System Analysis and Design

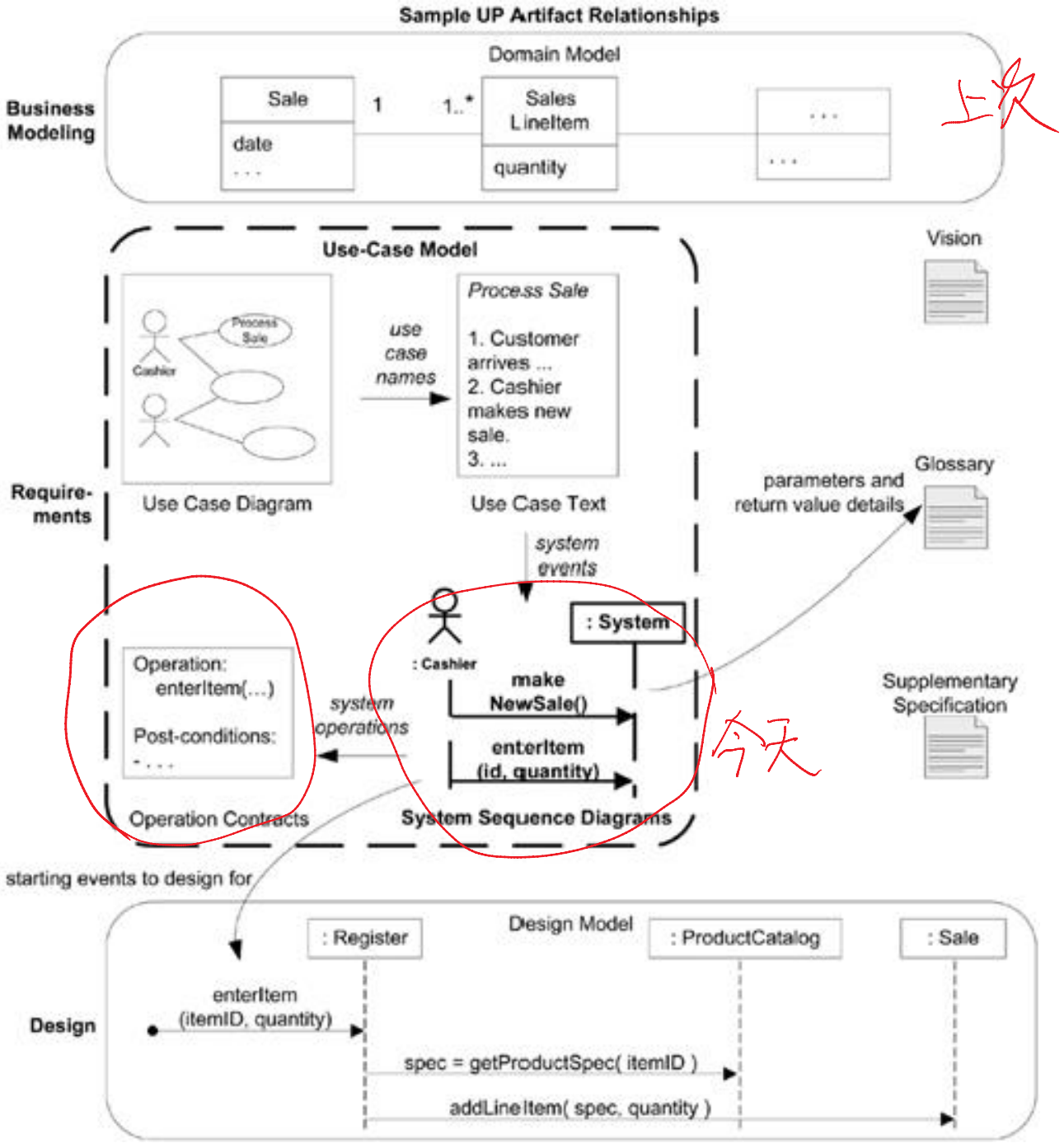
L11. System Sequence Diagrams and Operation Contracts

Topics

- What is an System Sequence Diagram?
- Why Use SSDs?
- Applying UML: Sequence Diagrams
- Iterative and Evolutionary SSDs

- Operation Contracts
- Postconditions
- How to Create and Write Contracts
- Operation Contracts Within the UP

Sample UP artifact influence



What is an System Sequence Diagram (SSD)?

