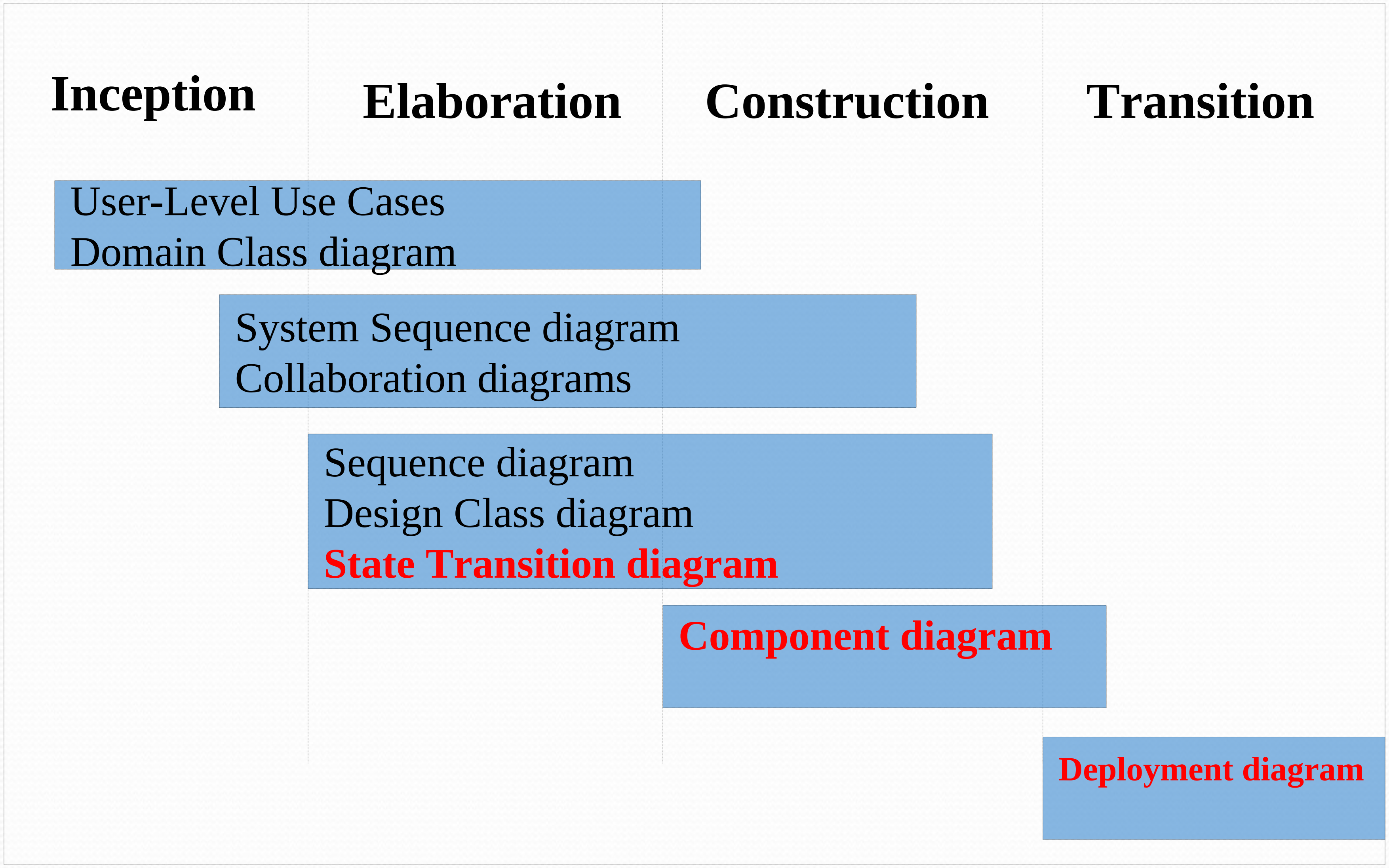
COMP 3711

(OOA and OOD)

More UML Diagrams

Larman Chapter 28, 29, 33, 34, 39

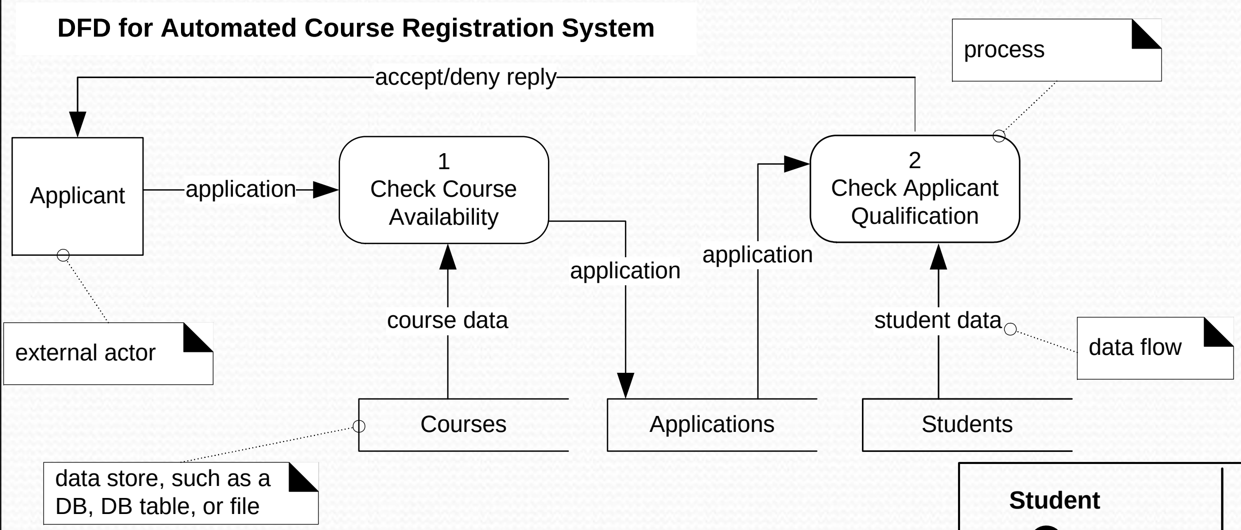
UML Diagrams Over UP

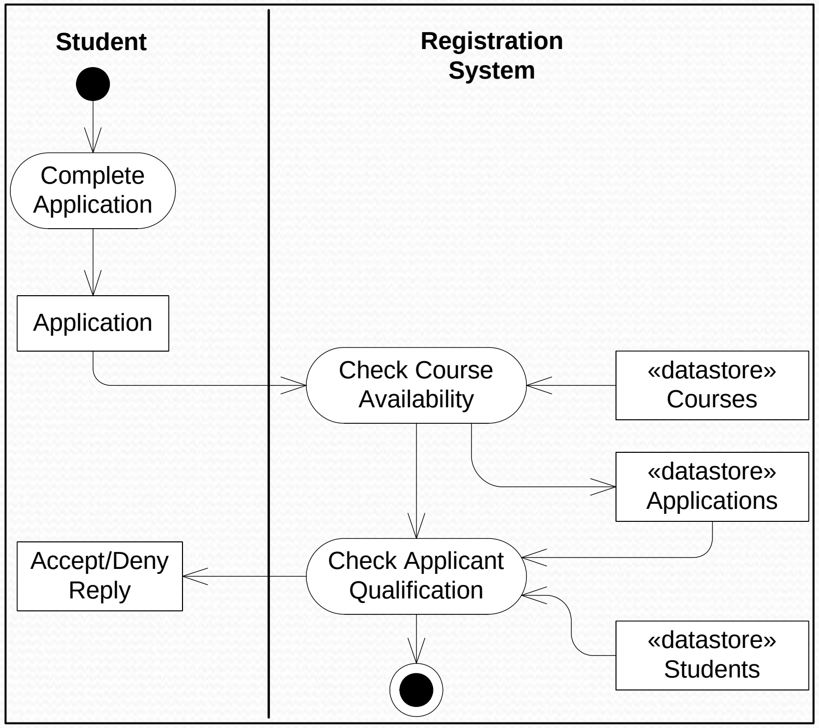


UML Activity Diagram

- Show sequence of activities (including parallel activities) – dynamic model
- Generally used for visualizing business workflows and processes and use-case (Business Object Modeling)
- Replace the traditional DFD (Data-Flow Diagram)
- Remember: “*Pictures worth a thousand words*”

