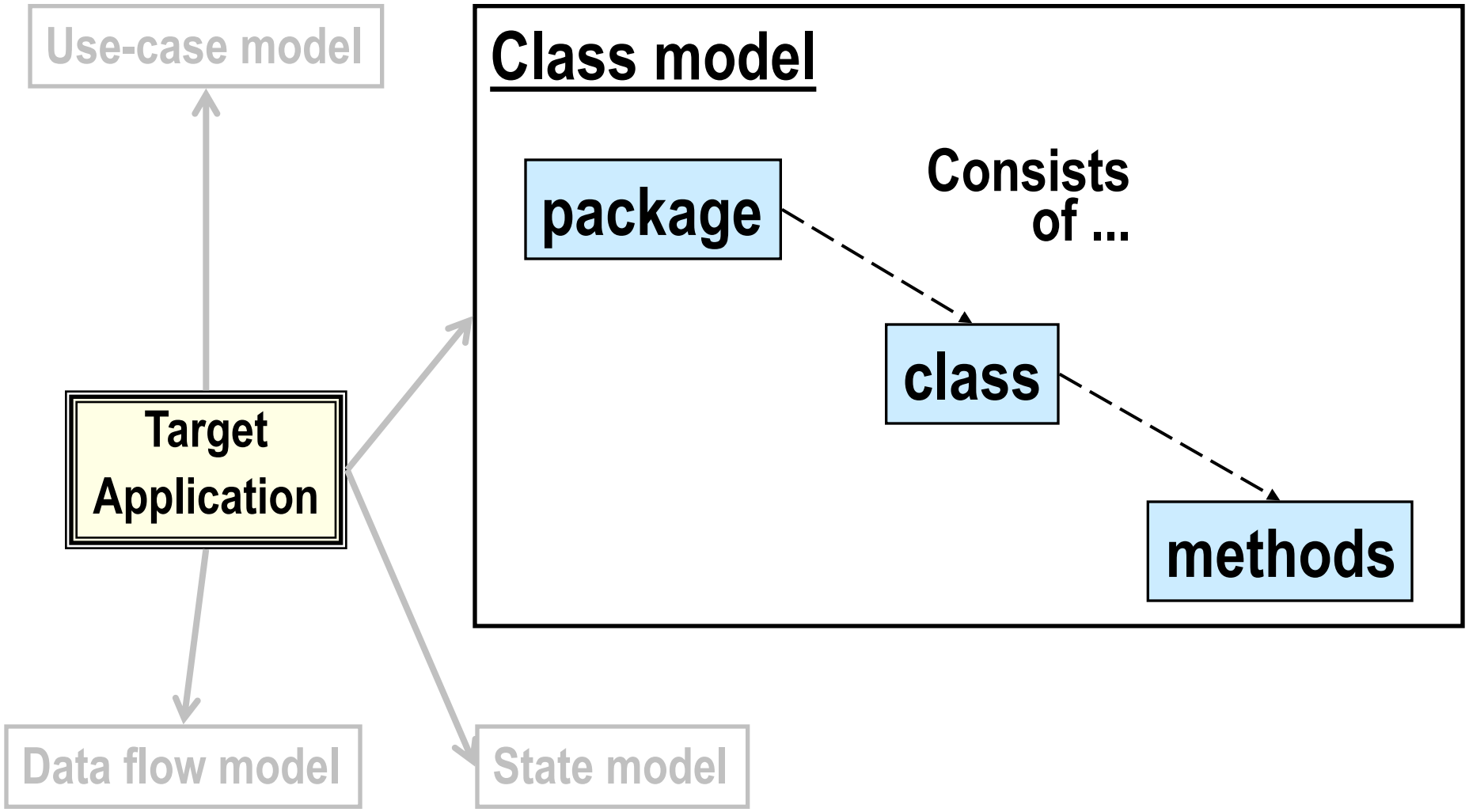


* Video game example

Role of Use Case Models

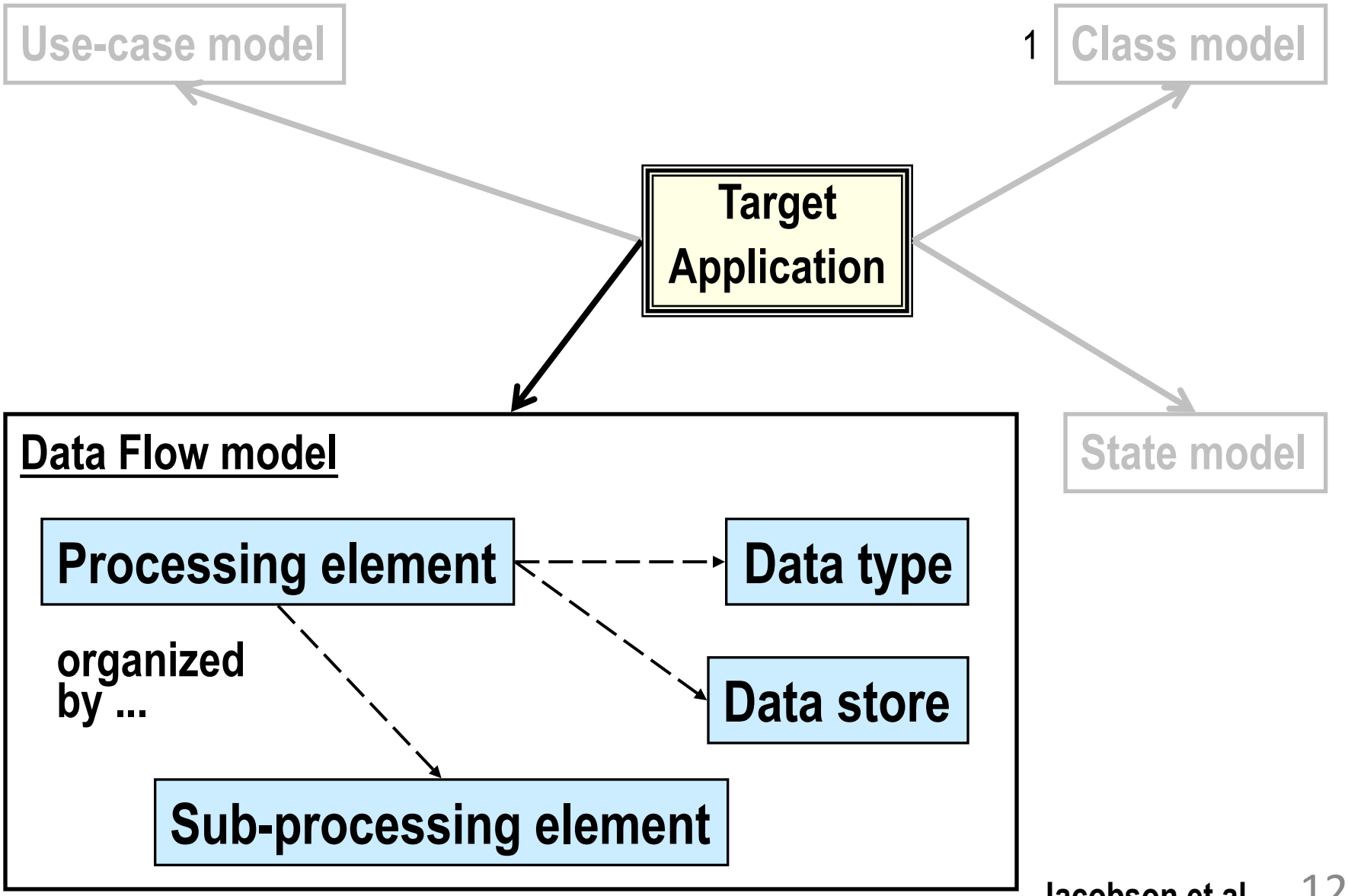


Role of Class Models



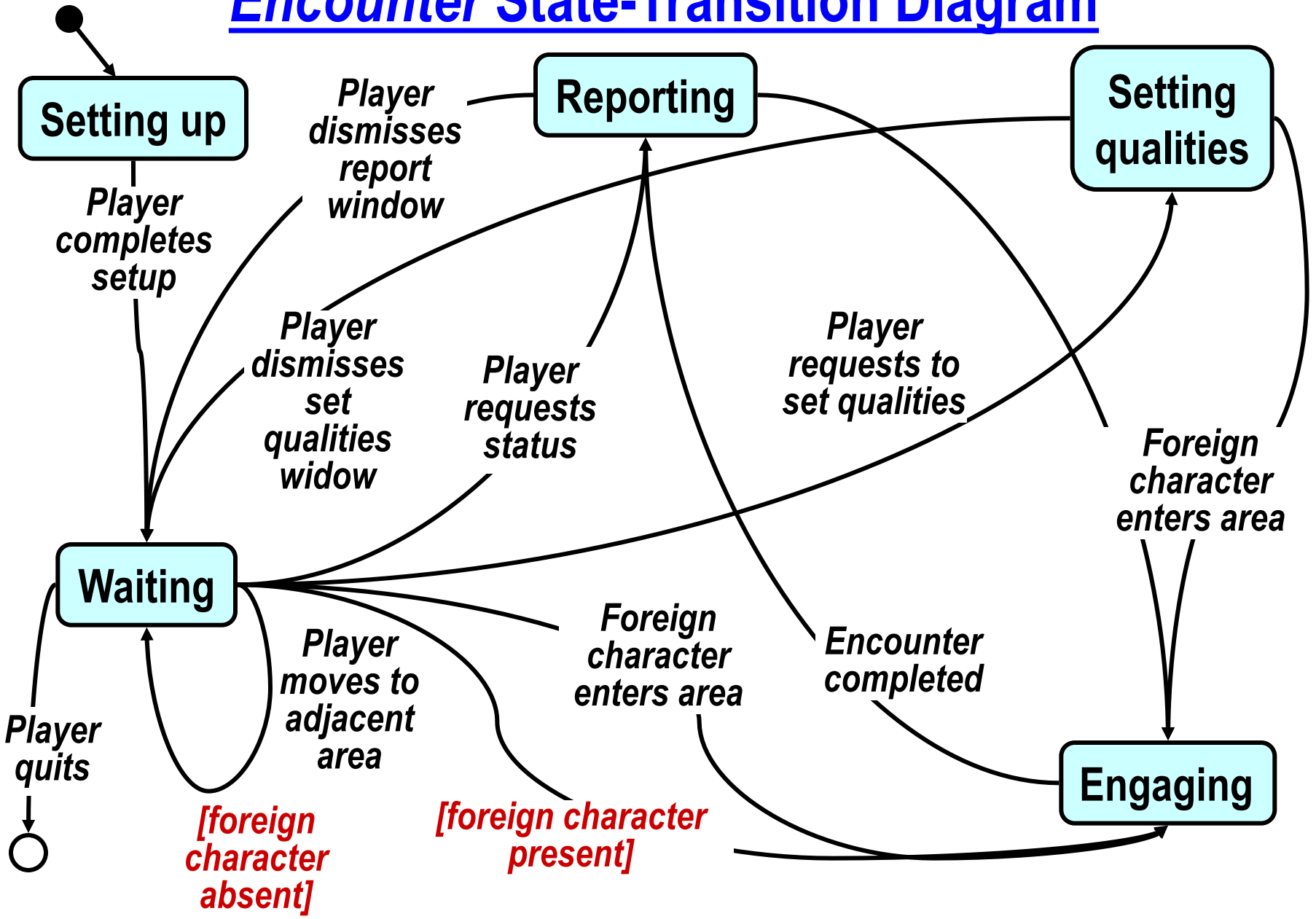
Jacobson et al

Role of Component Model

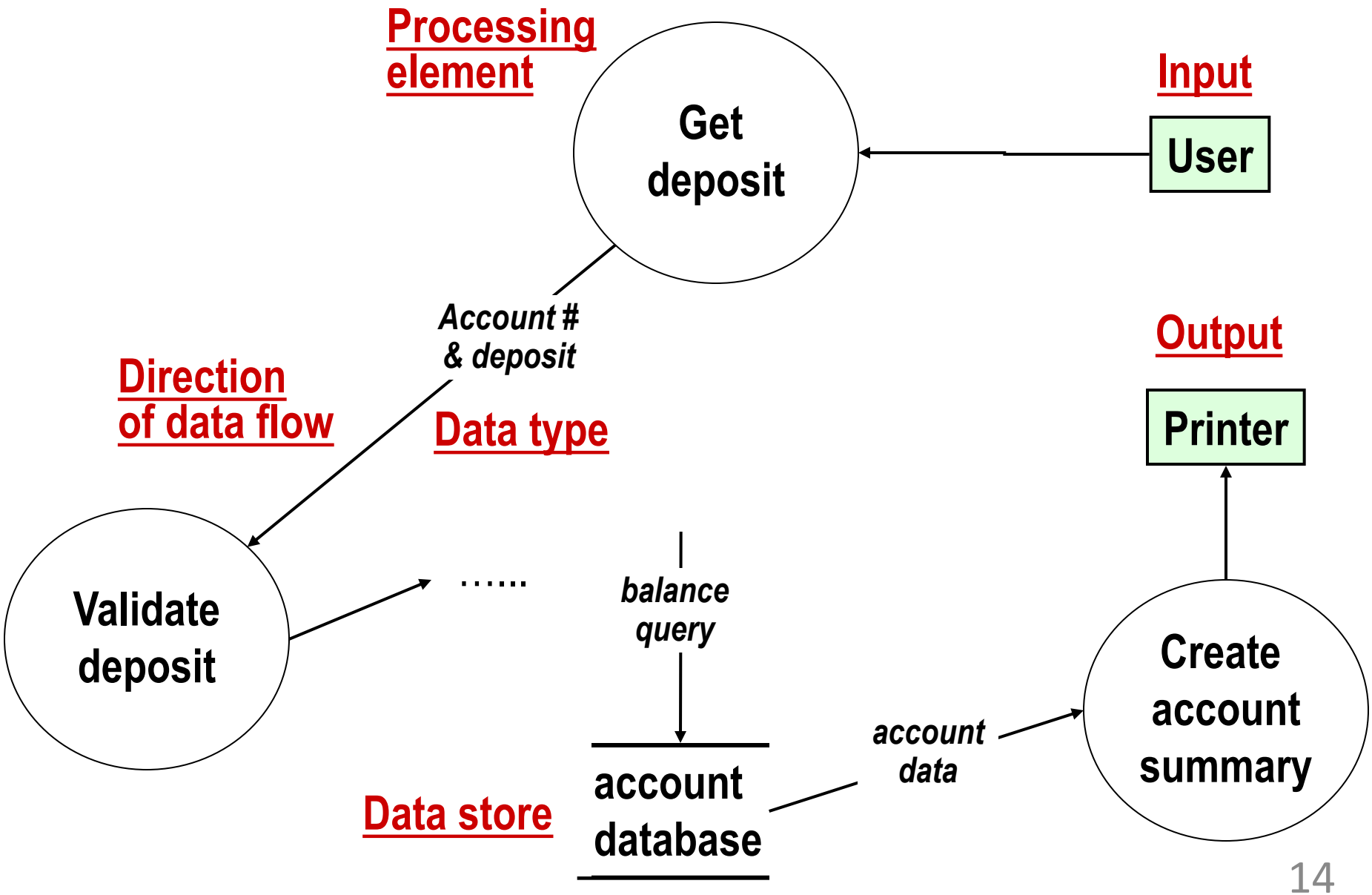


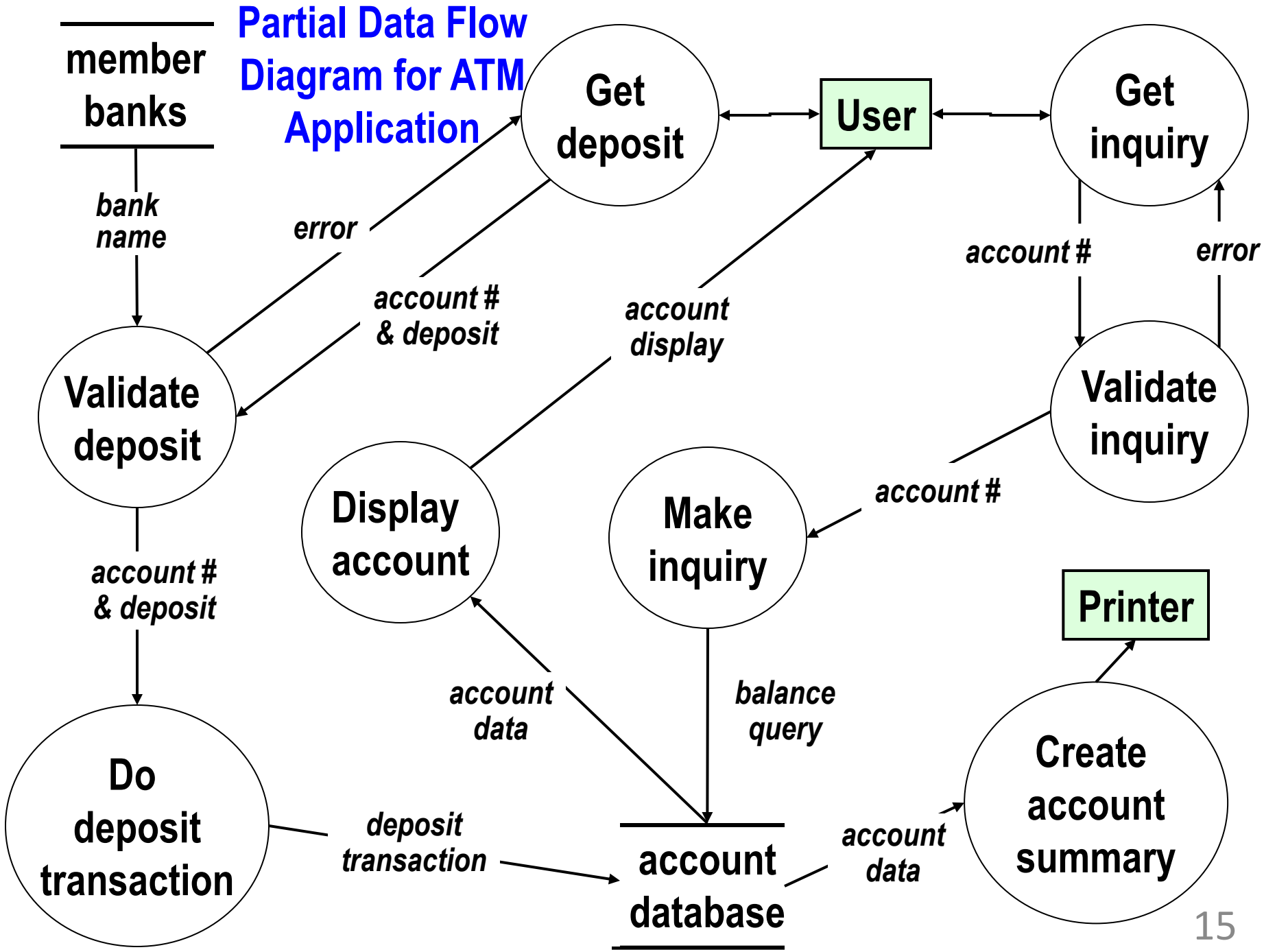
Jacobson et al

Encounter State-Transition Diagram

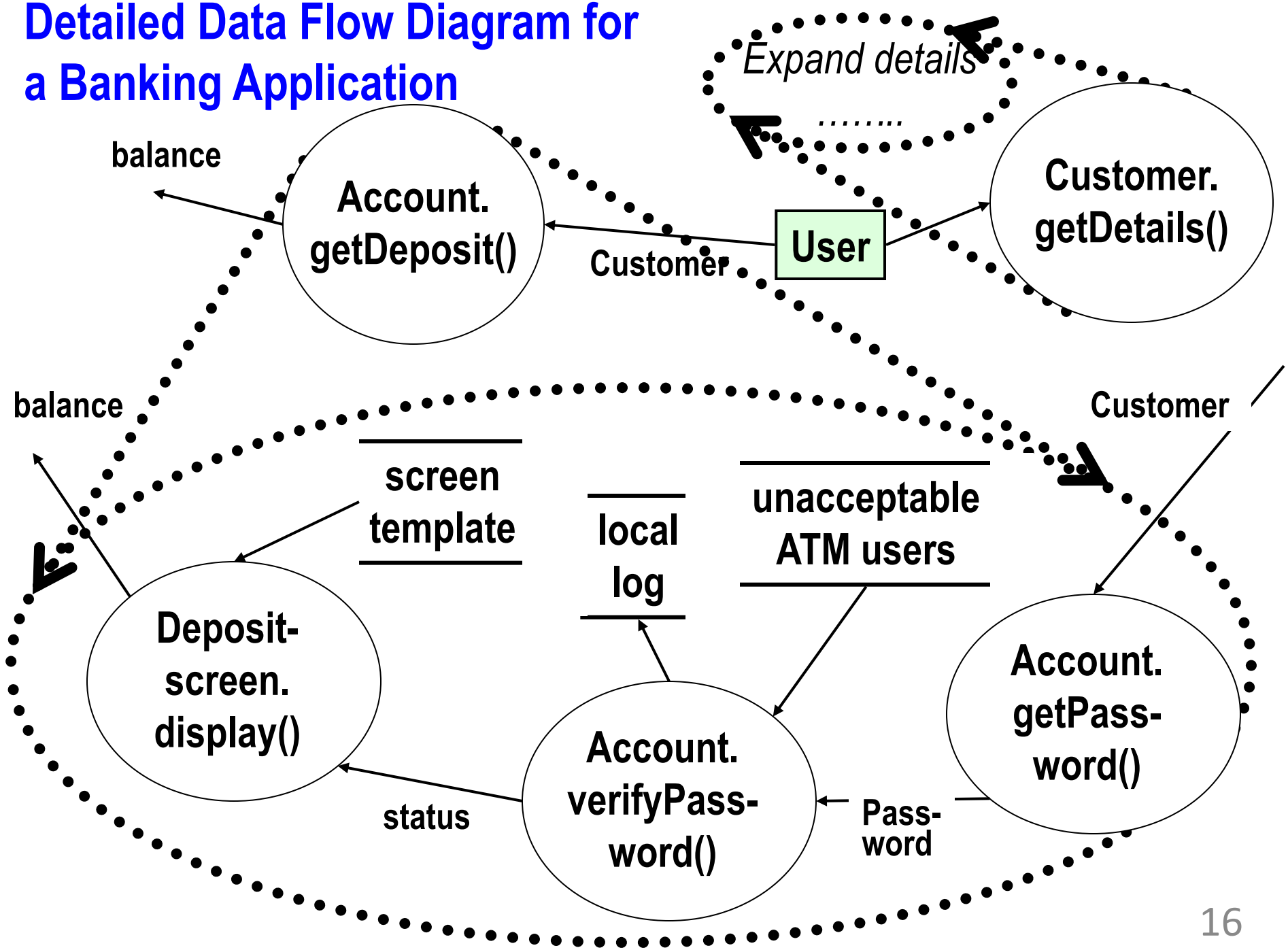


Data Flow Diagram: Explanation of Symbols