- A diagram that illustrates input and output events related to the system
- It shows, for one particular scenario of a use case, the events that external actors generate, their order, and inter-system events
- SSDs can also be used to illustrate collaborations between systems,
  - such as between the NextGen POS and the external credit payment authorizer.
- Systems are treated as black boxes; describe what they do but not how ;
- The emphasis of the diagram is events that **cross the system boundary** from actors to systems.

What is the Relationship Between SSDs and Use Cases?

- An SSD shows system events *for one scenario of a use case*, therefore it is generated from inspection of a use case
- Draw an SSD for a main success scenario of each use case, and frequent or complex alternative scenarios.

System Sequence Diagram 系统顺序图

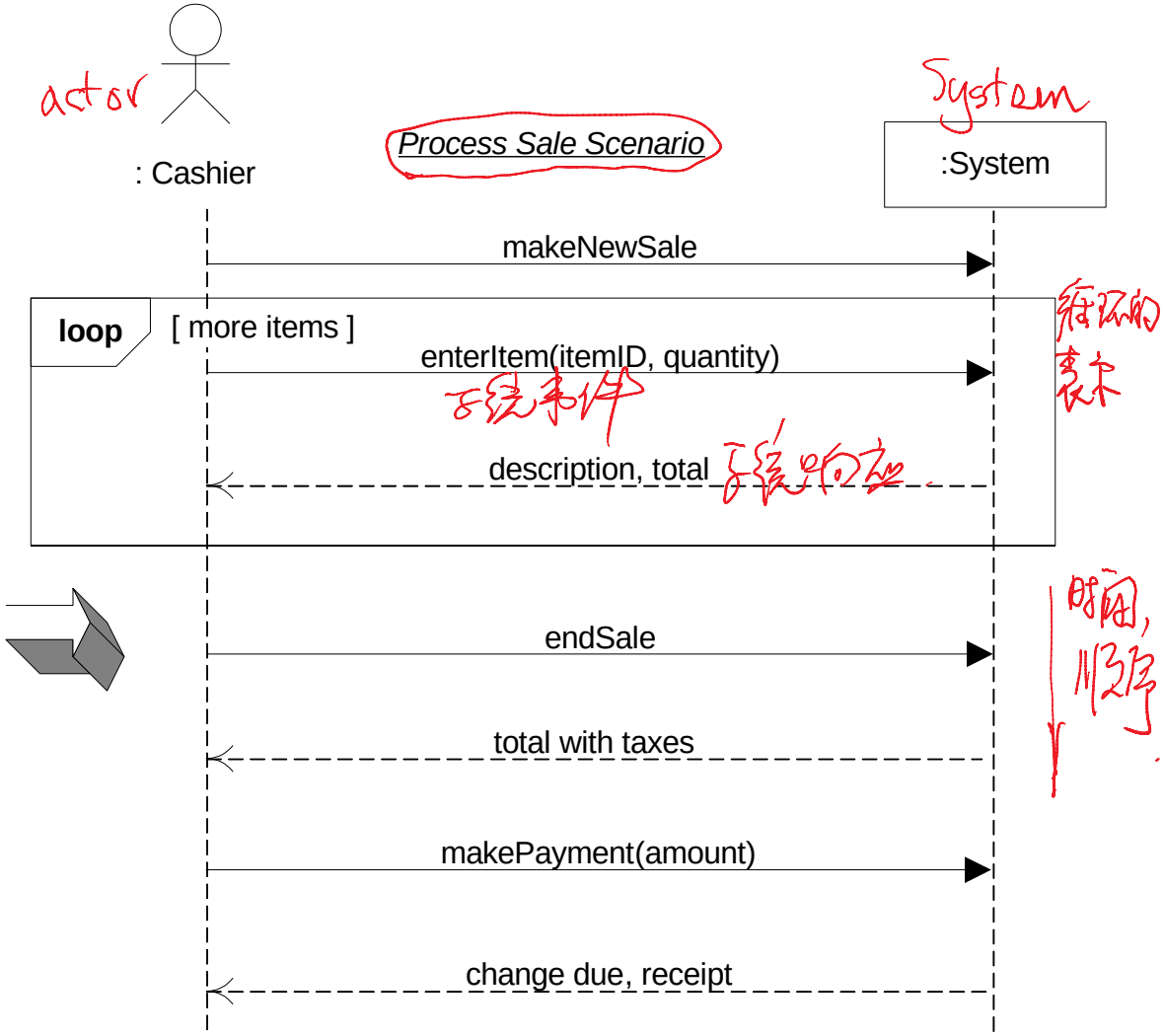
- SSDs are derived from use cases; they show one scenario.
- During interaction between system and actor, an actor generates **system events** to a system, usually requesting some **system operation** to handle the event.

use case text

Simple cash-only *Process Sale* scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...

use case scenario



Why Use SSDs?

