88823459
การวิเคราะห์และออกแบบระบบเชิงวัตถุ

พีระศักดิ์ เพียรประสิทธิ์

Outline

- The System Sequence Diagram—Identifying Inputs and Outputs
- The State Machine Diagram—Identifying Object Behavior
- Integrating Requirements Models

วัตถุประสงค์การเรียนรู้

- มีความรู้ความเข้าใจการอ่านและเขียน System Sequence Diagrams
- มีความรู้ความเข้าใจการอ่านและเขียน State Machine Diagrams จนนำไปสู่พฤติกรรมของวัตถุ
- การใช้งานแผนภาพ UML ต่างๆ เพื่ออธิบายถึงความต้องการที่เป็นแบบฟังก์ชัน (functional requirements)

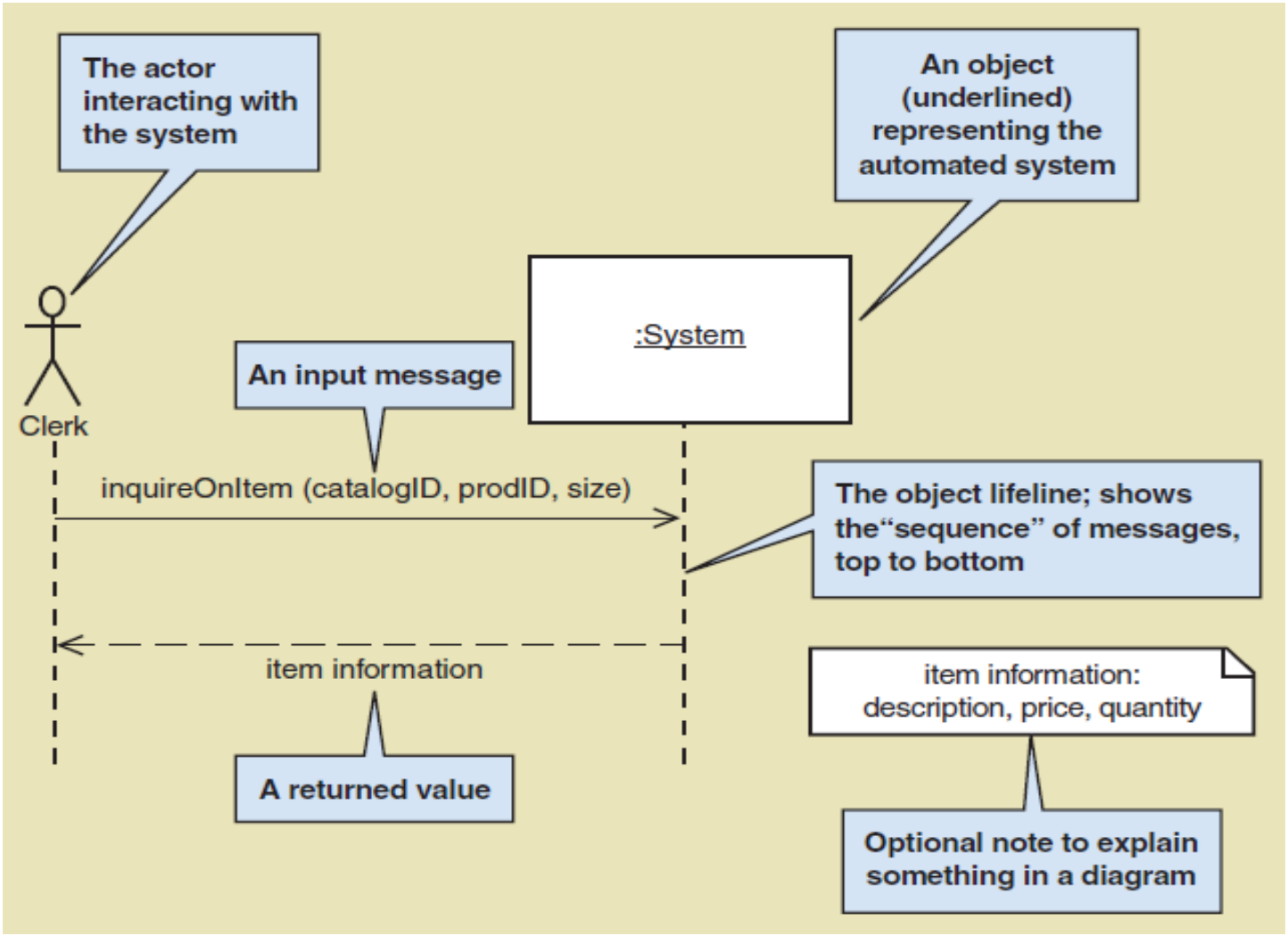
Overview

- System Sequence Diagrams (SSDs) แสดงถึงการนำเข้า (Input) และการส่งออก (Output) ของแต่ละ Use Case เสมือนข้อความ (messages)
- State Machine Diagrams แสดงถึงสถานะของวัตถุที่เป็นไปได้ในช่วงเวลาที่มีการเรียกใช้งาน use case
- ในแต่ละยูสเคสนั้นนักออกแบบได้แสดงรายละเอียดของยูสเคสด้วยคำอธิบายยูสเคส (Use Case Description) แผนภาพกิจกรรม นอกจากนั้นนักวิเคราะห์ระบบจะใช้แผนภาพซีเควนซ์ระบบ (System Sequence Diagram) แผนภาพโดเมนคลาส (Domain Model Class Diagram) และแผนภาพสถานะ (State Machine Diagram) เพื่อแสดงถึงรายละเอียดของระบบ

System Sequence Diagram (SSD)

- A UML sequence diagram แสดงถึงการปฏิสัมพันธ์ระหว่าง actor และอ็อบเจกต์ต่างๆ ในระบบ เพื่อจะสามารถดำเนินพฤติกรรมให้เป็นไปตามยูสเคส
- Special case for a sequence diagram
  - แสดงเฉพาะ actor และ object หนึ่งเท่านั้น
  - The one object represents the complete system
  - Shows input & output messaging requirements for a use case
- Actor, :System, object lifeline
- Messages

สัญลักษณ์ที่ใช้ใน System Sequence Diagram (SSD)