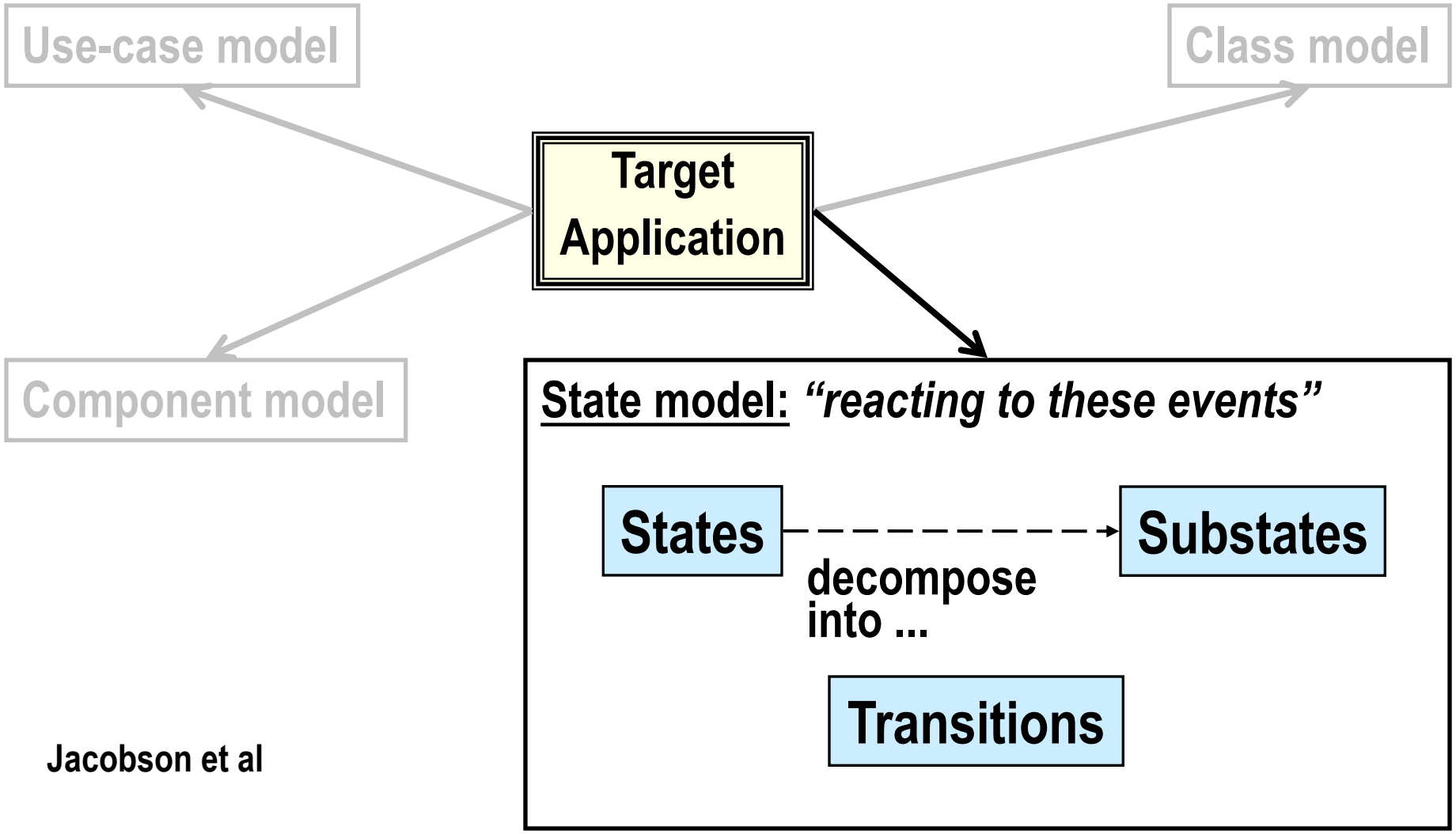



Partial Data Flow Diagram for ATM Application
member banks
bank name
Get deposit
User
Get inquiry
error
account # & deposit
account display
account #
error
Validate deposit
Validate inquiry
Display account
Make inquiry
account #
account # & deposit
Printer
account data
balance query
Do deposit transaction
deposit transaction
account database
account data
Create account summary