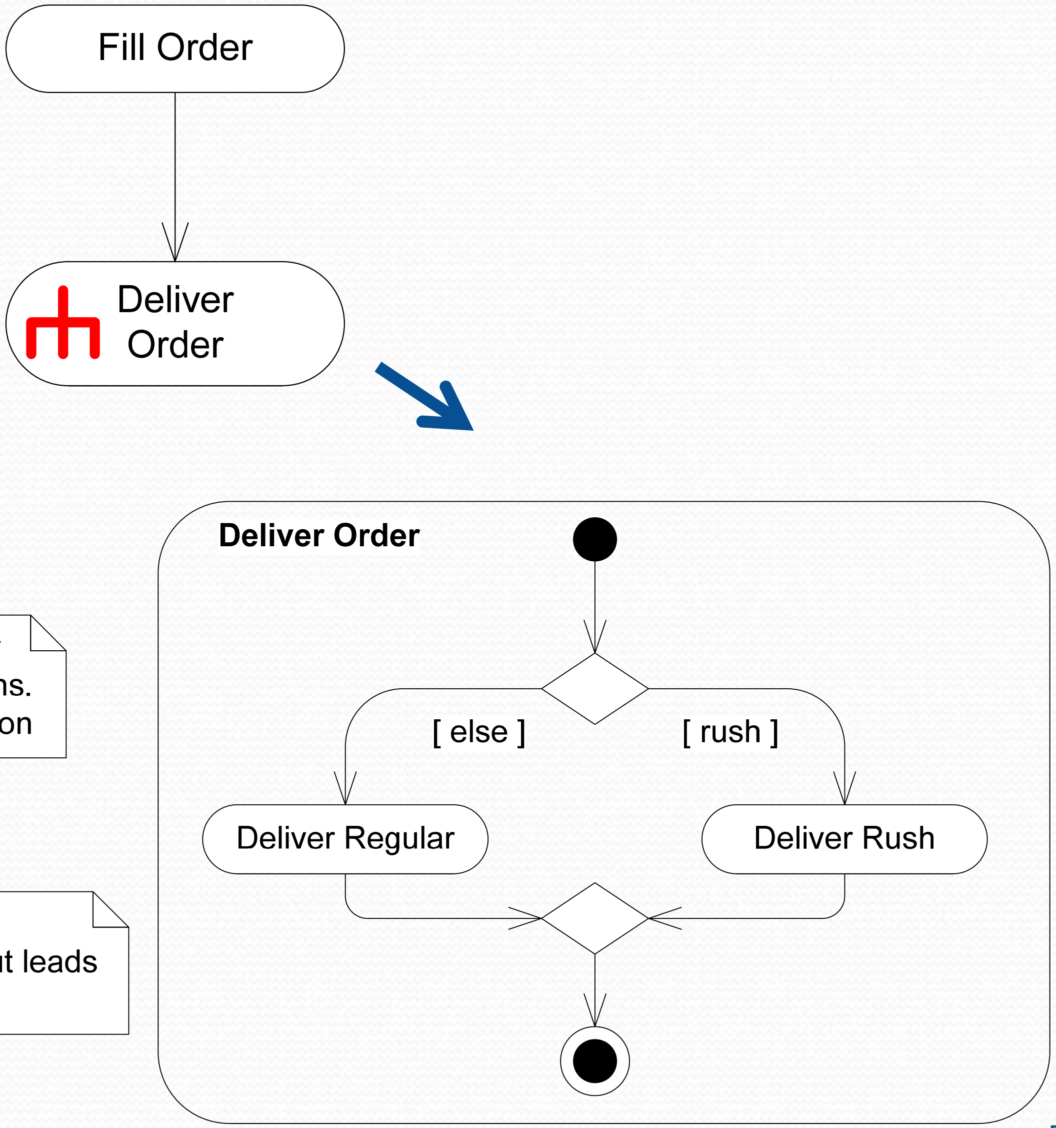
Example - Activity Diagram





Larman 28.2, 28.3

UML – Activity Diagram



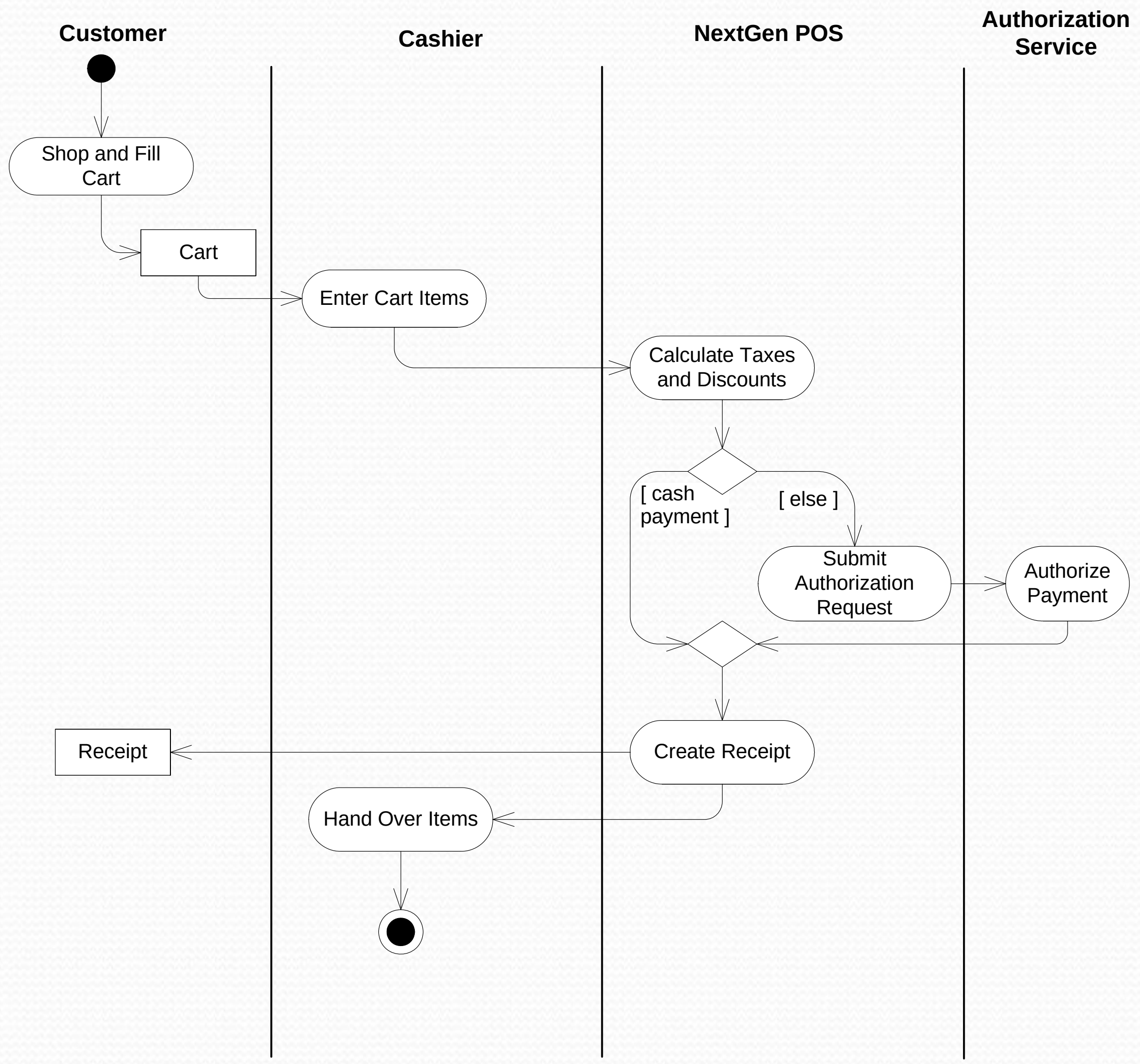
the "rake" symbol (which represents a hierarchy) indicates this activity is expanded in a sub-activity diagram

Decision: Any branch happens. Mutual exclusion

Merge: *Any* input leads to continuation.

