AARHUS UNIVERSITET

Software Architecture in Practice

Architectural description

Overview

Introduction to the POS case

Motivation

Architectural description

- Formal description languages
- Using UML

A Simple Case

NextGen Point-Of-Sales (POS) System

- Record sales and handle payments
  - Typically used in retail stores
- Hardware
  - Terminal
  - Barcode scanner
- Interfaces with external systems
  - Inventory
  - Accounting
  - …

From [Larman, 2001]

- See also note on architecture description using UML, [Christensen et al., 2007]

Why Focus on Description?

- Architecture as a means for communication among stakeholders
  - Need suitable (stakeholder-dependent) representations
  - Architect -> developer
    - Needs precise understanding of design choices
    - Architecture as “blueprint” for development
  - Architect -> customer
    - Precision needs to be balanced with abilityt to understand
      - Box-and-line vs. formal
- Architecture as basis for design and evaluation
  - Precise semantics of description beneficial to analyze non-trivial properties
  - Support analytical evaluation
    - questioning techniques
    - automated tools

Architecture Description Languages (ADLs)

Languages for describing software architectures

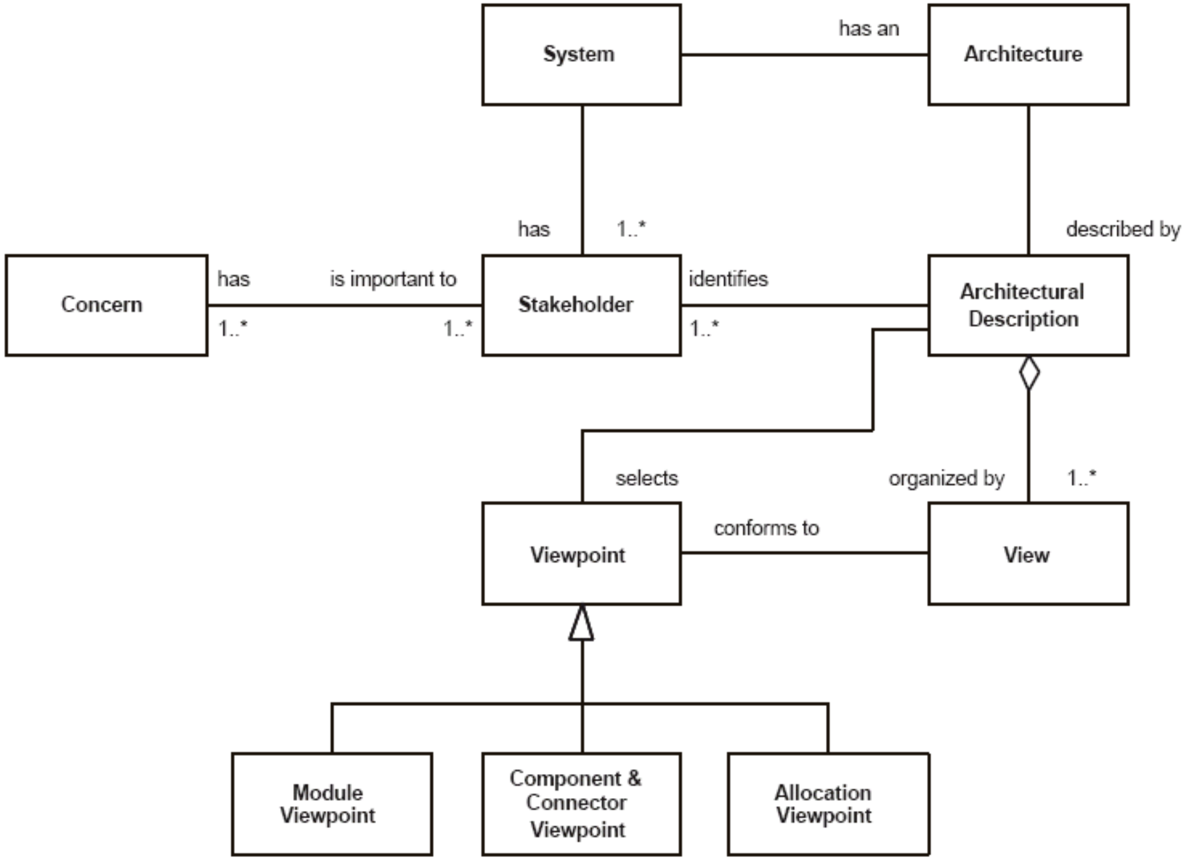
- Broad categories
  - Box-and-line drawings
    - Not really an ADL
  - Formal descriptions
  - Multiple view-based descriptions

This course

- Unified Modeling Language (subset) as example of multiple-view descriptions
  - Core, [Christensen et al., 2007]
- Formal description languages
  - Just a taste…

An Ontology of Architectural Descriptions

Here: ontology = model of concepts



Developed from [IEEE 1471, 2000]

Elements of an Architectual Description

Architectural views (N)

- What is the software architecture?
  - Multiple viewpoints, here
    - Module viewpoint
    - Component & Connector viewpoint
    - Allocation viewpoint

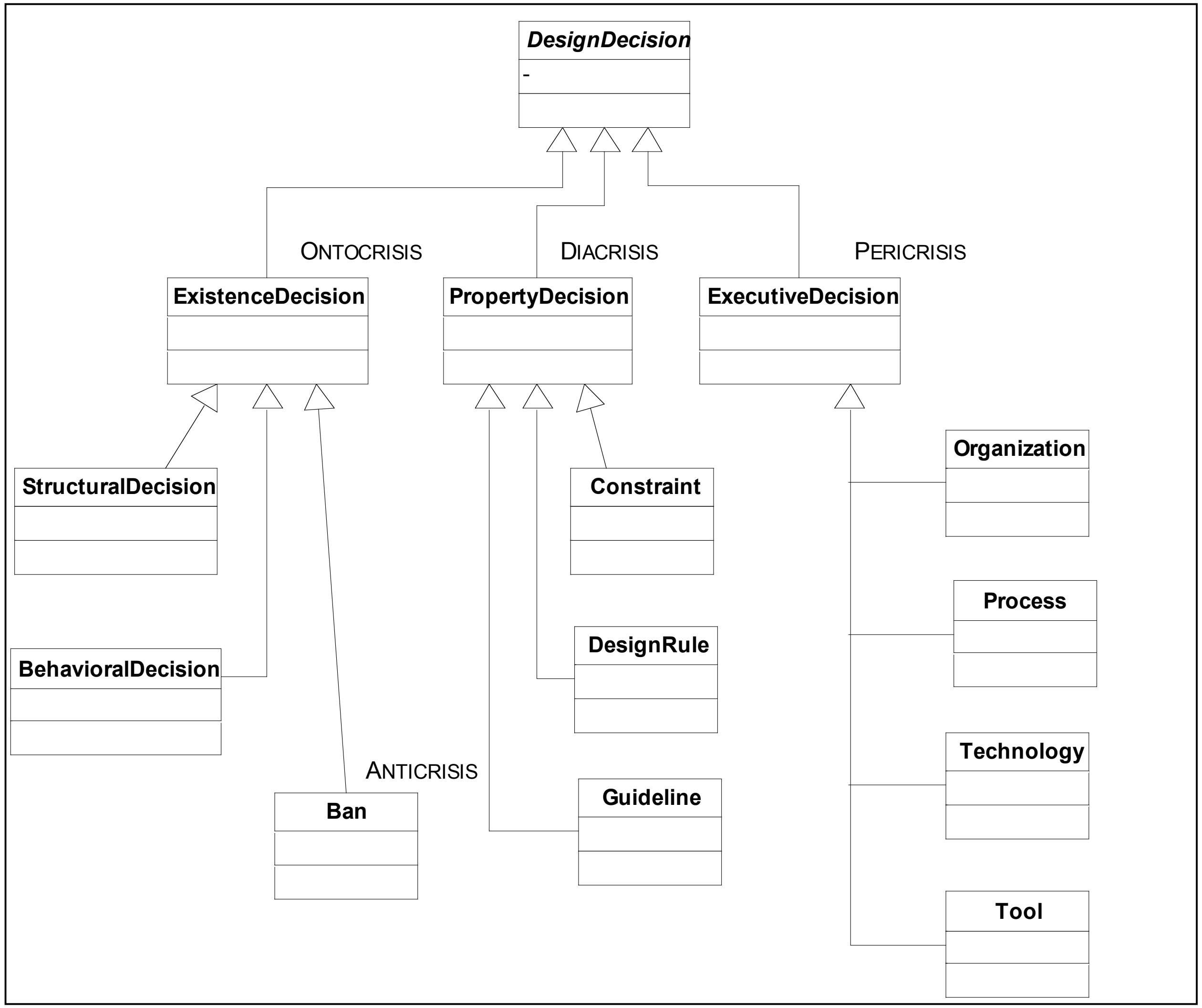
Architectural requirements (+1)