Message Notation

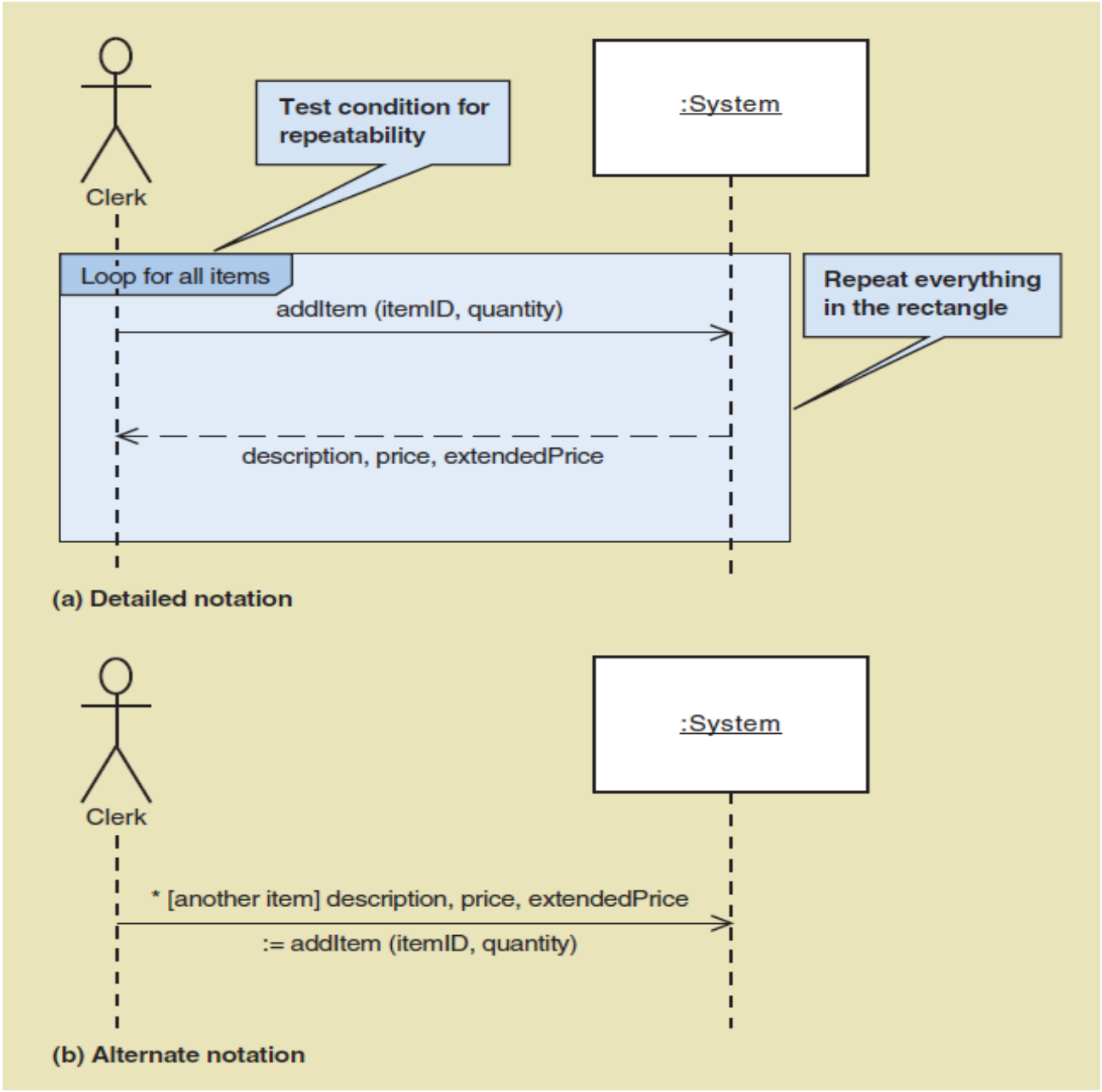
[true/false condition] return-value := message-name (parameter-list)

- An asterisk (*) indicates repeating or looping of the message.
- Brackets [ ] indicate a true/false condition. This is a test for that message only. If it evaluates to true, the message is sent. If it evaluates to false, the message isn't sent.
- Message-name is the description of the requested service. It is omitted on dashed-line return messages, which only show the return data parameters.
- Parameter-list (with parentheses on initiating messages and without parentheses on return messages) shows the data that are passed with the message.
- Return-value on the same line as the message (requires :=) is used to describe data being returned from the destination object to the source object in response to the message.

สัญลักษณ์บนข้อความของ System Sequence Diagram

* [เงื่อนไขจริง/เท็จ] ข้อมูลผลลัพธ์ := ชื่อข้อความ (รายการของพารามิเตอร์)

- ดอกจัน (*) แสดงการทำซ้ำหรือวบลูปของข้อความ
- วงเล็บ [ ] แสดง เงื่อนไขจริง/เท็จ
  - ถ้าผลการทดสอบเงื่อนไขเป็นจริง จะทำการส่งข้อความ
  - ถ้าผลการทดสอบเงื่อนไขเป็นเท็จ จะไม่ส่งข้อความ
- ชื่อข้อความ แสดงชื่อของบริการที่ร้องขอบริการ
- รายการของพารามิเตอร์ ภายในวงเล็บ แสดงถึงข้อมูลที่ต้องส่งไปพร้อมกับข้อความ
- ข้อมูลผลลัพธ์ ใช้เพื่ออธิบายข้อมูลผลลัพธ์ที่ได้จากอ็อบเจกต์ปลายทางของข้อความที่ร้องขอ

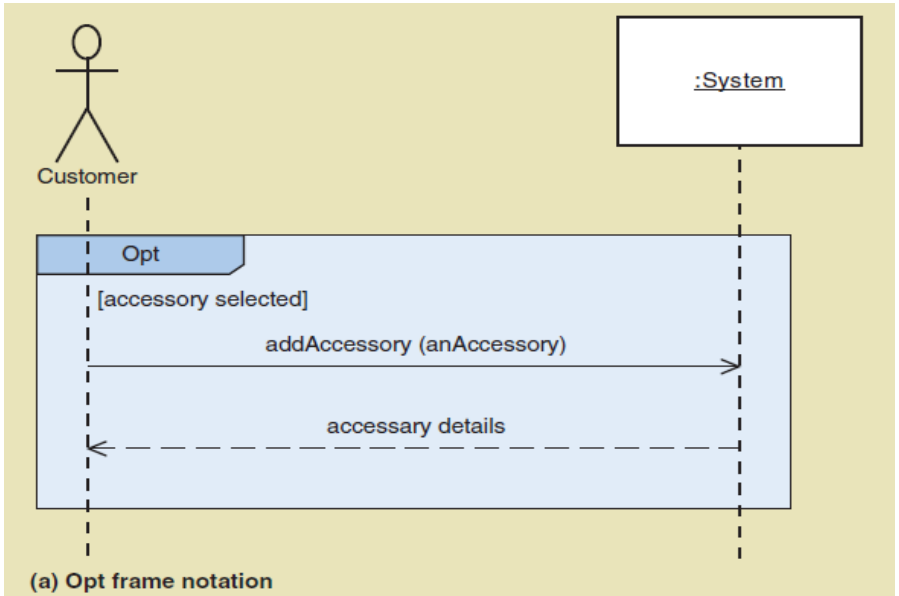


ตัวอย่าง การเพิ่มรายการสั่งซื้อในใบรายการสั่งซื้อ มักจะมีการเพิ่มรายการสั่งซื้อหลายๆ รายการ