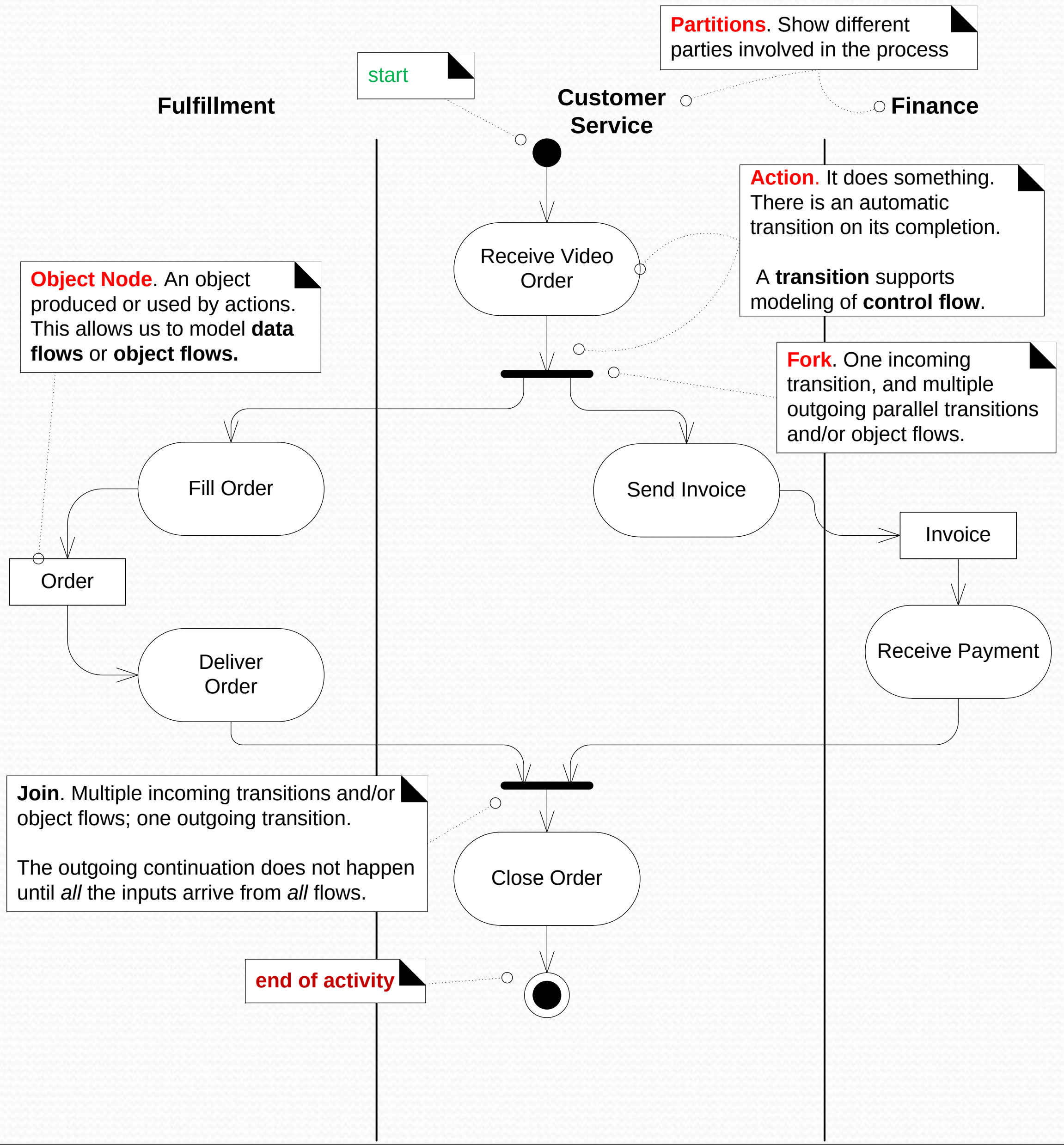
Larman 28.4, 28.5

Example – NextGen Activity Diagram For Process Sale Use Case



Larman 28.7

Activity Diagram Notations



Partitions. Show different parties involved in the process
start
Fulfillment
Customer Service
Finance
Action. It does something. There is an automatic transition on its completion.
A transition supports modeling of control flow.
Receive Video Order
Object Node. An object produced or used by actions. This allows us to model data flows or object flows.
Fork. One incoming transition, and multiple outgoing parallel transitions and/or object flows.
Fill Order
Send Invoice
Invoice
Order
Receive Payment
Deliver Order
Join. Multiple incoming transitions and/or object flows; one outgoing transition.
The outgoing continuation does not happen until all the inputs arrive from all flows.
Close Order
end of activity

Larman 28.1

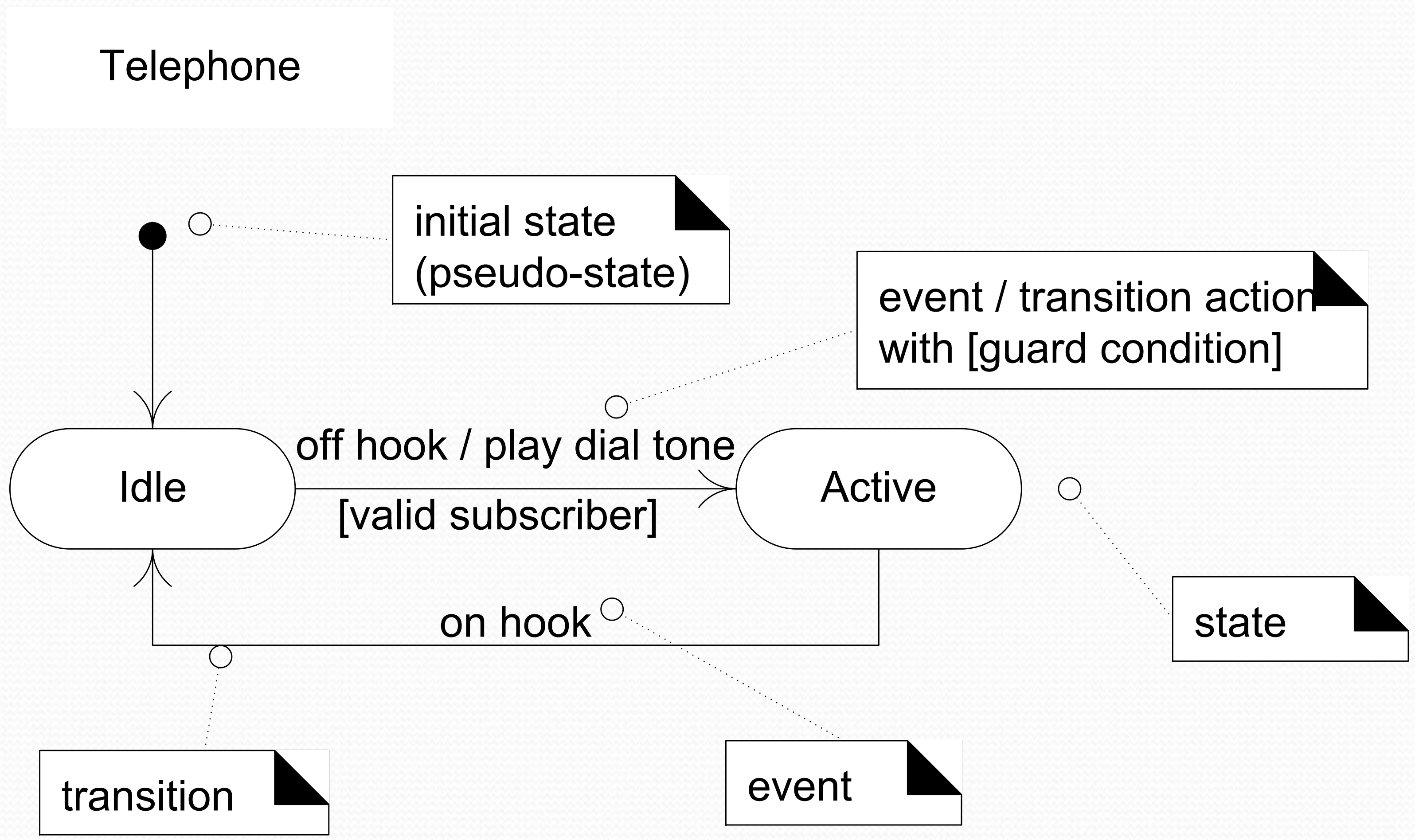
UML State Machine Diagram

- State Machine Diagram = State Transition Diagram = Statechart
- Shows transition behaviour of an object in reaction to an event through out the object lifecycle
- An “event” is a significant or noteworthy occurrence
- A “state” is condition of an object at a moment in time between events, 1 start state, multiple or 0 stop states
- A “transition” is a relationship between two states as the object moves from prior state to subsequent state

