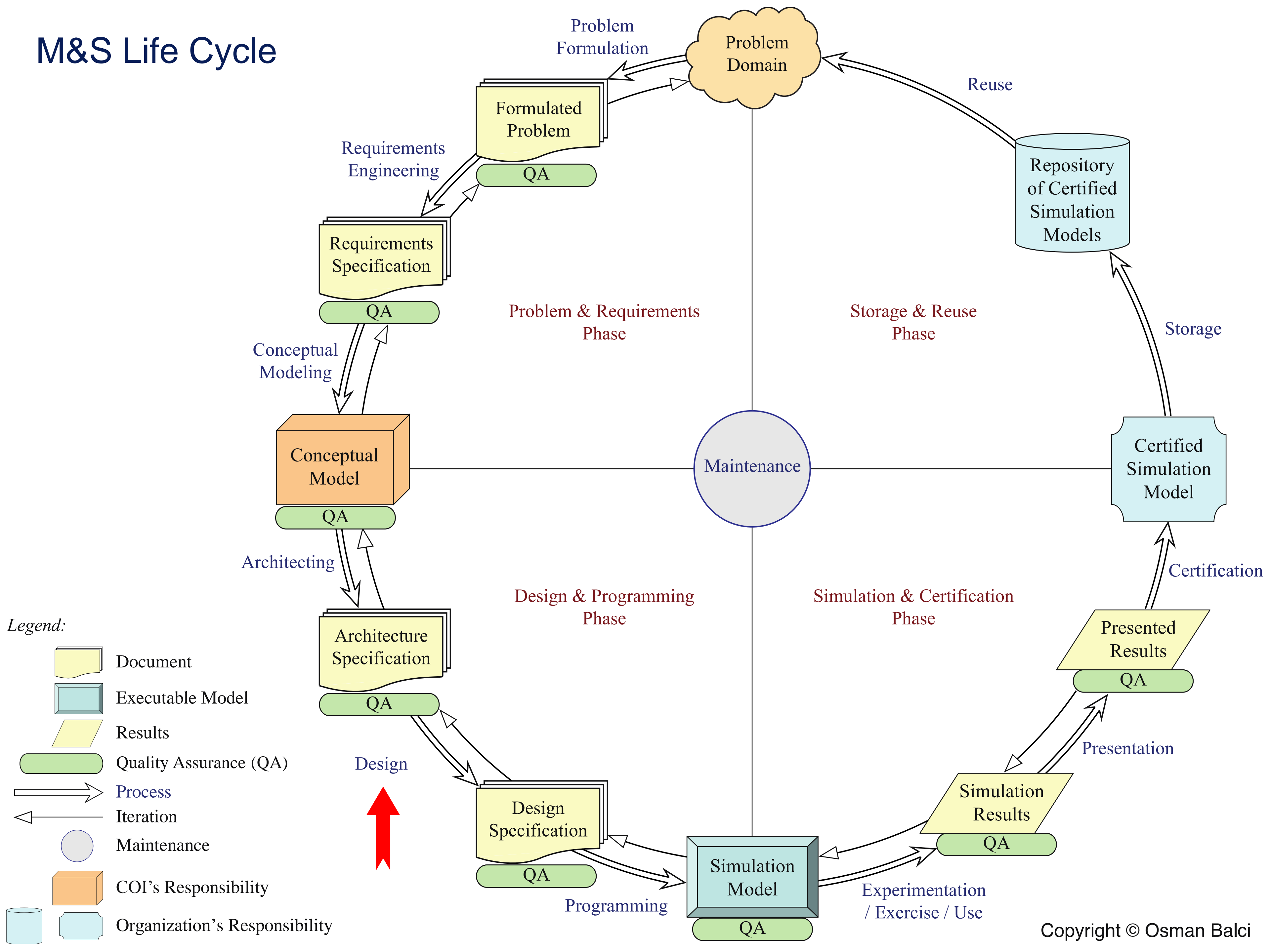
Design of an M&S Application

OSMAN BALCI
Professor

Department of Computer Science
Virginia Polytechnic Institute and State University (Virginia Tech)
Blacksburg, VA 24061, USA

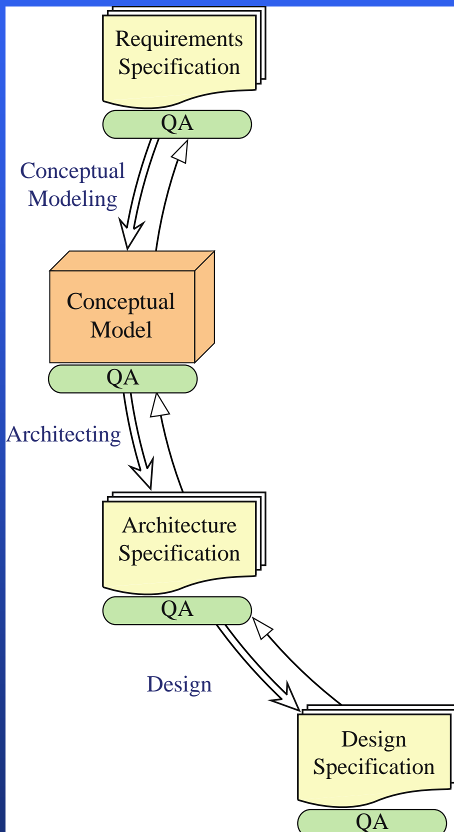
https://manta.cs.vt.edu/balci

Copyright © Osman Balci


M&S Life Cycle
Problem Formulation
Problem Domain
Reuse
Formulated Problem
QA
Requirements Engineering
Requirements Specification
QA
Repository of Certified Simulation Models
Problem & Requirements Phase
Storage & Reuse Phase
Storage
Conceptual Modeling
Conceptual Model
QA
Maintenance
Certified Simulation Model
Architecting
Certification
Design & Programming Phase
Simulation & Certification Phase
Presented Results
QA
Legend:
Document
Architecture Specification
QA
Executable Model
Results
Quality Assurance (QA)
Design
Presentation
Process
Iteration
Simulation Results
QA
Design Specification
QA
Maintenance
Simulation Model
QA
COI's Responsibility
Programming
Experimentation / Exercise / Use
Organization's Responsibility
Copyright © Osman Balci

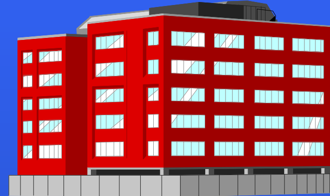
Design

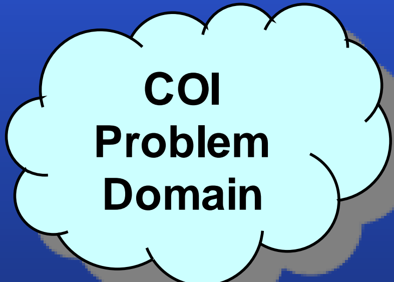
- The Design process deals with the instantiation (creation) of a design of the M&S application from the Architecture Specification.
- The Design process takes the M&S Requirements Specification, Conceptual Model, and Architecture Specification as input and produces a design specification of the M&S application as the output work product.
- M&S Application Design QA integrates the assessments of
  a) quality of the M&S application design specification work product,
  b) quality of the design process,
  c) quality of the people employed in the design, &
  d) project characteristics related to the life cycle stage for design.



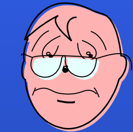
Reuse-based M&S Application Design Using a Conceptual Model (CM)

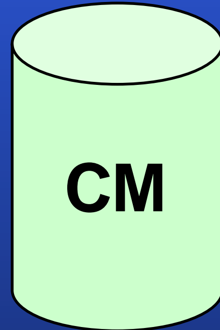
COI Lead Organization / Sponsor





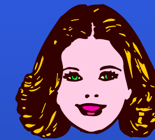
SMEs



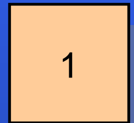


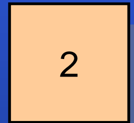
Reuse

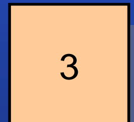
M&S Application Designers

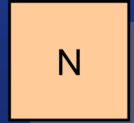


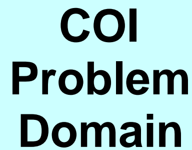
M&S Application Designs







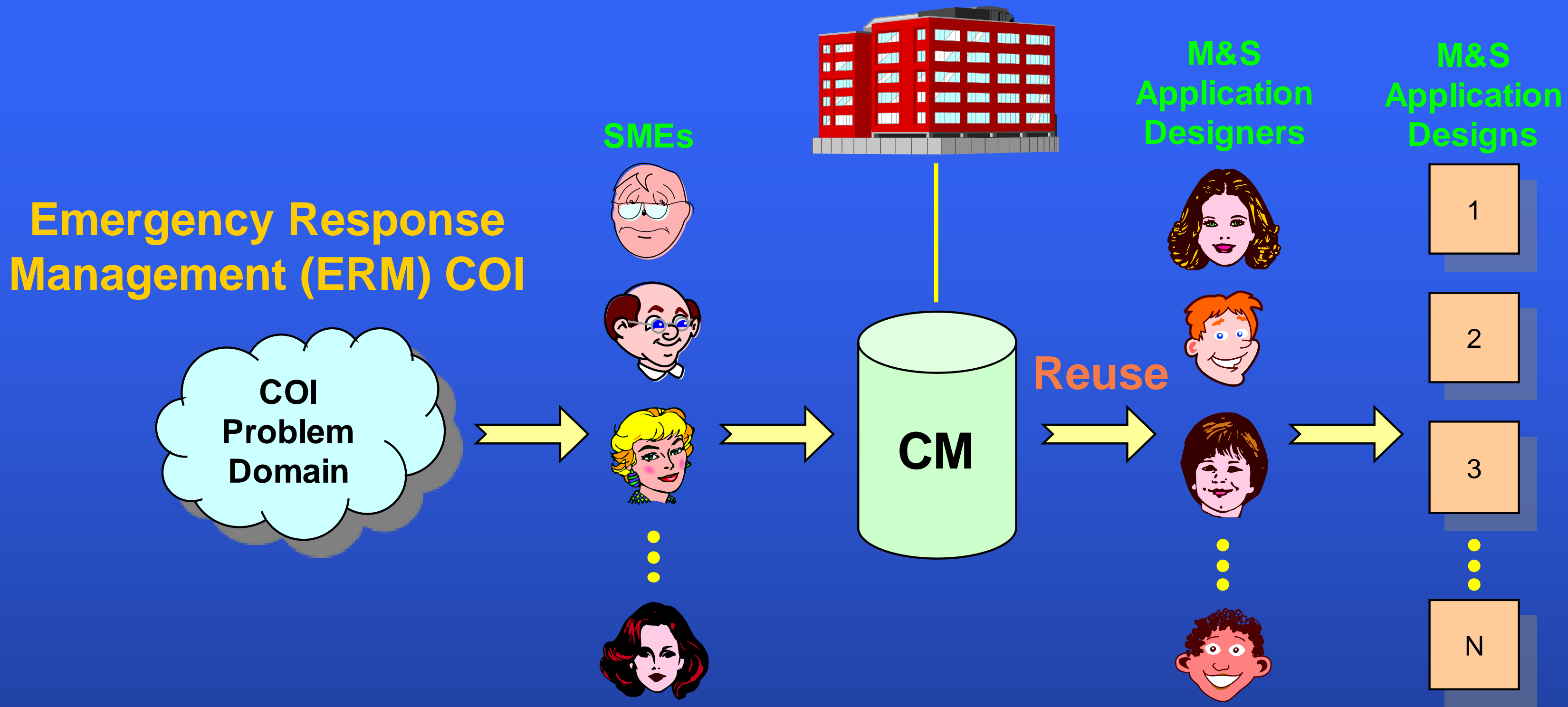




An Example

- COI: Emergency Response Management (ERM)
- Problem Domain:
  - Based on the U.S. National Response Plan, states and cities in the U.S. are expected to have an ERM plan.
    - Problem 1: Asses the operational effectiveness of a given ERM plan.
  - Under a given ERM plan, first responders, decision makers, authorities involved, and citizens are expected to be trained.
    - Problem 2: Conduct the training under the ERM plan.
- M&S is the only effective approach for solving Problems 1 and 2 above.

An Example



- For hundreds of cities and states, there exist hundreds of ERM plans, requiring the development of hundreds of M&S applications for:
  - Assessing the operational effectiveness of an ERM plan.
  - Training first responders, decision makers, authorities involved, and citizens under a given ERM plan.