- Why is the software architecture the way it is?
  - Scenario-based requirements
    - E.g., paths through significant use cases
  - Quality-attribute-based requirements
    - Primary concerns (e.g., critical quality requirements)
    - Quality attribute specifications
  - Design decisions

Architectural Requirements: POS Scenarios

Process Sale: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the item

Types of Design Decisions [Kruchten, 2005]



Architectural Requirements: POS Qualities

Architectural drivers

- Availability
  - The system shall be highly available since the effectiveness of sales depends on its availability
- Portability
  - The system shall be portable to a range of different platforms to support a product line of POS systems
- Usability
  - The system shall be usable by clerks with a minimum of training and with a high degree of efficiency

This is not operational!

- More later on quality attribute scenarios…

Architectural Views

How is the functionality of the system mapped to runtime components and their interaction?

- Component & connector viewpoint/structure

How is the functionality of the system to be mapped into implementation?

- Module viewpoint/structure

How are software elements mapped onto environmental structures?

- Allocation viewpoint/structure

Component & Connector Viewpoint

> The software architecture of a computing system is the structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass et al., 2003]

Elements

- Components
  - Functional behaviour
  - What part of the system is doing what?

Relations

- Connectors
  - Control and communication aspects
  - Define protocols for control and data exchange
    - Incoming and outgoing operations
    - Mandates ordering of operations
    - Define roles for attached components

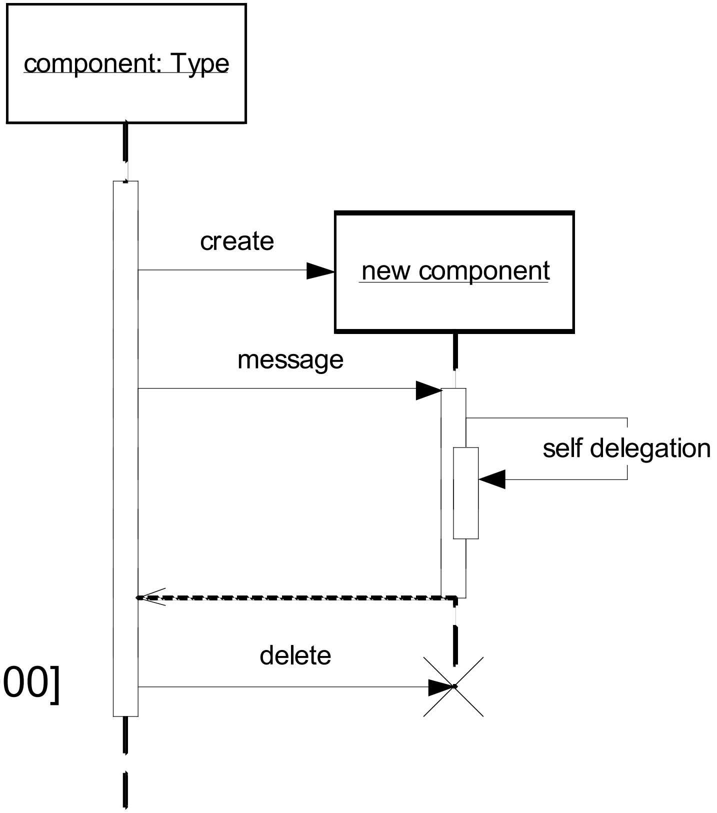
Mapping to UML

- Object diagrams, interaction diagrams
- Components = active objects
- Connectors = links + annotations, messages