- Software systems respond to events [**System behavior**].
- Systems react to 3 things:
  - External events,
  - Timer events,
  - And faults or exceptions
- Therefore, identify *precisely* the external events

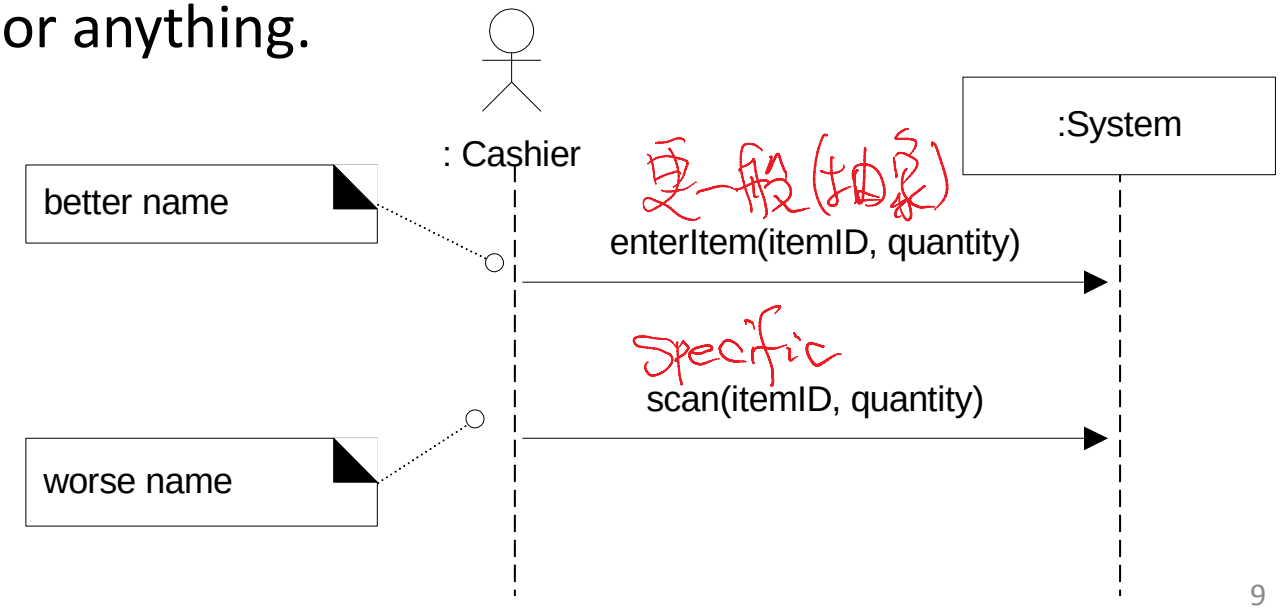
UML顺序图

Applying UML: Sequence Diagrams

- The UML does not define something called a "system" sequence diagram but simply a "sequence diagram."
- The qualification is used to emphasize its application to systems as black boxes.
- Later, sequence diagrams will be used in another context to illustrate the design of interacting software objects to fulfill work.

Naming Events and Operations

- Keep the names as abstract or conceptual as possible
- Start the name of the event with a verb
- enterItem(itemID) is better than scan(itemID)
  - it captures the *intent* of the operation
  - It remains abstract and noncommittal
  - It is not a design choices such as by via laser scanner, keyboard, voice input, or anything.



What SSD Info in the Glossary?

