



[www.primaryview.org](http://www.primaryview.org)

*from the notebooks of John Artim*

# Use Cases and Software Requirements Specification

## Concept Modeling for Analysis and Specification

John M. Artim, ©2003, 2004. Permission to copy, modify, or distribute is granted.

# Acknowledgements

- Feel free to use this material, in whole or part, for your own training needs but please indicate that you based your work, with permission, on a copyrighted work by John M. Artim that is available from [www.primaryview.org](http://www.primaryview.org).
- A Microsoft® PowerPoint file of this notebook is available by request. Send an email to John Artim ([jartim@uistyle.com](mailto:jartim@uistyle.com)) requesting the Use Case Basics Notebook PowerPoint file in the Use Cases and Software Requirements Specification series. Please include your name, affiliation, and your intended use of the material.

# Goals of this Notebook

---

- Discuss concept modeling in support of requirements specification
- Introduce object and state modeling in UML notation

**Browsing time for this notebook is about 1-1/2 hours**



# Background for this Notebook

---

- You should have a basic understanding of use case modeling such as is outlined in the *Use Case Basics* notebook



# Major Topics in this Notebook

---

- What is a concept model and what is it good for?
- How are concepts described with an Object Model?
- How are concepts with a strong state character described?

# Why Concept Modeling?

- Concept modeling helps to maintain unambiguous language within use case scenarios
  - The concepts—read for this, the language—in the scenarios of a use case model must be used consistently
    - Any inconsistency means that the requirements are inconsistently expressed
    - This *will* lead to an unpredictable implementation



# Concept Modeling and Use Cases

---

- From *The Basics of Use Case Authoring* notebook, Task Modeling includes:
  - **How, When, Who, and What**
- A Concept Model is concerned with the **What** covered by a use case model including:
  - The content of the text in the narrative steps
  - Preconditions
  - Postconditions
  - And optionally, triggers

# Concept Modeling

- In simplest form a concept model is simply a glossary of the terms used in scenarios
  - The glossary is used to promote consistent usage of terms throughout all scenarios
- A concept model can also be an object model
  - The purpose is the same—consistent use of language in scenarios
  - An object model extends the notion of glossary by structuring the concept definitions
  - This structure highlights ambiguity, duplication, and inter-relationships among concepts

# Concept Modeling Basics

- At a minimum each concept must have its own definition where:
  - Each concept has one and only one preferred term—avoid multiple synonyms in your scenarios!
  - Overlapping concepts are defined by using one concept's preferred term within the second concept's definition
  - All significant terms used in use case scenarios should be included in this glossary (concept model)

# Example Glossary—Catalog Sales

- **Address**—A physical location identified by a street name, number, city name, state or province name, country name, and postal code.
- **Catalog**—A set of one or more *catalog items* organized around some marketing theme.
- **Catalog Item**—A listing within a *catalog* representing a single product.
- **Customer**—A person who receives goods or services on a for-fee basis.
- **Order**—A set of one or more *order items* requested by a *customer* for purchase and shipment to some *address*.
- **Order Item**—A single item within an *order* representing a customer request to purchase a quantity of one *catalog item*.
- **Product**—An item for sale fulfilling some customer need and identified by a SKU number.

# Glossary—Pros and Cons

- Pros
  - Easy to construct
  - Easy to share with subject matter experts (SMEs)
  - A text editor and a good dictionary are all you need
  - You'll need a glossary anyway!
- Cons
  - Hard to maintain without unnecessary overlap or inconsistency
  - Hard to automatically check scenario language against a glossary (possible with scripting)
  - Not always expressive enough

# A Glossary is not Enough?

---

- If you are:
  - Working in a large domain?
  - Working with other analysts?
  - Sharing a common reference implementation or model?
- Consider adding more structure to your concept definitions using an object model

# Concept Modeling with Object Models

- Each concept is represented by a class
- The properties of each concept are represented by attributes
- The relationships between concepts are represented by different types of association:
  - Basic associations represent a concept where one concept definition references another concept
  - Aggregation associations represent a whole-part relationship between two concepts
  - Inheritance associations represent a type-of relationship between two concepts
- Key behaviors of the concept are defined as responsibilities

# What is a Class?

- Technically, a class is an encapsulated package of state and function
- For our purposes as analysts, a class is a way to define a concept that:
  - Separates the part of the concept that is distinct and individual from the parts that are defined in other concepts
  - Allows a rich set of relationships among concepts to be defined
  - And is maintainable!