+ textual description of responsibilities

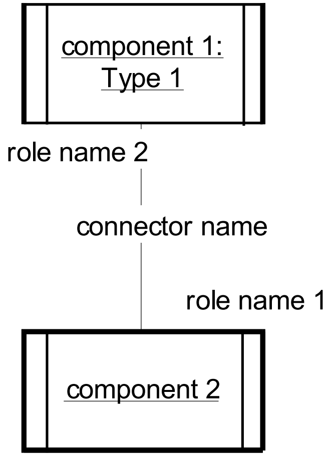
Basic C&C Elements

Sequence Diagrams

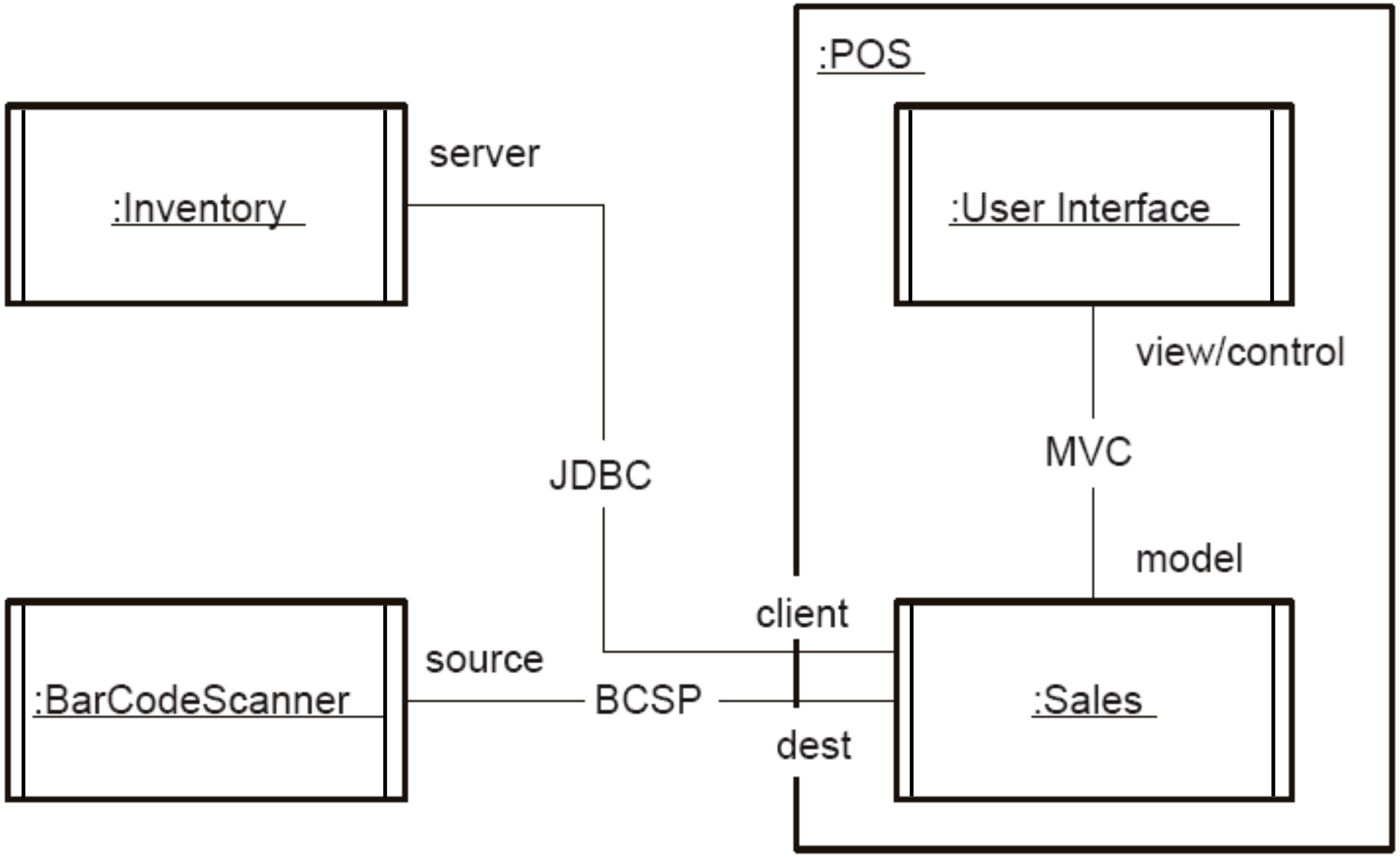


[Fowler, 2000]

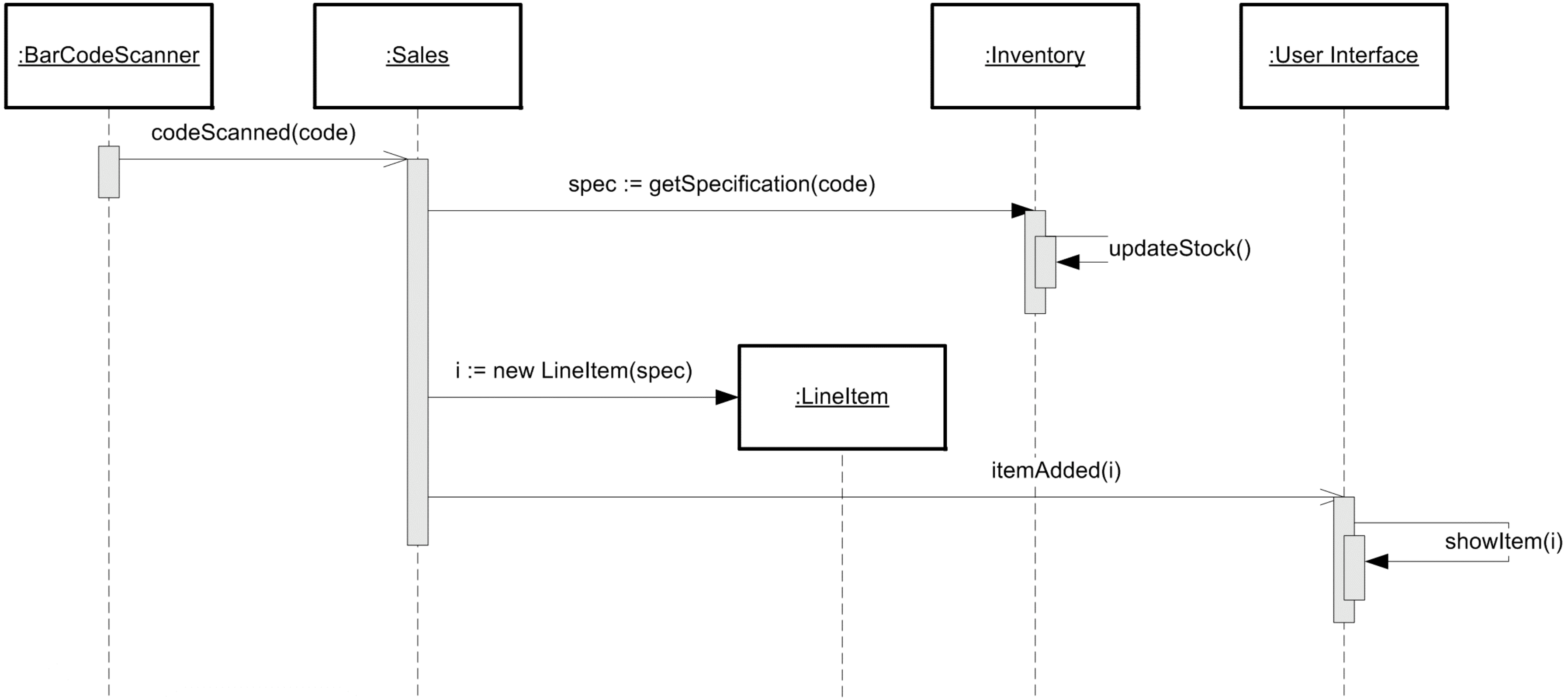
Active Objects



POS Example: C&C View



POS Example: C&C View (2)



Module Viewpoint

The software architecture of a computing system is the structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass et al., 2003]

Elements

– Classes, packages, interfaces

Relations

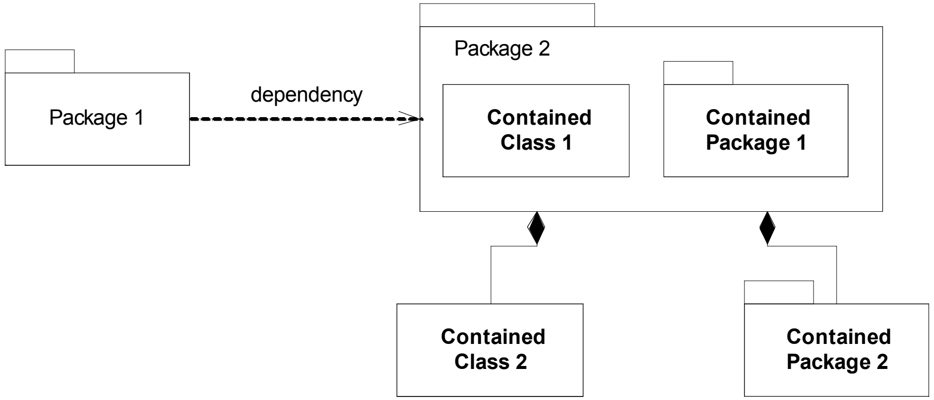
– Associations, generalizations, realizations, dependencies

Mapping to UML

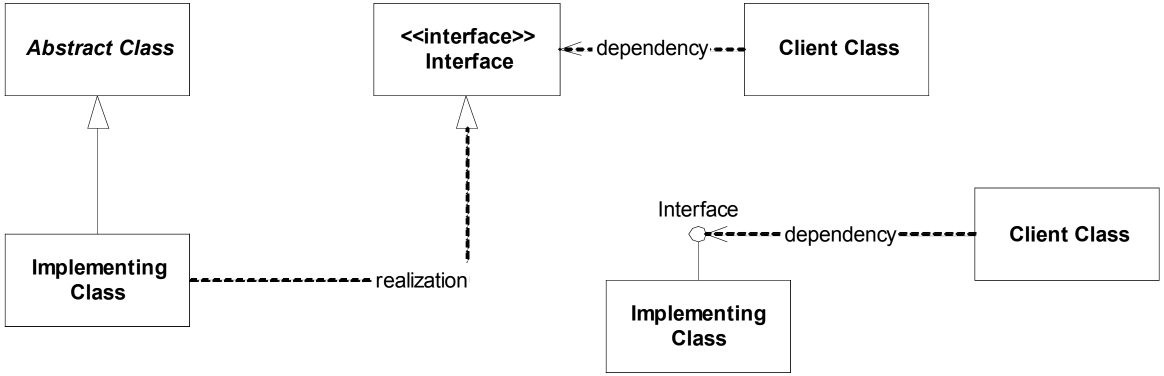
– Class diagrams…

Basic Module Elements (1)