- Since SSD elements (operation name, parameters, return data) are terse
  - these may need proper explanation so that during design it is clear what is coming in and going out. 需要细化说明.
- The Glossary is a great place for these details.
  - Guideline: In general for many artifacts, show details in the Glossary
- E. g., in the Process Sale example, “receipt” is shown.
  - However, a receipt can be a complex document that should be defined.
  - the UP Glossary can have a *receipt* entry

Use Case for Process Sale

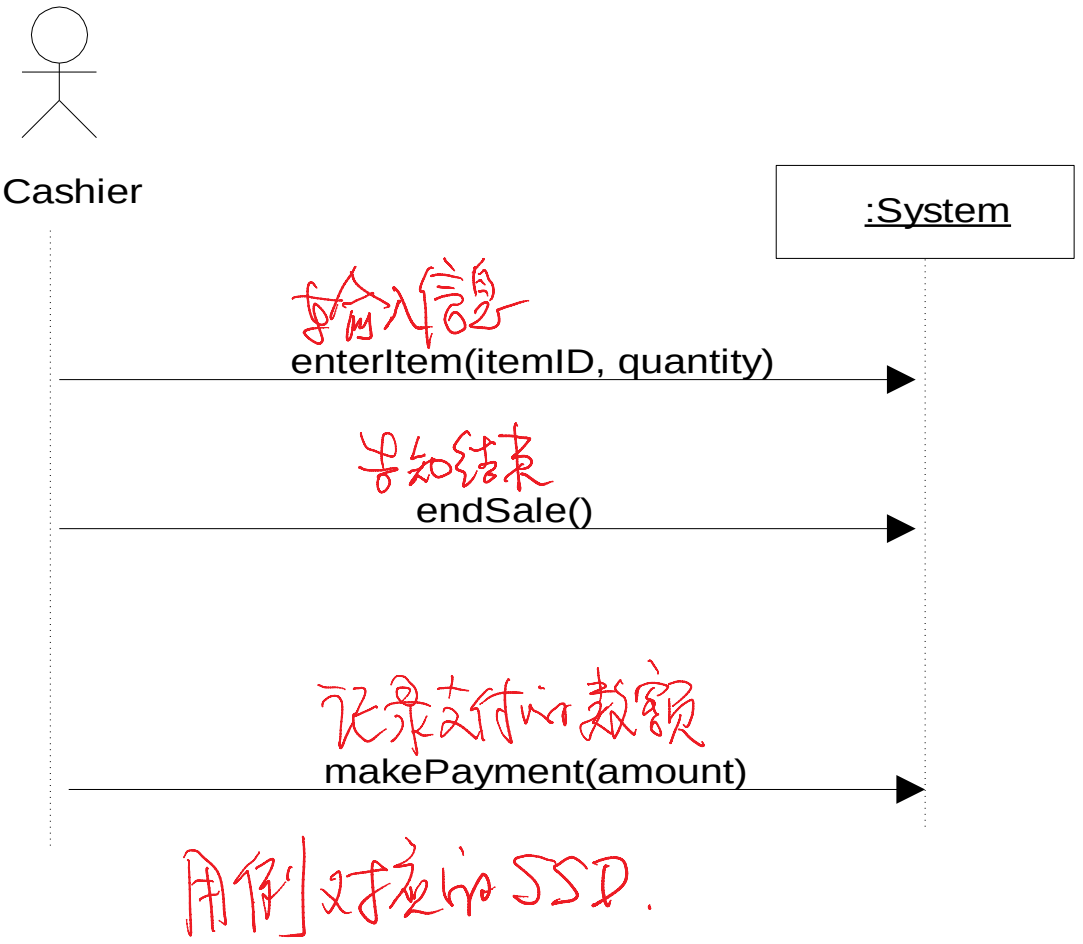
For all items, the Cashier records the itemID and quantity .

On completion of item entry, the Cashier indicates to the POS that the sale is complete.

The Cashier tells the Customer the total, and the Customer gives a payment to the Cashier.