# Class Descriptions

- A class is defined by:
  - An operational definition
  - A list of attributes of the class
  - A list of responsibilities of the class
  - A list of associations the class has to other classes
    - Basic associations
    - Aggregations (whole-part relationships)
    - Inheritance associations (type-of relationships)

# Separation of Concepts

- The core of a concept is defined by:
  - An operational definition
  - A list of attributes of the class
  - A list of responsibilities of the class
- The relationship of one concept to other concepts is defined by:
  - The associations the class has to other classes
- This is the key to OO and good requirements
  - **Without strong encapsulation boundaries use cases and OO is just another pretty collection of diagrams!**

# Attribute

---

- An attribute represents a property of a concept
  - Names, quantities, size, weight, color, shape

# Responsibility

- A responsibility represents a bundle of function the class (concept) must be able to perform as a part of what defines it
  - A payment can debit itself
  - An address can validate itself
  - An order item can confirm sufficient stock is available
  - An order can confirm itself
    - Which might lead to order items confirming stock available and to a payment debiting itself

# Association

- An association represents some way in which one concept is defined, in part, by another concept
- The connection or relation of ideas, feelings, sensations, etc.; correlation of elements of perception, reasoning, or the like—Random House Unabridged Dictionary, © 1996

# Example of an Association

- An *Order* is defined, in part, by:
  - The order items that it contains
  - The customer who is ordering it
  - The payment that the order requires
  - The employees who have helped fulfill it
  - The address to which the order items are shipped

# Association Types

---

- Generic Association
- Aggregation
  - Also called a composition or whole-part relationship
- Inheritance

# Association

- A generic association is a connection between two or more concepts that indicates that they collaborate to create a greater concept
  - The association can be named to give life to the association itself
  - Or the role of one or more of the classes at the ends of the association can be named
- All other association types are specializations of this generic type

# Aggregation or Whole-Part

- A whole-part relationship between two concepts implies that the parts form a portion of the definition of the whole
- For example, and *Order* is a whole and *Order Item* is a part of that whole
  - Having one or more *Order Items* is a portion of what defines an *Order*

# Inheritance

- An inheritance association indicates that one class represents a generic concept and other classes represent specific concepts based on the generic
- For example,
  - The concept *Payment* represents a generic way to pay for goods or services
  - The concept *Credit Charge* represents a *Payment* by credit card
  - And the concept *Purchase Order* represents a *Payment* by Purchase Order
- This separates the common definition of the concept of *Payment* from the specifics of each kind of *Payment*