Types Of Events

- External event (System event)
    - Caused by an actor outside the system boundary
- Internal event
    - Caused by something inside the system boundary, i.e. when a method is invoked via a message from another internal object
- Temporal event
    - Caused by the occurrence of a specific date and time or by the passage of time
        - driven by a real-time or simulated-time clock

State Machine Diagram Notations


Telephone
initial state (pseudo-state)
event / transition action with [guard condition]
Idle
off hook / play dial tone
[valid subscriber]
Active
state
on hook
transition
event

Larman 29.1, 29.2

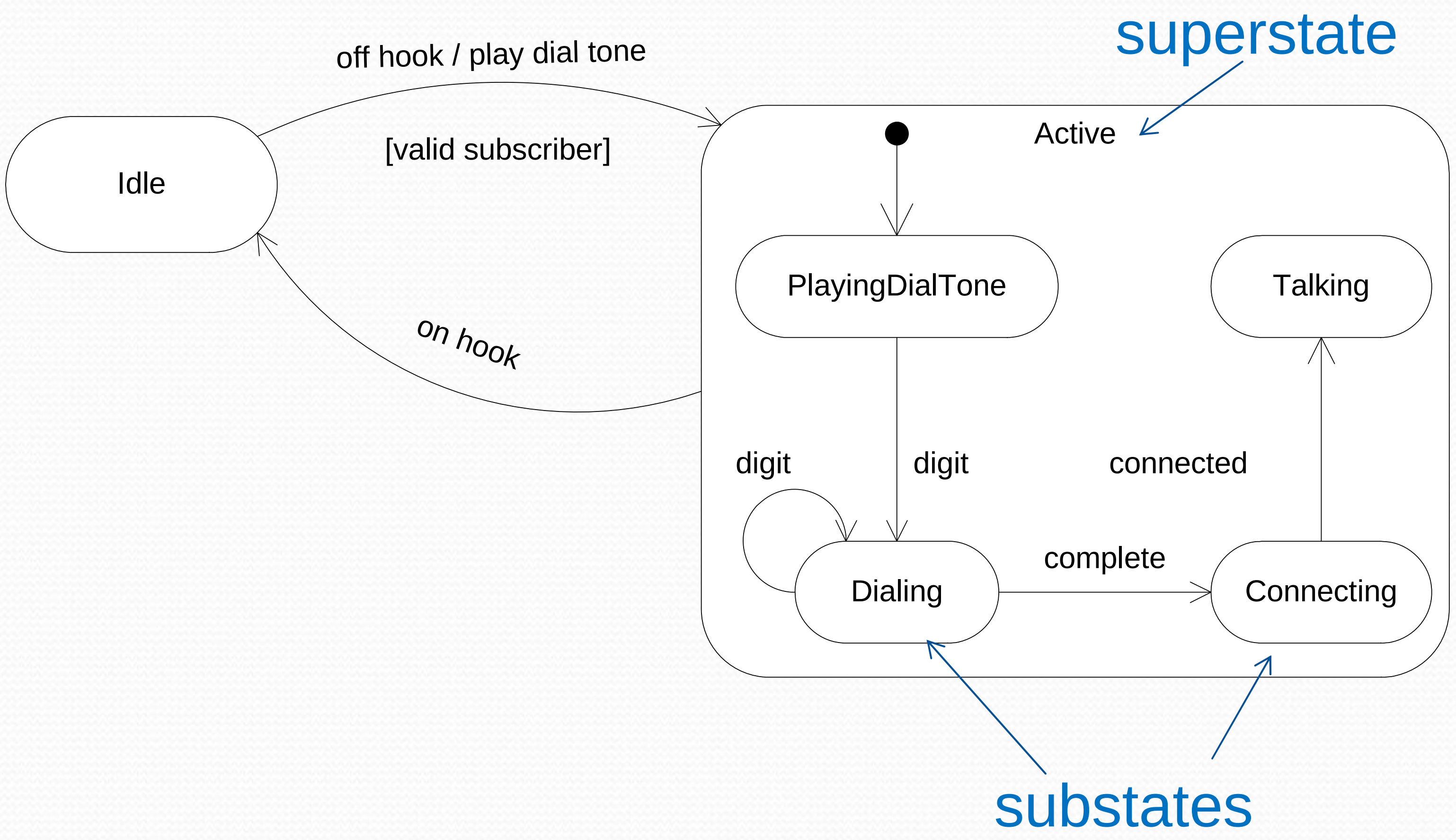
What Objects To Model?

- No need to show all events

- Model state-dependent objects with complex behaviour (those react differently to events depending on their state or mode), for example:
  - physical devices
  - transactions
  - role mutators
  - communication protocols
  - UI (navigation, controllers, sessions, event handling)
  - use case system operations