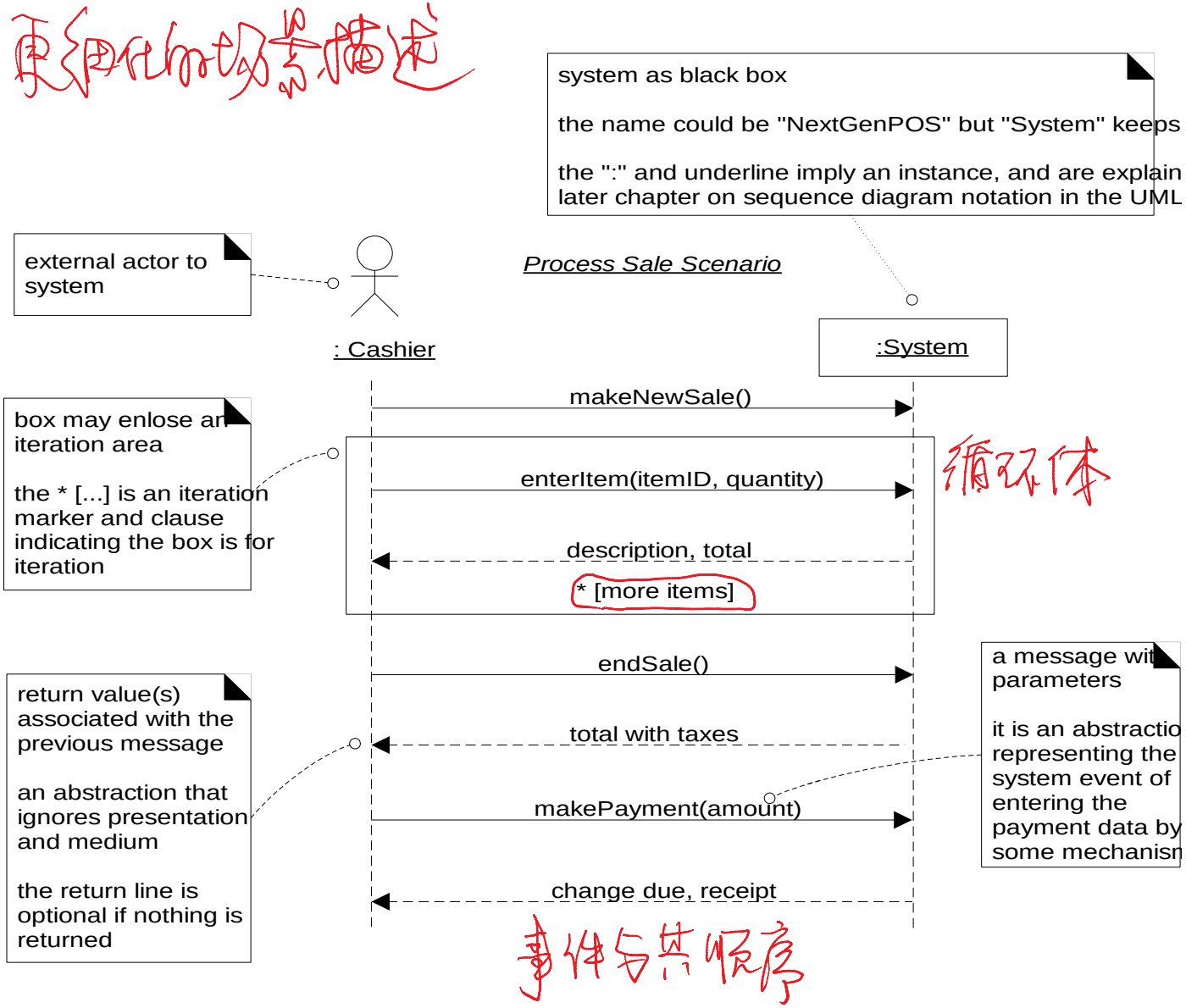
The Cashier records the cash received amount.

用例.



用例对应的SSD.

SSD for Process Sale Scenario



- There is a loop for getting more items that returns a description and a total
- The cashier can end the sale.
- The system returns the total with taxes

Iterative and Evolutionary SSDs

- SSDs are part of the Use-Case Model
- Draw an SSD for a **main success scenario** of each use case, and **frequent or complex alternative scenarios**.
- Don't create them for all scenarios unless necessary
- Use them to clarify a complex process
- Have lots of explanatory text with the diagram
- SSDs are not usually motivated in inception, unless you are doing rough estimating
- Create most of them during elaboration

What Are Operation Contracts?

- An operation is a specification of a transformation or query that an object may be called on to execute.
- In UML, an operation has a signature (name and parameters). A method is an implementation of an operation
- They use a pre- and post-condition form to describe detailed changes to objects in a domain model as the result of an operation
- They're part of the use case model
- They're too detailed to be part of inception; if you write them, do so during elaboration

Example

- An operation contract for the *enterItem* system operation.
- The critical element is the ***postconditions*** 核心是后置条件的描述

Operation: enterItem(itemID: ItemID, quantity: integer)
Cross References: Use Cases: Process Sale
Preconditions: There is a sale underway.
Postconditions:

- A SalesLineItem instance sli was created. 创建实例
- sli was associated with the current Sale. 形成关联
- sli.quantity became quantity. 修改属性
- sli was associated with a ProductDescription, based on itemID match. 形成关联

The elements of operation contracts.