M&S Application Design Depends on the M&S Area

M&S Areas	M&S Application Design Approaches
A. Based on Model Representation:	**A. Model Representation:**
1. Discrete M&S	1. Logic
2. Continuous M&S	2. Differential equations
3. Monte Carlo M&S	3. Statistical random sampling
4. System Dynamics M&S	4. Rate equations
5. Gaming-based M&S	5. Logic
6. Agent-based M&S	6. Knowledge, “intelligence”
7. AI-based M&S	7. Knowledge, “intelligence”
8. VR-based M&S	8. Computer generated visualization
B. Based on Model Execution:	**B. Model Execution:**
9. Distributed / Parallel M&S	9. Distributed processing / computing
10. Web-based M&S	10. Java EE, Microsoft .NET
C. Based on Model Composition:	**C. Model Composition:**
11. Live Exercises	11. Synthetic environments
12. Live Experimentations	12. Synthetic environments
13. Live Demonstrations	13. Synthetic environments
14. Live Trials	14. Synthetic environments
D. Based on What is in the Loop:	**D. What is in the Loop:**
15. Hardware-in-the-loop M&S	15. Hardware + Simulation
16. Human-in-the-loop M&S	16. Human + Simulation
17. Software-in-the-loop M&S	17. Software + Simulation

M&S Application Design Depends on the M&S Area

M&S Areas

A. Based on Model Representation:
1. Discrete M&S
2. Continuous M&S
3. Monte Carlo M&S
4. System Dynamics M&S
5. Gaming-based M&S
6. Agent-based M&S
7. AI-based M&S
8. VR-based M&S

B. Based on Model Execution:

9. Distributed M&S
10. Web-based M&S

C. Based on Model Composition:

11. Live Exercises
12. Live Experimentations
13. Live Demonstrations
14. Live Trials

D. Based on What is in the Loop:

15. Hardware-in-the-loop M&S
16. Human-in-the-loop M&S
17. Software-in-the-loop M&S

Discrete M&S Design Approaches:

- **Object-Oriented Design**
- **Procedural Design**

  Procedural Programming

  Conceptual Frameworks:
  - Event Scheduling
  - Activity Scanning
  - Three-Phase Approach
  - Process Interaction

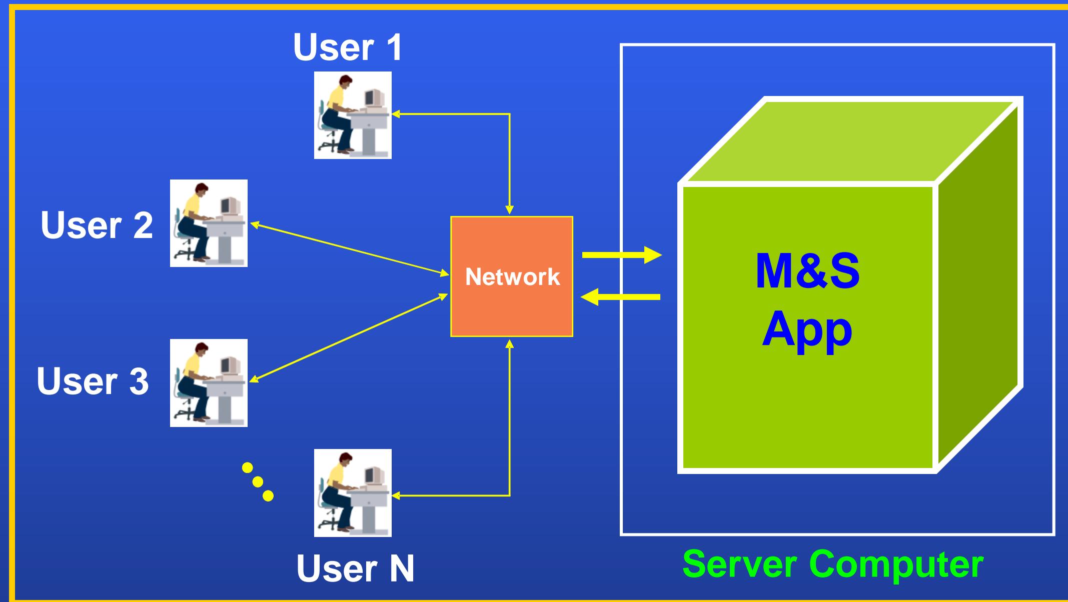
M&S Application Types

- M&S applications can be designed under two categories:
  - Stand-Alone M&S application
    - Intended to run on one computer for a single user
    - Commonly used for problem solving
    - Also used for training a single person (e.g., flight simulator)
  - Network-Centric M&S application
    - Intended to run on at least one server computer and used by many people over a network
    - Ideal for simulations for training people who are geographically dispersed (e.g., OneSAF, JSAF)
    - Also used for problem solving by many people over a network (e.g., BEST)

Network-Centric M&S Application Design

A design is instantiated (created) from the architecture specified for the M&S application.

Example:



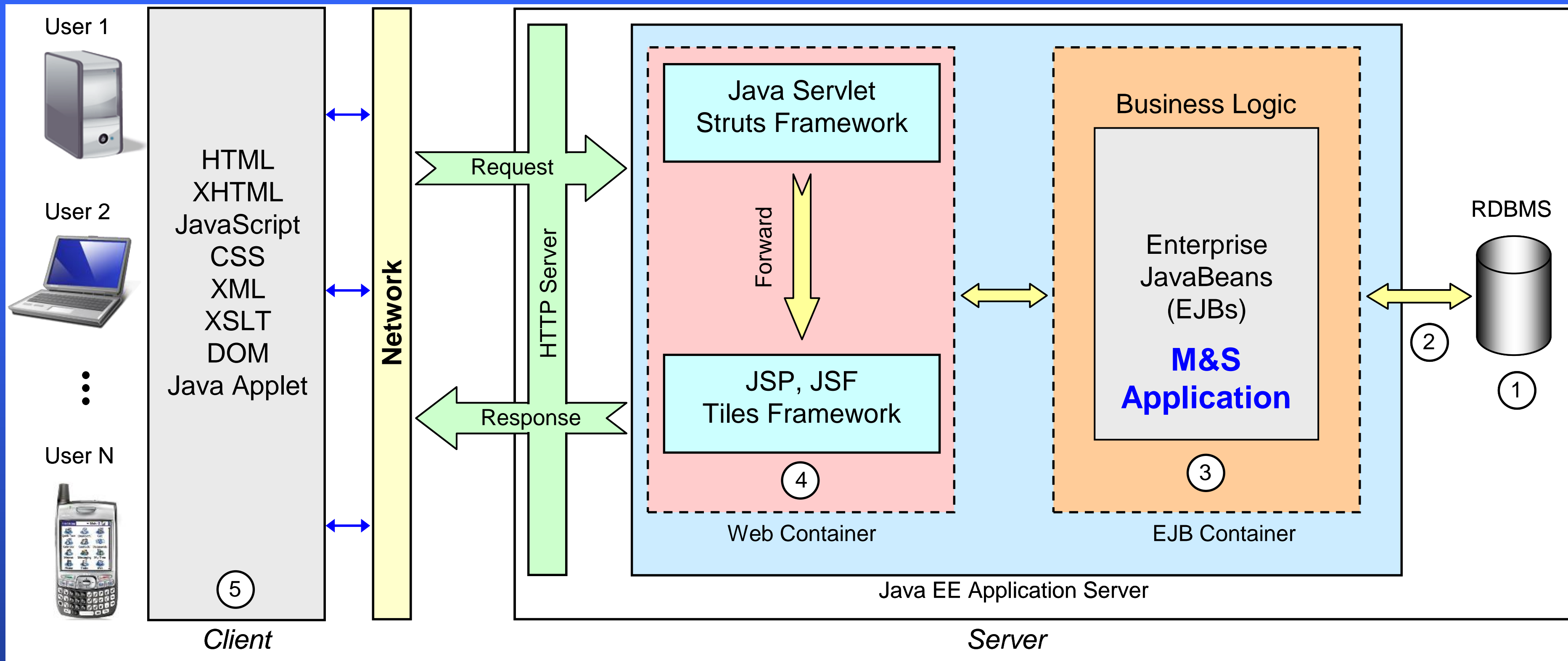
Instantiation

Instantiation

M&S Application Design
based on
the Java Platform
Java Enterprise Edition (EE)

M&S Application Design
based on
the Microsoft Platform
.NET Framework

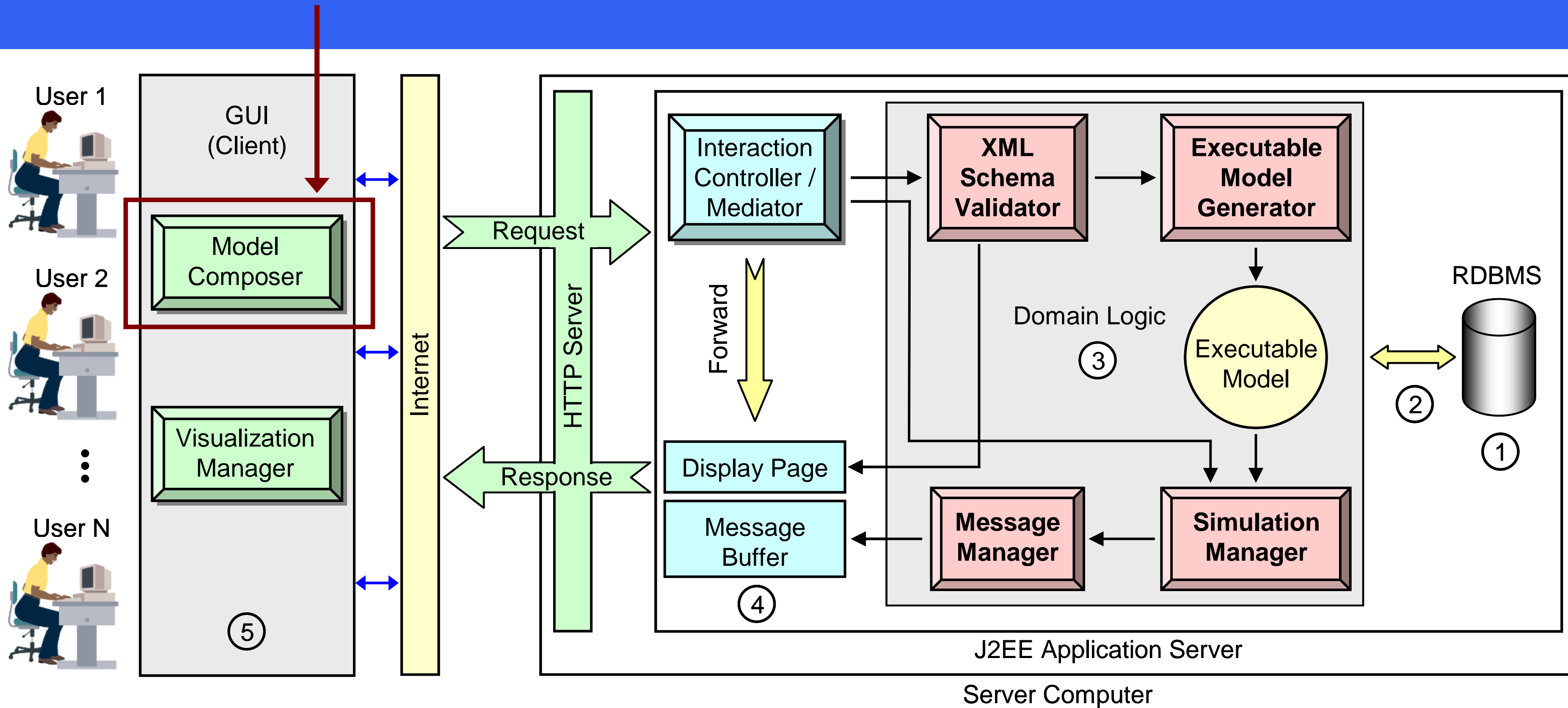
Network-Centric M&S Application Design using the Java Platform



- Java EE architecture layer 3 represents the M&S application core.
- EJBs can be designed and implemented as reusable model components.
- A simulation model can be designed by way of reuse of the EJB model components.

Network-Centric M&S Application Design: WebQS3

Design by way of reuse of components made available in a library.