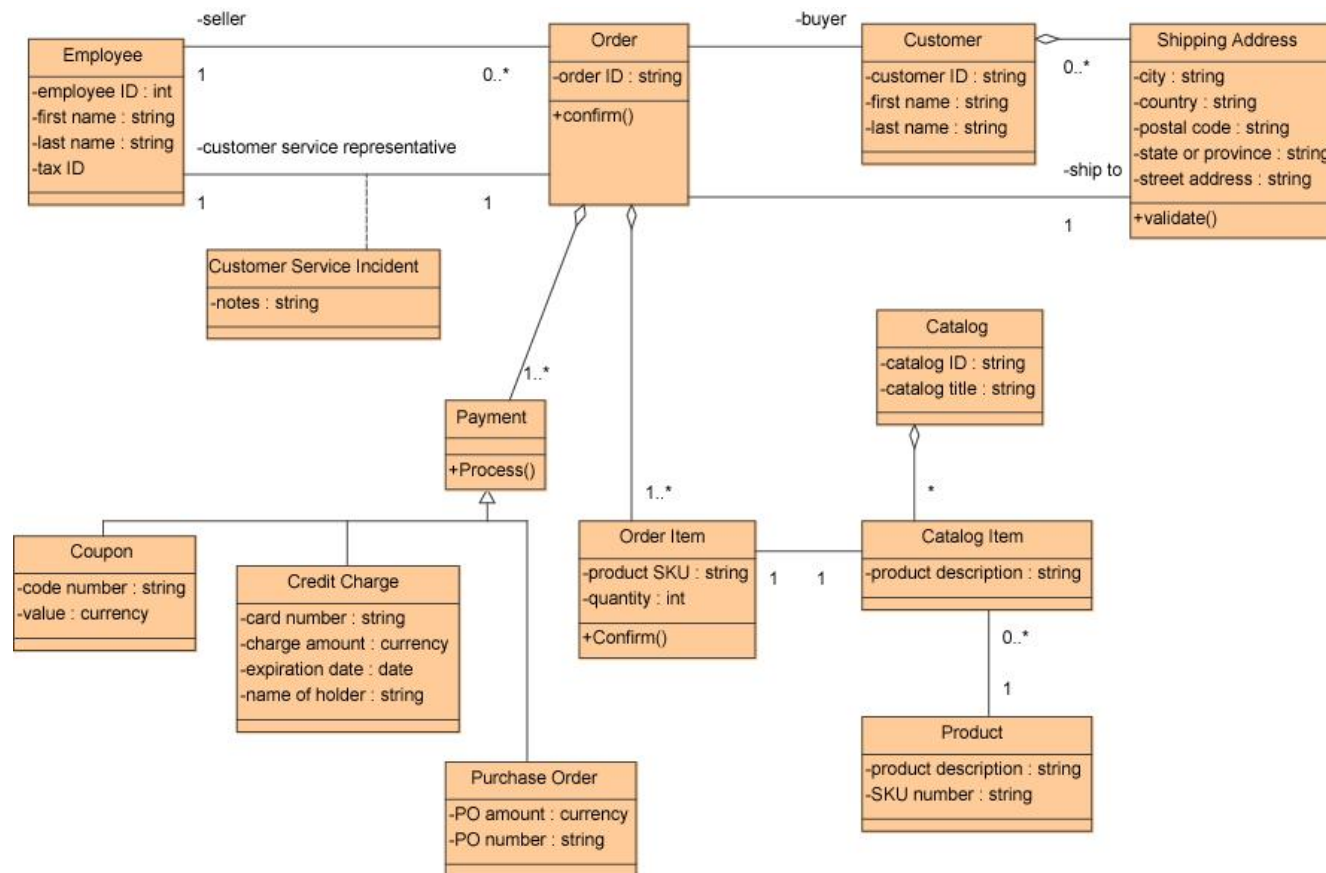
# Example of a Class Definition—Order

- Brief Description
  - A set of one or more order items requested by a customer for purchase and shipment to some address
- Attributes
  - Order ID
- Responsibilities
  - Confirm
- Whole-Part Relationships
  - The Order Items that it contains
- Associations
  - The Customer who is ordering it
  - The Payment that the Order requires
  - The Employee who took the Order
  - The Employees who have helped fulfill it
  - The Address to which the Order Items are shipped

# Example of a Responsibility Definition—Confirm

- Each Attribute, Responsibility, Association, and so on, must similarly be defined as appropriate for that item
- *Confirm*
  - The action of confirming completion of an *Order* including:
    - *Confirming each Order Item*
    - *Process the Payment*

# Example—Customer Service Class Diagram



# Pros and Cons of Class Modeling

- Pros
  - Can see an overview of the entire glossary in a diagram or two
  - Can get tool support for maintaining consistency between classes and scenarios
  - Easier to maintain
    - Each class points to other classes that are affected by changes
- Cons
  - Tool costs are higher
    - Unless you use a drawing tool such as Visio, but then you have more work to do!
  - Harder to learn to do
    - But not that hard!
  - Initial authoring costs higher
    - But maintenance costs lower!