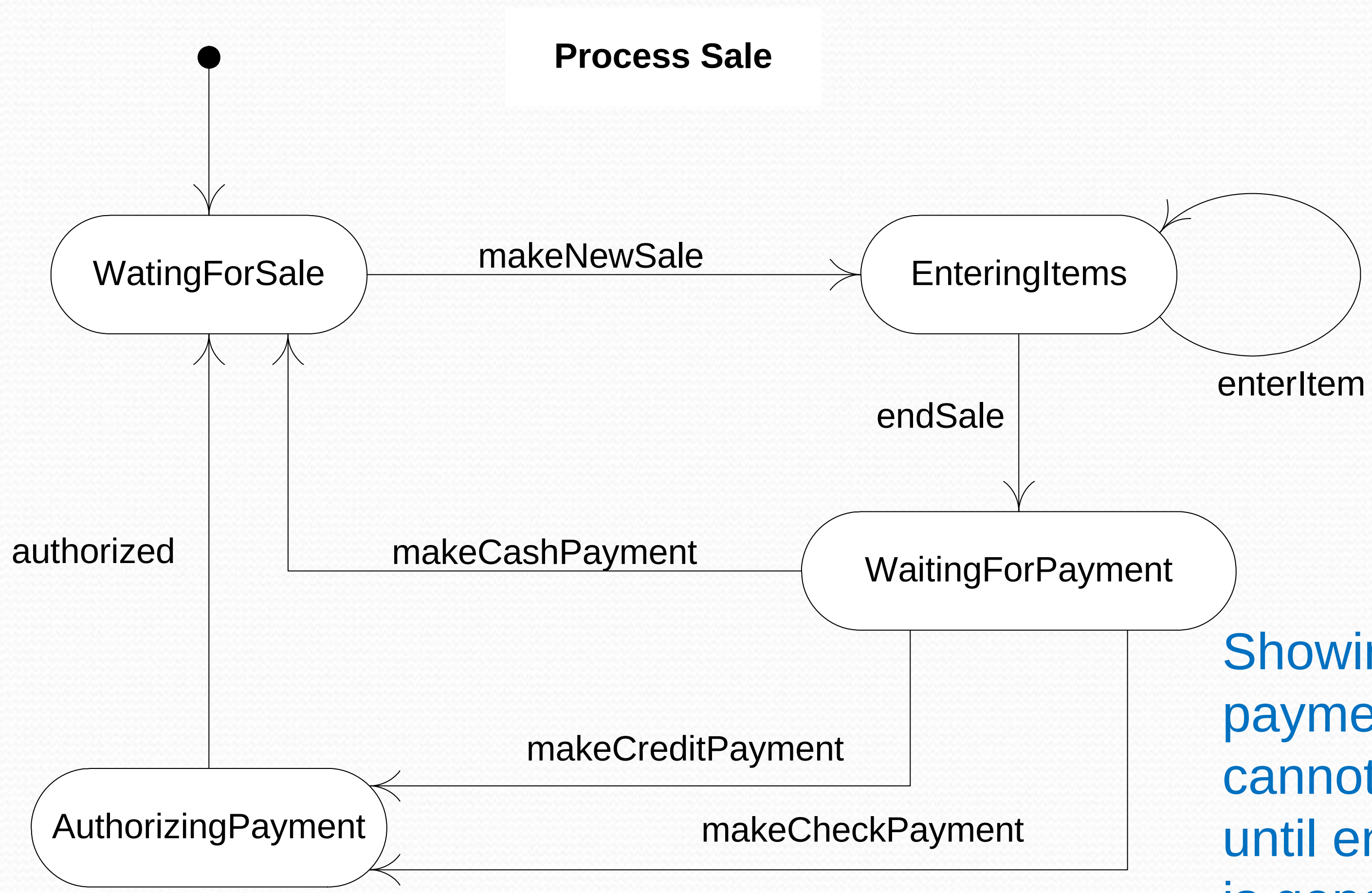
Nested States

- Substates can be shown by nesting in a superstate box



Larman 29.3

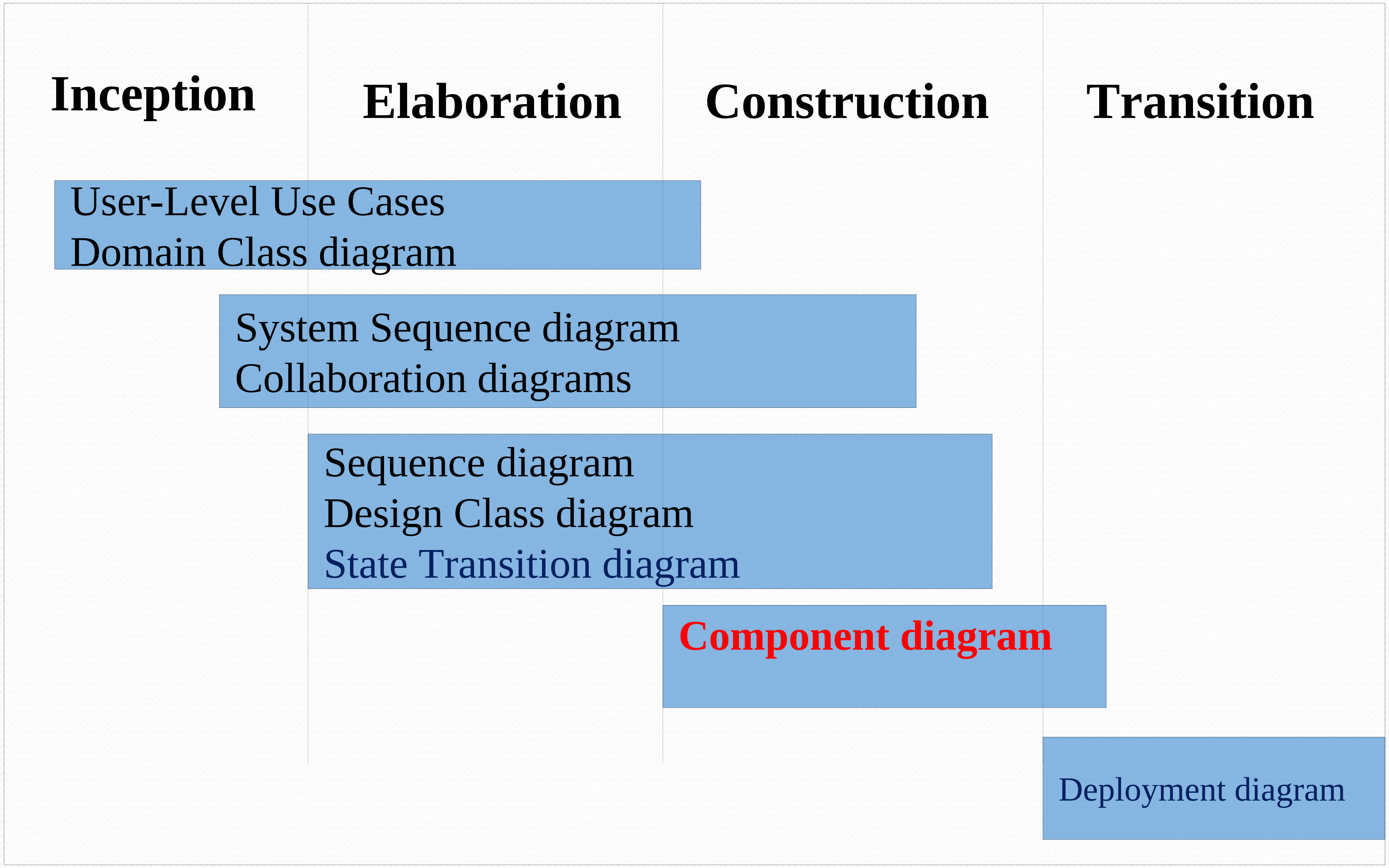
NextGen Example



Showing payment cannot occur until endSale is generated

Larman 29.5

More UML Diagrams



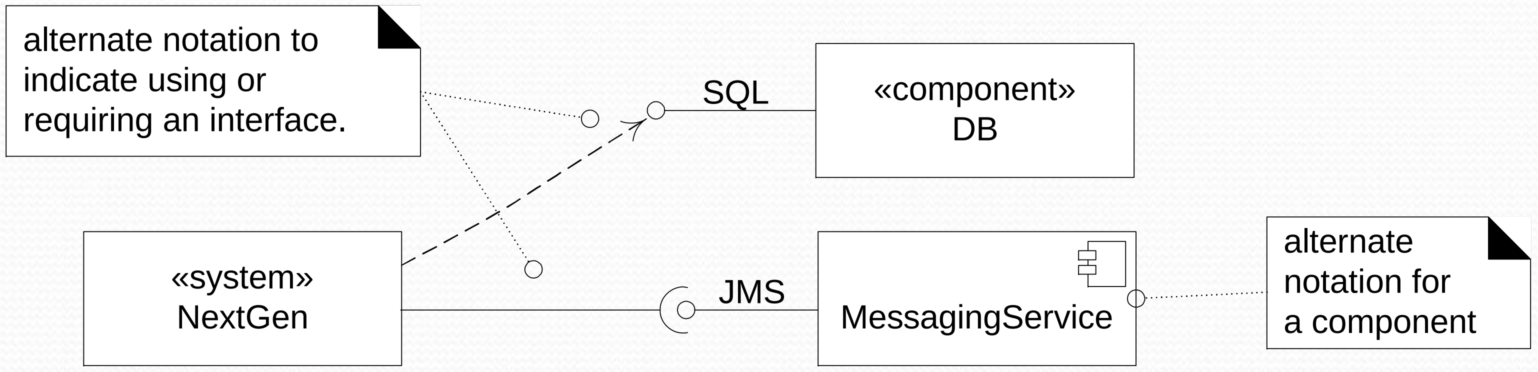
UML – Component Diagram

- *Quote from UML spec[OMG-03b] “A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behaviour in terms of provided and required interfaces. As such, a component serves as a type, whose conformances is defined by these provided and required interfaces”*

- Represent design-level perspective and not concrete software perspective

NextGen Component Diagram

- Emphasis the importance of interfaces and modularity (self-contained and replaceable)


alternate notation to indicate using or requiring an interface.
SQL
«component» DB
«system» NextGen
JMS
MessagingService
alternate notation for a component

Larman 38.2 A Component Diagram

Architectural Analysis

- Architectural Analysis is concerned with identification and resolution of non-functional requirements (e..g security, platform, fault-tolerance, etc.)