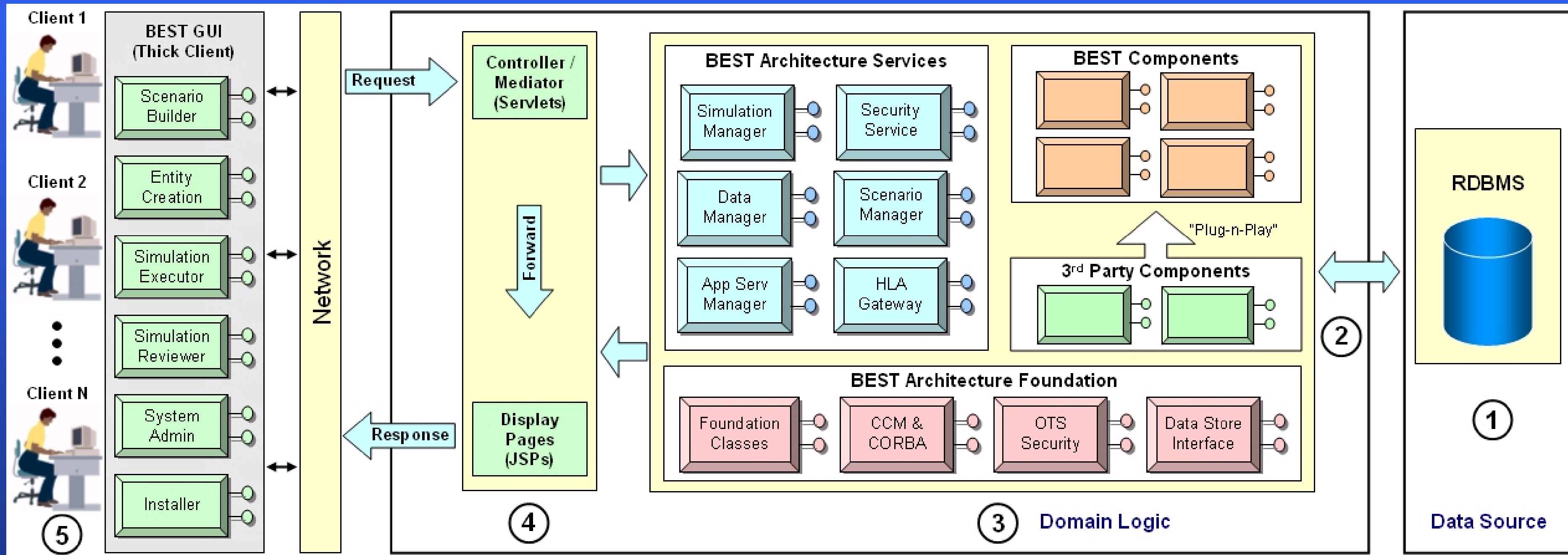
Play the WebQS3 Editor Video

Play the WebQS3 Simulator Video

David S. Myers and Osman Balci (2009), "A Web-Based Visual Simulation Architecture," *International Journal of Modelling and Simulation* 29, 2, 137-148.

Network-Centric M&S Application Design: BEST

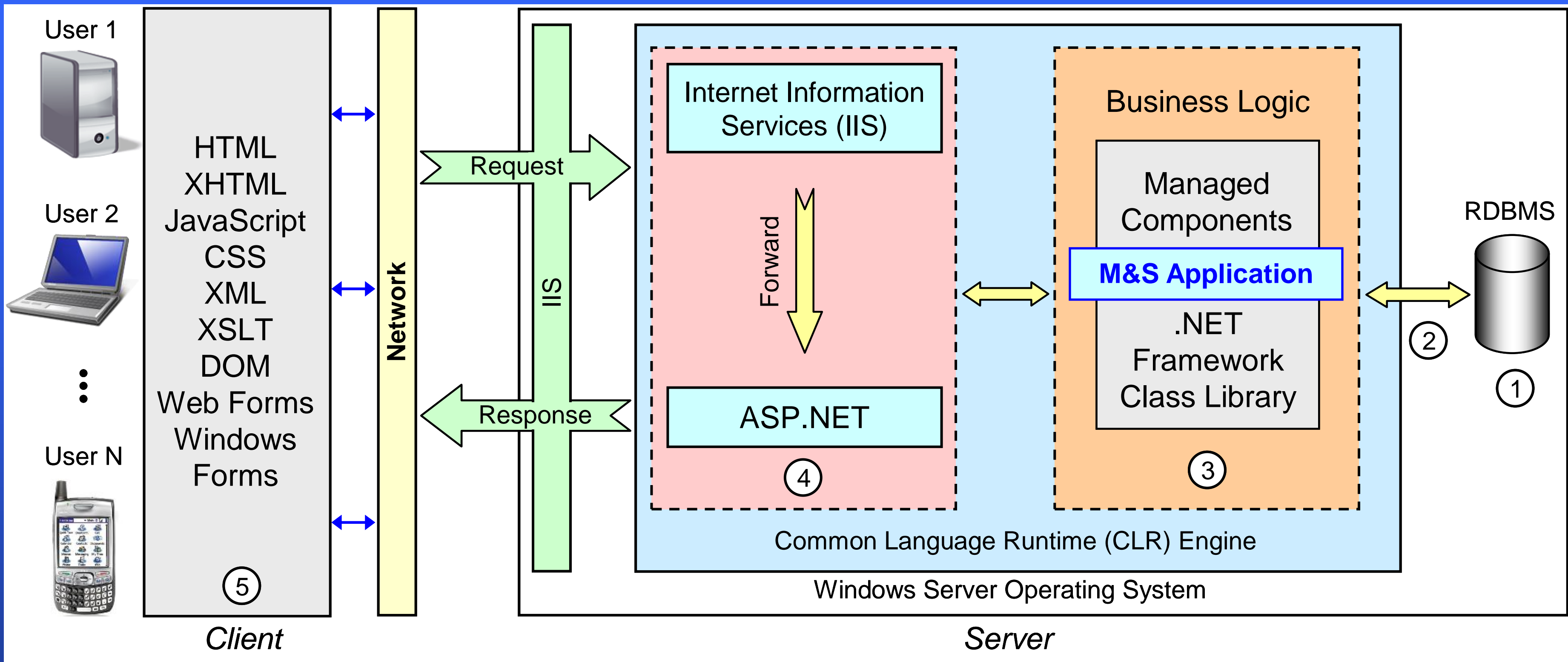
Model Design under the CORBA Distributed Objects Architecture



Java Platform Client-Server Architecture

BEST = Battlespace Environment and Signatures Toolkit
A simulation integrated development environment (IDE)

Network-Centric M&S Application Design using the .NET Framework

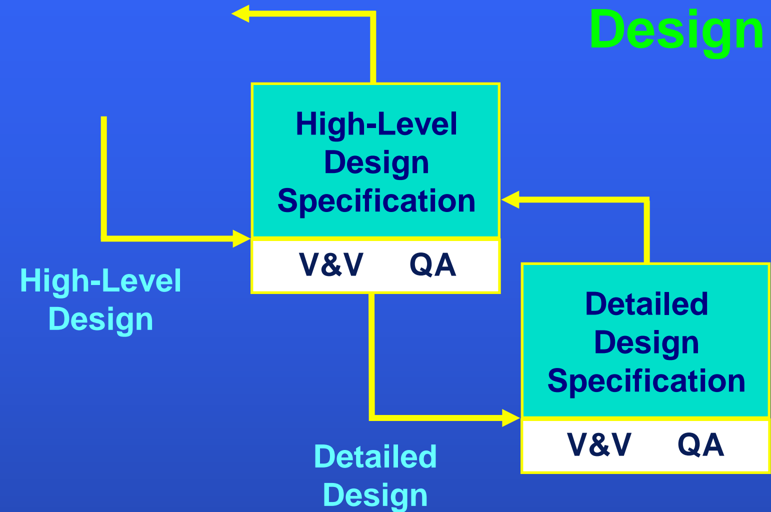


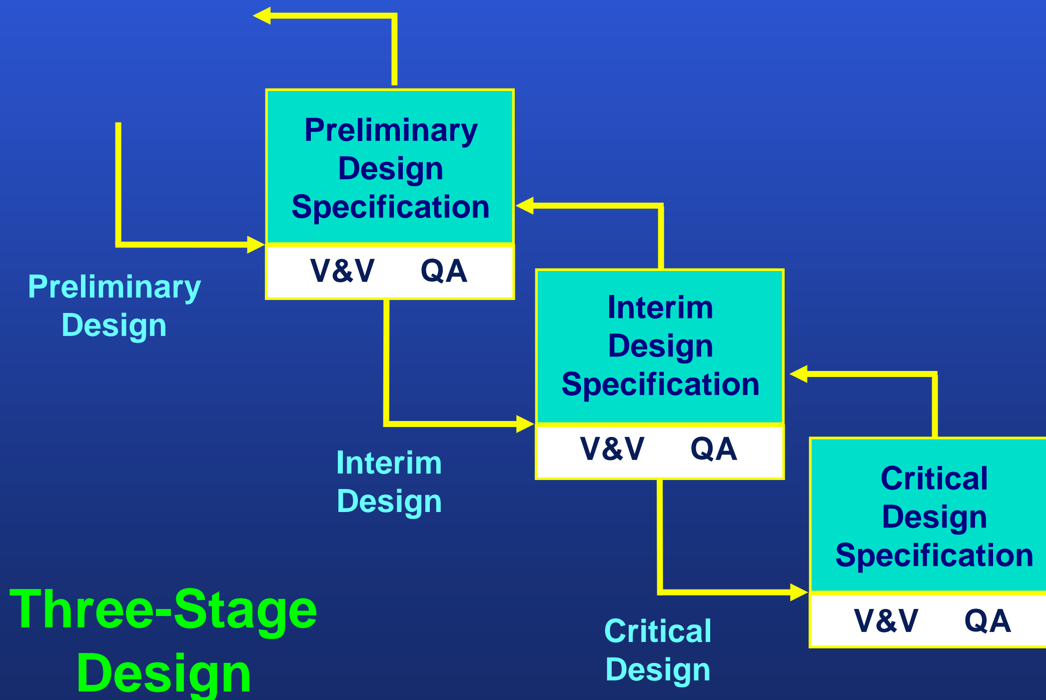
- Microsoft .NET Framework client-server architecture layer 3 represents the M&S application core.
- A simulation model can be designed by using the .NET Framework Class Library.

M&S Application Design Stages

Number of design stages is determined based on the M&S application complexity.