# State Modeling

- State modeling helps to simplify talking about classes which exhibit state behavior
- These are often concepts that are operated on by a cascade of business processes where each process transforms the object from one state to the next
- This can be most helpful where there are branching choice points where one state can lead to a choice of two or more states depending on what happens in the business processes
  - This shows up in the coming example ...

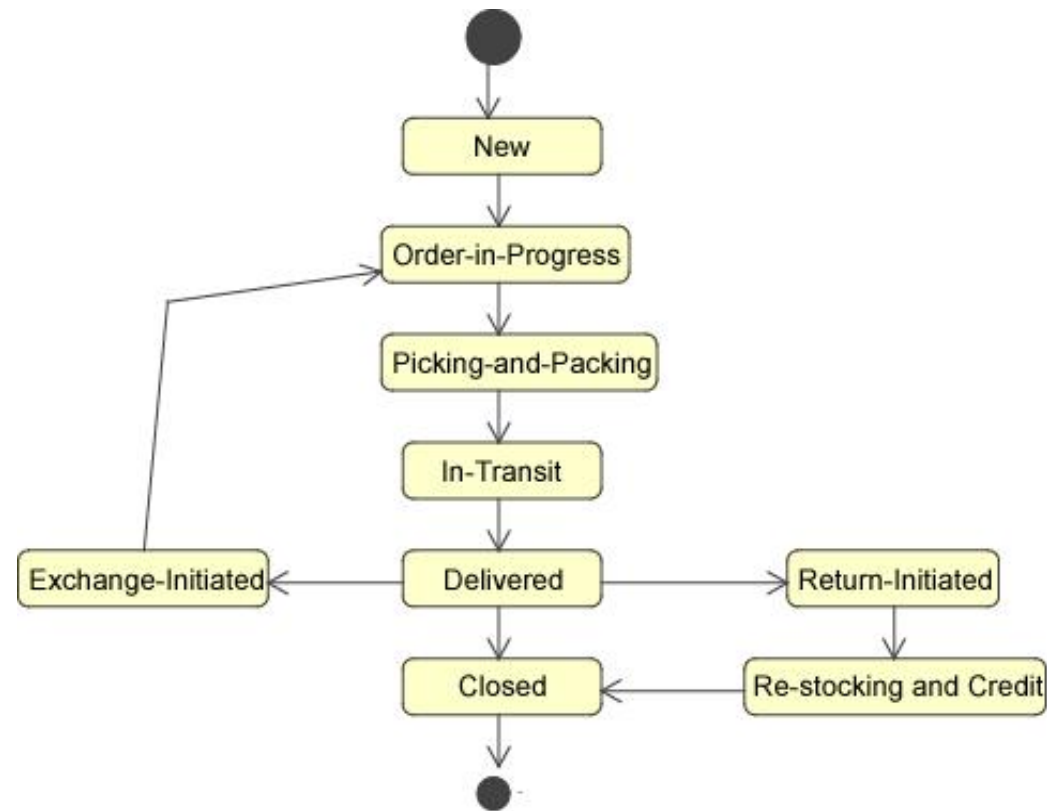
# State Model of a Class

- A state model represents a class's encapsulated data as a discrete series of states where each state leads to one or more other states