- Iterative UP allows Architectural Analysis to perform iteratively with the first development iteration

- Identify variation points and evolution points for Architectural Analysis
  - variation points (e.g. multiple tax calculators)
  - evolution points (e.g. support multi-language)

Architectural Factors

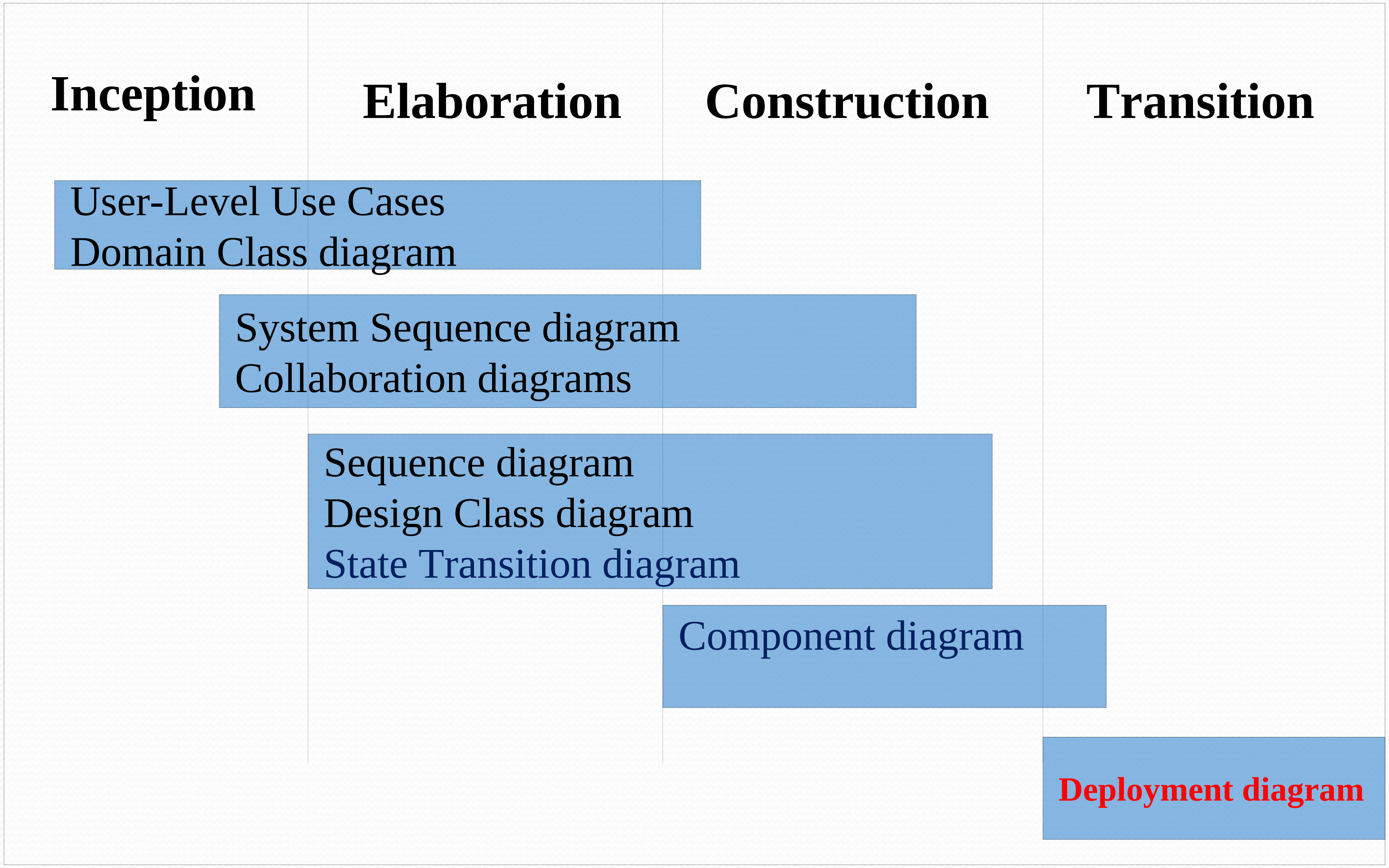
- Also known as Architectural Drivers
- Those of architectural significant requirements: such as high level FURPS+ (e.g. Reliability, Performance. Supportability)
- Describe in Factor Table using the following:
  - Measures and quality scenarios
  - Variability (current flexibility and future evolution)
  - Impact of factor on stakeholders, architecture
  - Priority for success (high or low)
  - Difficulty or Risk (high, medium, low)

Architectural Decisions

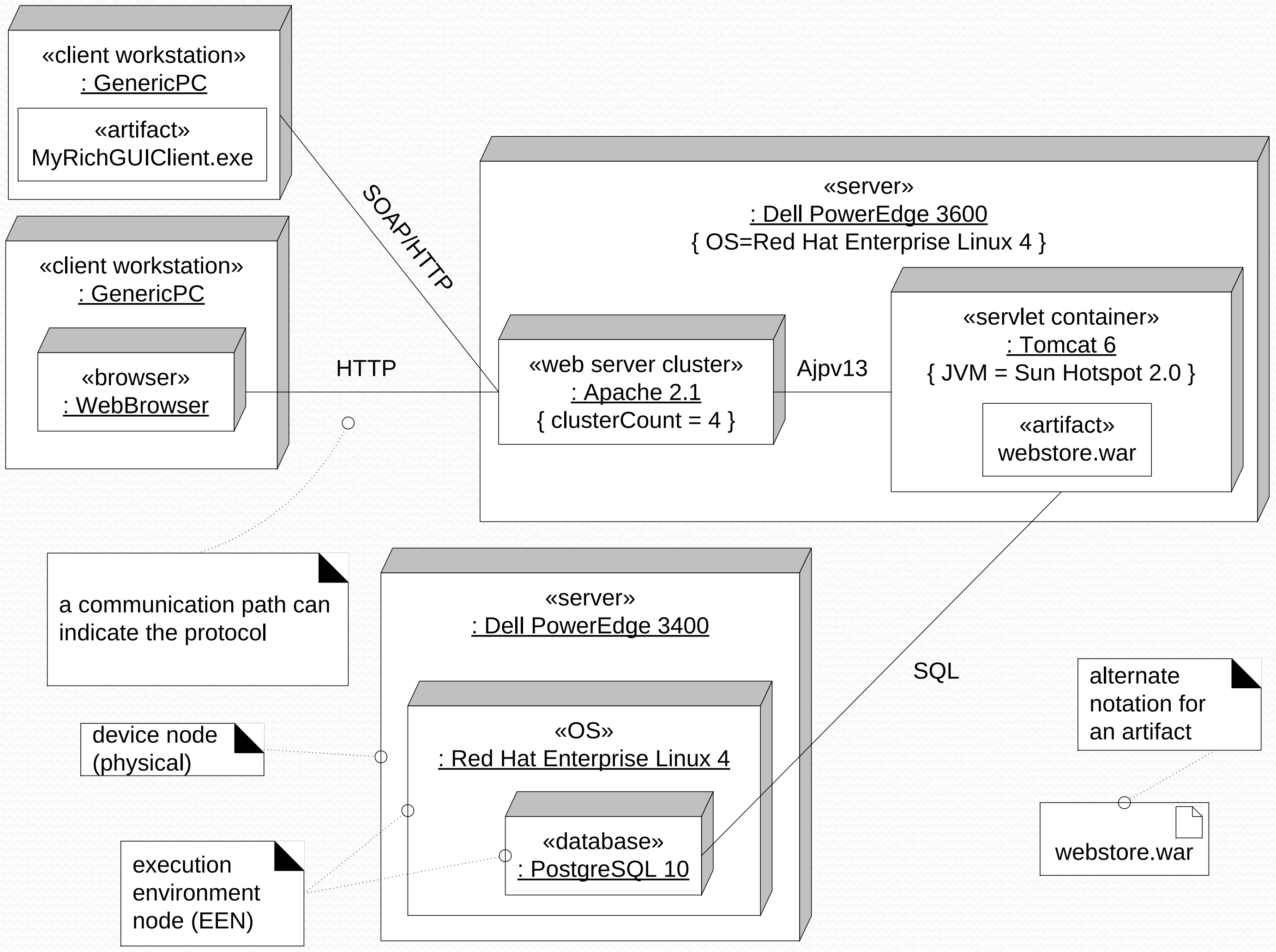
- One of the principles applied in architectural analysis is to separate concerns and localization impact (e.g. data persistency, security, etc.)