Packages



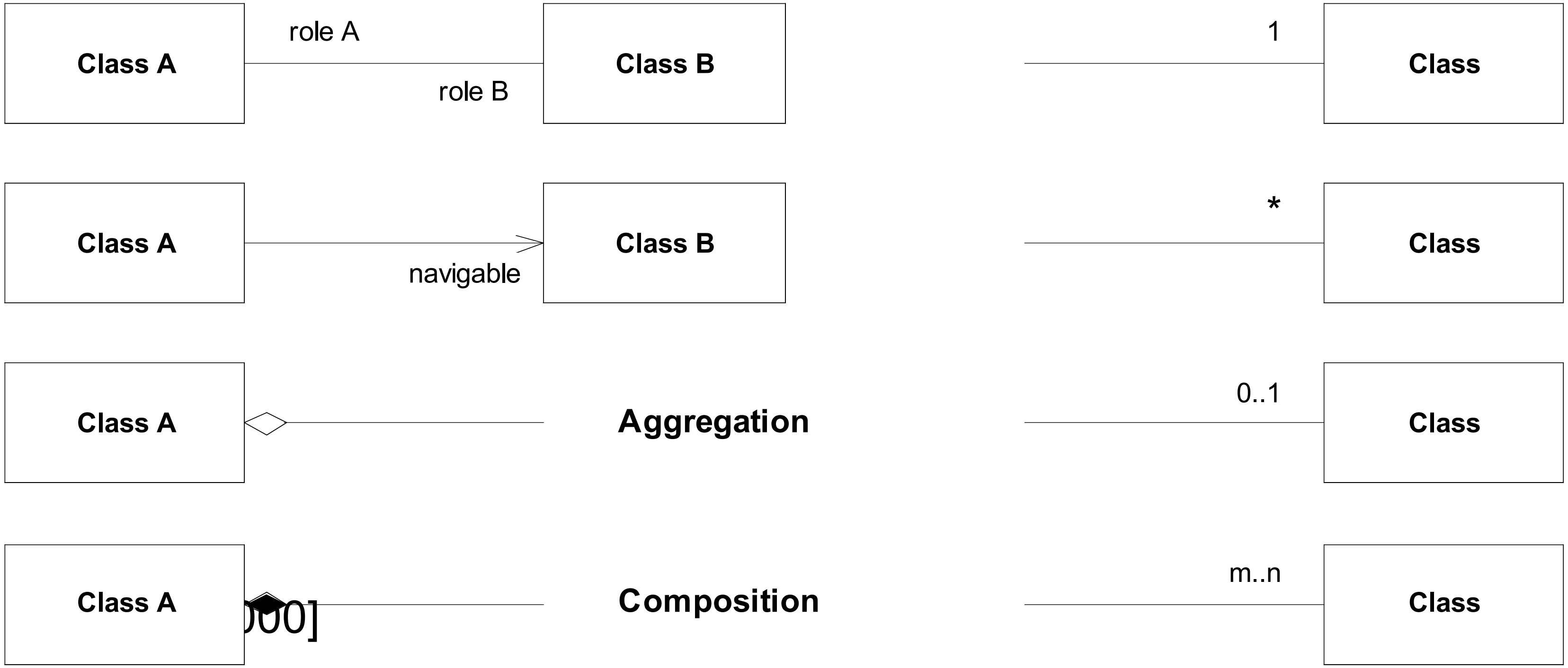
Interfaces



[Fowler, 2000]

Basic Module Elements (2)

Associations



Class A

role A

role B

Class B

1

Class

Class A

navigable

Class B

*

Class

Class A

Aggregation

0..1

Class

Class A

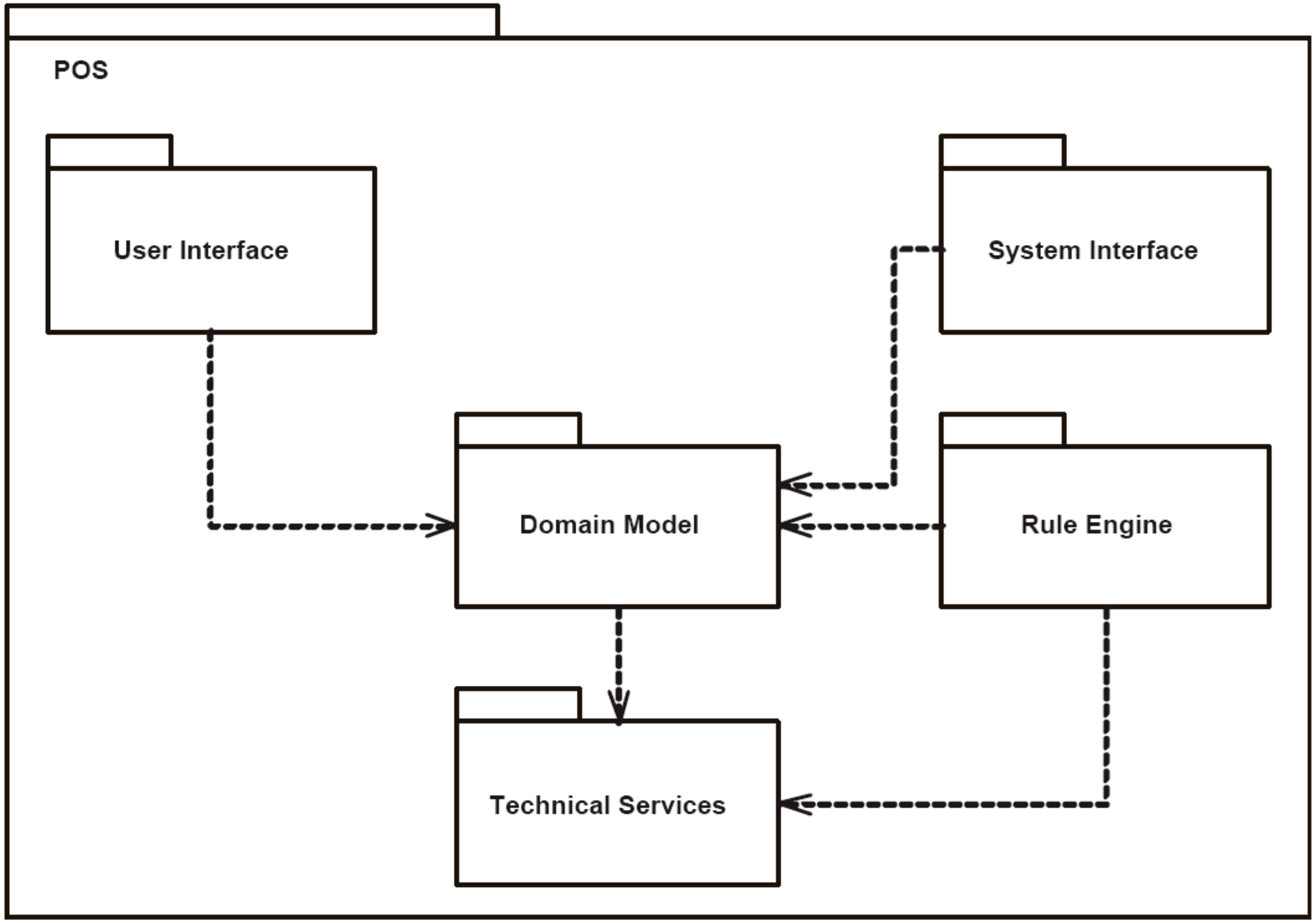
000]