# Order Object—State Characteristics

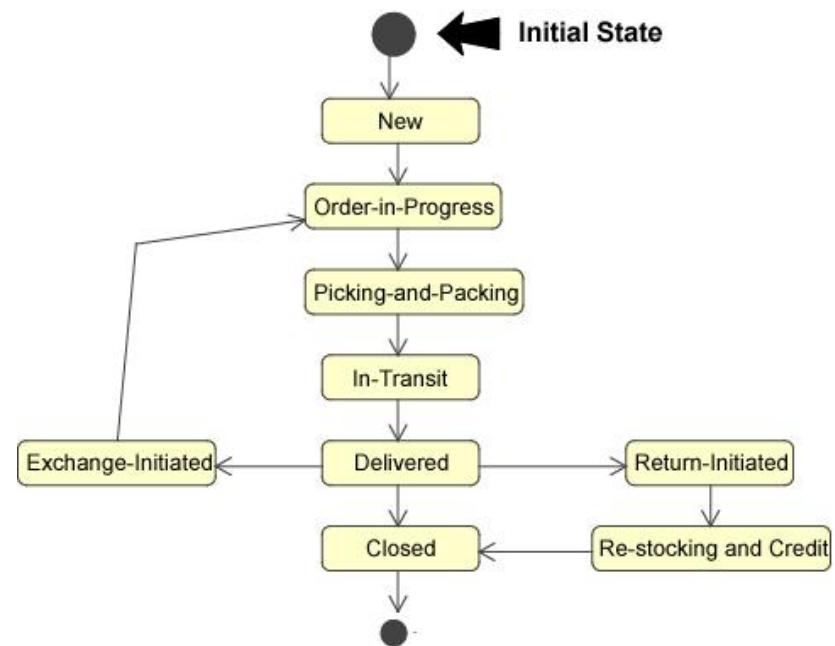
- For some concepts, the behavior of the concept is dominated by the set of states it can be in
- The *Order* class displays state characteristics and can be in these states:
  - New
  - Order-in-Progress
  - Picking-and-Packing
  - In-Transit
  - Delivered
  - Closed
- **Note:** the set of states to choose is dependent on the business processes supported—be careful!

# Order—State Transition Diagram



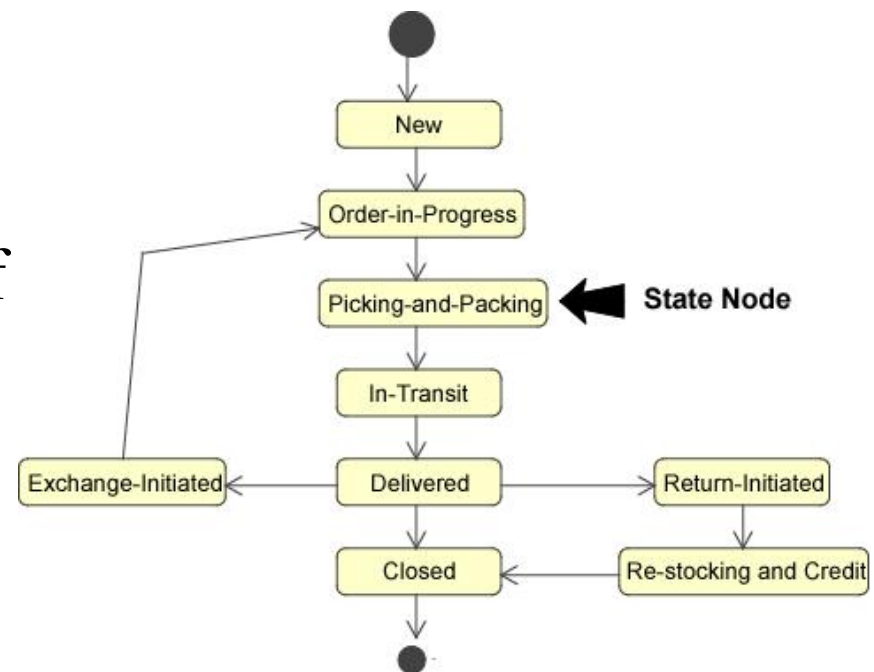
# Initial State

- The Initial State is a standard state that indicates the entry state for a class
- All state models must start with a single initial state!