Detailed Data Flow Diagram for a Banking Application

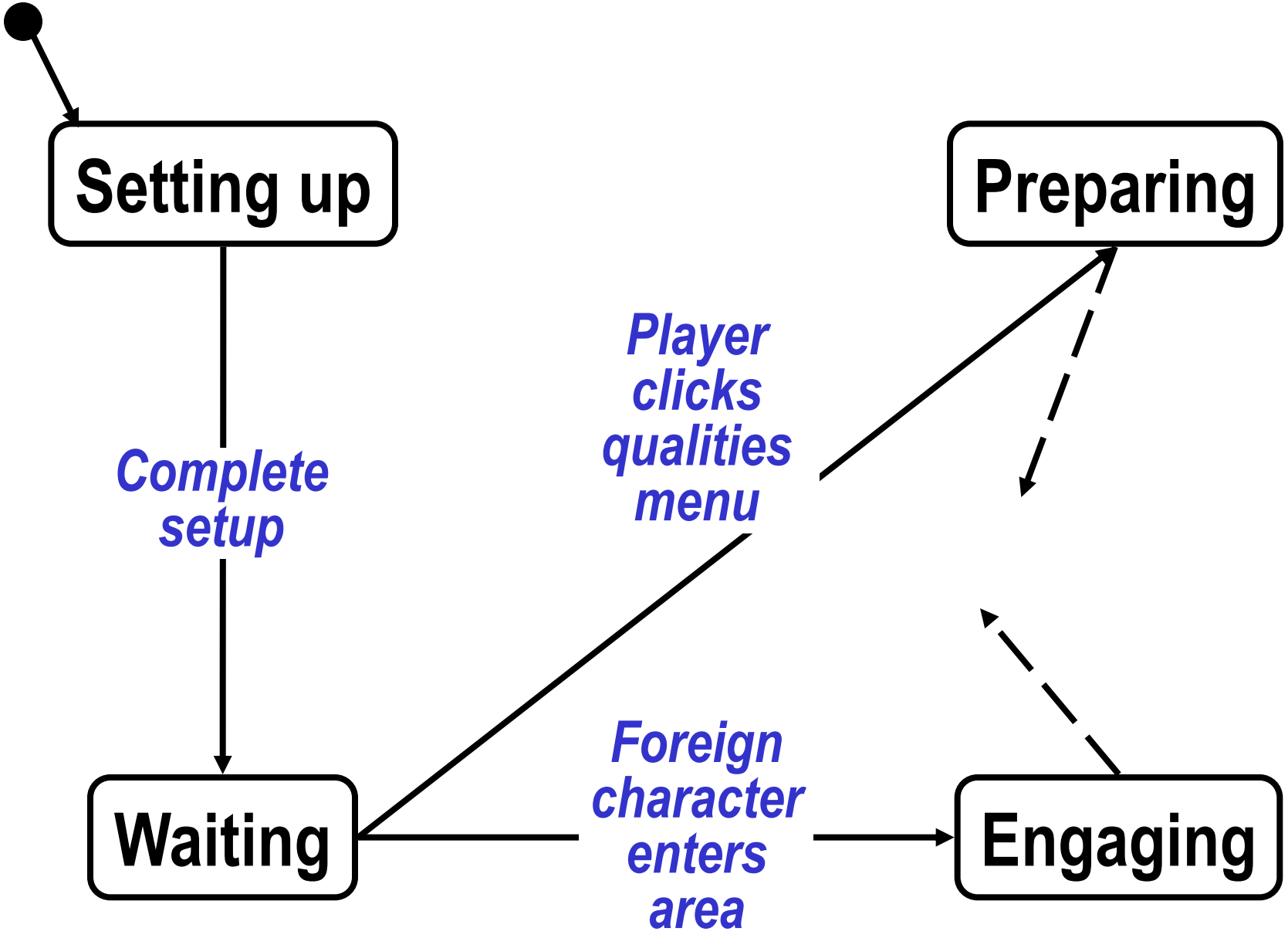


Role of State Models

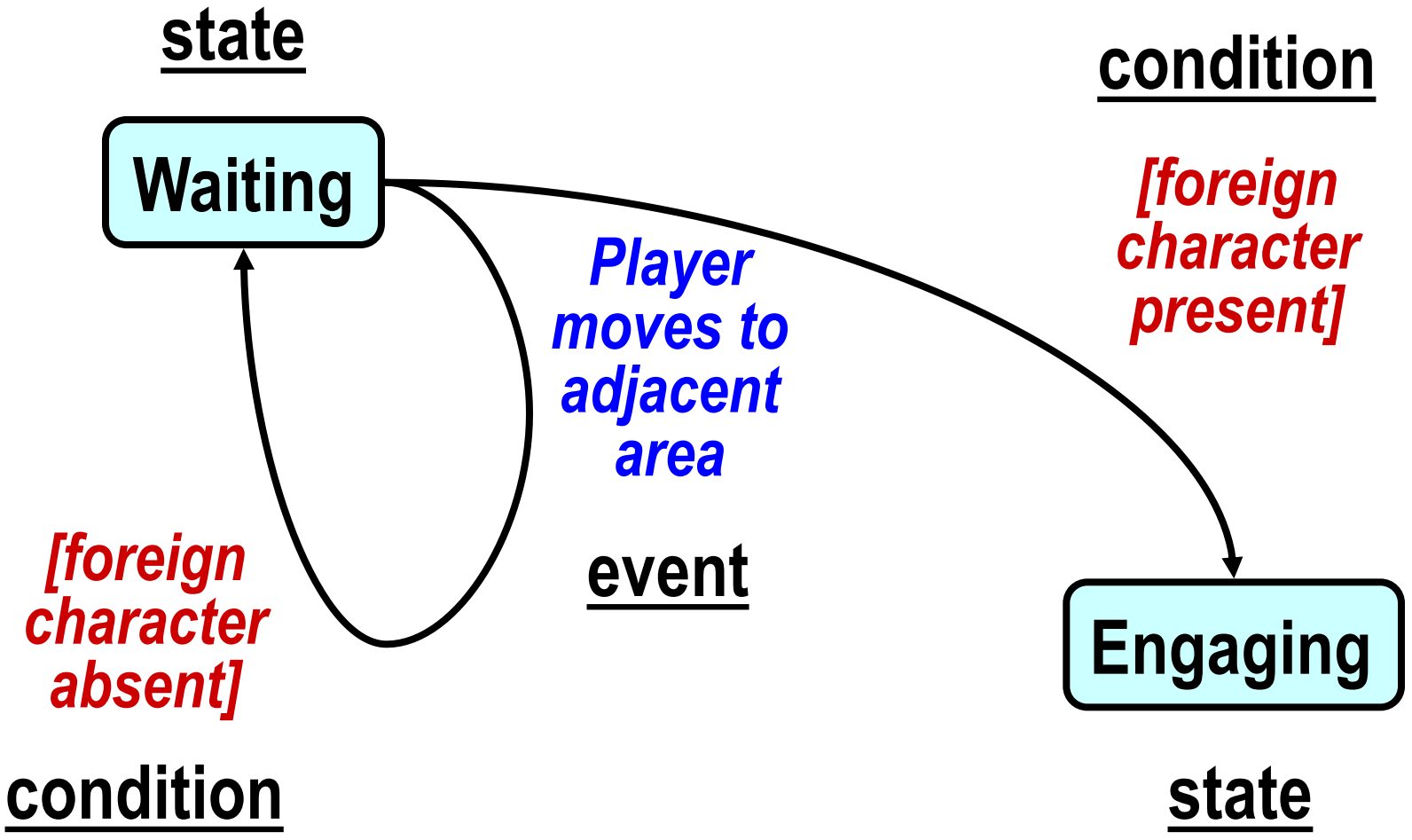


Jacobson et al

Partial *Encounter* Video Game State Model



Using Conditions in State-Transition Diagrams

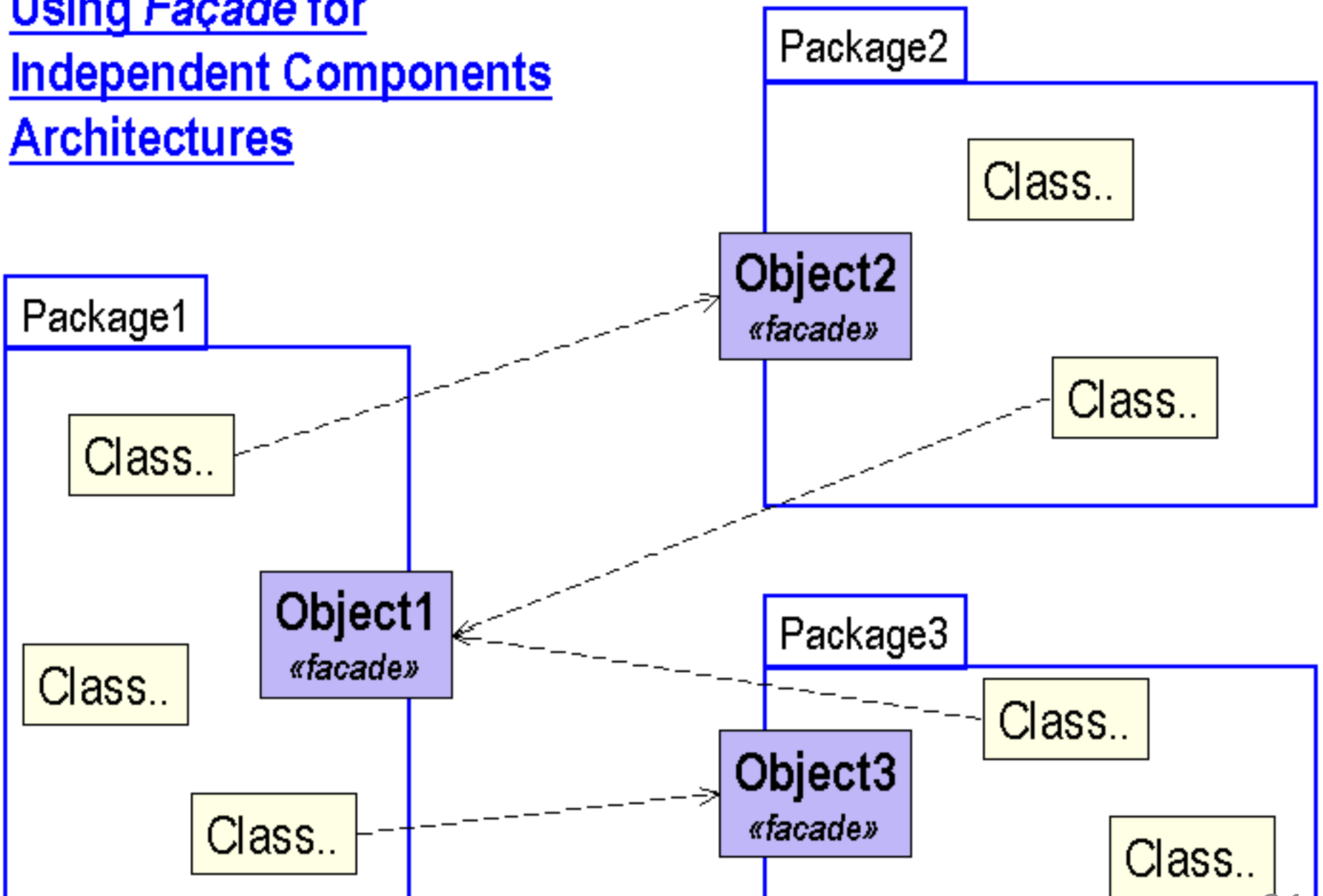


Conditions can be placed on events, that influence what resulting state is transitioned into.