Composition

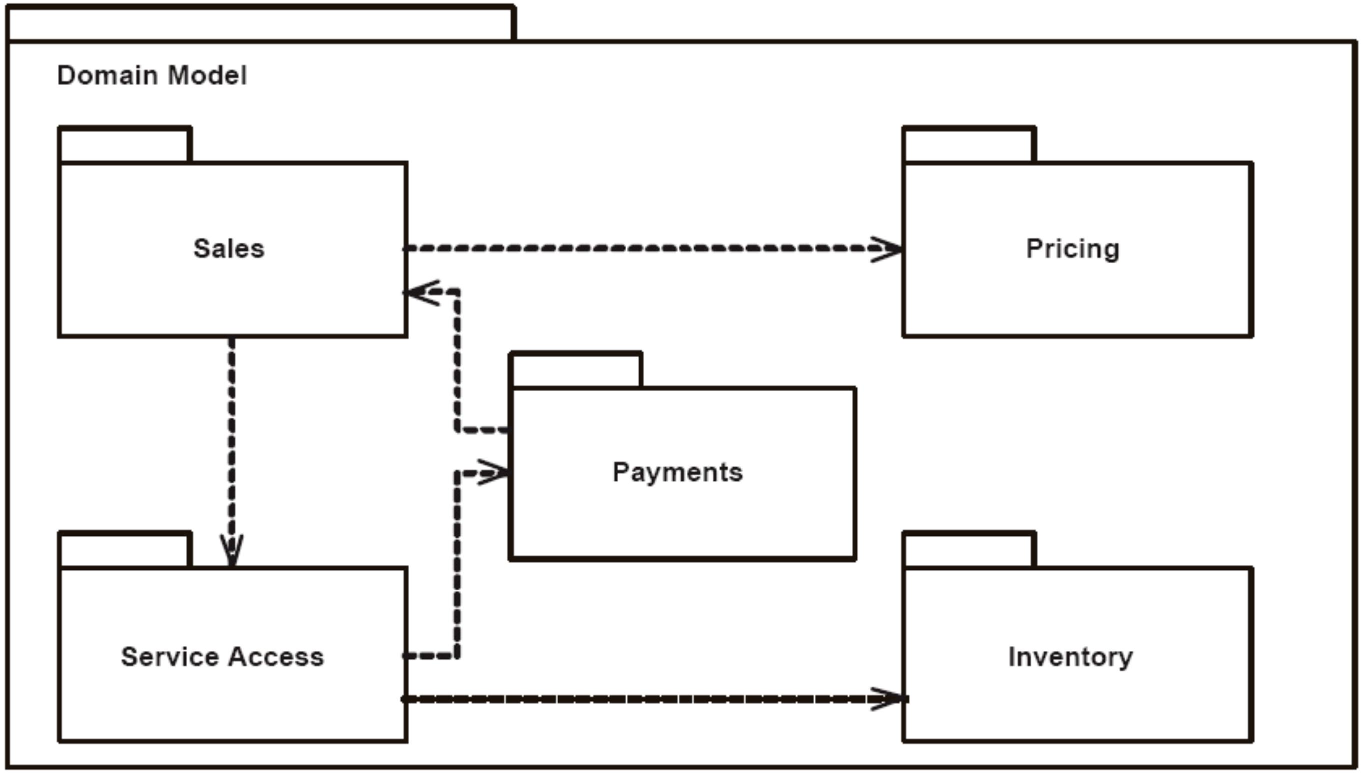
m..n

Class

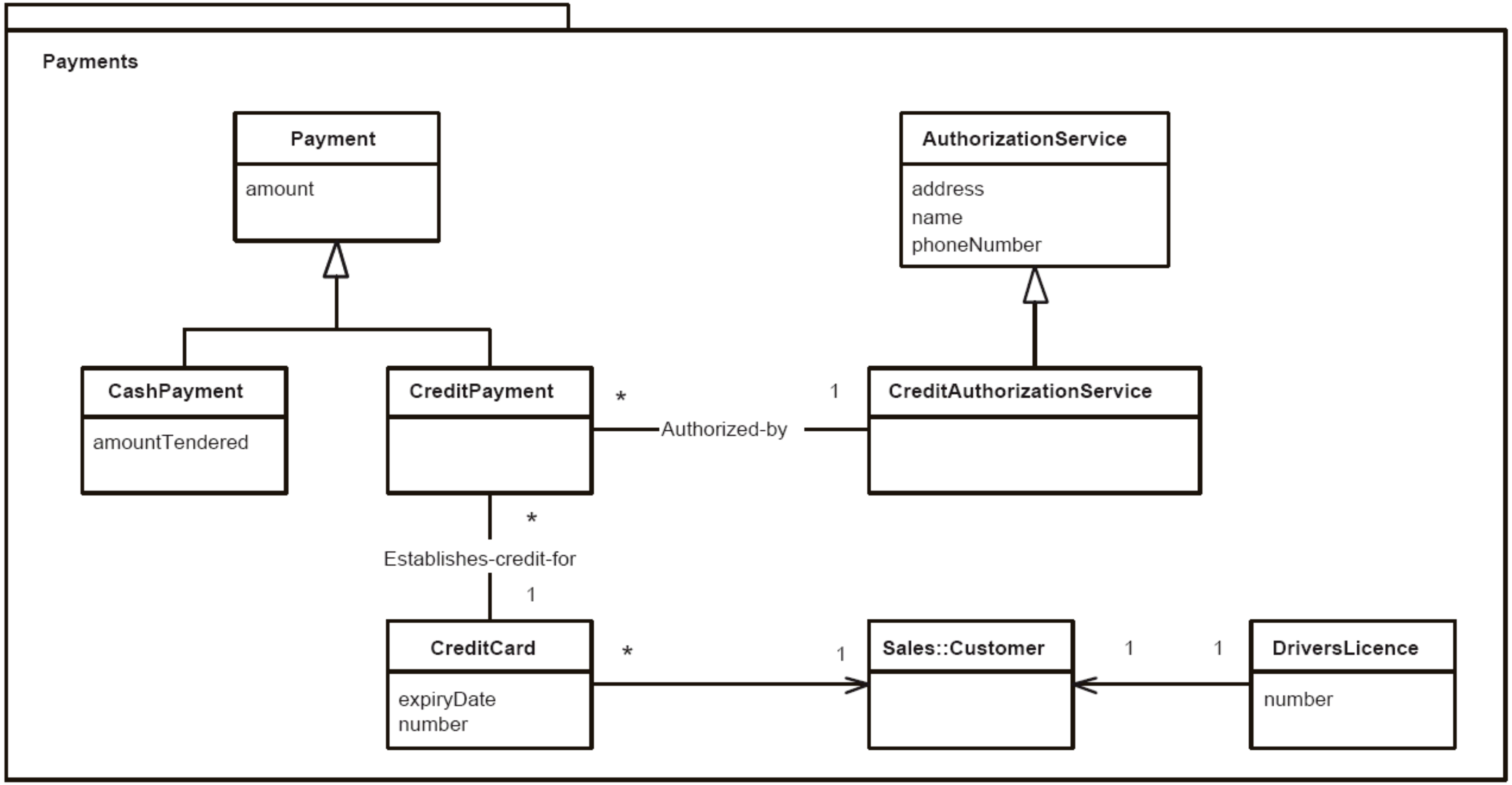
POS Example: Module View (1)



POS Example: Module View (2)



POS Example: Module View (3)



Allocation Viewpoint

> The software architecture of a computing system is the structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass et al., 2003]