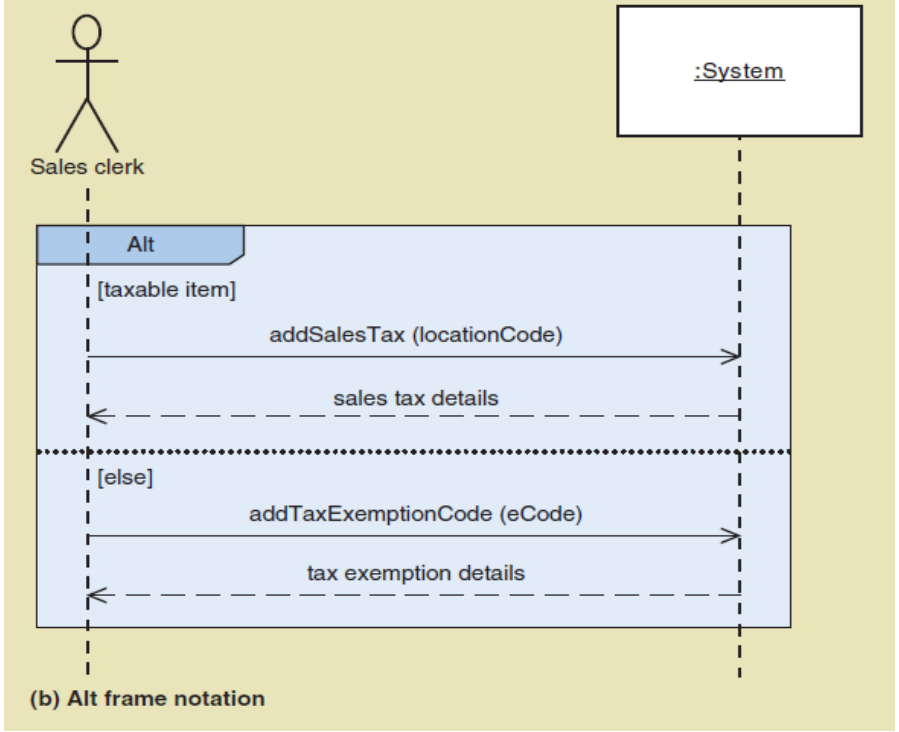
a) ใช้กรอปสี่เหลี่ยมที่แสดงถึงลูปการทำซ้ำ

b) ใช้เครื่องหมาย * แสดงการทำซ้ำ

ทั้งภาพ a และภาพ b มีความหมายเหมือนกัน



(a) Opt frame notation



(b) Alt frame notation

ตัวอย่าง SSD

Opt Frame (optional) เป็นการแสดงถึงทางเลือกเพิ่มเติม เช่น การเลือกอุปกรณ์เสริม

Alt Frame (if-else) เป็นการแสดงถึงทางเลือกอย่างใดอย่างหนึ่ง เช่น การคำนวณภาษี หรือการขอยกเว้นภาษี