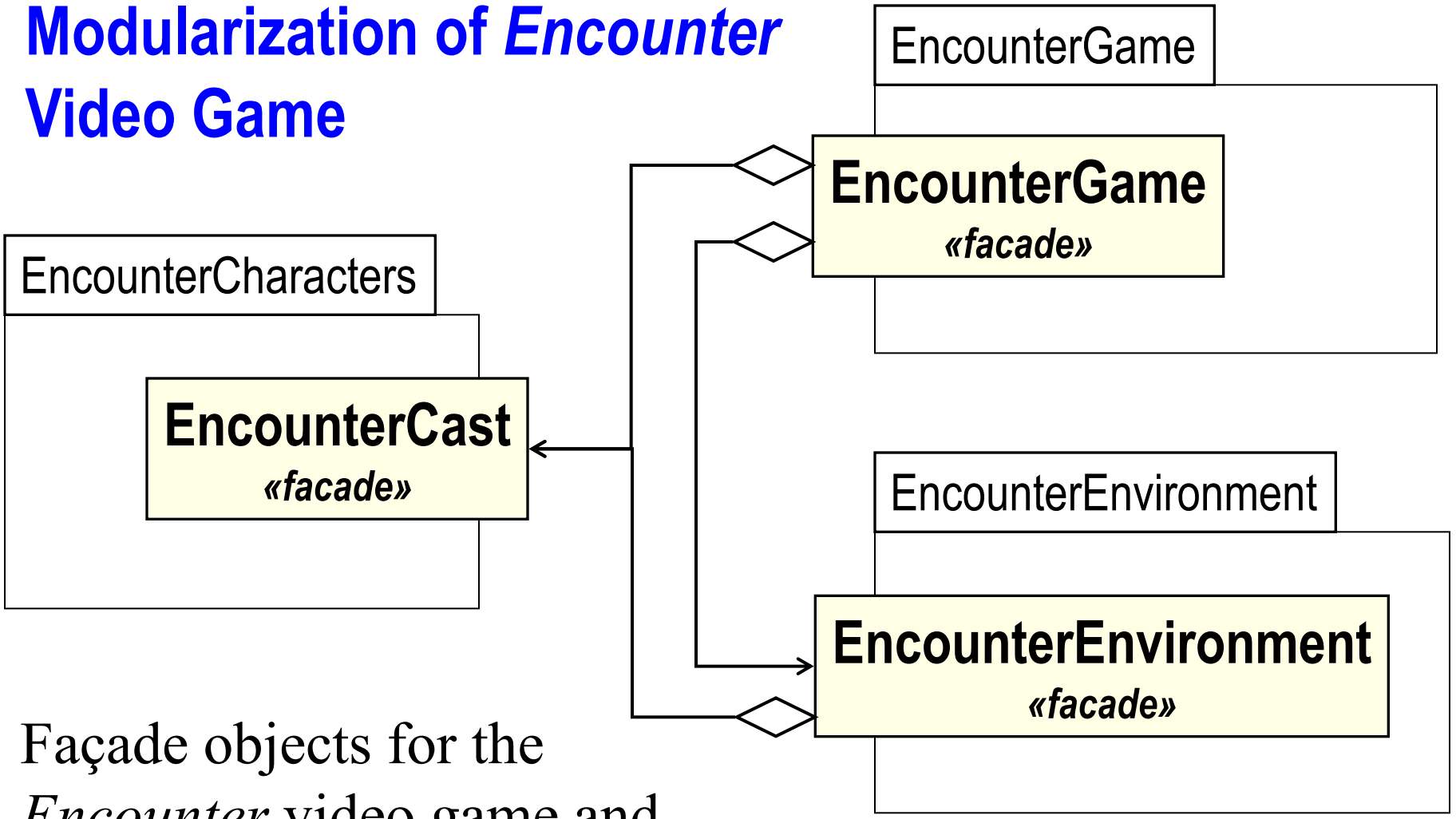
Design Goal At Work: → *Correctness/Modularity* ←

We want to decompose designs into modules, each well-knit, and depending on few others.

Using *Façade* for Independent Components Architectures



Architecture and Modularization of *Encounter* Video Game



Façade objects for the *Encounter* video game and their relationships.

To Begin Selecting a Basic Architecture

1. Develop a mental **model** of the application.
   - as if it were a small application

   e.g., personal finance application ...

   … "works by receiving money or paying out money, in any order, controlled through a user interface".
2. Decompose into the required components.
   - look for high cohesion & low coupling

   e.g., personal finance application ...

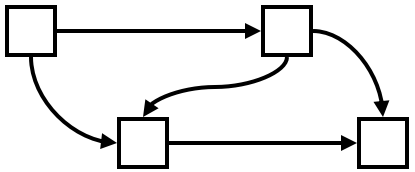
   … decomposes into Assets, *Sources, Suppliers, & Interface*.
3. Repeat this process for the components.
4. Consider using ***Façade*** for each package.

1. Context of Architecture
2. Models
3. **Classifying Architectures**
4. Frameworks

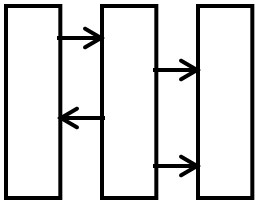
A ***framework*** is a collection of **models**, typically a class model, that forms a template for a category of **architectures**.

A Classification of Software Architectures (Garlan & Shaw)

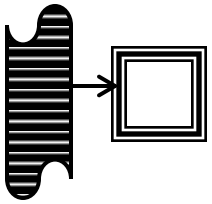
- ***Data Flow***
  - Data flowing between functional elements



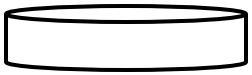
- ***Independent Components***
  - -- executing in parallel, occasionally communicating



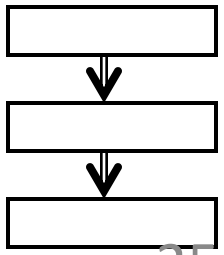
- ***Virtual Machines***
  - Interpreter + program in special-purpose language



- ***Repositories***
  - Primarily built around large data collection



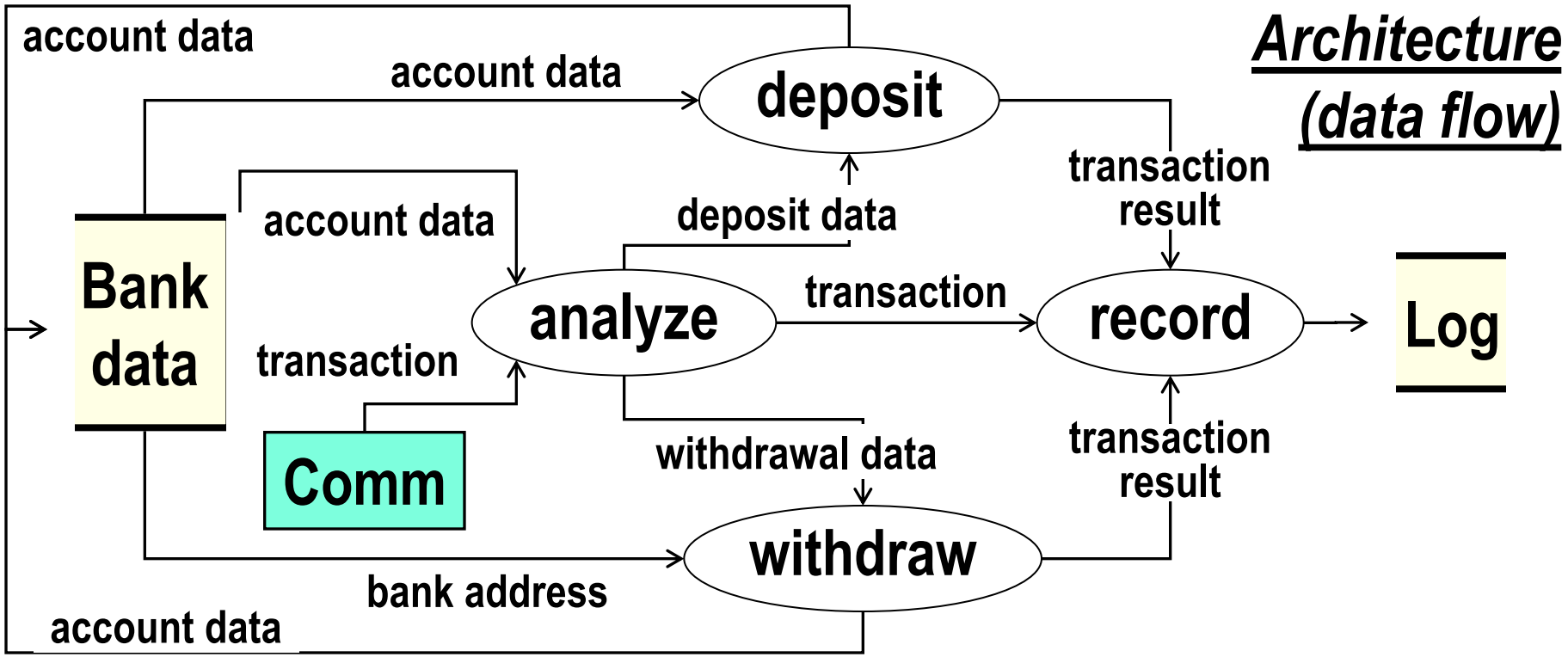
- ***Layered***
  - Subsystems, each depending one-way on another subsystem



Design Goal At Work: → *Reusability* ←

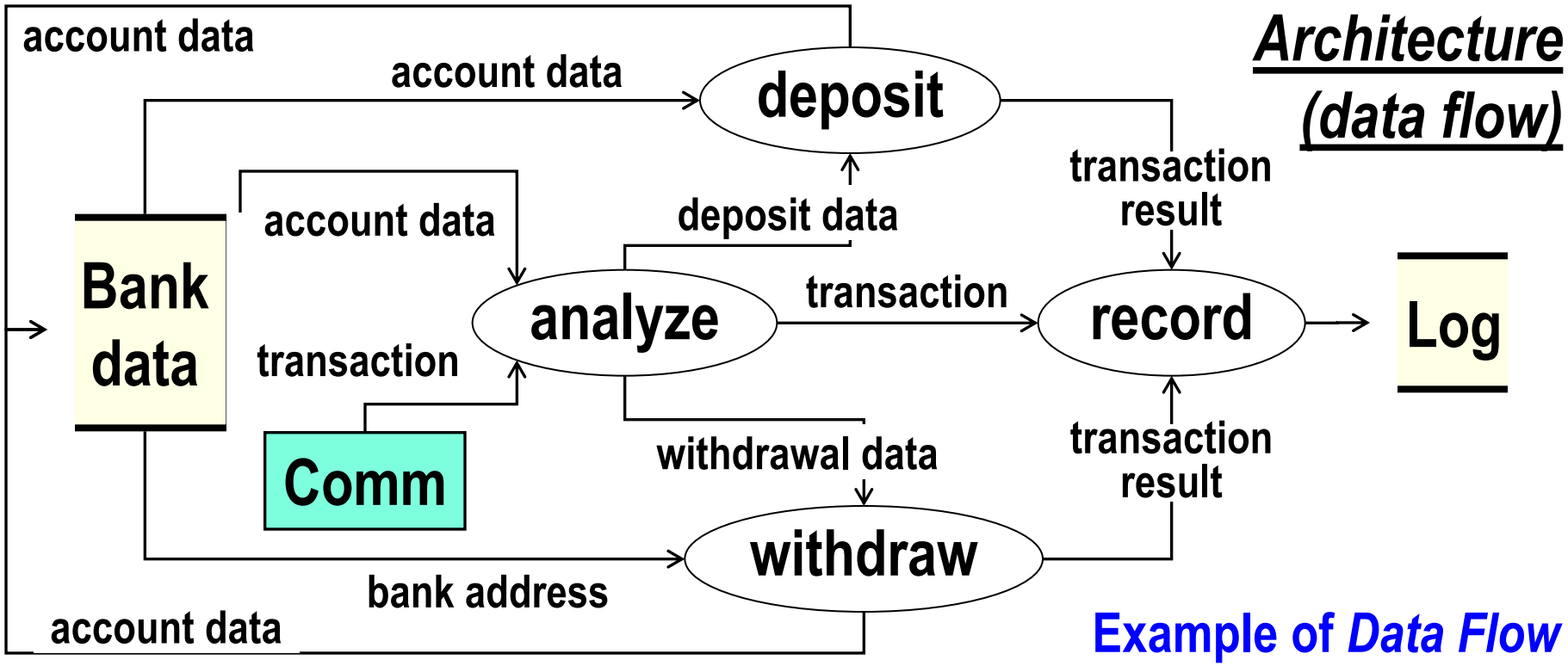
We classify architectures so as to use them for several applications.