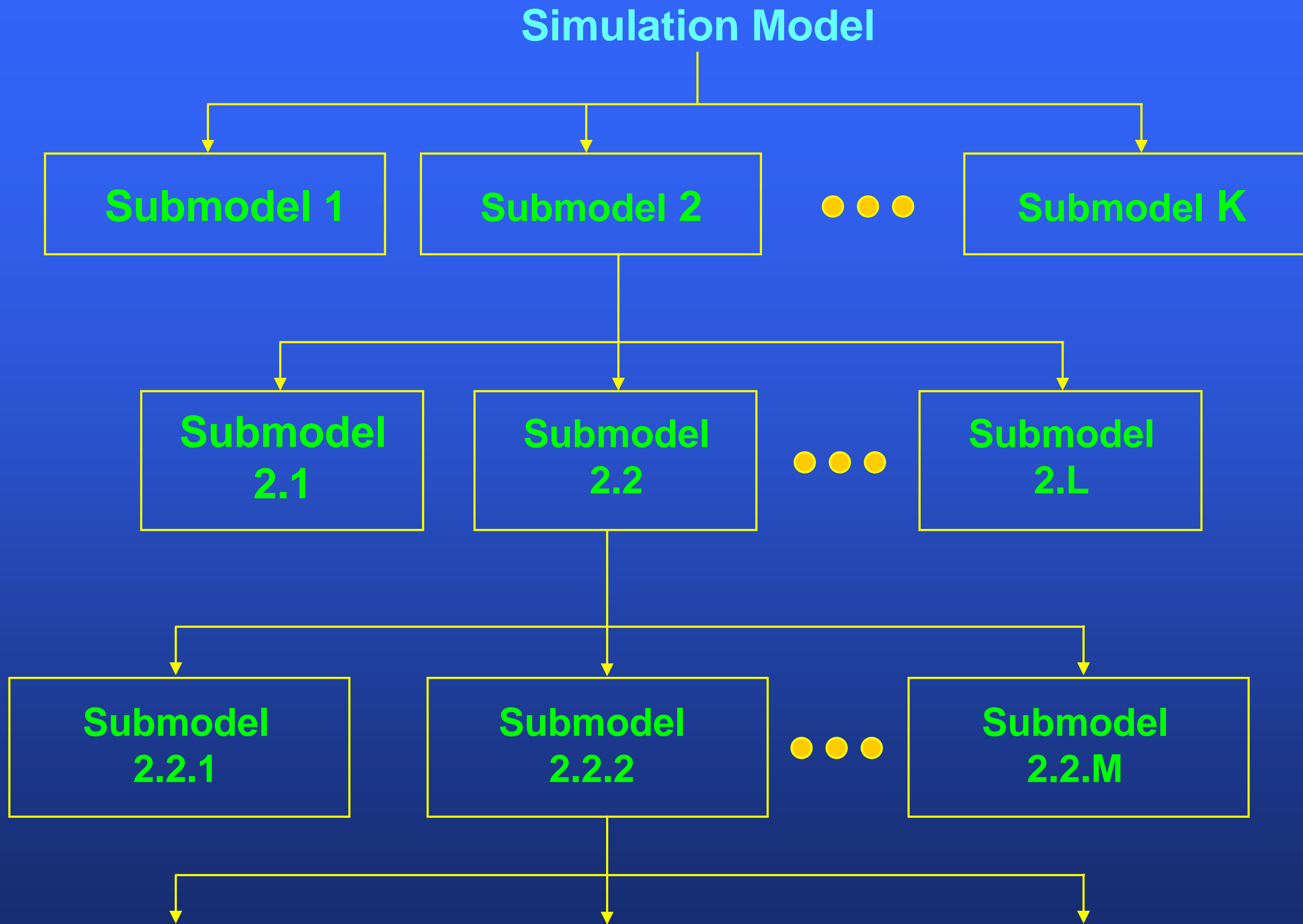
Two-Stage Design





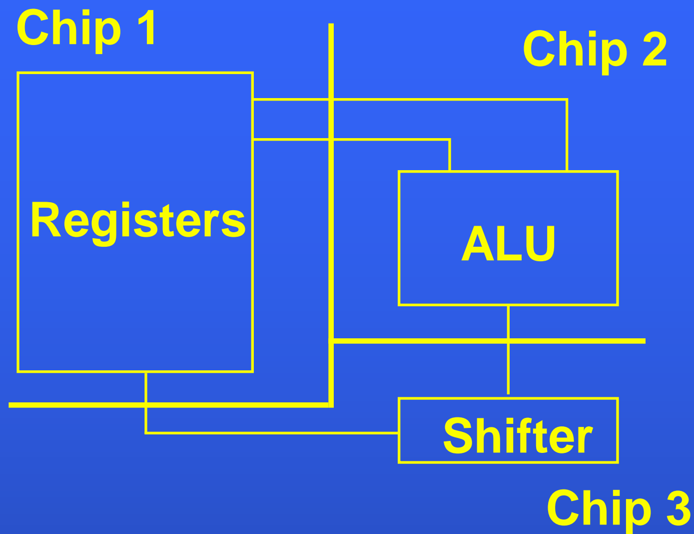
Three-Stage Design

Decomposition / Modularization



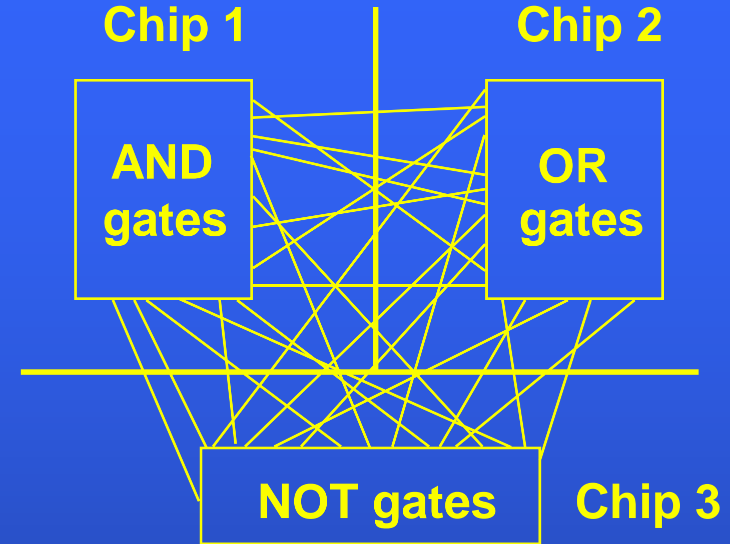
Decomposition / modularization is the key solution to reducing and managing complexity.

Design 1



- Functionally same as the other one
- Much easier to understand
- Much easier to modify
- Faults can be localized
- Much easier to extend or enhance
- Chips can be reused
- Maximal relationship within each chip and minimal relationship between chips

Design 2



Functionally same as the other one

Much harder to understand

Much harder to modify

Faults can not be localized

Much harder to extend or enhance

Chips can not be reused

Minimal relationship within each chip and maximal relationship between chips

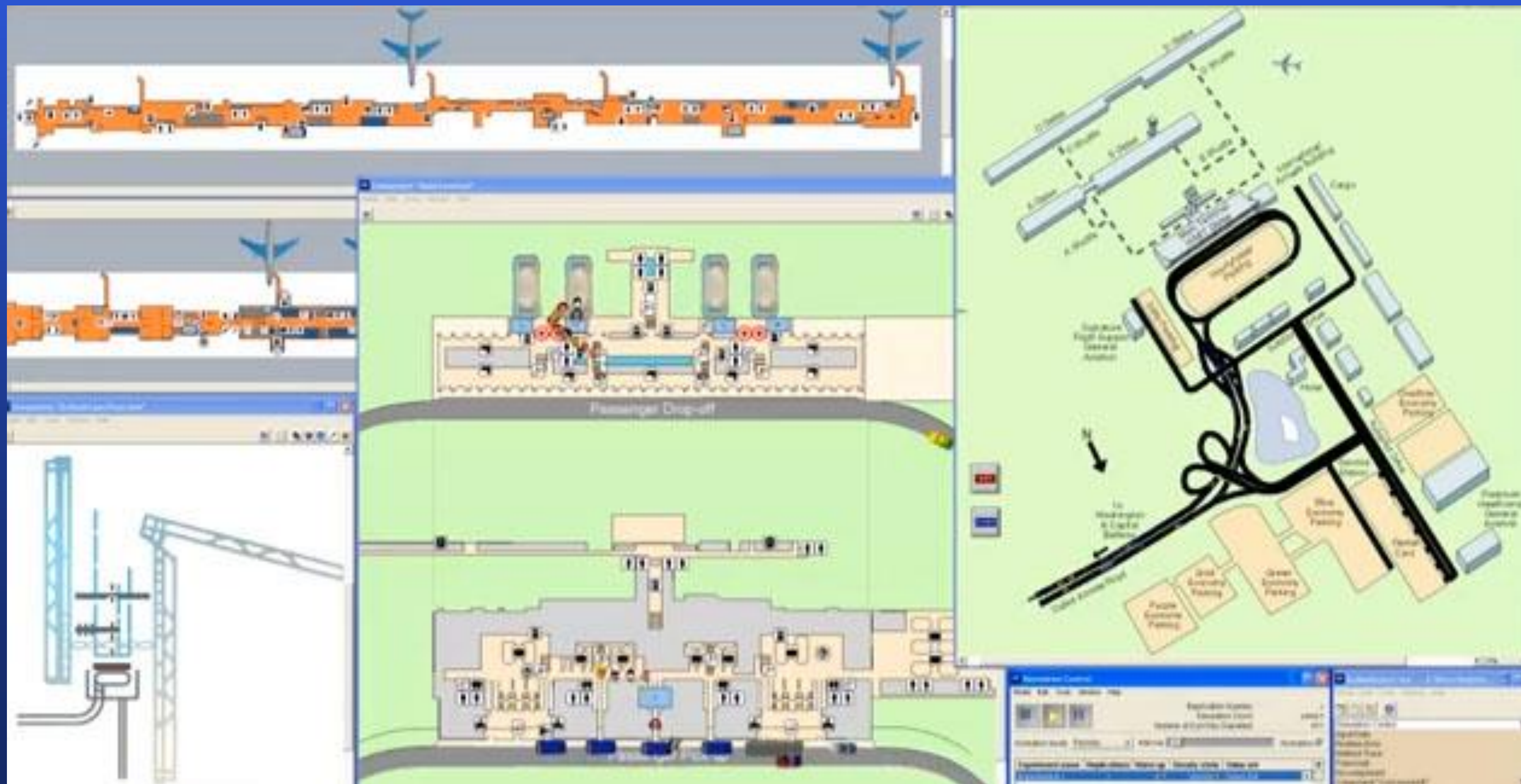
Modularity

- **Objective:** Decompose the simulation model into submodels in such a way as to reduce the cost of maintenance and increase reusability.
- Submodel **COHESION** is the degree of interaction among the elements included within the submodel.
- Submodel **COUPLING** is the degree of dependency between submodels.
- The **maintenance effort is reduced** and **reusability is increased** when there is maximal interaction within each submodel (cohesion) and minimal dependencies (coupling) between submodels.
- **Design Principle:** Design a submodel or model component (class) in such a way that the submodel has
  - the **highest possible cohesion** and
  - the **lowest possible coupling**.

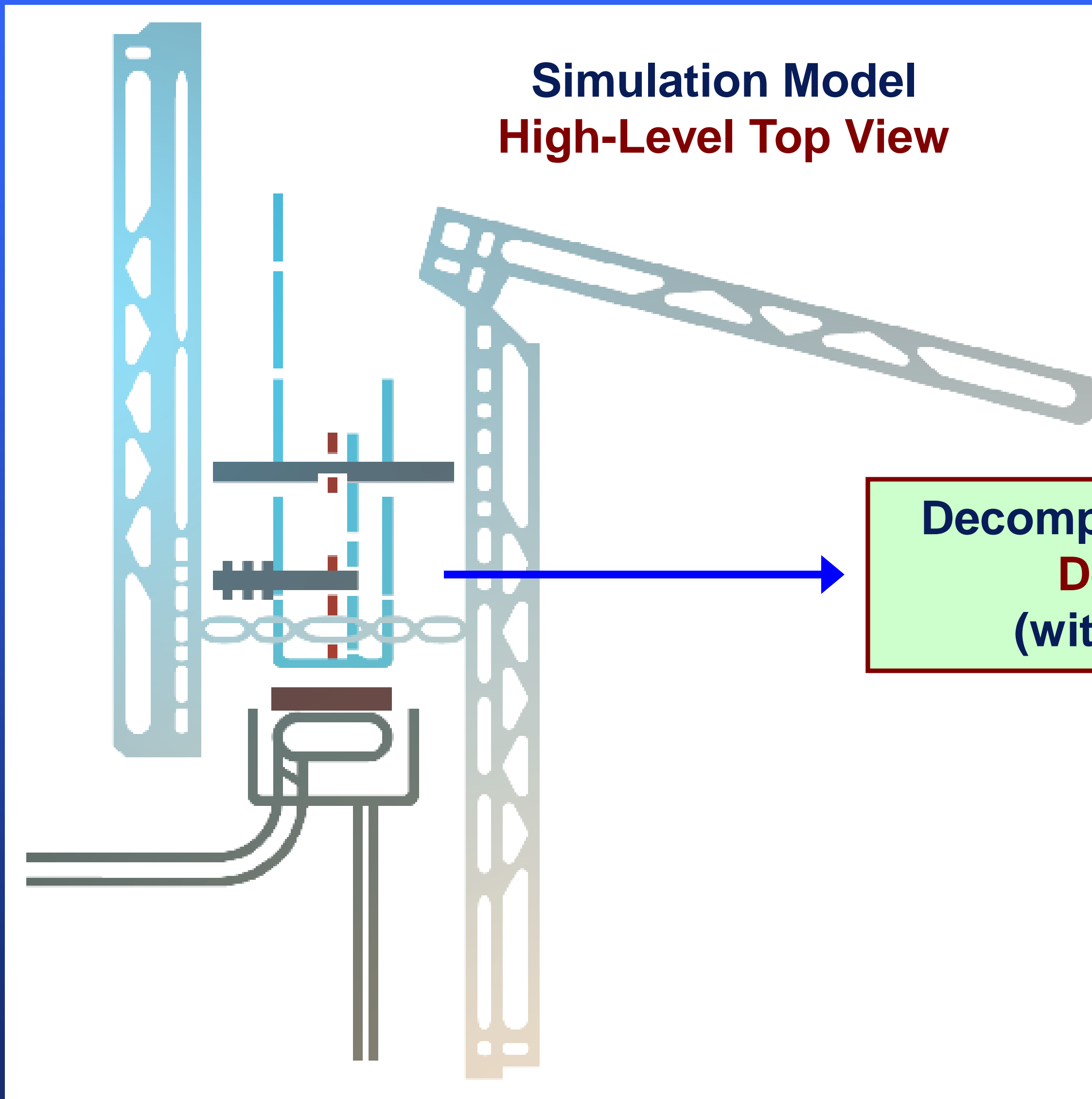
Example Discrete M&S Application Design Using the Object-Oriented Paradigm