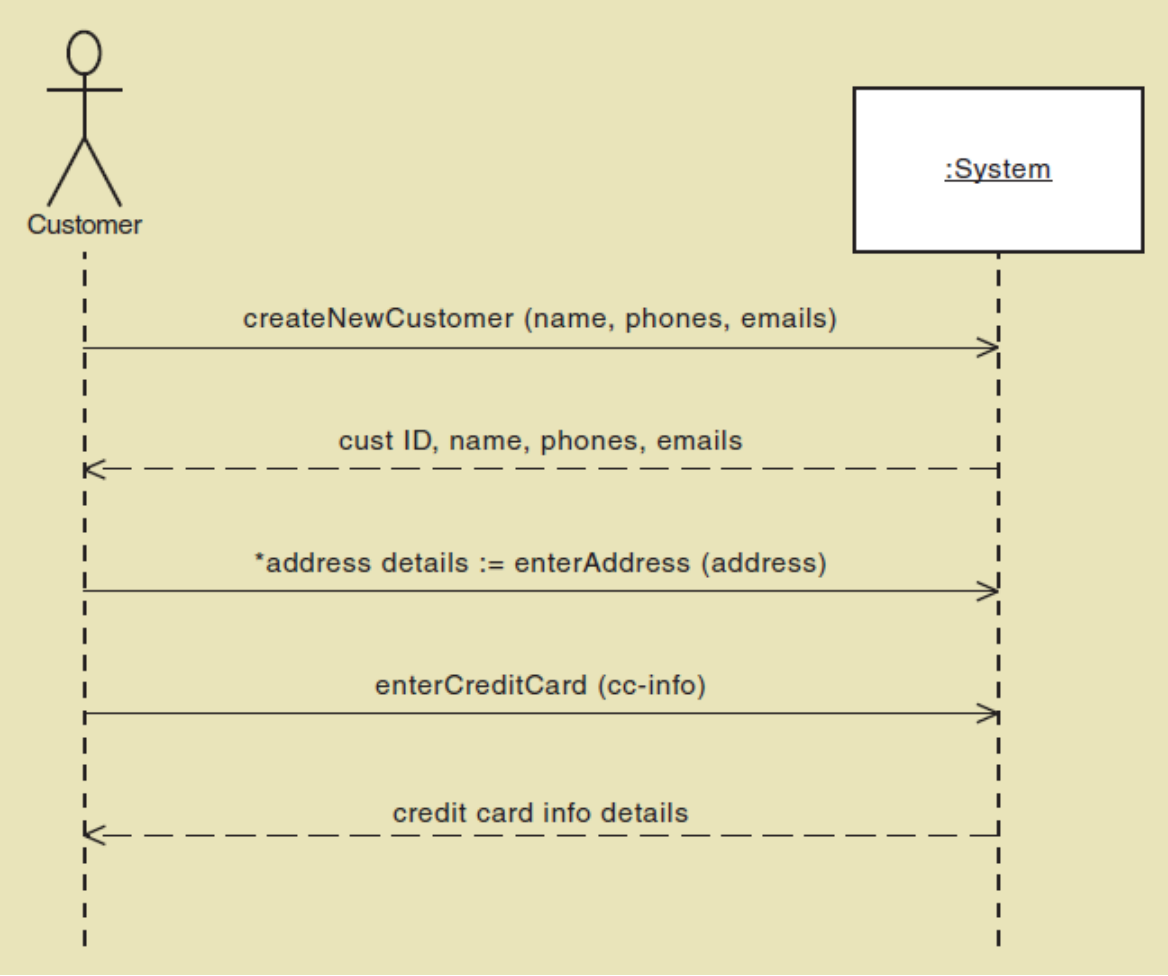
ขั้นตอนการพัฒนา SSD

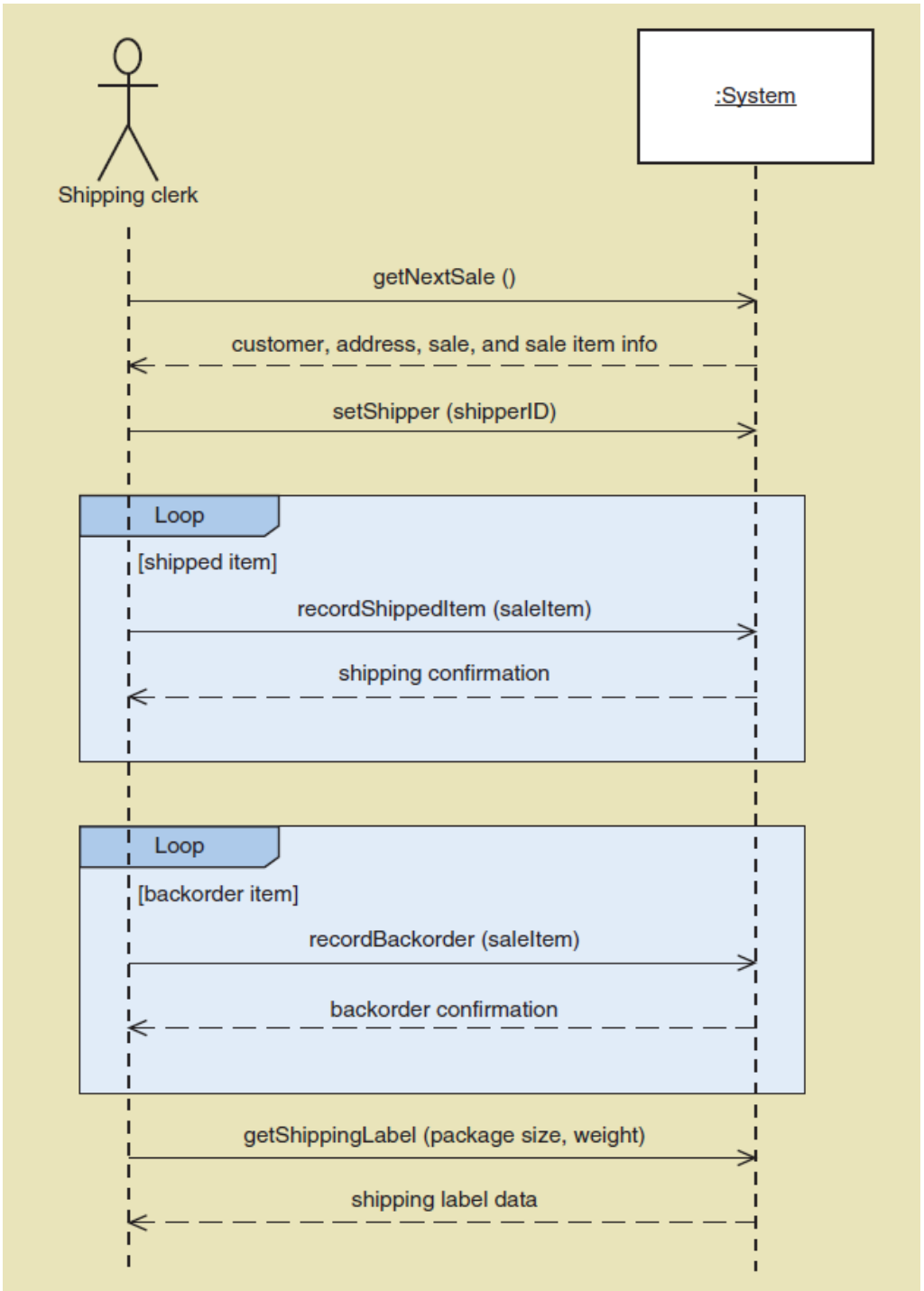
1. ระบุข้อความ (message) ที่ใช้ input
   - พิจารณาจาก flow of activities ของ use case หรือ activity diagram
2. เขียนบรรยายที่ actor เรียกใช้งานระบบ
   - ข้อความควรเป็นลักษณะ คำกริยา-คำนาม
   - พิจารณา parameter ที่ระบบต้องการ
3. ระบุเงื่อนไขพิเศษ บนข้อความที่ input เข้าระบบ เช่น การทำซ้ำ (loop) ทางเลือกต่างๆ ทั้ง optional และ altetnate
4. ระบุ output ที่ส่งค่าคืนจากระบบ (return values)
   - หากข้อความนั้นเป็นการกระทำกับวัตถุตัวเองให้ระบุดังนี้ aValue:= getValue(valueID)
   - หาสิ่งที่คืนค่าออกมาชัดเจน ให้ทำการเขียนบนเส้นประ

Steps for Developing SSD

1. Identify input message
   - See use case flow of activities or activity diagram
2. Describe the message from the external actor to the system using the message notation
   - Name it verb-noun: what the system is asked to do
   - Consider parameters the system will need
3. Identify any special conditions on input messages
   - Iteration/loop frame
   - Opt or Alt frame
4. Identify and add output return values
   - On message itself: aValue:= getValue(valueID)
   - As explicit return on separate dashed line

SSD ของยูสเคส Create customer account





SSD ของยูสเคส Ship items

State Machine Diagram

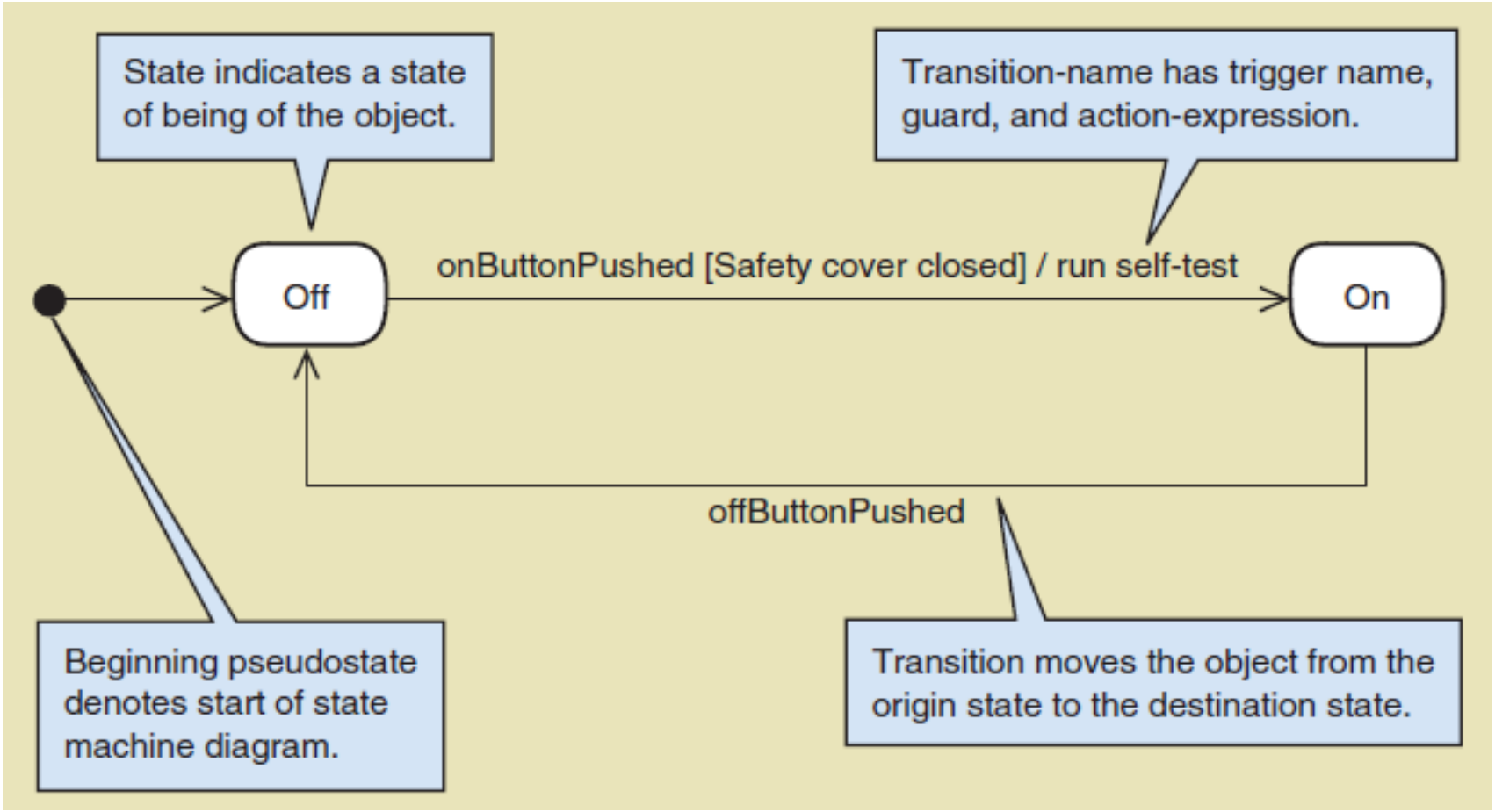
- State machine diagram
  - A UML diagram ที่แสดงถึงสถานะของวัตถุ (object) และการเปลี่ยนแปลงสถานะ
- State
  - สถานะของวัตถุ ณ เวลาหนึ่ง เช่น เมื่อวัตถุนั้น ได้ดำเนินการบางอย่าง รอเหตุการณ์บางอย่าง
- Transition
  - เหตุการณ์ที่ทำให้ object เปลี่ยนสถานะ
- Action Expression
  - รายละเอียดของกิจกรรมที่ดำเนินการ เสมือนเป็นส่วนนึงของการเปลี่ยนแปลง