Requirement: **Maintain wired financial transactions.**



Example of *Data Flow* Architecture

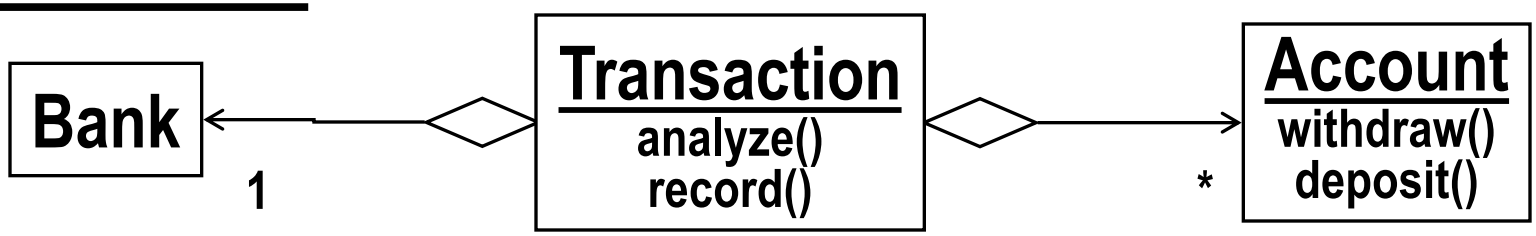
Requirement: **Maintain wired financial transactions.**



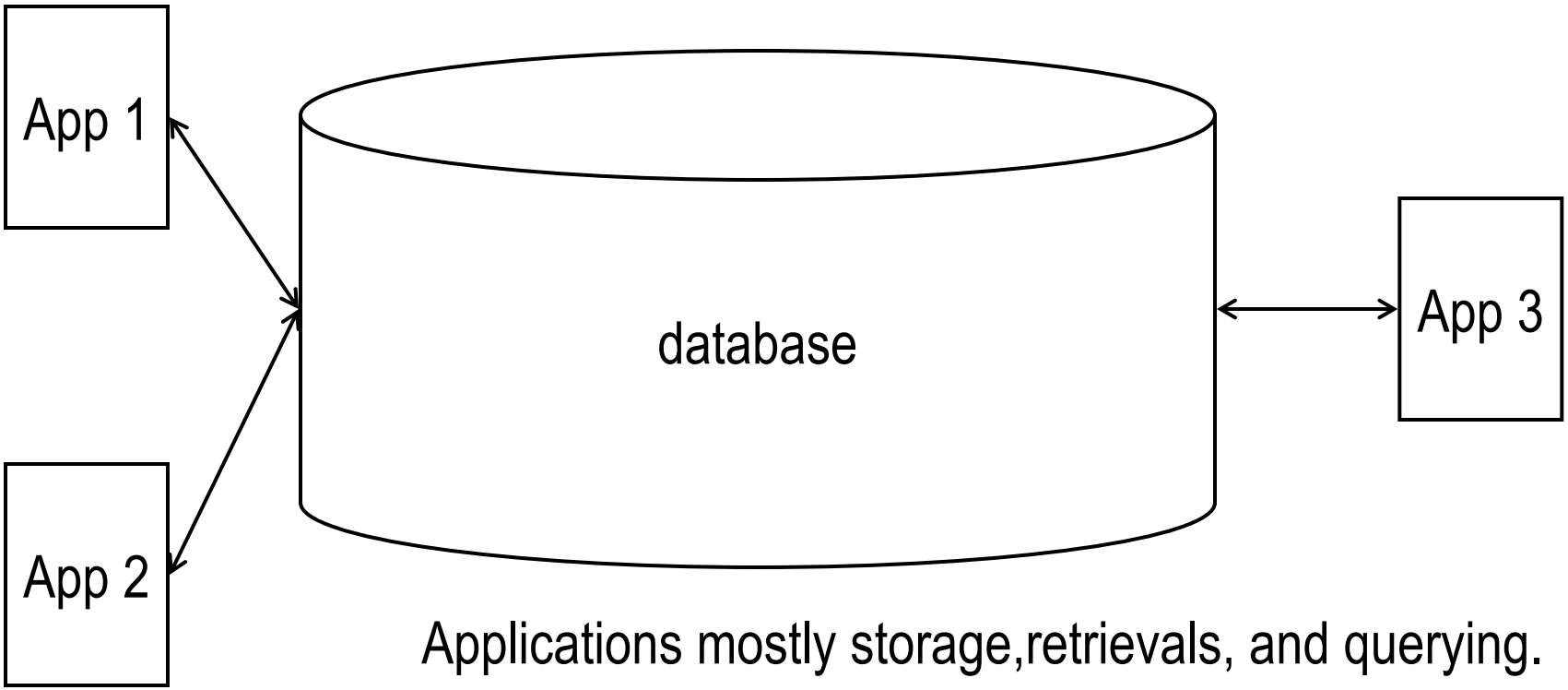
Architecture (data flow)

Example of *Data Flow* Architecture and Corresponding Class Model

A Class model:

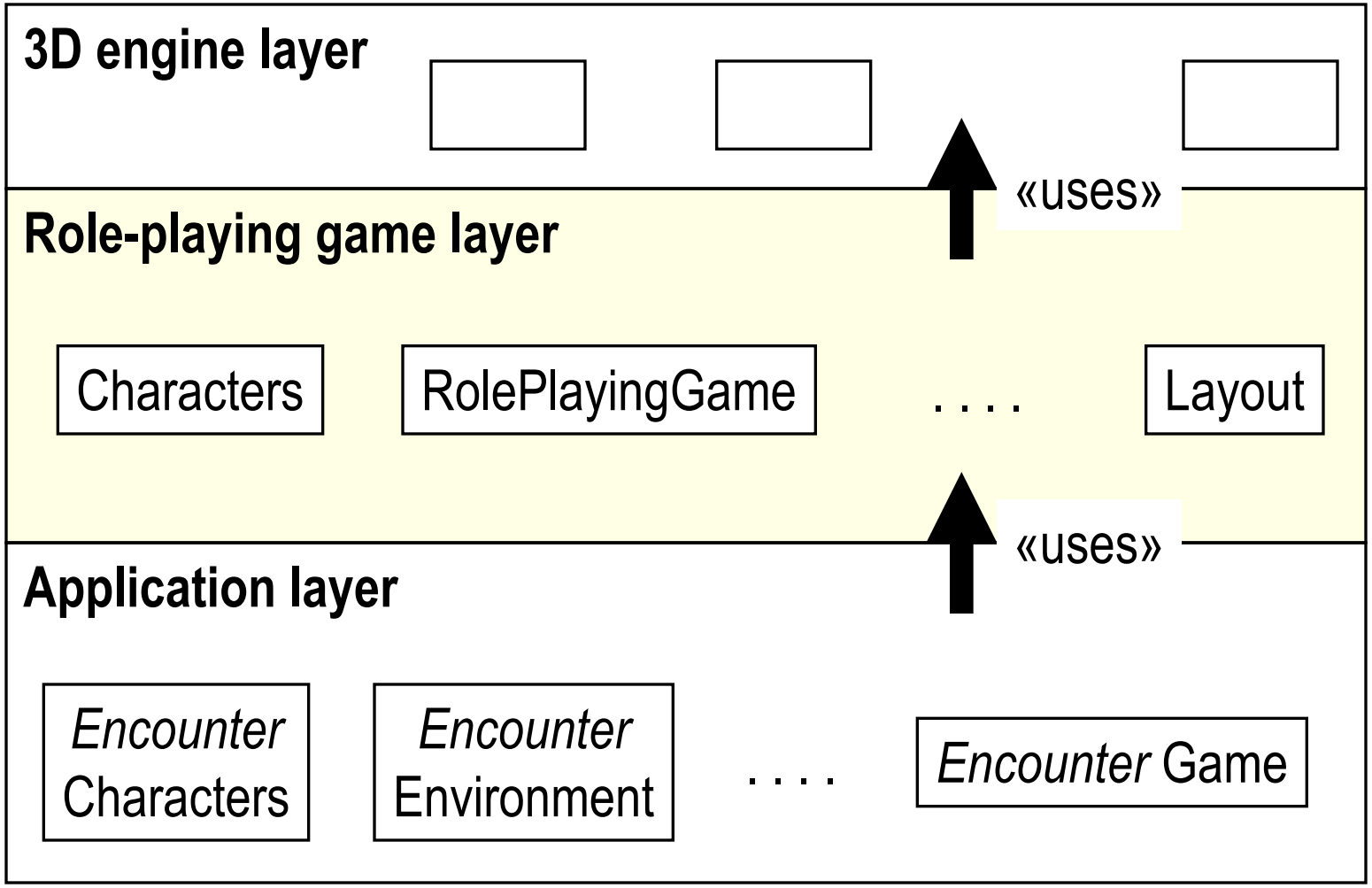


Repository Architecture



Applications mostly storage,retrievals, and querying.

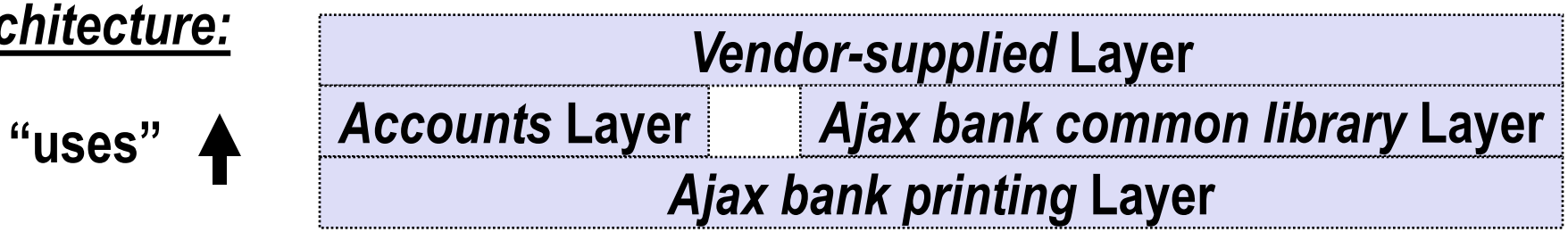
Layered Architecture



Layered Architecture Example Using Aggregation

Requirement: **Print monthly statements**

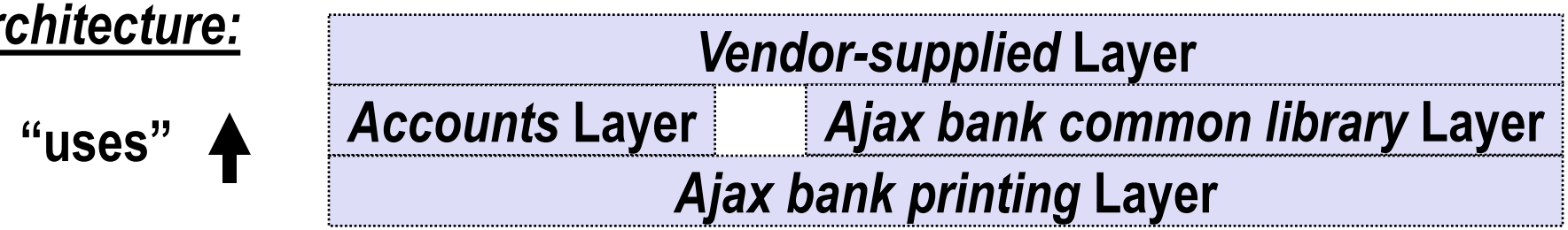
Architecture:



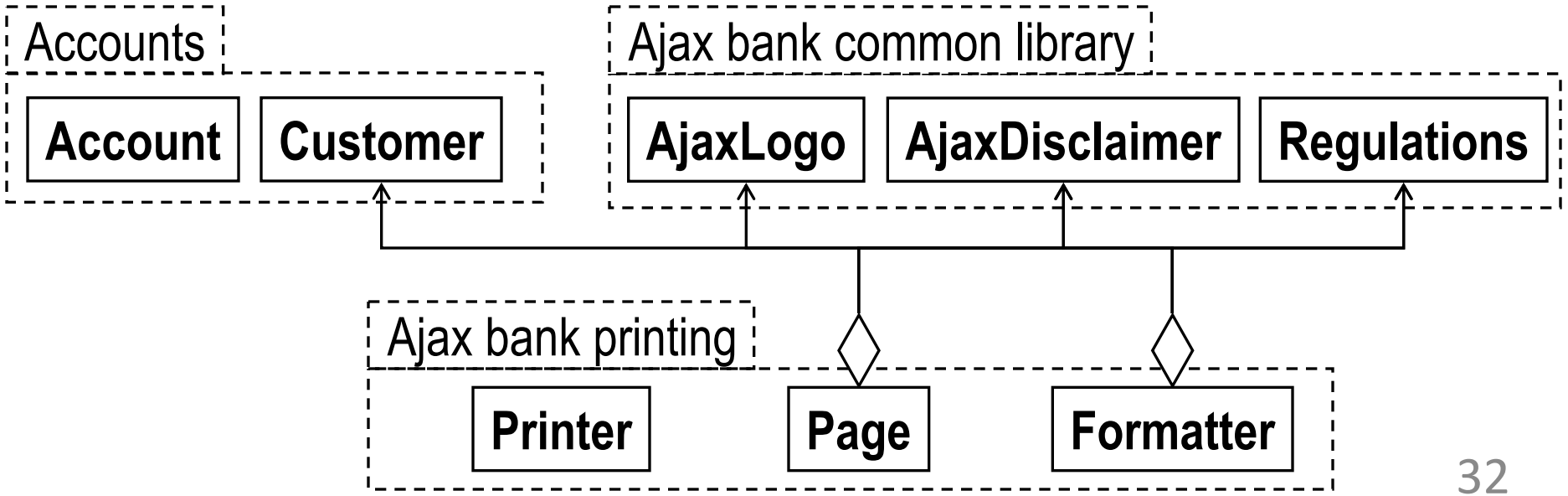
Layered Architecture Example Using Aggregation

Requirement: **Print monthly statements**

Architecture:



Class model: (relationships within packages and *Vendor-supplied* layer not shown)



Key Concept:
A Software Architectures is Essentially ...