Object-Oriented Simulation Model Design of Washington Dulles International Airport

Click to play the Video of Object-Oriented Visual Simulation of Dulles Airport



Example Decomposition / Modularization: Dulles Airport Simulation



Decomposed into a Submodel
Detailed Top View
(without the runways)

Washington Dulles International Airport, Dulles, Virginia

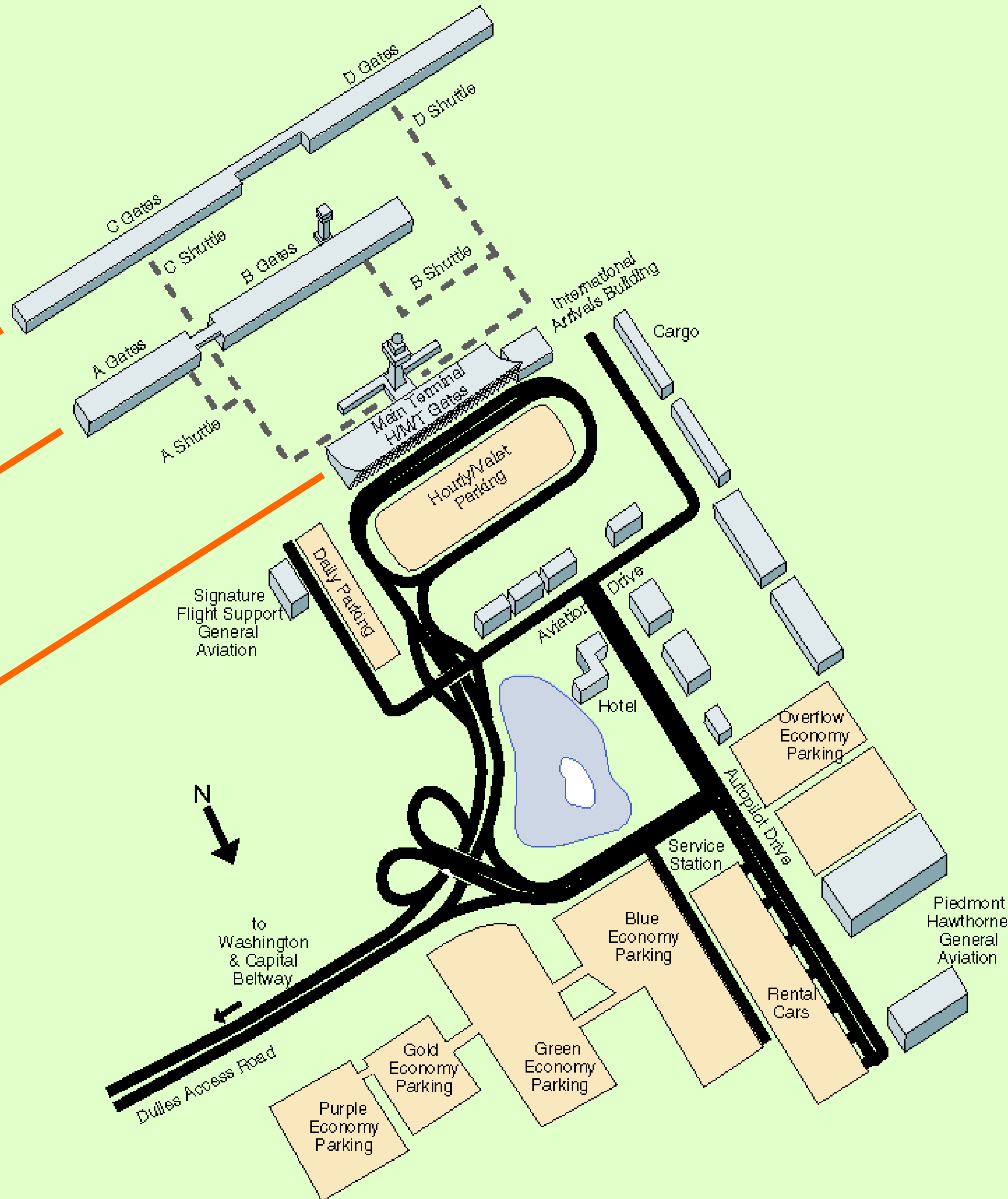
Submodel
Detailed Top View
(without the runways)

Decomposed into:

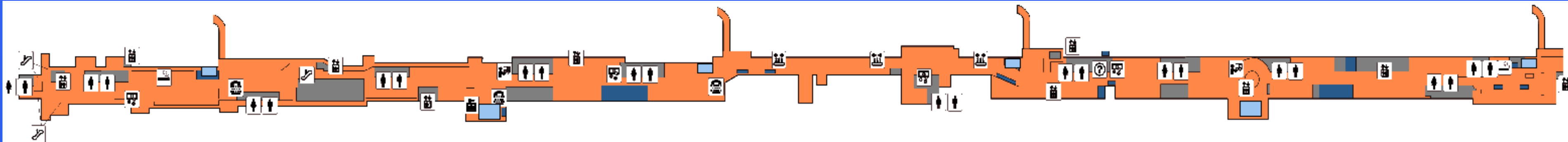
Concourse CD submodel

Concourse AB submodel

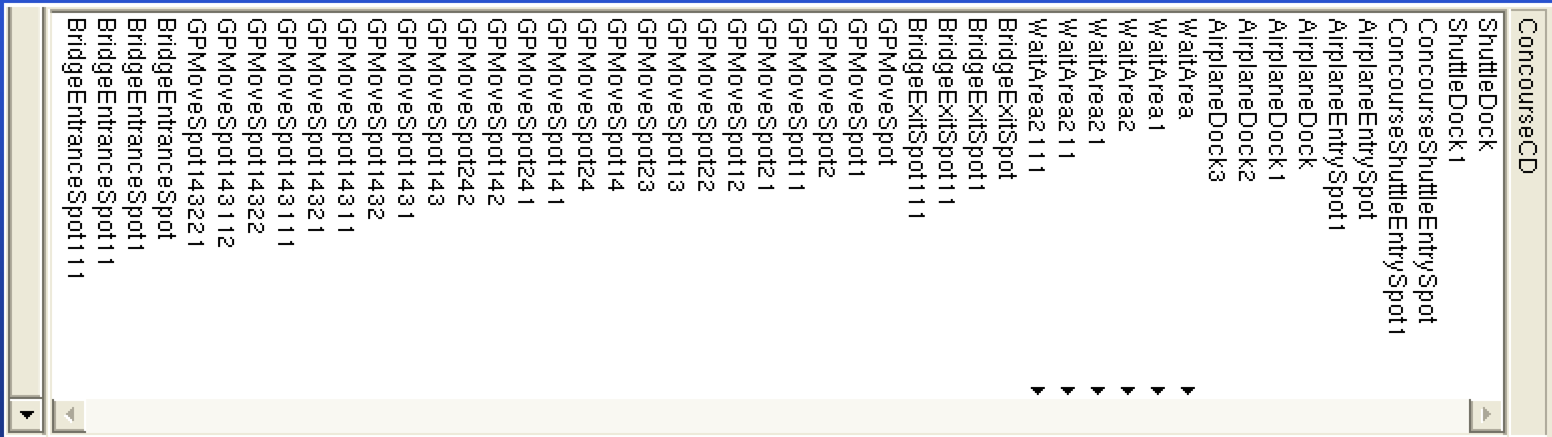
Main Terminal submodel



Submodel Concourse CD

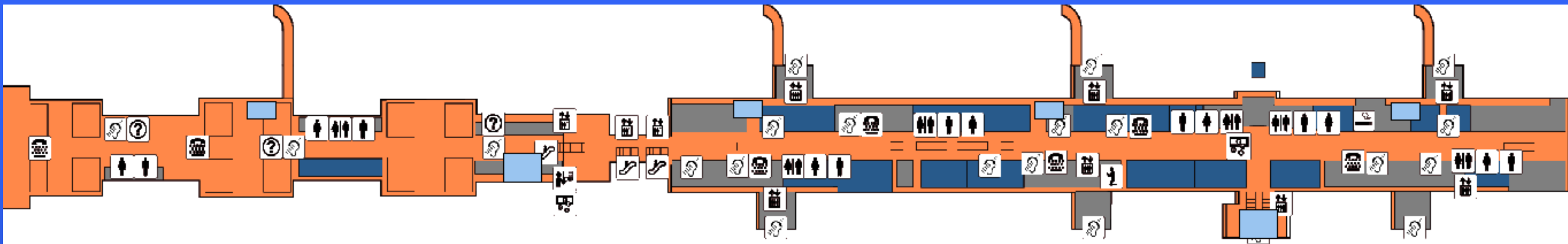


Decomposed into:



Arrows in the submodel browser indicate further decomposition

Submodel Concourse AB

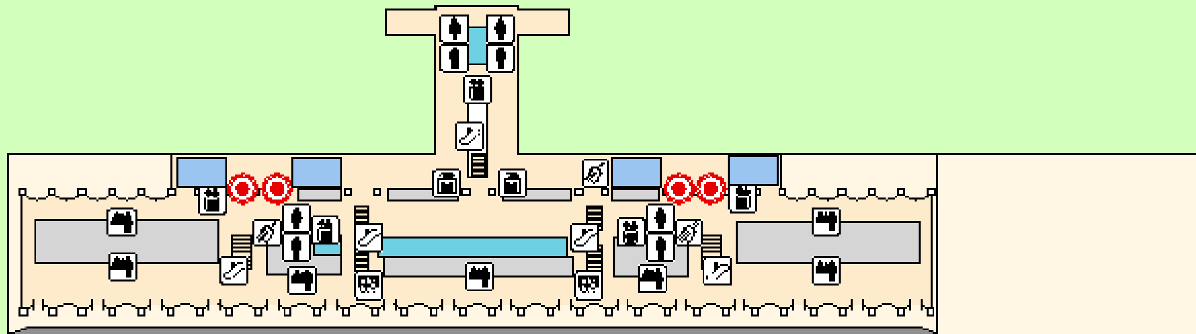


Decomposed into:

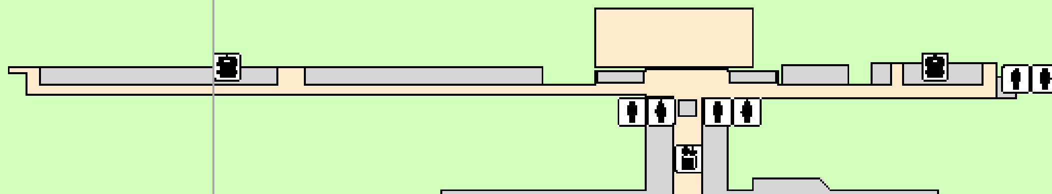
ConcourseAB
shuttleDockA
shuttleDockB
ShuttleEntrySpot
ShuttleEntrySpot1
AirplaneEntrySpot
AirplaneDock
AirplaneDock1
AirplaneDock2
AirplaneDock3
AirplaneEntrySpot1
WaitArea
WaitArea1
BridgeExitSpot
BridgeExitSpot1
BridgeExitSpot11
BridgeExitSpot12
WaitArea2
WaitArea21
WaitArea211
WaitArea212
GPMoveSpot
GPMoveSpot1
GPMoveSpot2
GPMoveSpot3
GPMoveSpot4
GPMoveSpot5
GPMoveSpot41
GPMoveSpot411
GPMoveSpot4111
GPMoveSpot4112
GPMoveSpot41111
GPMoveSpot413
GPMoveSpot41112
GPMoveSpot4131
GPMoveSpot41121
GPMoveSpot41311
GPMoveSpot41132
GPMoveSpot31
GPMoveSpot412
GPMoveSpot41321
BridgeEntranceSpot
BridgeEntranceSpot1
BridgeEntranceSpot11
BridgeEntranceSpot12