- For each architectural factor, record alternative solutions, decisions, influential factors, motivations for noteworthy issues and decisions in:
  - technical memos
  - UML-SAD (Software Architecture Document)
  - Issue cards
  - architectural approach documents

More UML Diagrams

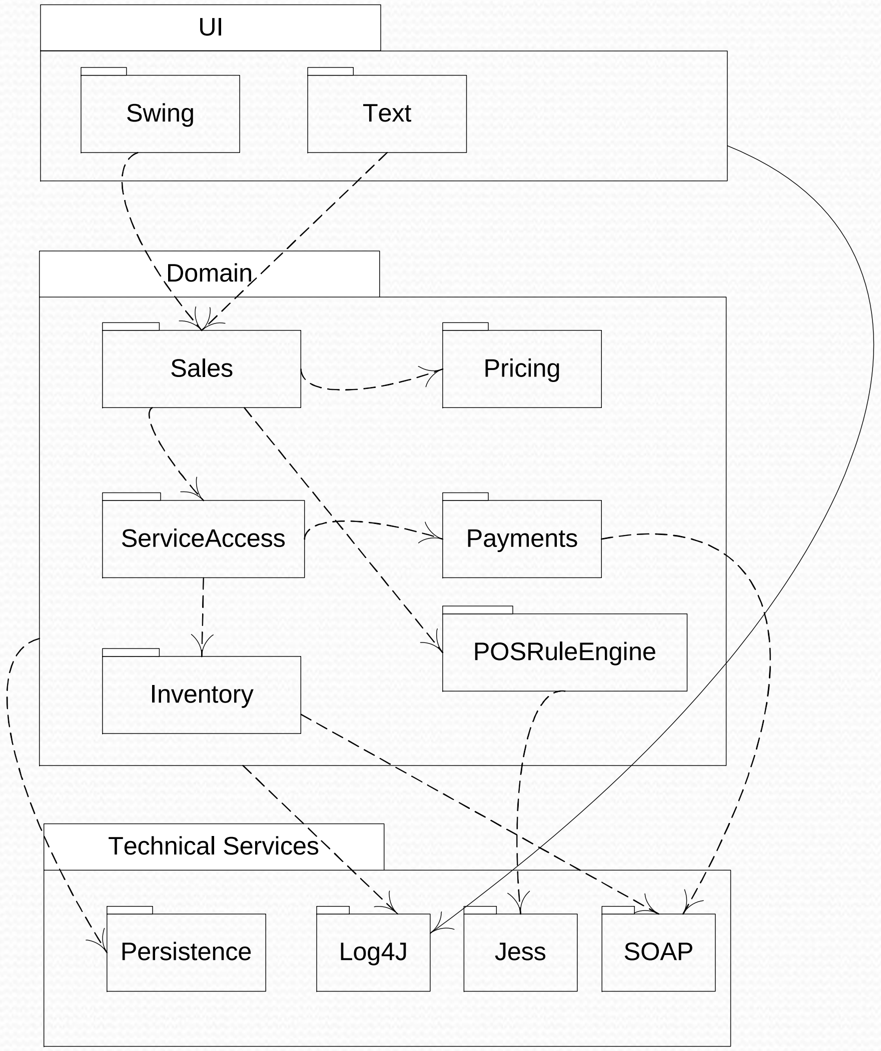


NextGen Deployment Diagram



Larman 38.1 A Deployment Diagram is often used to communicate the physical and deployment architecture

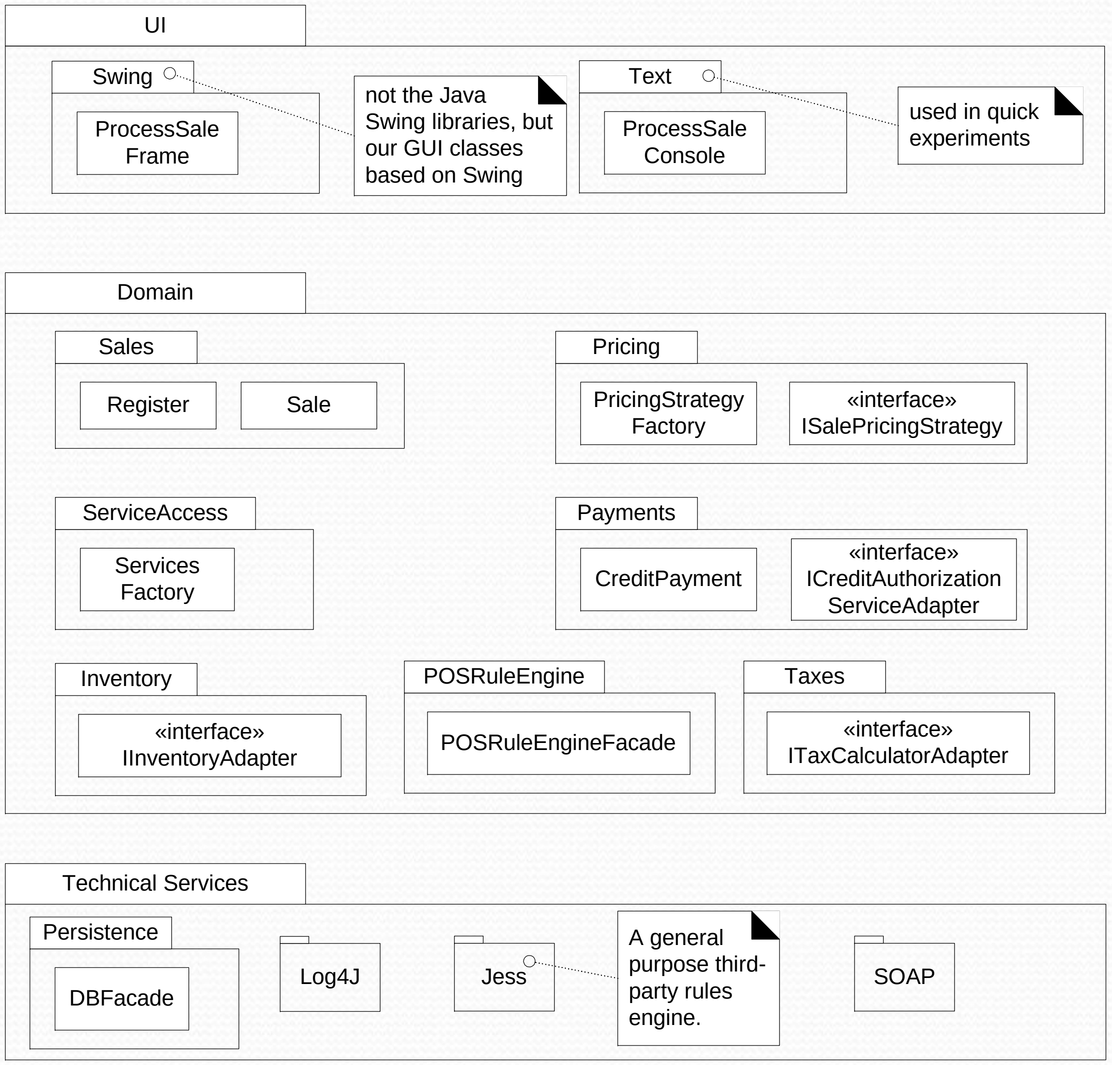
NextGen Logical Architecture Diagram



Showing key note-worthy elements for “big ideas” and ignoring the Application layer