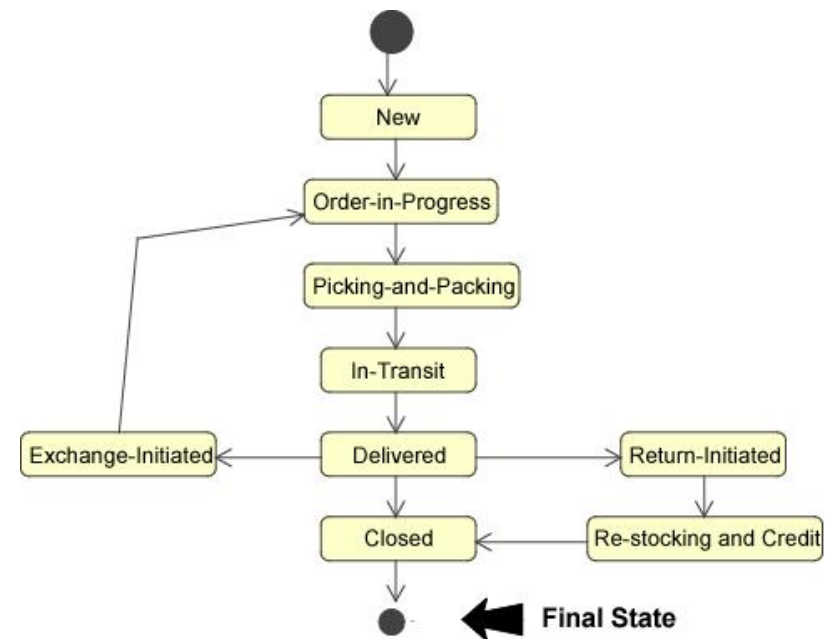
# State

- Each state in a State diagram is represented by an oval with the name of the state inside the oval



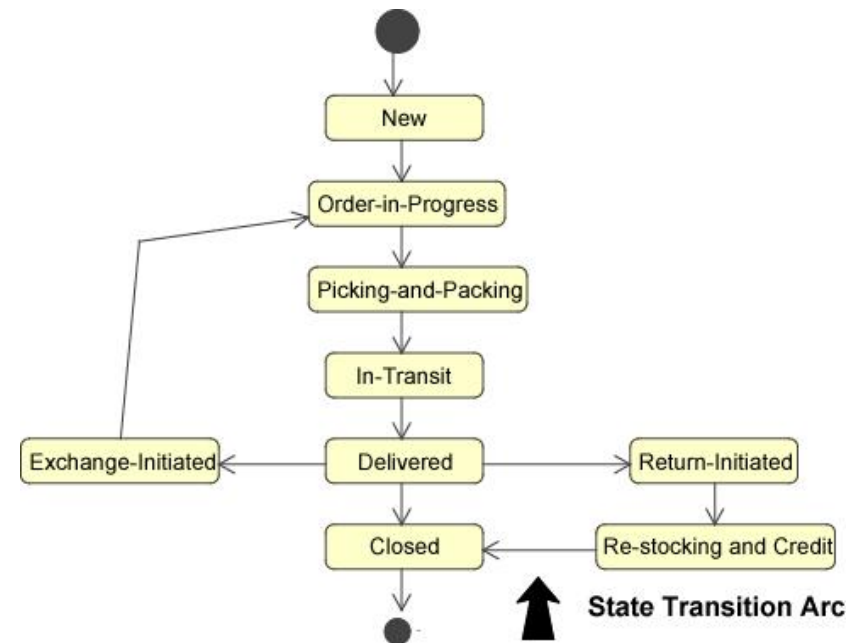
# Final State

- The Final State is a standard state that indicates the exit state for a class



# State Transition

- To be valid, each state must have at least one transition leading into it, and at least one transition leading out of it
- A state transition arc is a line with an arrowhead at one end that symbolizes a transition from one state (at the tail of the arc) to another state (at the head)



# Conclusions

---

- Concept Modeling in some form helps drive consistent scenarios
  - This leads to cleaner requirements
- Glossaries are a cheap and easy way to get a good chunk of this consistency
- To get good consistency for large domains or when sharing work among many analysts class models are a better tool
- State diagrams provide an easy way to visualize concepts that are defined in part by their state behavior