Arrows in the submodel browser indicate further decomposition

Submodel Main Terminal



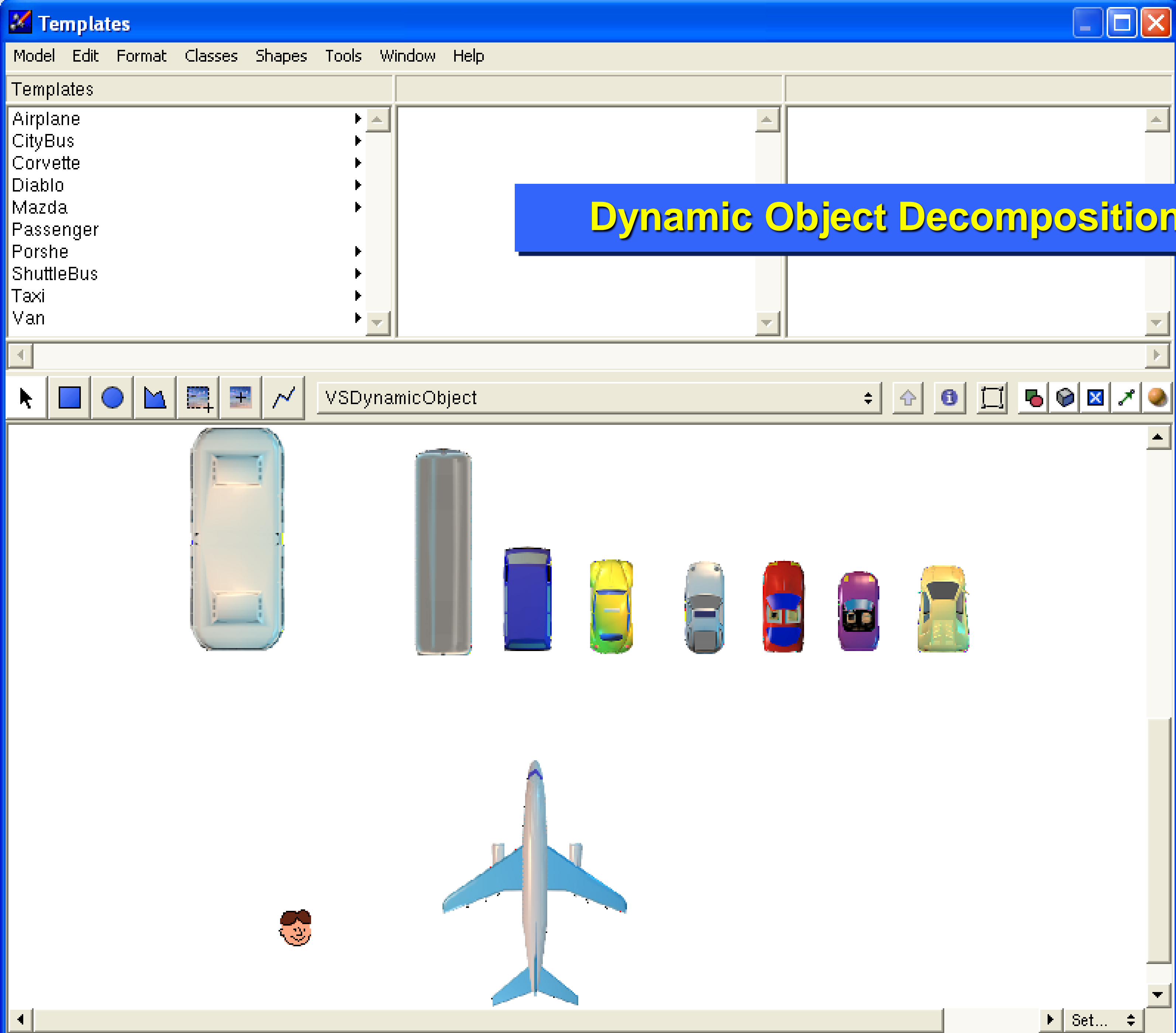
Passenger Drop-off



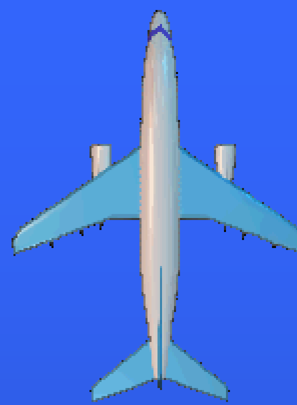
Passenger Pick-up

MainTerminal

DropOffBeginning
PickUpBeginning
DropOffSpot
DropOffSpot1
DropOffSpot2
DropOffSpot3
DropOffSpot4
DropOffSpot5
DropOffSpot6
PickUpSpot
PickUpSpot1
PickUpSpot2
PickUpSpot3
PickUpSpot4
PickUpSpot5
PickUpSpot6
GPMoveSpot
GPMoveSpot1
GPMoveSpot3
GPMoveSpot5
GPMoveSpot6
GPMoveSpot8
GPMoveSpot12
GPMoveSpot19
ShuttleDock
ShuttleDock1
ShuttleDock2
ShuttleDock3
GPMoveSpot2
ShuttleEntrySpot
ShuttleEntrySpot1
ShuttleEntrySpot2
ShuttleEntrySpot3
GPMoveSpot4
GPMoveSpot7
GPMoveSpot9
GPMoveSpot10
GPMoveSpot11
GPMoveSpot13
GPMoveSpot14
GPMoveSpot15
GPMoveSpot16
GPMoveSpot17
GPMoveSpot171
GPMoveSpot1711
GPMoveSpot1712
GPMoveSpot1713
GPMoveSpot17121
GPMoveSpot17122
GPMoveSpot171211
GPMoveSpot1712111
GPMoveSpot1712112
GPMoveSpot1712113
GPMoveSpot1712114
GPMoveSpot17121131
Component
Component1
Component2
Component3
GPMoveSpot17121121
GPMoveSpot171211211
GPMoveSpot1712112111


Templates
Model Edit Format Classes Shapes Tools Window Help
Templates
Airplane
CityBus
Corvette
Diablo
Mazda
Passenger
Porshe
ShuttleBus
Taxi
Van
VSDynamicObject
Set...
Dynamic Object Decomposition

Dynamic Object Airplane Decomposition



Templates - Airplane

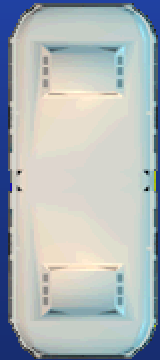
Model Edit Format Classes Shapes Tools Window Help

Templates	Airplane
Airplane	Move Spot3
CityBus	Move Spot4
Corvette	Move Spot5
Diablo	Move Spot6
Mazda	Move Spot7
Passenger	Move Spot8
Porshe	Move Spot9
ShuttleBus	Seat - 10A
Taxi	Seat - 10B
Van	Seat - 10C
	Seat - 10D
	Seat - 10E
	Seat - 11A

VSDynamicObject

40%

Dynamic Object Shuttle Bus Decomposition



Templates - ShuttleBus

Model Edit Format Classes Shapes Tools Window Help

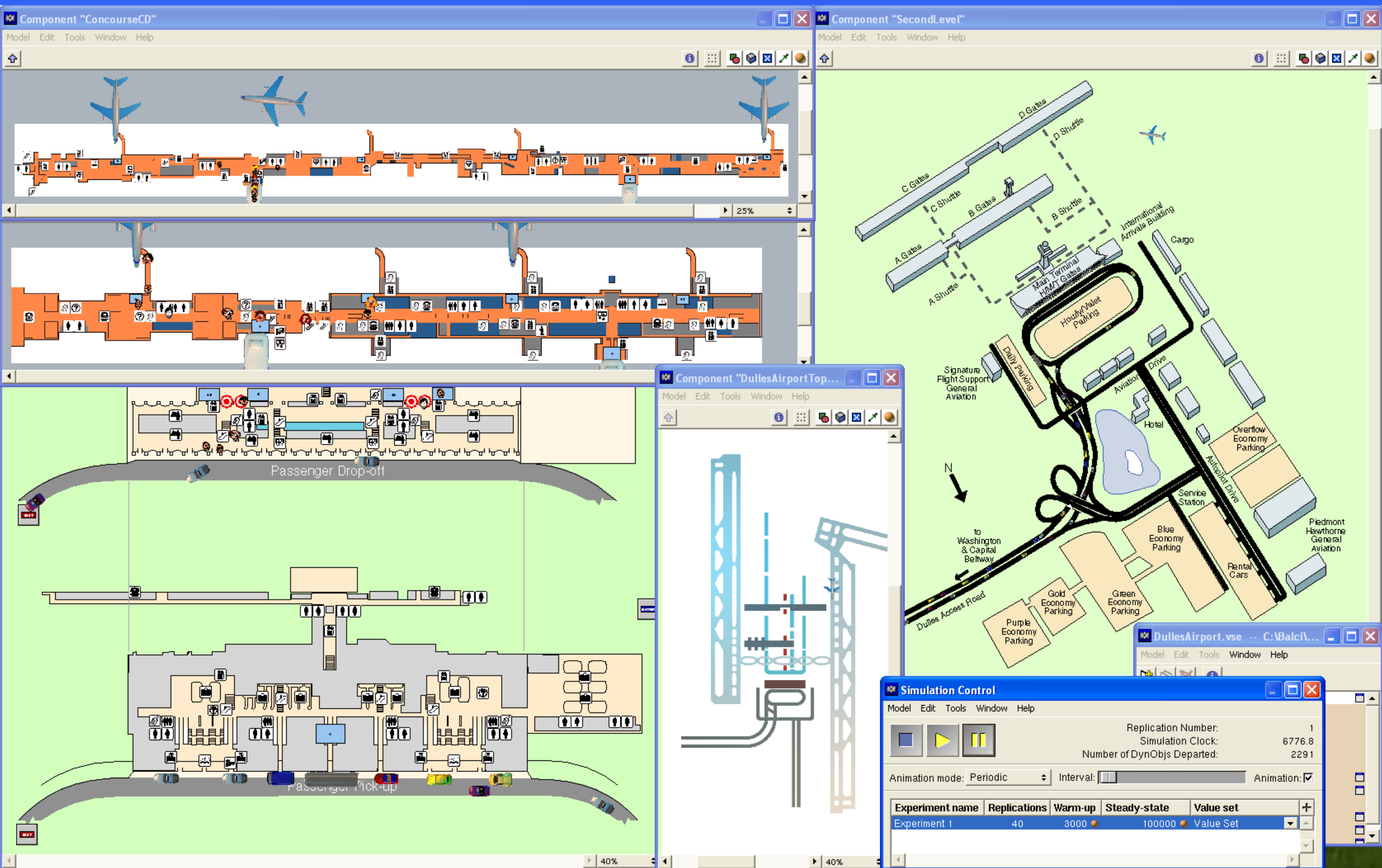
Templates	ShuttleBus
Airplane	GPDoor
CityBus	GPMoveSpot
Corvette	GPMoveSpot1
Diablo	GPMoveSpot5
Mazda	GPMoveSpot6
Passenger	Seat 1
Porshe	Seat 10
ShuttleBus	Seat 11
Taxi	Seat 12
Van	Seat 13
	Seat 14
	Seat 15