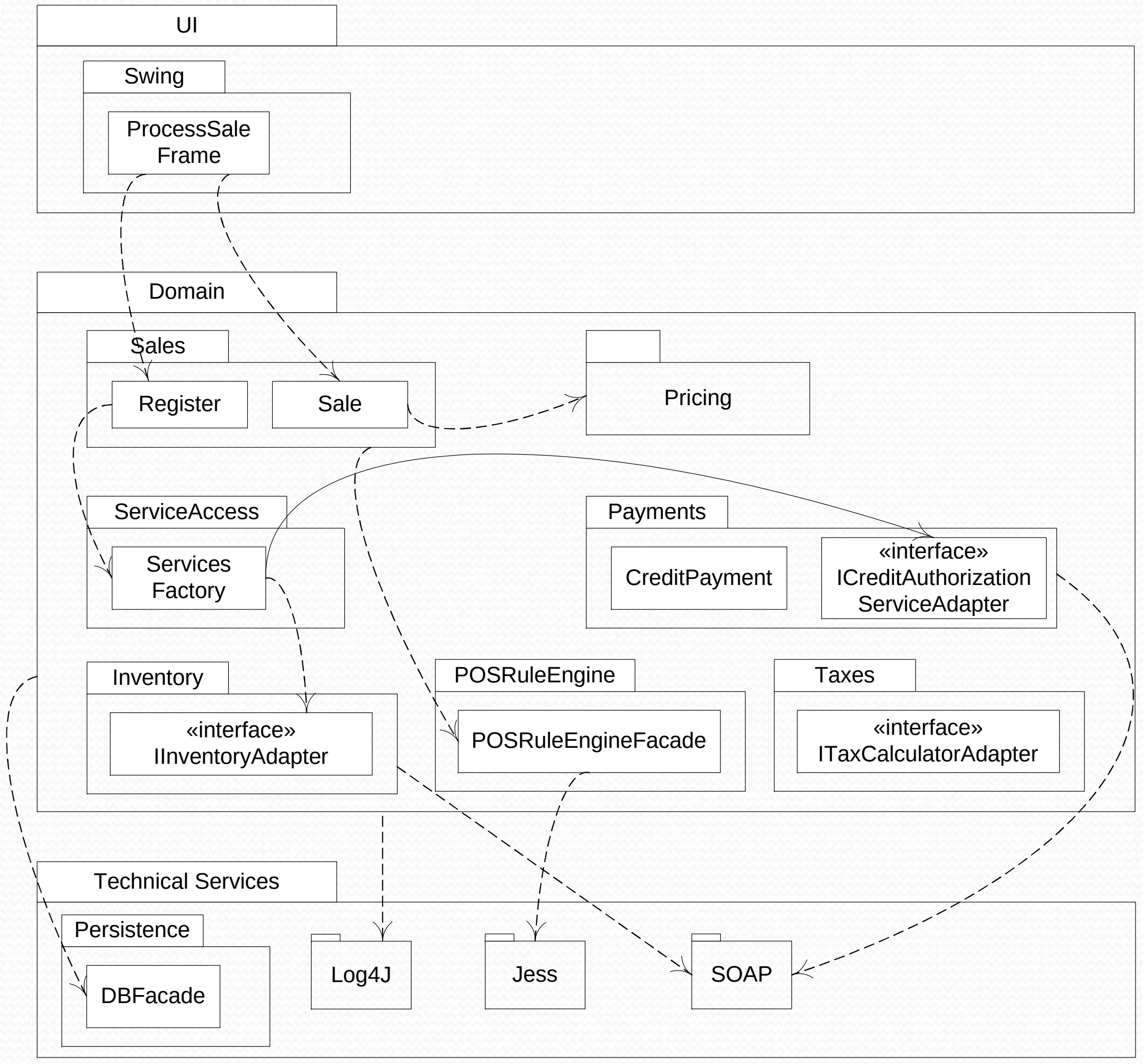
Larman 34.3

NextGen Logical View Packages


UI
Swing
ProcessSale Frame
not the Java Swing libraries, but our GUI classes based on Swing
Text
ProcessSale Console
used in quick experiments
Domain
Sales
Register
Sale
Pricing
PricingStrategy Factory
«interface» ISalePricingStrategy
ServiceAccess
Services Factory
Payments
CreditPayment
«interface» ICreditAuthorization ServiceAdapter
Inventory
«interface» IInventoryAdapter
POSRuleEngine
POSRuleEngineFacade
Taxes
«interface» ITaxCalculatorAdapter
Technical Services
Persistence
DBFacade
Log4J
Jess
A general purpose third-party rules engine.
SOAP

Larman 34.1

Logical View Showing Couplings



Larman 34.2

UML Rational Rose Four Views

- Use Case View *(**what** the system will do)*
  - *Actors, Use Cases and Use Case diagrams*

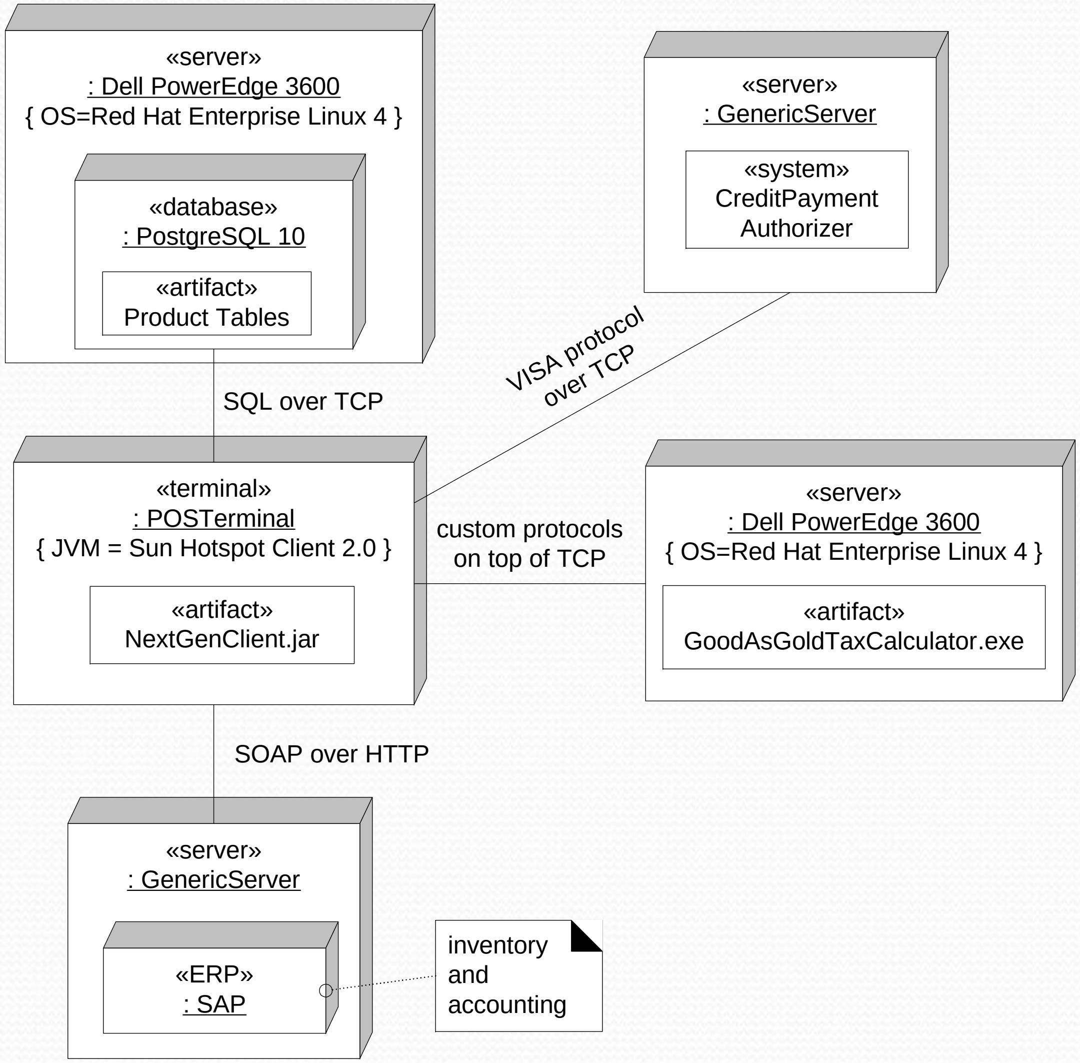
- Logical View *(**how** the system will implement behavior in the use cases)*
  - *Class diagrams, Integration diagrams, State Transition diagrams*

UML Rose Four Views

- Component View *(physical modules of the system)*
  - *Component diagrams, Code libraries, executable files, runtime libraries*

- Deployment View *(physical deployment of the system)*
  - *Physical architecture (not logical architecture): processes, processors, devices*
  - *Fault tolerance, network bandwidth, disaster recovery, response time*

NextGen Deployment View



Larman 39.2