State Machine Diagram (continued)

- Pseudo state
  - จุดเริ่มต้นของ a state machine diagram (black dot)
- Origin state
  - สถานะเดิมของวัตถุก่อนการเปลี่ยนแปลง
- Destination state
  - สถานะของวัตถุหลังการเปลี่ยนแปลง
- Guard condition
  - การทดสอบข้อเท็จจริง (true/false) เพื่อที่จะทราบว่าการเปลี่ยนแปลงนั้นสามารถดำเนินการได้

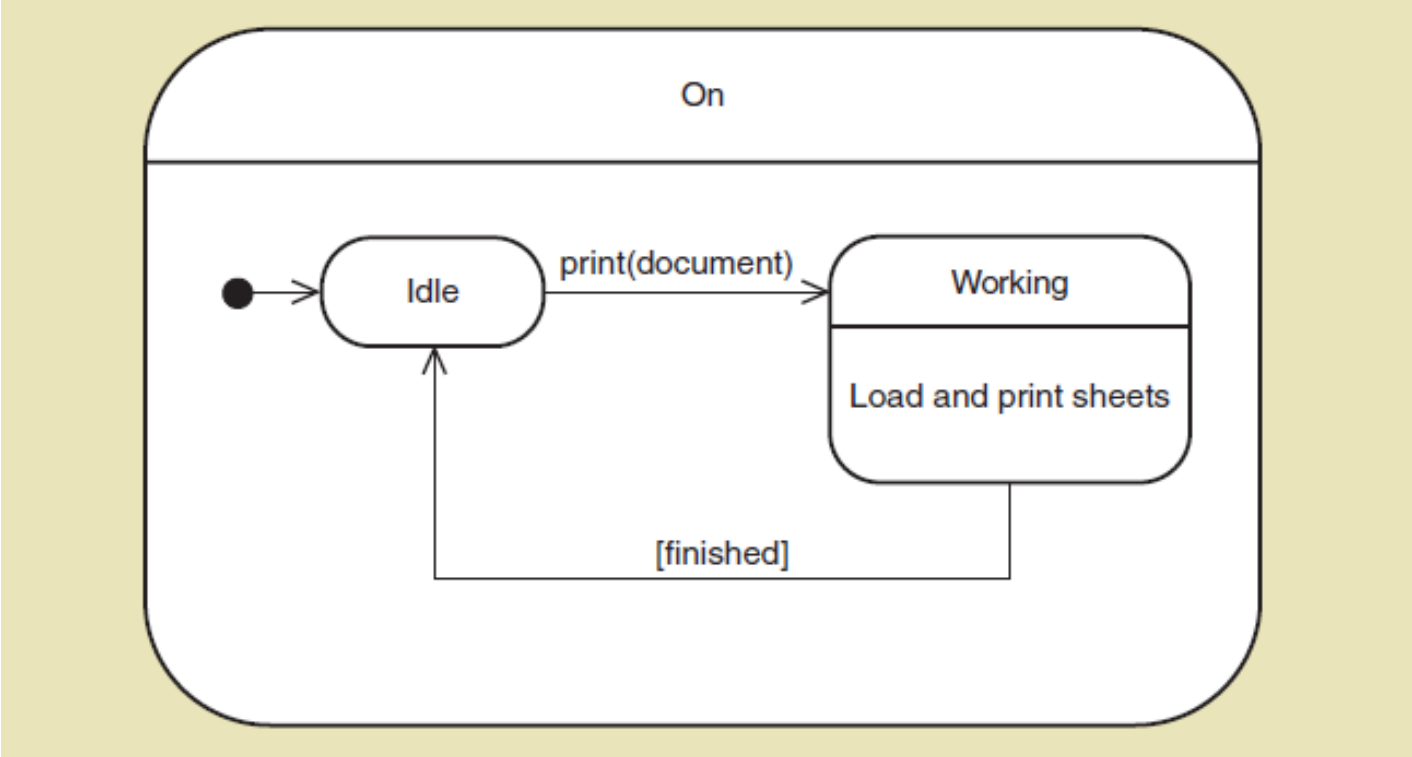
State Machine Diagram

ตัวอย่าง สถานะของเครื่องพิมพ์เอกสาร (Printer)