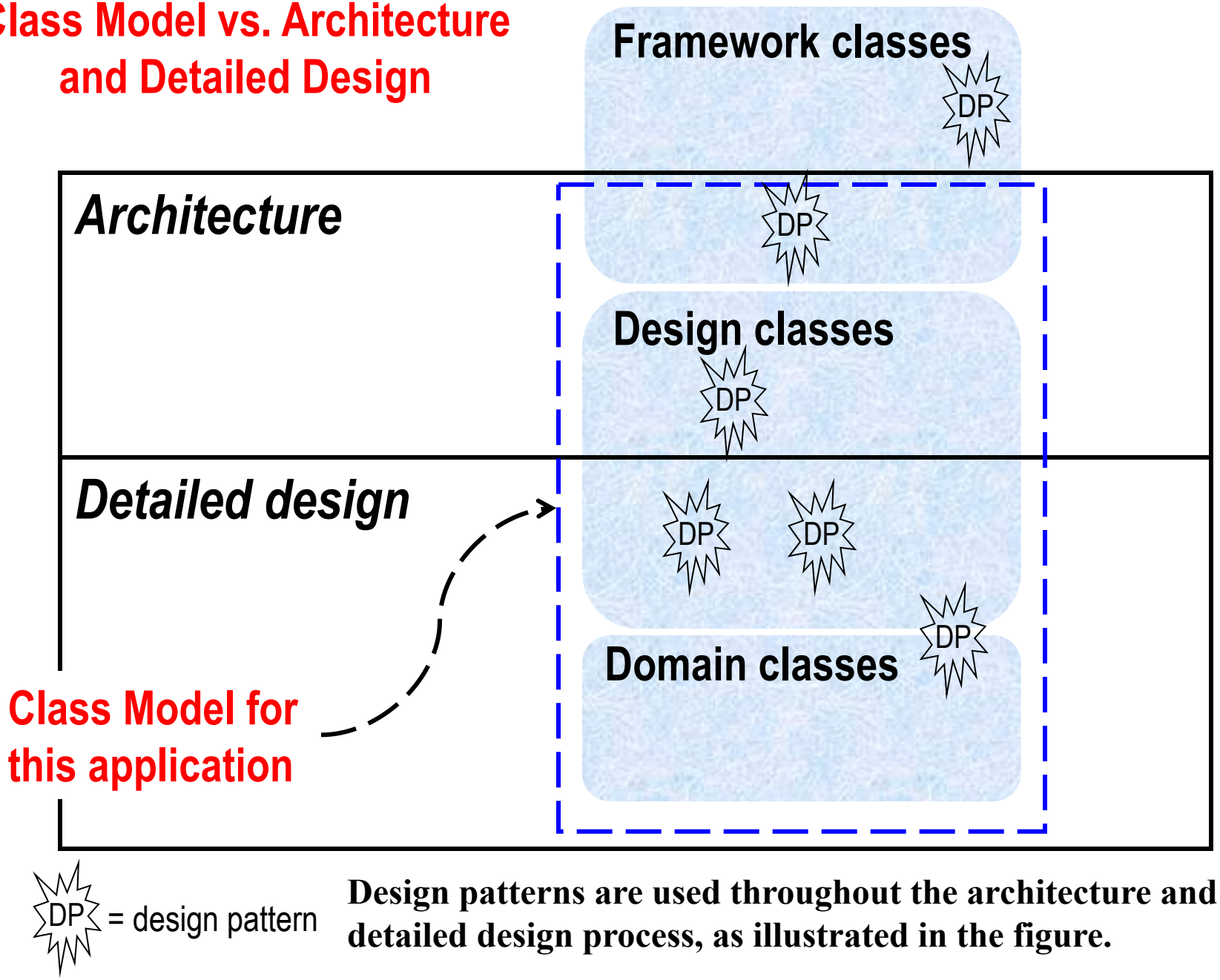
- a Data Flow,
- a set of Independent Components,
- a Virtual Machine,
- a Repository,
- or Layers.

1. Context of Architecture
2. Models
3. Classifying Architectures
4. **Frameworks**

A ***framework*** is a collection of **models**, typically a class model, that forms a template for a category of **architectures**.

A **framework** is a collection of models, typically a class model, that forms a template for a category of architectures.

Class Model vs. Architecture and Detailed Design



Design patterns are used throughout the architecture and detailed design process, as illustrated in the figure.

Key Concept: → We Use Frameworks … ←

… to reuse

- ❑ classes,
- ❑ relationships among classes,
- ❑ or pre-programmed control.

Selected Framework Goals

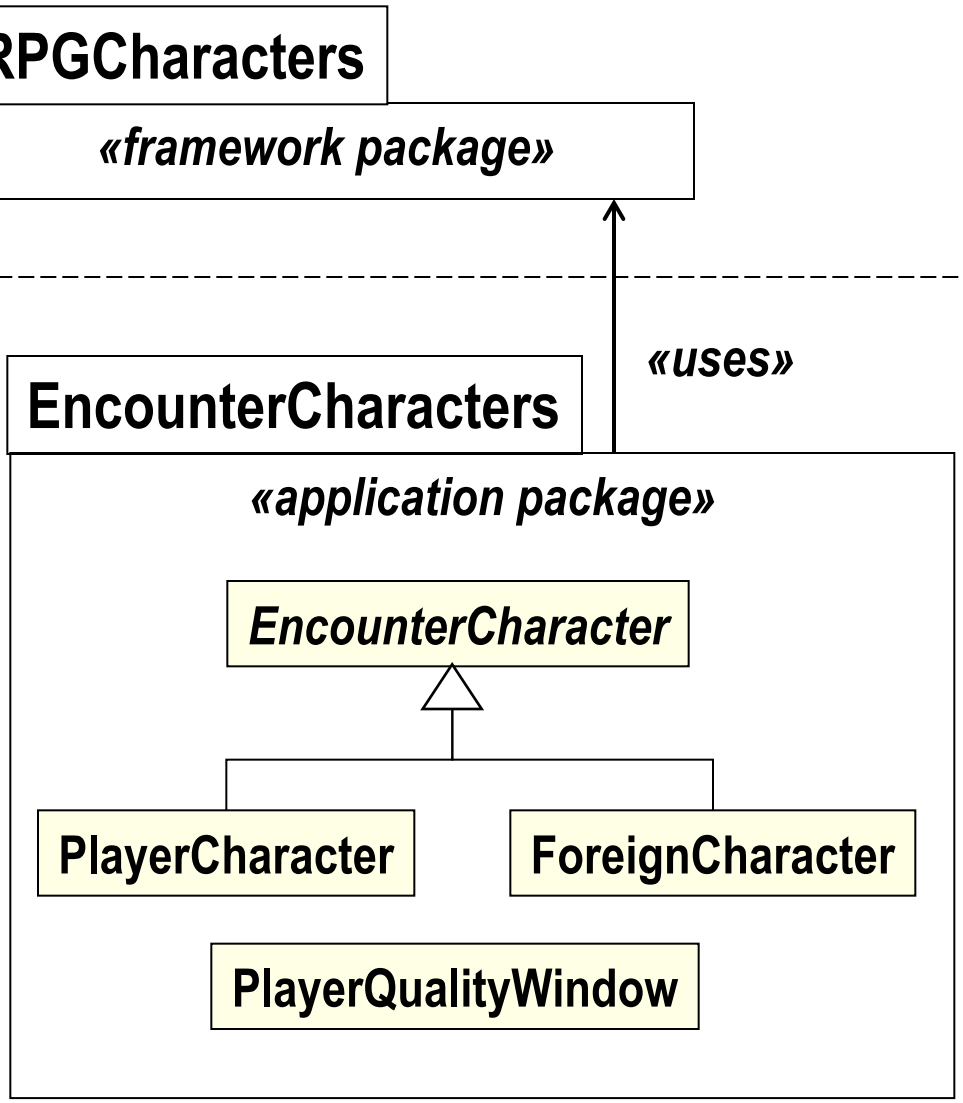
- *Persistence Service*
  - Store instances between executions
- *Identity Service*
  - Identify objects sufficiently to retrieve them across executions
- *Pooling*
  - - of objects: Reusing objects at runtime to save time and space
  - - of threads
  - - of database connections
- *Security*

RPG Video Game Layering

Framework layer

Application layer

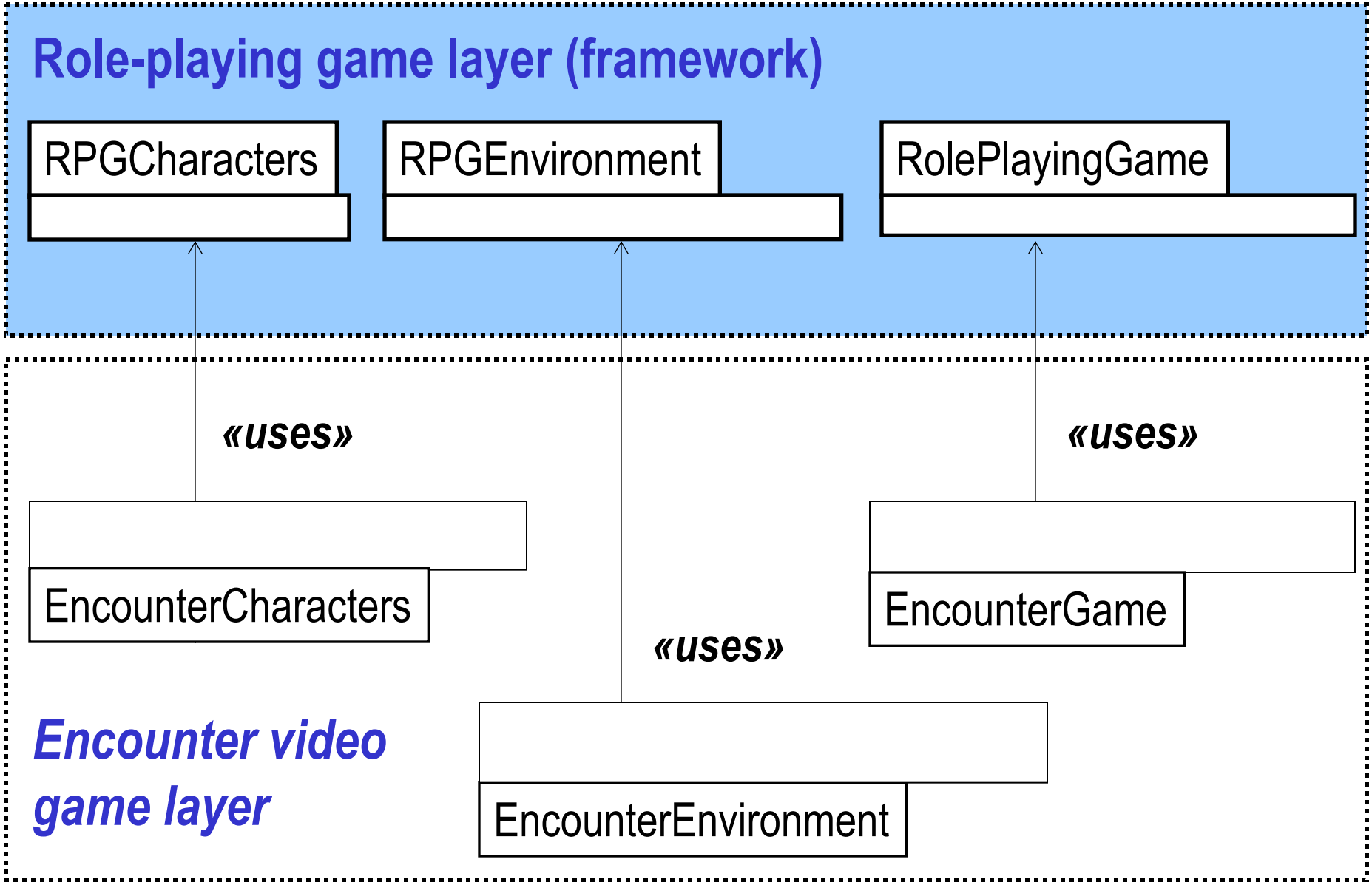
- development organization creates role-playing games over time.
- Include a package that contains as much information and structure about characters in all role-playing games as possible.