Elements

- Software elements: Components, objects
- Environmental elements: Nodes

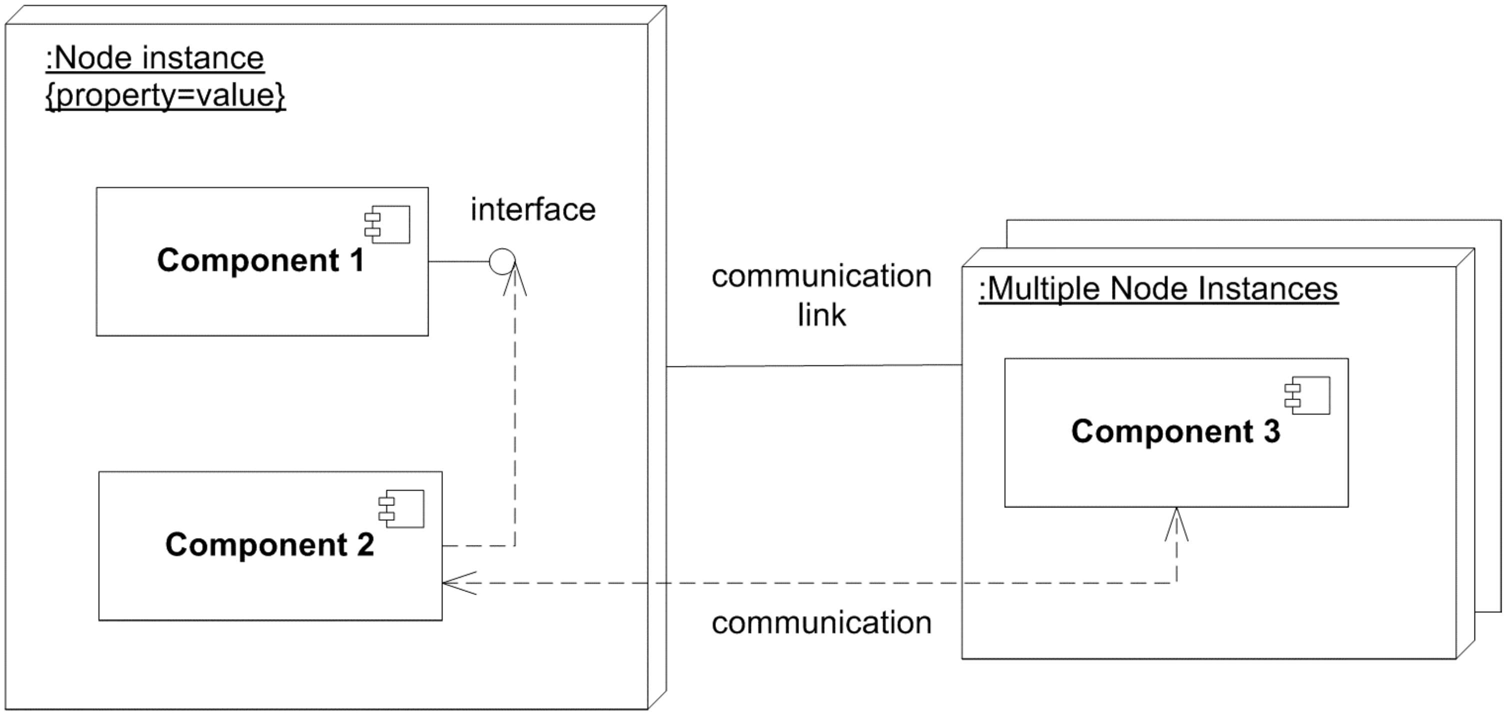
Relations

- Allocated-to
- Dependencies
- Connections (communication paths)

Mapping to UML

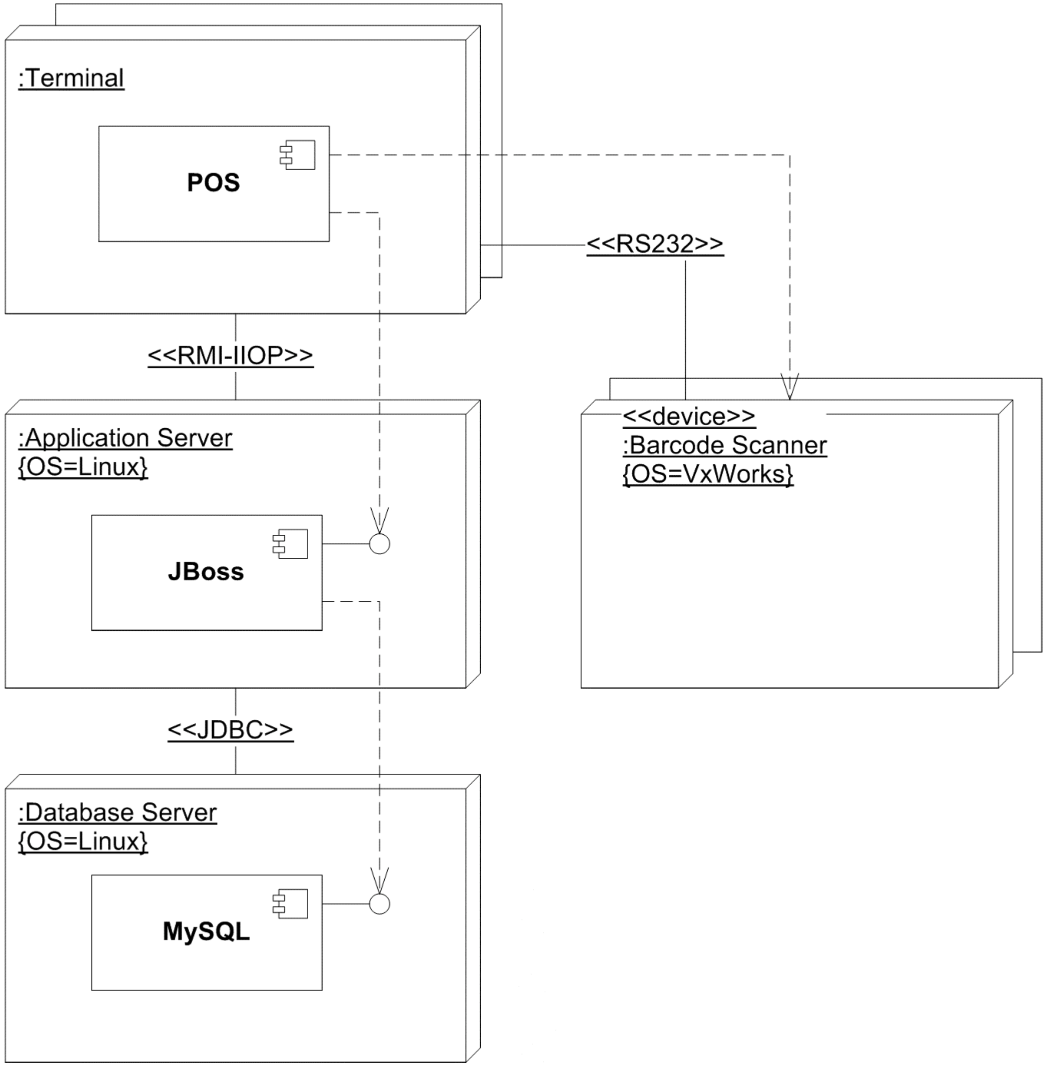
- Here: focus on deployment
  - Deployment and component diagrams

Basic Allocation Elements



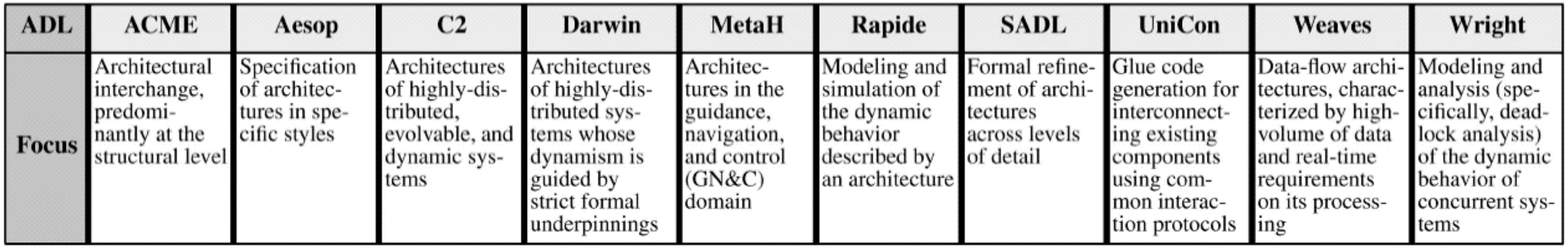
(Note: *UML* components are used here)
[Ambler: *Agile modelling]*