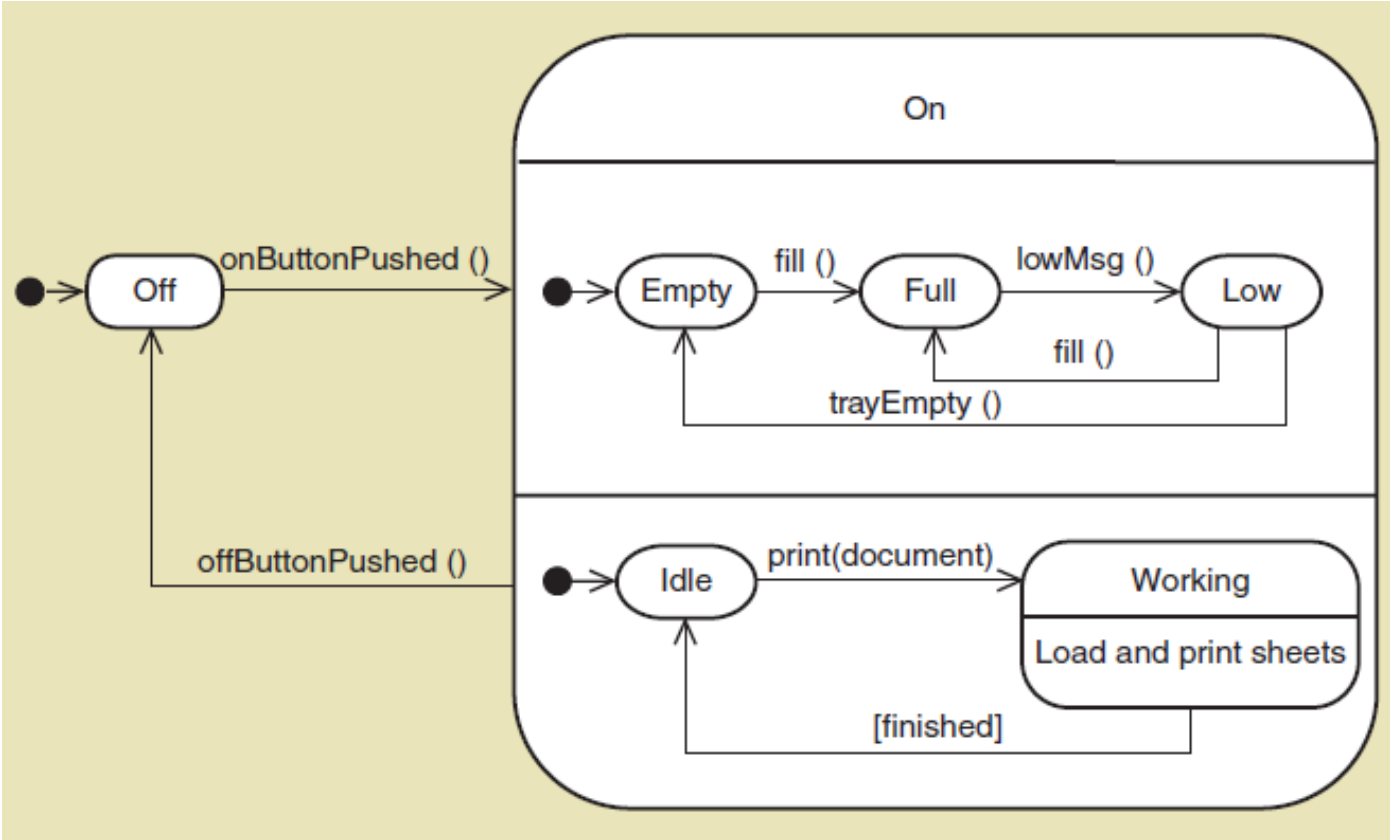
Composite States

- ภายใต้สถานะใดสถานะหนึ่งนั้น ภายใต้สถานะนั้นประกอบไปสถานะอื่นและสามารถเปลี่ยนเป็นสถานะอื่นได้ เช่น เครื่องพิมพ์มีสถานะ “เปิด” และอาจอยู่ในสถานะ “Idle” หรือสถานะ “Working”



Concurrent Paths

- Multiple paths in composite state
- Printer On paths are independent

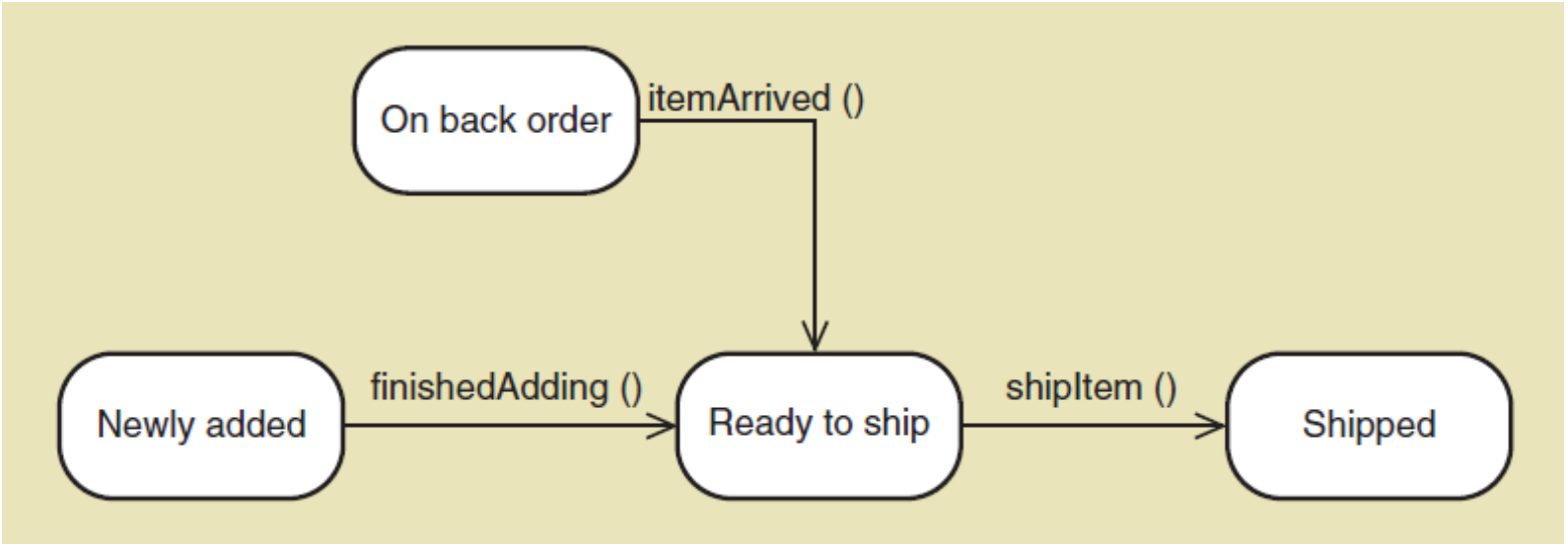


Steps for Developing State Machine Diagram

1. ตรวจสอบคลาสจาก class diagram แล้วเลือก classes ที่อาจต้องใช้ในการสร้าง state machine diagrams
2. ในแต่ละคลาส ให้ระบุสถานะ status conditions (states) ของคลาส
3. เริ่มต้นสร้างแผนภาพ โดยการระบุเหตุการณ์ที่ทำให้สถานะของเปลี่ยนแปลง
4. เรียงลำดับของสถานะและรวมสถานะให้เป็น Composite States
5. ทบทวน independent, concurrent paths และเส้นทางอื่นๆ เพิ่มเติม