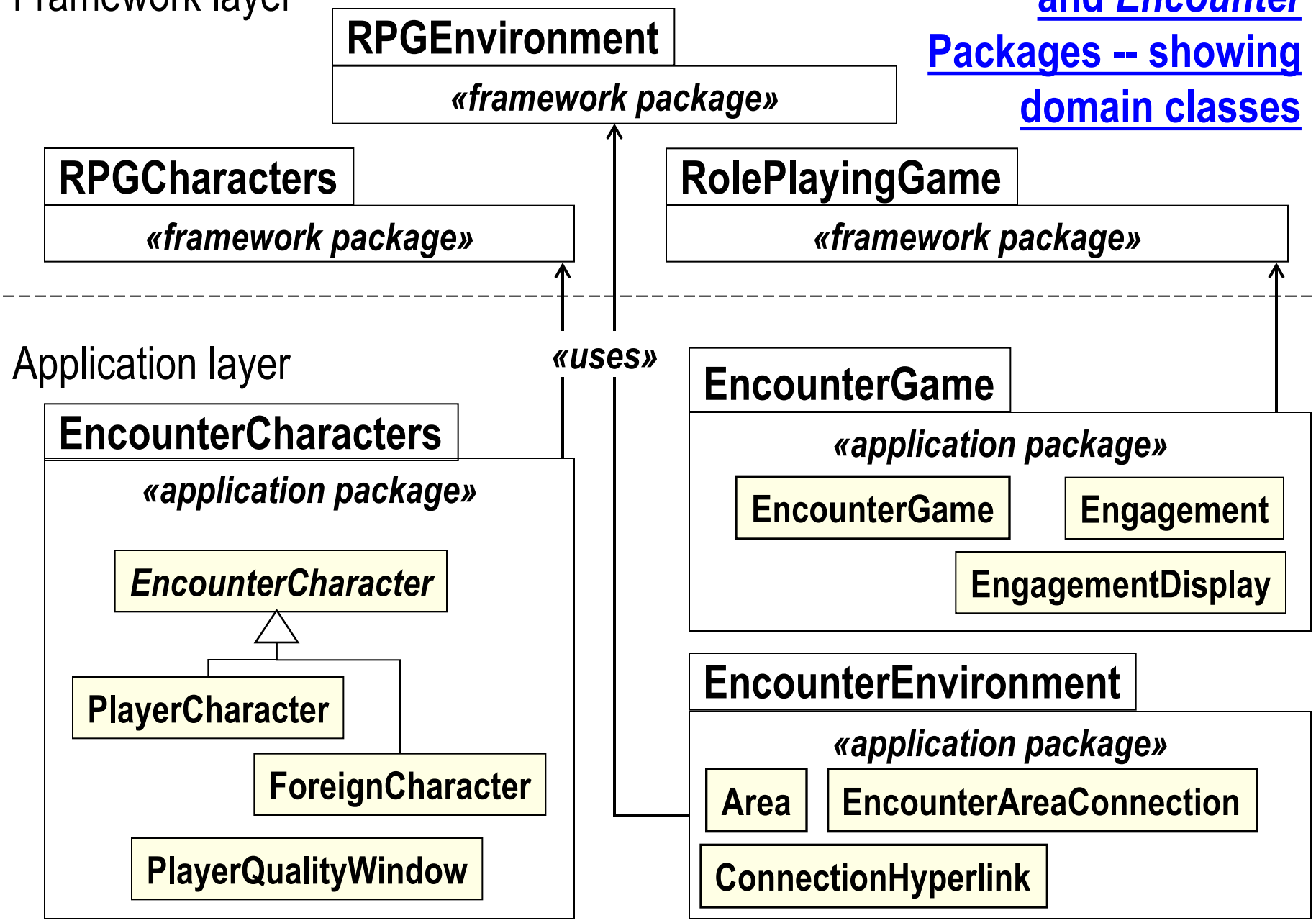
Design Goal At Work: → Reusability ←

We create frameworks because we **want to reuse** groups of classes and algorithms among them.

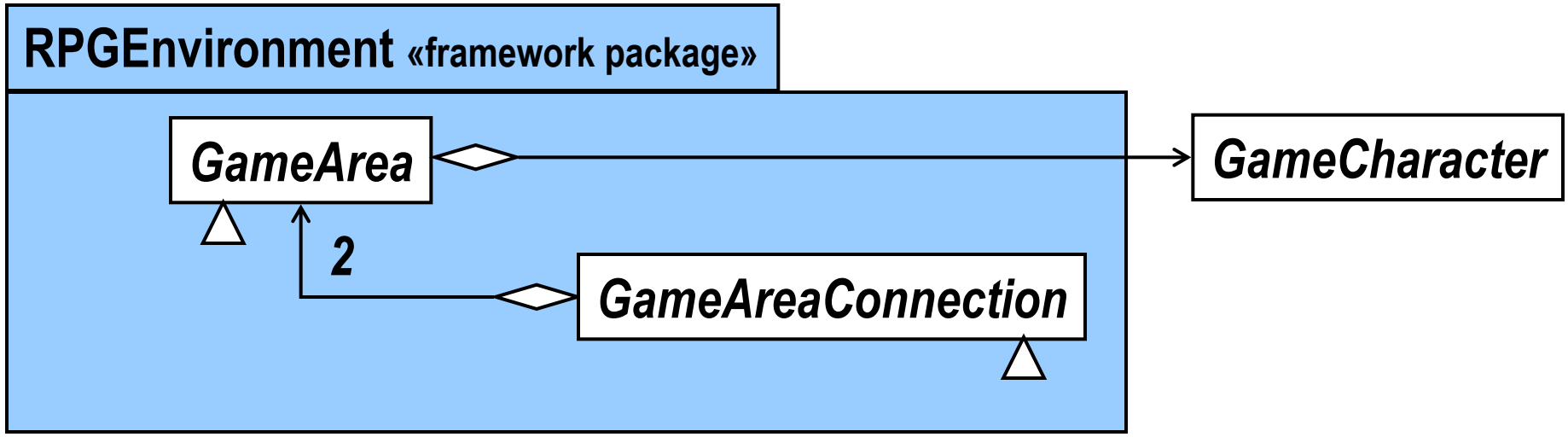
Framework For *Encounter* Video Game



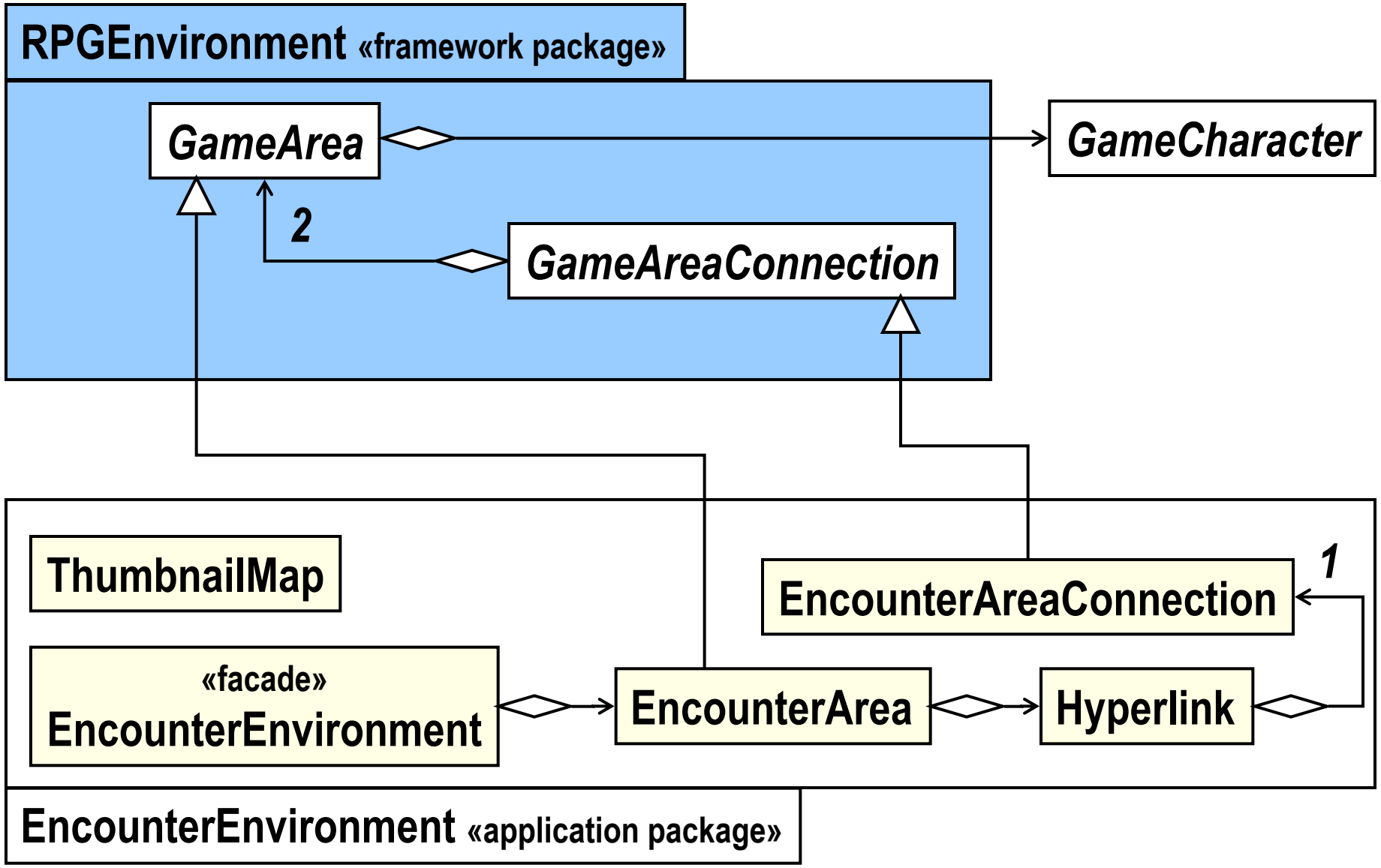
Role-Playing Game, and *Encounter* Packages -- showing domain classes