POS Example: Allocation View



Formal Description Languages

Architecture-specific



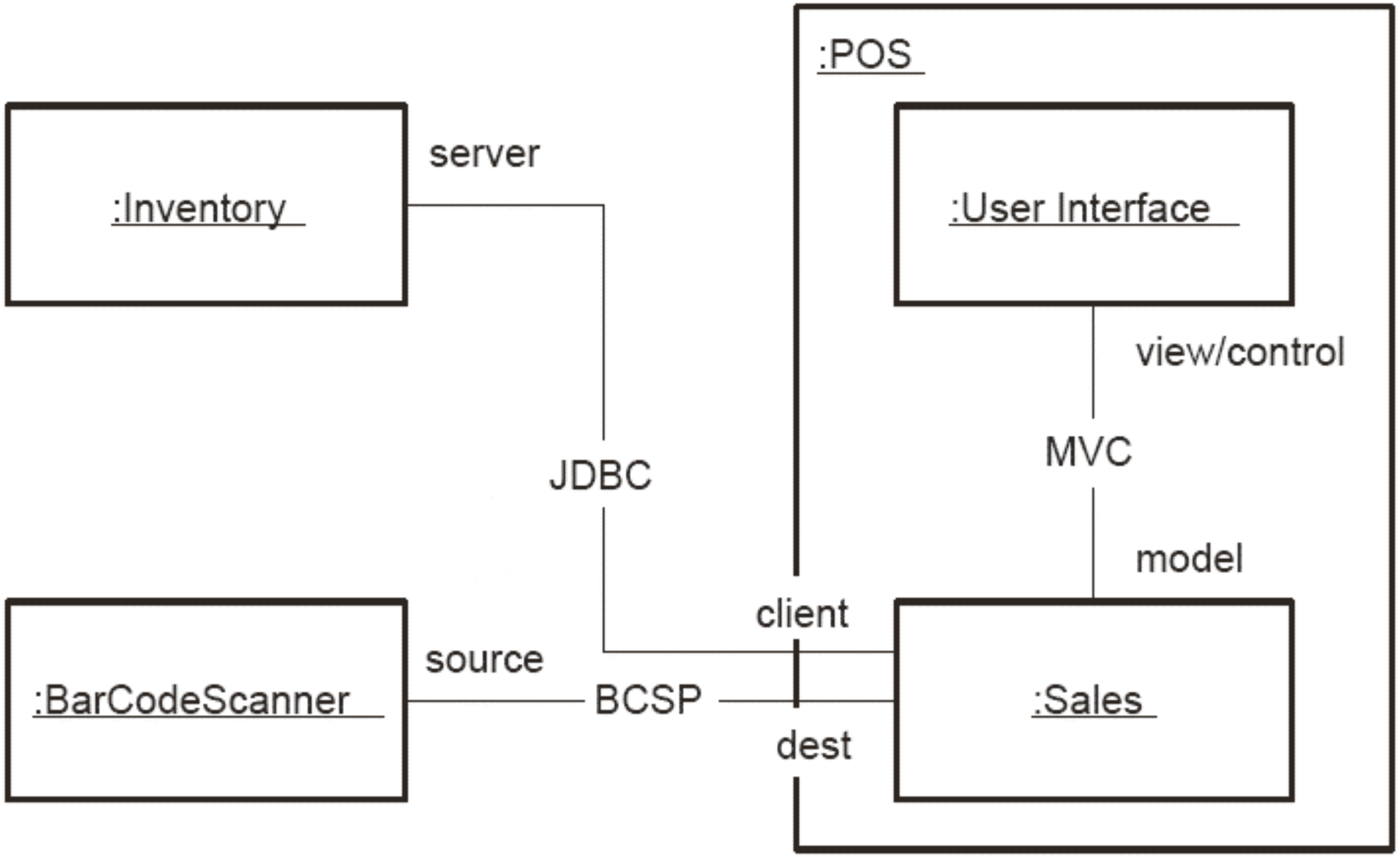
ADL	ACME	Aesop	C2	Darwin	MetaH	Rapide	SADL	UniCon	Weaves	Wright
Focus	Architectural interchange, predominantly at the structural level	Specification of architectures in specific styles	Architectures of highly-distributed, evolvable, and dynamic systems	Architectures of highly-distributed systems whose dynamism is guided by strict formal underpinnings	Architectures in the guidance, navigation, and control (GN&C) domain	Modeling and simulation of the dynamic behavior described by an architecture	Formal refinement of architectures across levels of detail	Glue code generation for interconnecting existing components using common interaction protocols	Data-flow architectures, characterized by high-volume of data and real-time requirements on its processing	Modeling and analysis (specifically, deadlock analysis) of the dynamic behavior of concurrent systems

Other

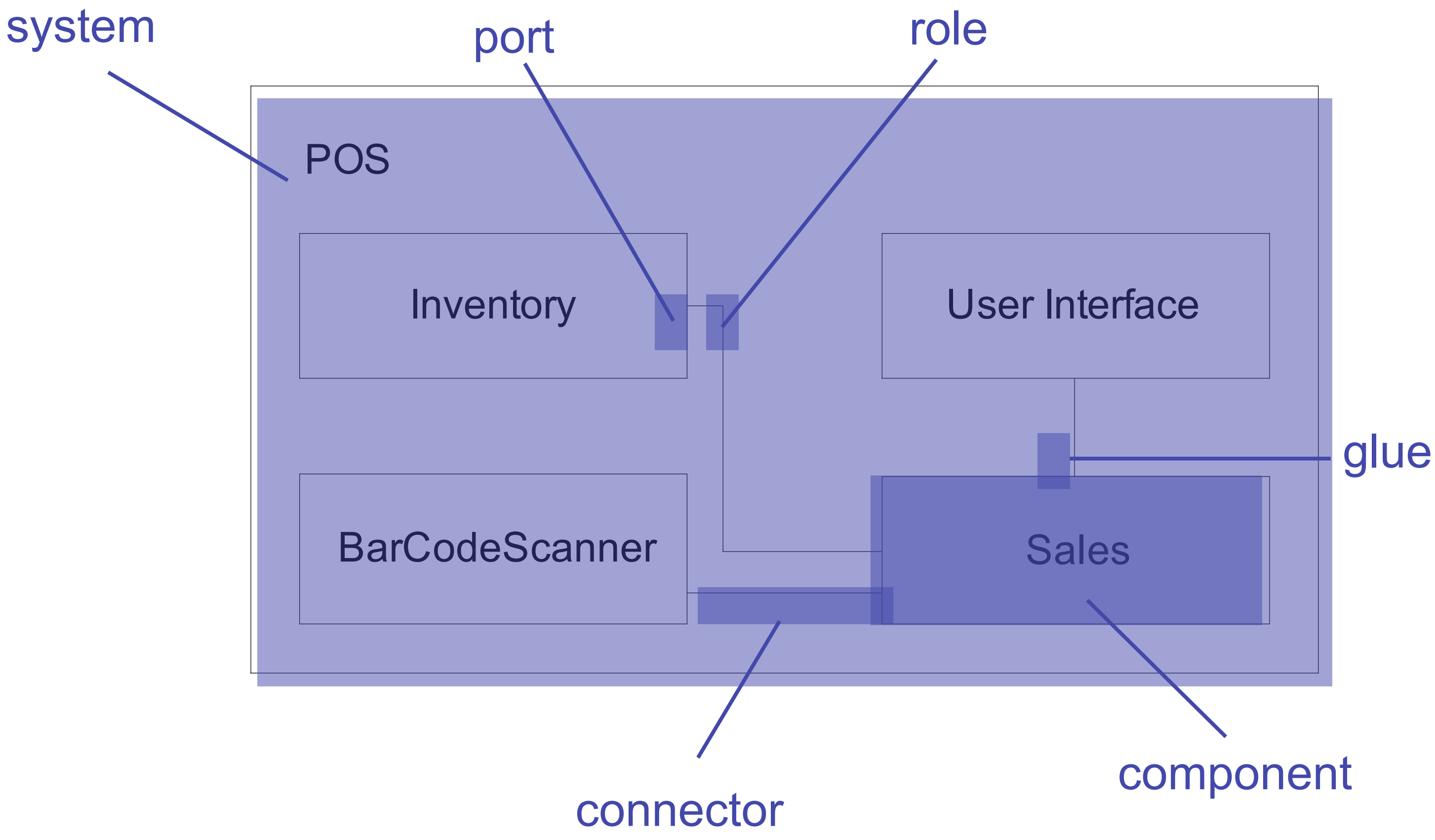
– E.g., rate-monotonic analysis on architectural components