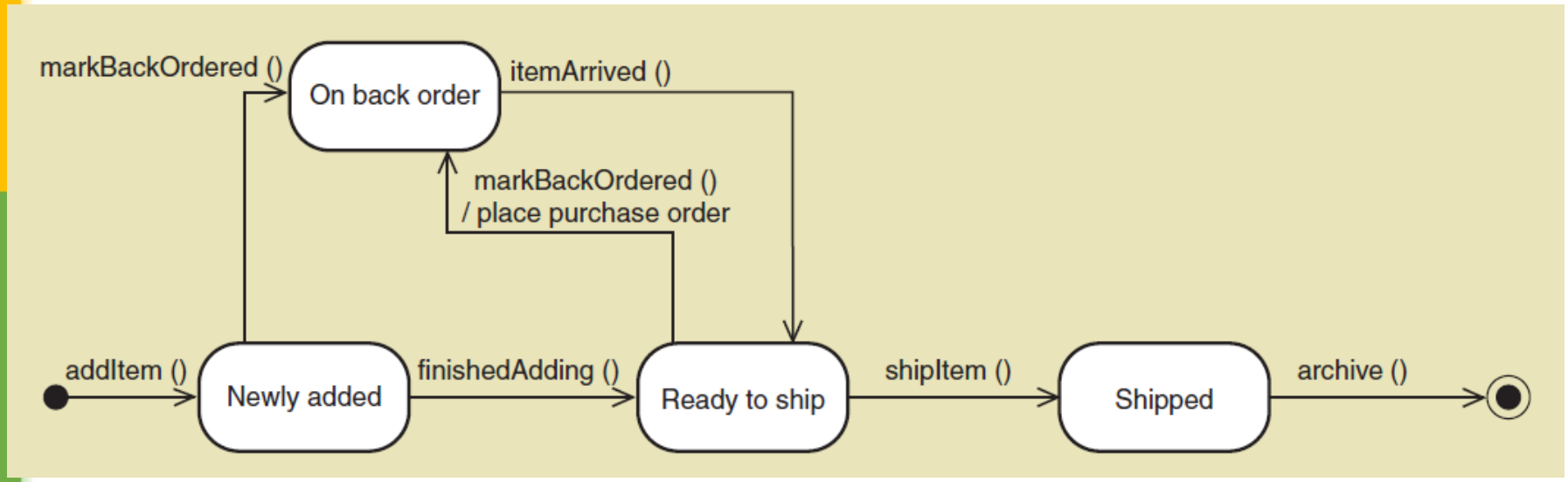
RMO Domain Class States for SaleItem Object

State	Transition causing exit from state
Newly added	finishedAdding
Ready to ship	shipItem
On back order	itemArrived
Shipped	No exit transition defined



Final State Machine Diagram for SaleItem Object

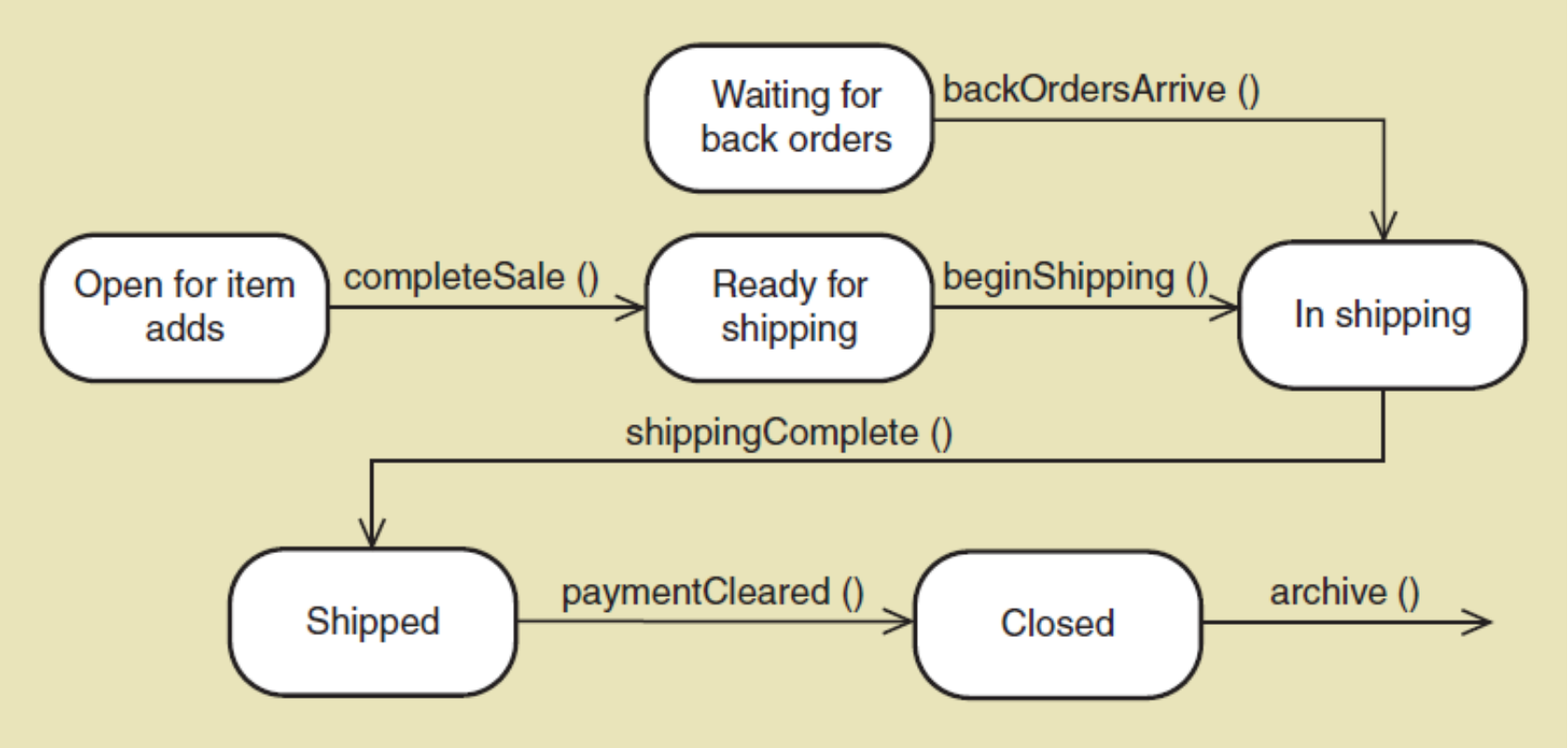
- addItem() and archive() transitions added
- markBackOrdered() transition added