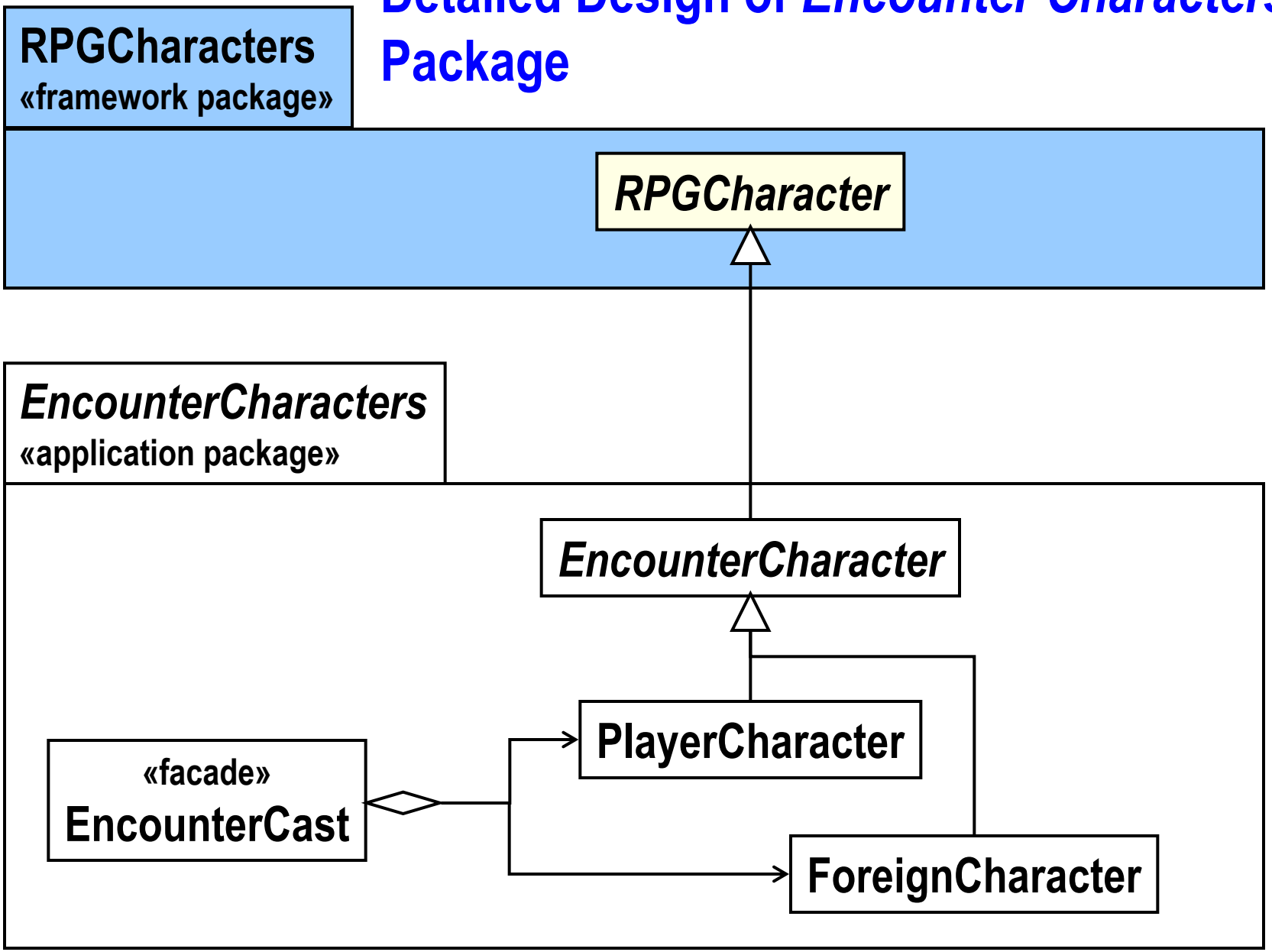
Encounter Environment Package



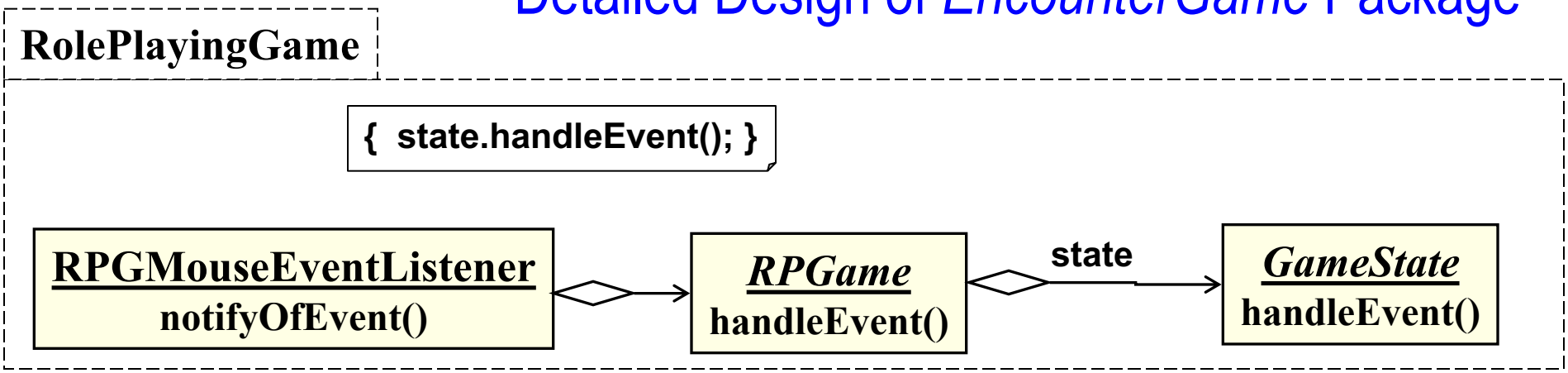
Encounter Environment Package



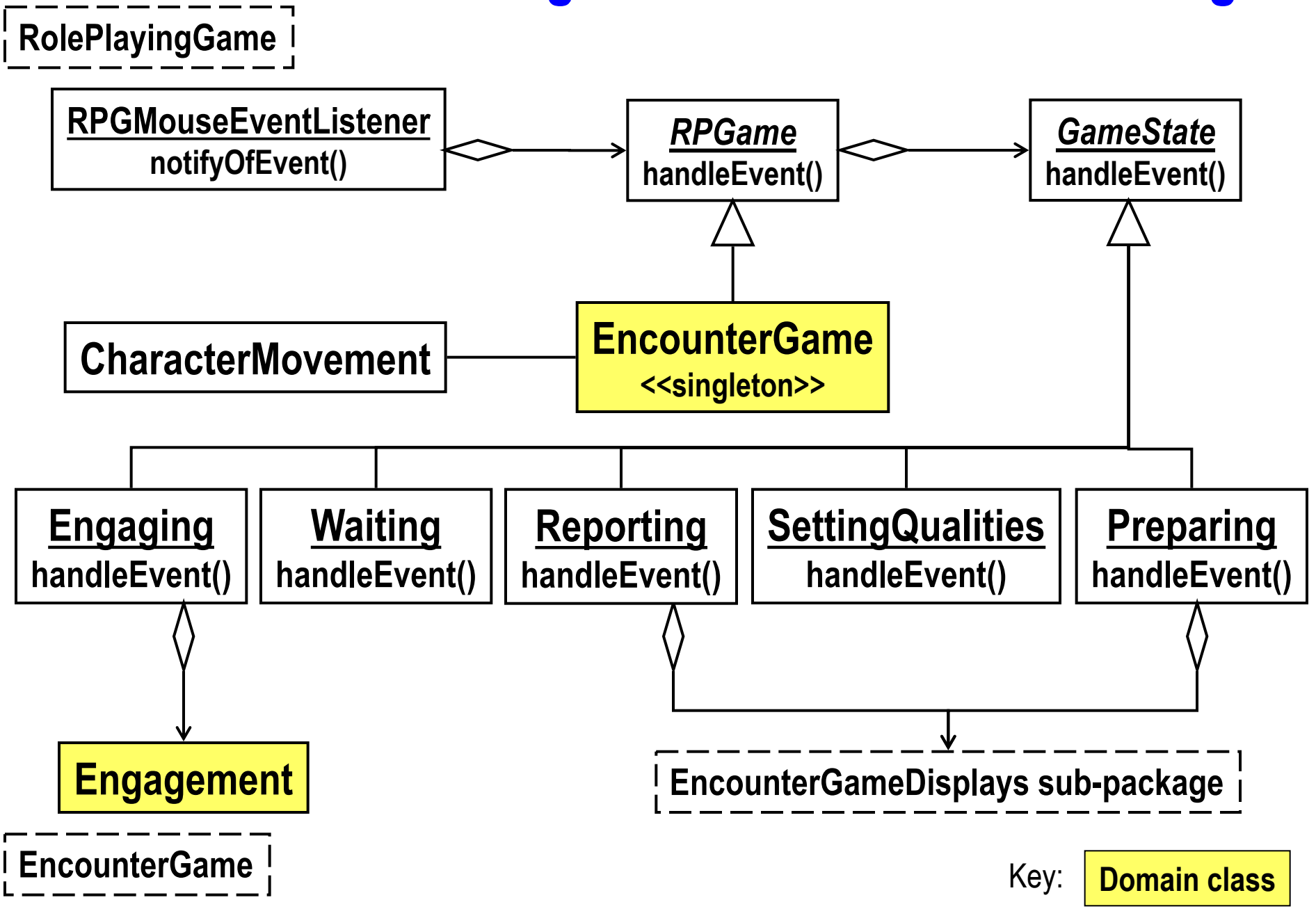
Detailed Design of *Encounter Characters* Package



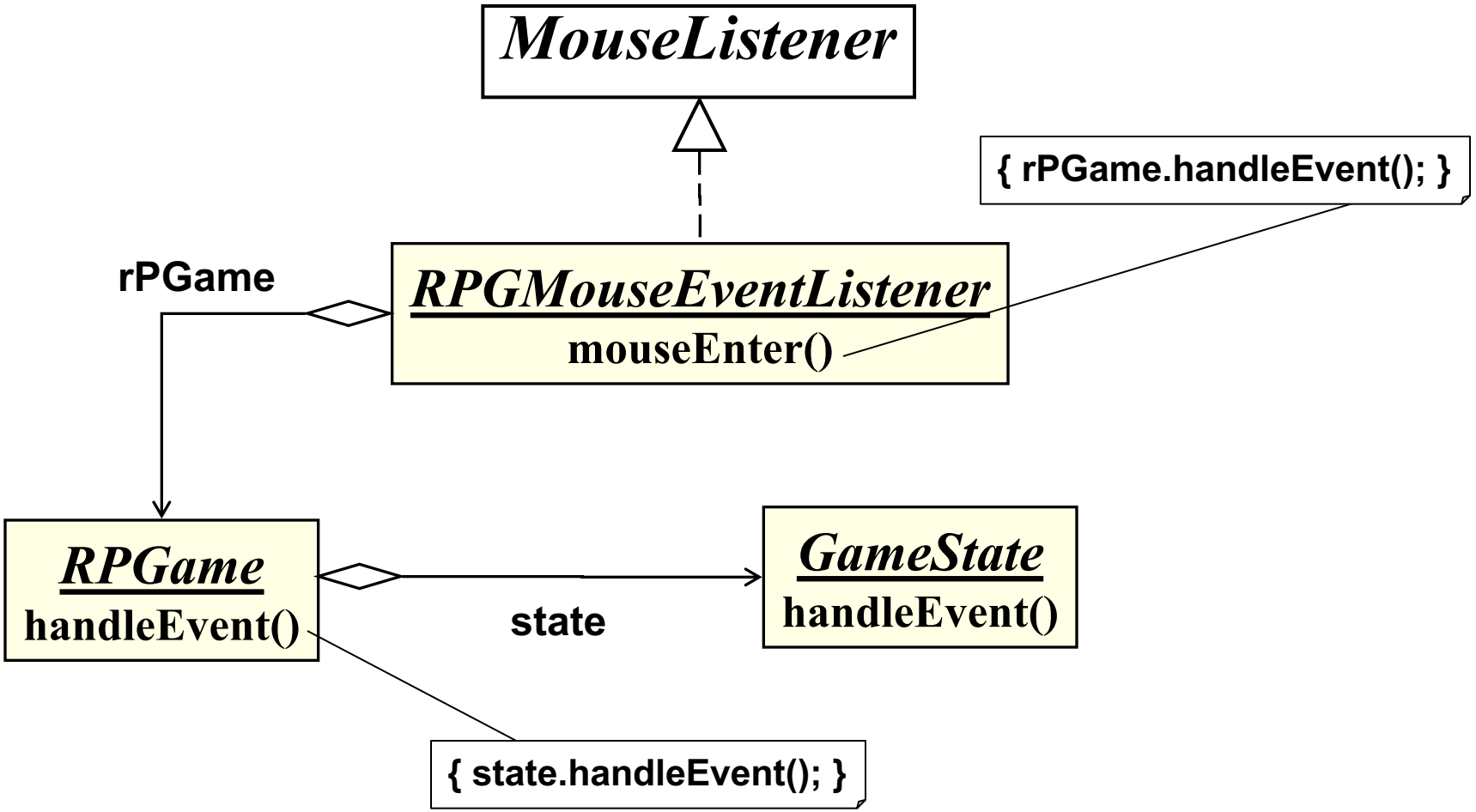
Detailed Design of *EncounterGame* Package



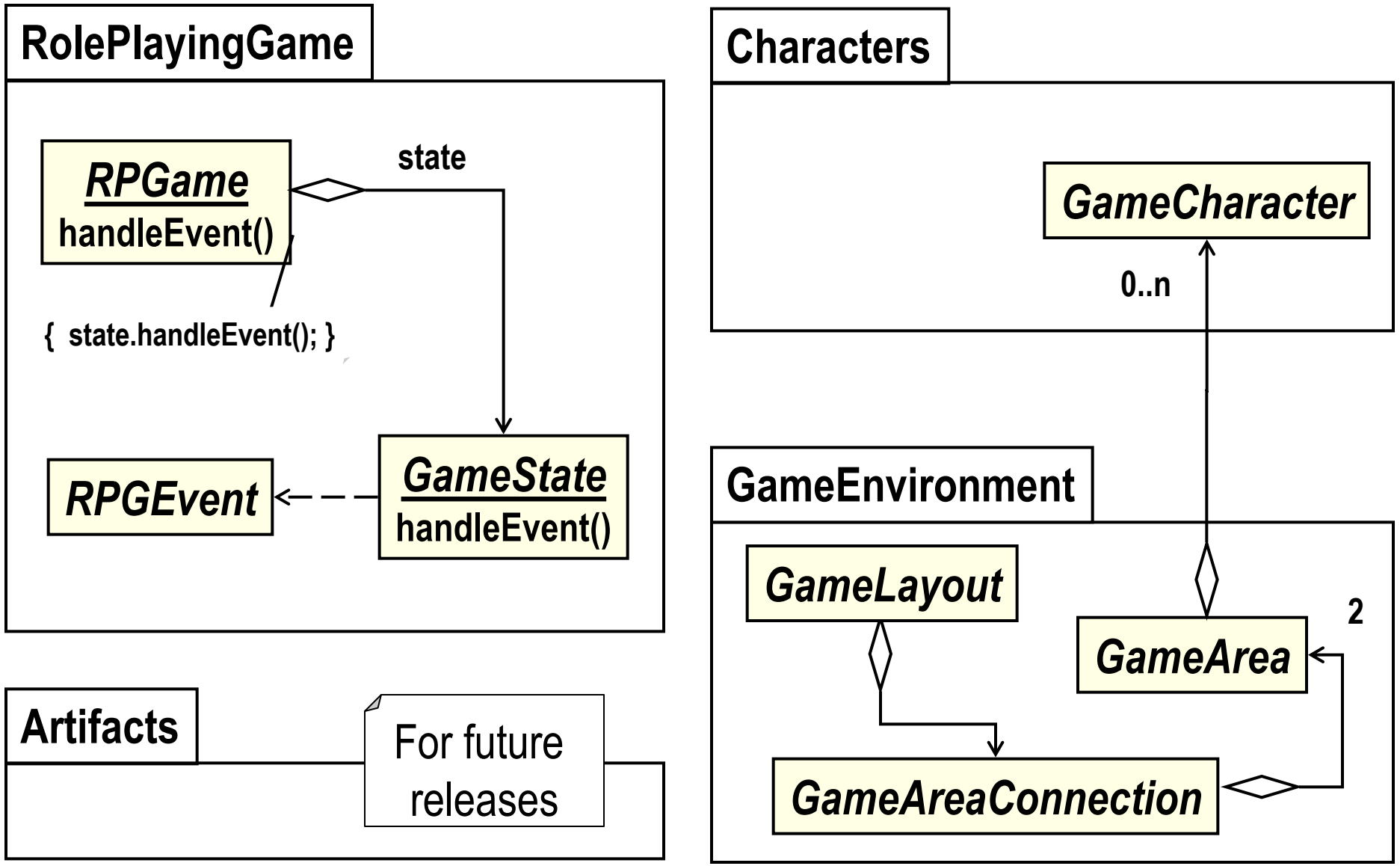
Detailed Design of EncounterGame Package



Detailed Design of *RolePlayingGame* Package



RPG Framework for Role-Playing Video Games



Completing Detailed Design

- Design the control of the application,
- Design for database access
- Design interaction with outside world.
- Audit relationships among classes
  - eliminate unnecessary connections,
  - adding requried detail classes
  - clarify all classes
- Ensure all required methods implemented

Figures in this presentation
are adapted from *Software Design: From Programming to Architecture* by Eric J. Braude (Wiley 2003), with permission.

Other References

- Jacobson, I., Object-Oriented Software Engineering, Addison Wesley, 1992
- Coad, P. and E. Yourdon, Object-Oriented Analysis, 2nd ed., Prentice-Hall, 1991