VSDynamicObject

40%

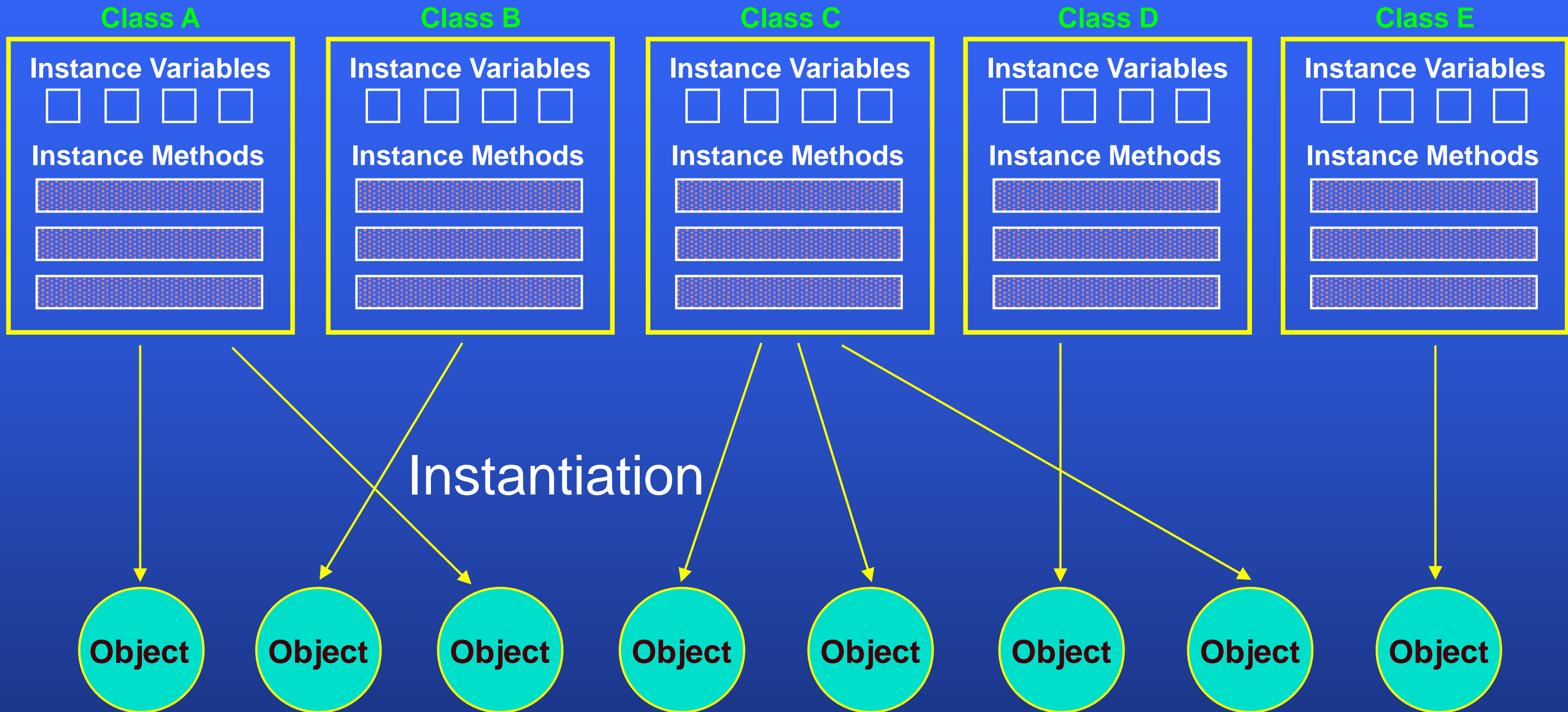
Dulles Airport Visual Simulation

Click to play the Video of Object-Oriented Visual Simulation of Dulles Airport



Discrete M&S Design using the Object-Oriented Paradigm

Classes are created at design time.



Some objects are created (instantiated) at design time and some at execution time.

Interactions among objects take place via message passing during execution.

Object-Oriented Simulation Model Design

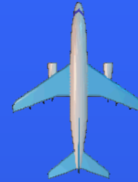
- Objects
- Classes
- Instantiation
- Variables (Attributes)
- Methods (Services, Operations)
- Inheritance
- Message Passing
- Encapsulation
- Polymorphism
- Dynamic Binding

Objects

Examples:

Passenger

Car

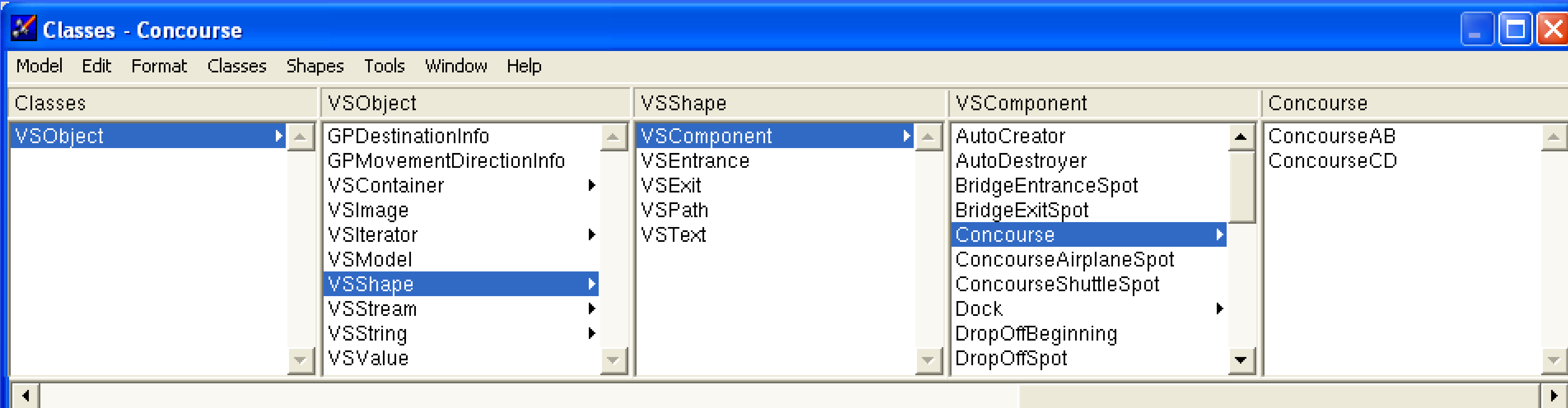

Airplane

Shuttle Bus

- Runway
- Dock
- Main Terminal
- Concourse
- Wait Area

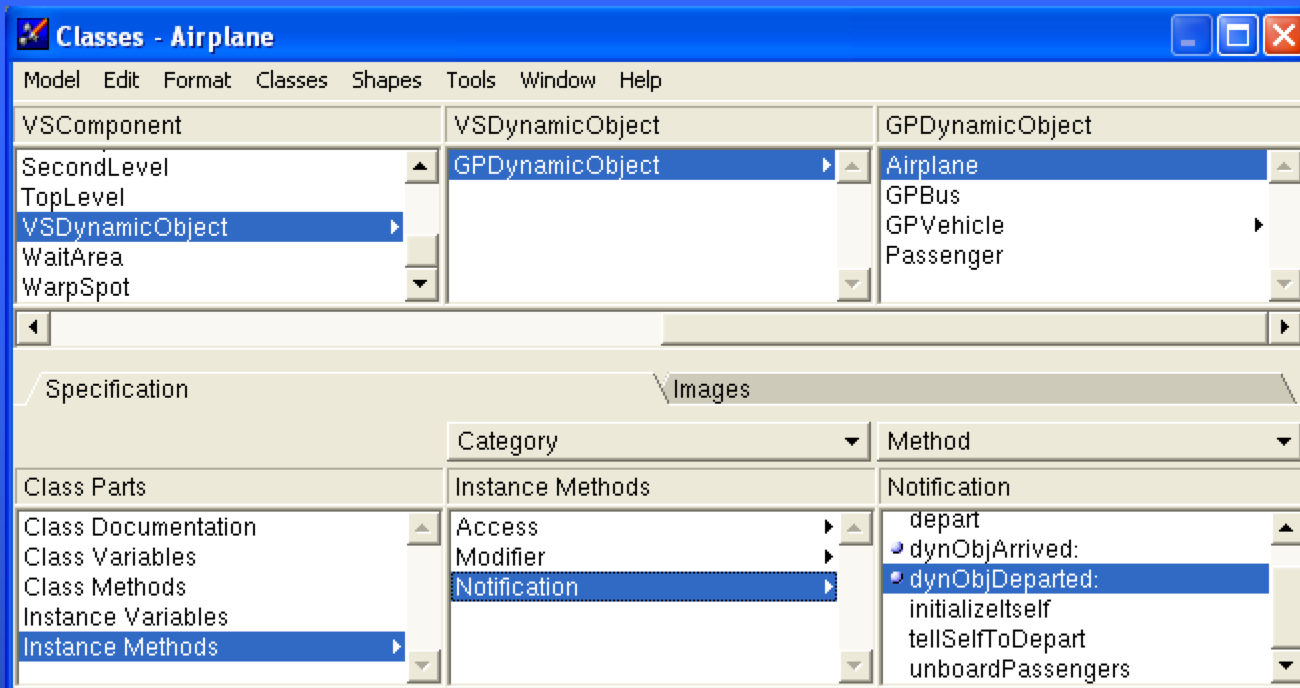
- **An object is an entity which has a state and a defined set of operations which operate on that state.**
- **The state is represented as a set of object attributes.**
- **The operations associated with the object provide services to other objects (clients) which request these services when needed.**
- **Objects are created according to some object class definition.**
- **An object class definition serves as a template for objects. It includes declarations of all the attributes and services which should be associated with an object of that class.**

Classes



- A **class** is a grouping or categorization of objects with the same characteristics, services, and behaviors.
- A class may be extended into other classes. Extending a class is called **subclassing** and an extended class is called a **subclass**.
- A **subclass** inherits all the characteristics, services, and behaviors of the parent class.
- It customizes what it inherits and/or provides more characteristics, services, or behaviors.
- The parent of a subclass is called the **superclass**.
- The top class having no superclass is called the **root class**.

Instantiation



- Creation of an object belonging to a class is called instantiation.
- The new object inherits all characteristics (instance variables) and behaviors (instance methods) specified in the class from which it is instantiated.
- Instance variables of a class are created for each object instantiated as a member of that class.
- Instance methods are inherited, but no method code is replicated.

Variables (Attributes)

- Characteristics (attributes) of an object are represented by variables. Typically three kinds of variables exist: class variables, instance variables, and local variables.
- Class Variables are attributes of a class and are declared in the class for use by the methods of that class and its subclasses and by the objects instantiated from that class and its subclasses.
- Instance Variables, declared in a class, are used by the instance methods of that class and are created for each object instantiated as belonging to that class or any of its subclasses, i.e., for each instance of a class, and hence the designator "instance".
- Local Variables are declared within a method for use only during the execution of that method. On completion of a method, all local variable values are lost.
- Example variables can have the following data types:
  - boolean, character, class reference, object reference, integer number, real number, enumeration.

Example Instance Variables

Classes - Airplane

Model Edit Format Classes Shapes Tools Window Help

VSComponent	VSDynamicObject	GPDynamicObject
RunwayEntrySpot	GPDynamicObject	Airplane
ScaleSpot		GPBus
SecondLevel		GPVehicle
TopLevel		Passenger
VSDynamicObject		
WaitArea		
WarpSpot		

Specification | Images

Category | Method

Class Parts

- Class Documentation
- Class Variables
- Class Methods
- Instance Variables
- Instance Methods

Documentation | Logic | edited

```
real            departureTime;
VSList ref      seatList;
integer         nextSeatNumber;
GPMoveSpot ref  door;
VSString ref    gateName;
VSString ref    oldGateName;
boolean         initializing;
boolean         arriving;
```

Methods (Services)

- Services provided and behaviors exhibited by an object are specified in methods. Two types of methods exist: class methods and instance methods.
- Class Methods are used to provide services specific to a class. For example, the method “new” which creates an instance of a class.
- Instance Methods, given in a class, are used to specify the services provided and behavior exhibited for each object instantiated from that class.
- Each instance (i.e., instantiated object) created as belonging to a class provides the services and behavior specified in the instance methods of that class.
- The method code is specified only once in the class and is not replicated for each instantiation of an object from that class.