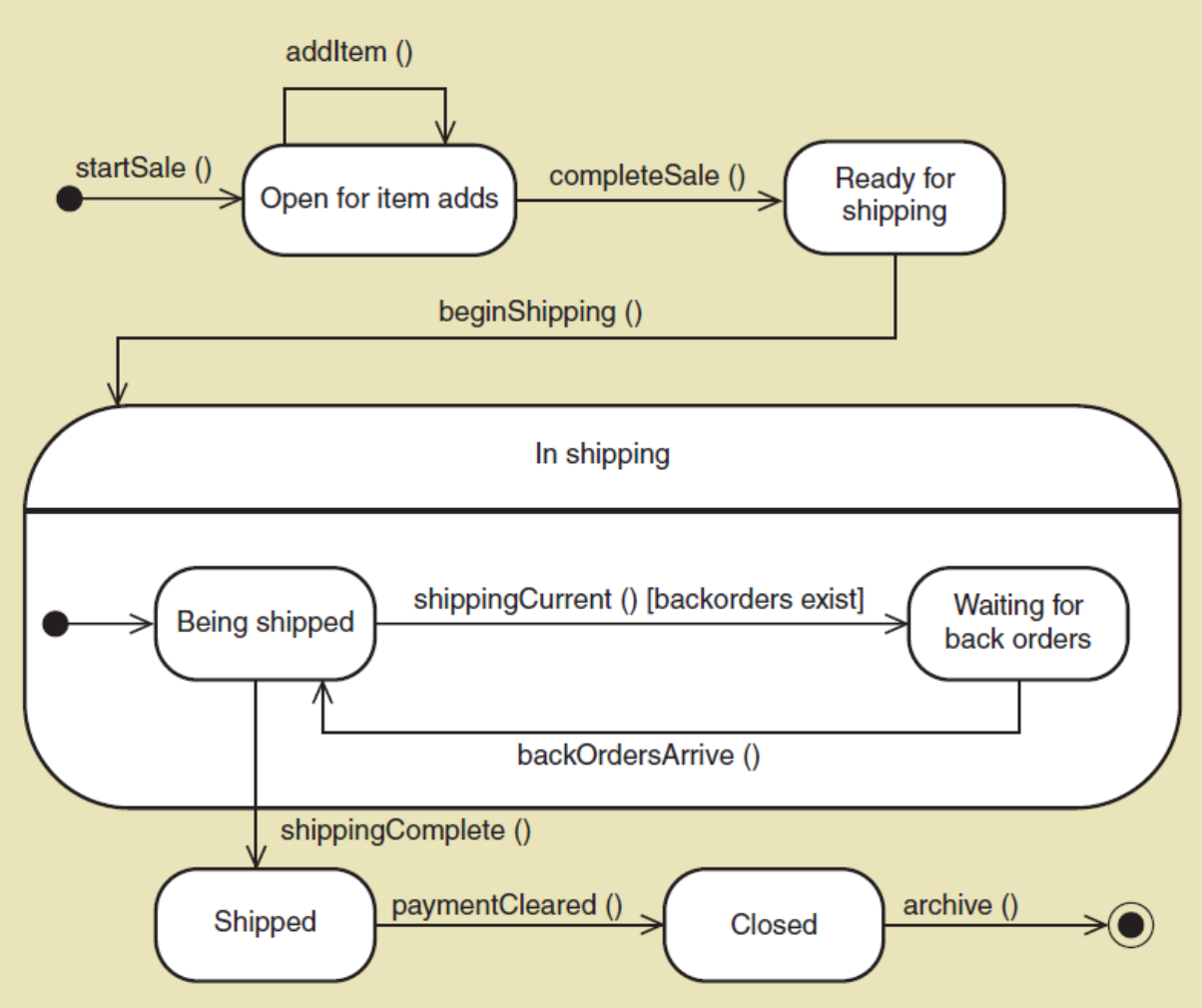
RMO Domain Class States for Sale Object

State	Exit transition
Open for item adds	completeSale
Ready for shipping	beginShipping
In shipping	shippingComplete
Waiting for back orders	backOrdersArrive
Shipped	paymentCleared
Closed	archive

Initial State Machine Diagram for RMO Sale Object



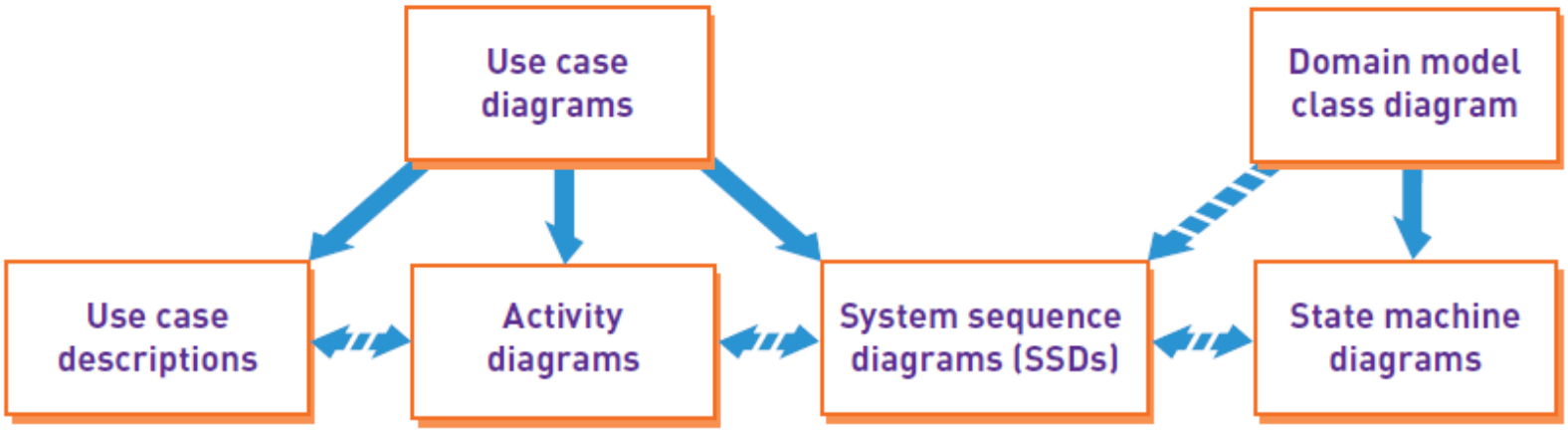
Final State Machine Diagram for Sale Object



Extending and Integrating Requirements Models

- Use cases
  - Use case diagram
    - Use case description
    - Activity diagram
    - System sequence diagram (SSD)
- Domain Classes
  - Domain model class diagram
    - State machine diagram

Integrating Requirements Models



Summary

- Chapters 3 and 4 identified and modeled the two primary aspects of functional requirements: use cases and domain classes
- This chapter focuses on additional techniques and models to extend the requirements models to show more detail
- Fully developed use case descriptions provide information about each use case, including actors, stakeholders, preconditions, post conditions, the flow of activities and exceptions conditions
- Activity diagrams (first shown in Chapter 2) can also be used to show the flow of activities for a use case

Summary (continued)

- System sequence diagrams (SSDs) show the inputs and outputs for each use case as messages
- State machine diagrams show the states an object can be in over time between use cases
- Use cases are modeled in more detail using fully developed use case descriptions, activity diagrams, and system sequence diagrams
- Domain classes are modeled in more detail using state machine diagrams
- Not all use cases and domain classes are modeled at this level of detail. Only model when there is complexity and a need to communicate details