– E.g., statechart-based analyses

UML Recap


:POS
:Inventory
server
:User Interface
view/control
MVC
JDBC
model
client
source
:BarCodeScanner
BCSP
:Sales
dest

Wright Concepts



Wright Basics

System

- Component types
  - Name
  - Ports
    - Logical points of interaction with environment
  - Spec
    - Abstract behaviour
- Connector types
  - Name
  - Roles
    - Possible behaviour of participant in an interaction
  - Glue
    - Coordination of behaviour of participants in an interaction
- Instances
  - Of components and connectors
- Attachments
  - Of component ports to connector roles

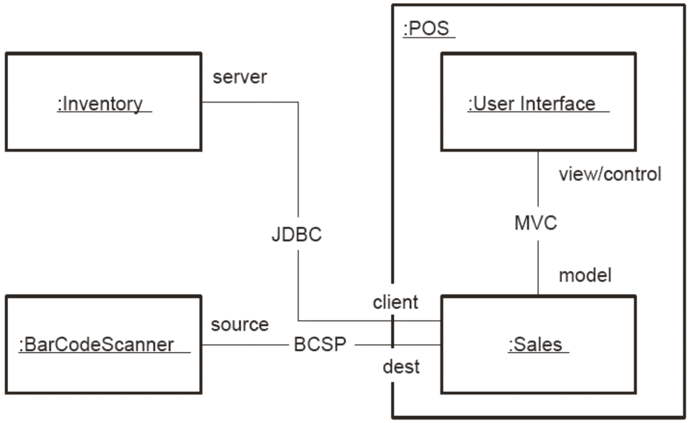
Translated into process algebra

- Modified version of Communicating Sequential Processes (CSP)

POS Example

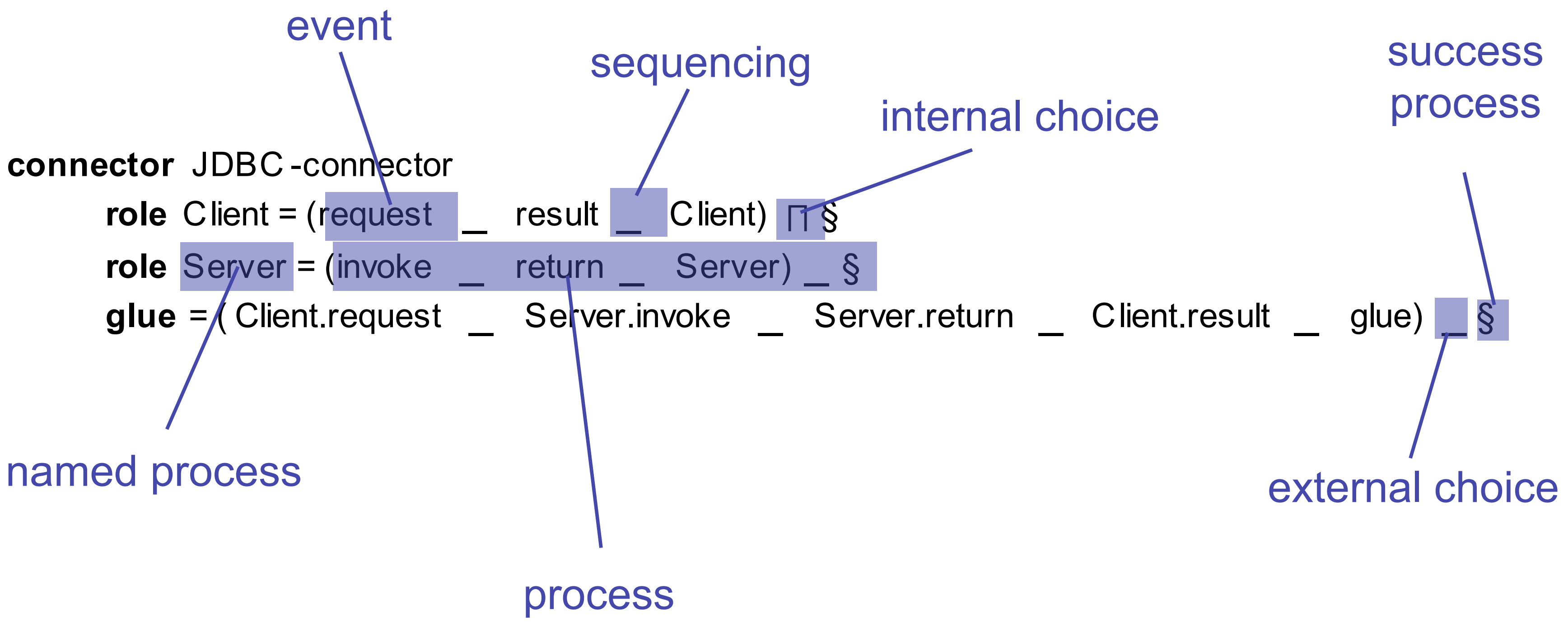
```
System POS
    component Inventory =
        port provide […]
        spec […]
    component Sales =
        port request […]
        port consume […]
        spec […]
    component BarCodeScanner =
        port produce […]
        spec […]
    connector JDBC-connector
        role client […]
        role server […]
        glue […]
    connector BCSP-connector
        role source […]
        role dest […]
        glue […]
```

```
Instances
    i:      Inventory
    s:      Sales
    b:      BarCodeScanner
    jdbc:   JDBC-connector
    bscp:   BCSP-connector
Attachments
    i.provide as jdbc.server;
    s.request as jdbc.client;
    s.consume as bcsp.dest;
    b.produce as bcsp.source
end POS
```



(Cave at: this example is most probably not correct Wright ☺)

POS Example: Behavior Specification



```
connector  JDBC -connector
        role  Client = (request  _   result  _  Client)  ⊓ §
        role  Server = (invoke   _   return  _   Server) _ §
        glue  = ( Client.request  _   Server.invoke  _   Server.return  _   Client.result  _   glue)  _ §
```

POS Example: Behavior Specification (2)

scoped process name

write event

read event

```
connector  BCSP-connector
        role  Source  = (produce ?x _   Source) ⊓ §
        role  Dest = ( consume !x _   Dest) ⊓ §
        glue =   let Continue =   Source. produce !x _   Continue  _ Dest. consume ?x _   Continue  _ §
                  in Source. produce !x _   Continue   _ §
```

label

Discussion

Component & Connector viewpoint does not map well to implementation?

- Few languages have interaction as first-class construct
- Neither to the UML – bound tightly to OO implementation
- On the other hand central in many approaches to architectural design

UML is not designed specifically for software architecture description?

- Many (irrelevant) modeling constructs
- But very widely used and supported

Not precise enough for formal analysis (?)