Example Instance Methods

Classes - Airplane

Model Edit Format Classes Shapes Tools Window Help

VSComponent	VSDynamicObject	GPDynamicObject
TopLevel	GPDynamicObject	Airplane
VSDynamicObject		GPBus
WaitArea		GPVehicle
WarpSpot		Passenger

Specification | Images

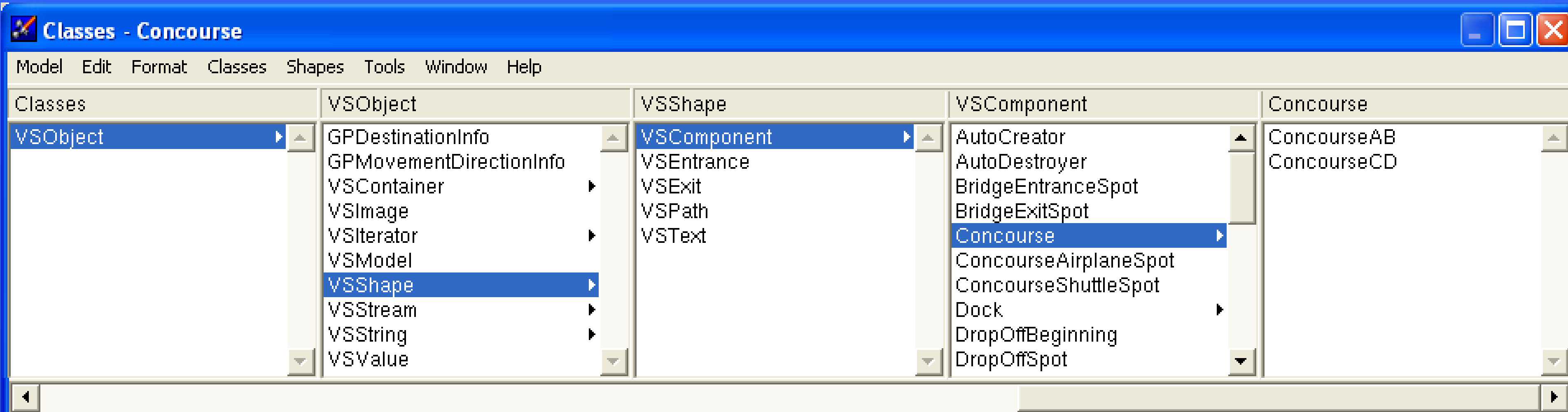
Category | Method

Class Parts	Instance Methods	Notification
Class Documentation	Access	checkPassengers
Class Variables	Modifier	depart
Class Methods	Notification	dynObjArrived:
Instance Variables		dynObjDeparted:
Instance Methods		initializeItself
		tellSelfToDepart

Documentation | Logic

```
returns
        nothing
parameters
        none
declarations
        none
logic
        if [numberOfDynObjs] <> 0 then
        {
                tell self to unboardPassengers;
        }
        else  //start boarding since airplane is empty
        {
                if ([location] <> nil ) then
                {
                        set nextSeatNumber to 1;
                        tell [location] to requestPassengersFromWaitArea:NUMBER_OF_SEATS;
                        tell self to performMessageNamed:"depart" afterTime:(airplaneWaitTime+20);
                }
        }
```

Inheritance



- When an object is declared as member of a class, it inherits the characteristics (instance variables) and behaviors (instance methods) of that class as well as the characteristics and behaviors of that class's superclass, and any ancestral classes, tracing back to the root class.
- Inheritance significantly facilitates reusability of earlier developed classes and decreases simulation model development time.

Message Passing

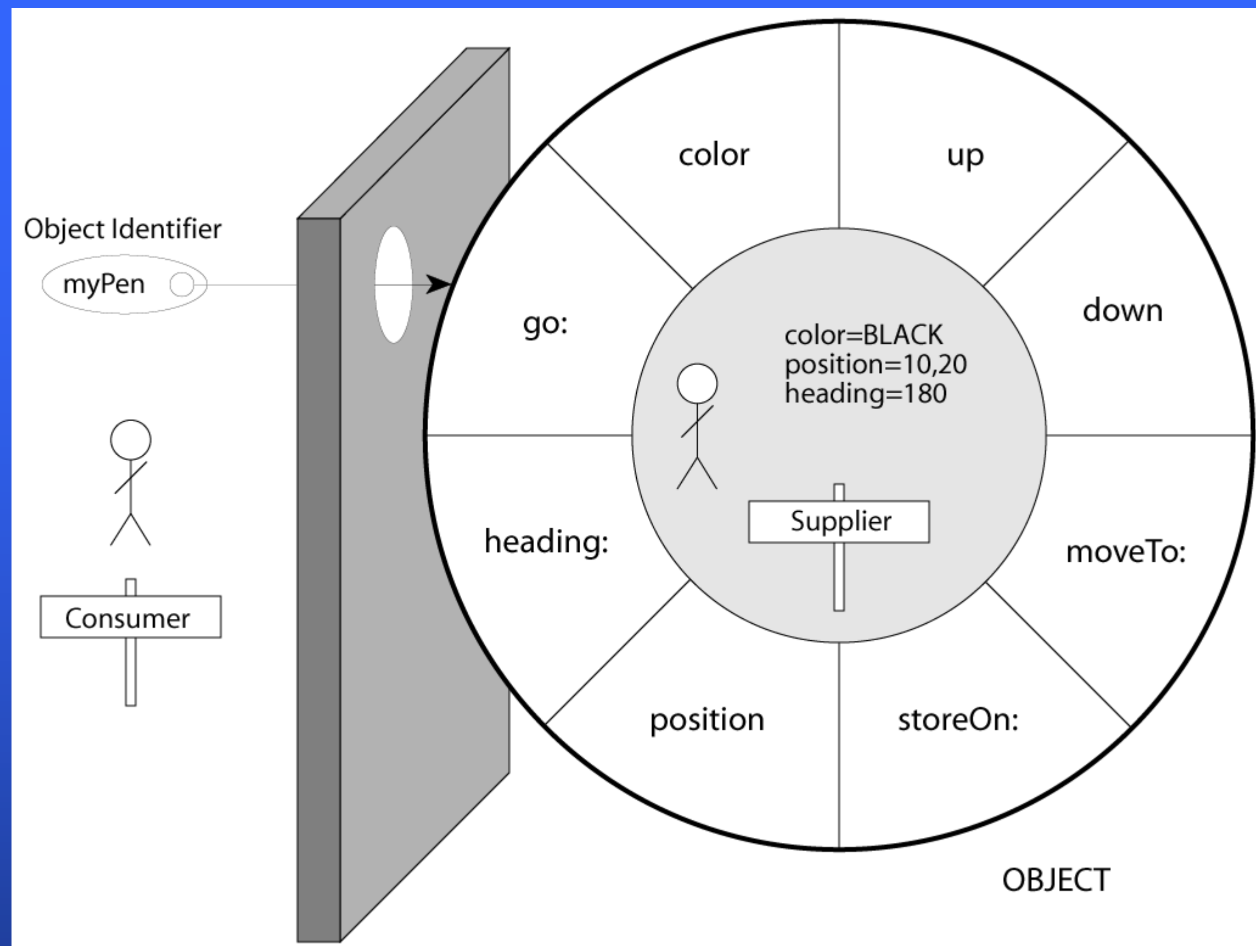
```
logic
    if [numberOfDynObjs] <> 0 then
    {
        tell self to unboardPassengers;
    }
    else        //start boarding since airplane is empty
    {
        if ([location] <> nil ) then
        {
            set nextSeatNumber to 1;
            tell [location] to requestPassengersFromWaitArea:NUMBER_OF_SEATS;
            tell self to performMessageNamed:"depart" afterTime:(airplaneWaitTime+20);
        }
    }
```

Airplane object's checkPassengers method logic

- All objects communicate with each other via message passing.
- Sending a message to an object implies the invocation of one of the receiving object's methods.
- Generally, it is said that "send message M to object A" as opposed to "send a message to object A to invoke its method M."
- Objects are identified by their unique addresses internally maintained by the implementation programming language. They are called object references. An object reference uniquely identifies the object which will receive the message.

Encapsulation

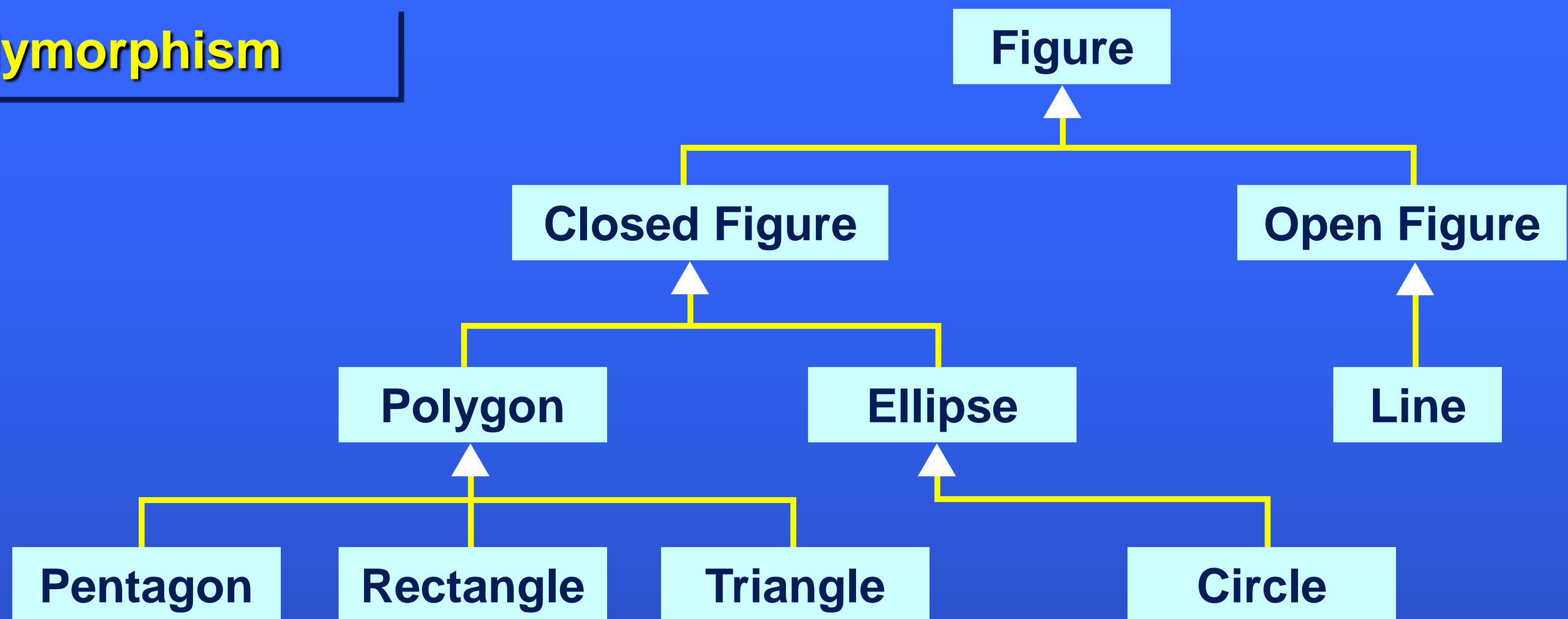
Encapsulation means that a consumer sees only the services that are available from an object, but not how those services are implemented.



Consumer / Supplier Viewpoints

- Only the supplier has visibility into the object's procedures and data.
- From a consumer's point of view, an object is a seamless capsule that offers a number of services, with no visibility as to how these services are implemented.

Polymorphism



- **Polymorphism** refers to the ability of an object to assume more than one form. For example, an object reference can be created to refer to a shape: obj ref shape;
- The object reference shape may refer to a rectangle, triangle, or circle depending on the logic at run time.
- For example, sending the computeArea message to the object pointed to by the object reference shape would invoke a different algorithm (procedure) depending on the form of that object, i.e., rectangle, triangle, or circle, at run time.

Dynamic Binding

- Dynamic Binding implies that the implementation code executed in responding to a particular message in the source code is not known at compile time, and is dispatched at run time depending on the form of the polymorphic object that receives the message. For example, an object reference can be created to refer to a shape:

    obj ref   shape;

- The object reference shape may refer to, or can assume the form of, a circle, rectangle, or triangle.
- For example, sending the computeArea message to an object of the Circle class, pointed to by the object reference shape, would invoke a different algorithm from one invoked when sending computeArea to an object of the Rectangle class, again pointed to by the object reference shape.
- Therefore, the implementation code executed in response to the computeArea message is determined and dynamically bound at run time depending on the form of the object, pointed to by the object reference shape, which receives the message.
- Polymorphism and dynamic binding are strongly related.

Unified Modeling Language (UML) Diagrams for Object-Oriented Design Specification

- **Structural Diagrams**
  - Class Diagram: Classes, interfaces, and collaborations
  - Object Diagram: Objects
  - Component Diagram: Components
  - Deployment Diagram: Nodes
- **Behavioral Diagrams**
  - Use Case Diagram: Organizes the behaviors of the system
  - Sequence Diagram: Focused on the time ordering of messages
  - Collaboration Diagram: Focused on the structural organization of objects that send and receive messages.
  - Statechart Diagram: Focused on the changing state of a system driven by events
  - Activity Diagram: Focused on the flow of control from activity to activity.