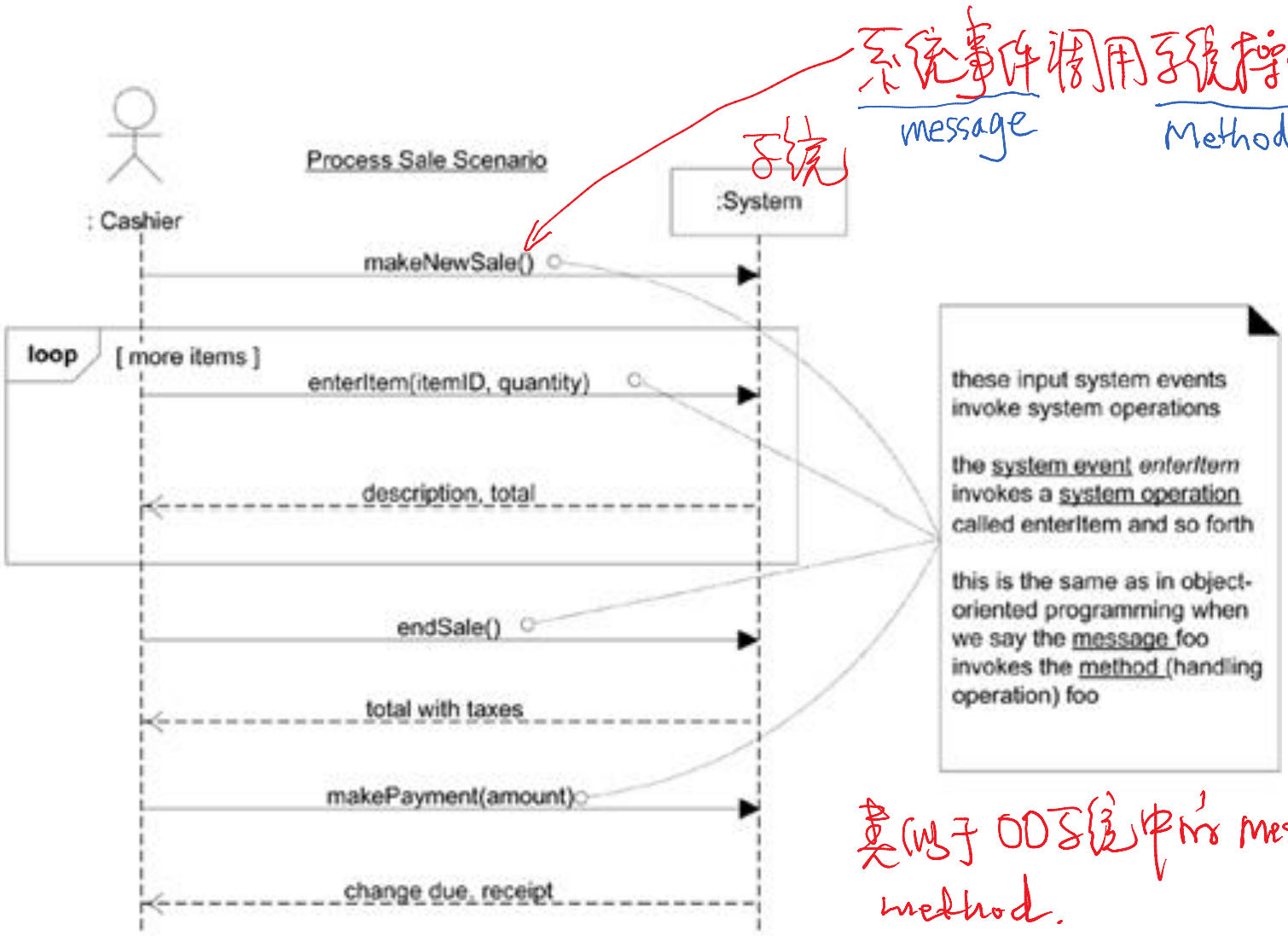
What are the Sections of a Contract?

- **Operation**
  - Name of operation, and parameters (Signature)
- **Cross References**
  - Use cases this operation can occur within
- **Preconditions**
  - Noteworthy assumptions about the state of the system or objects in the Domain Model before execution of the operation. These are non-trivial assumptions the reader should be told.
- **Postconditions** 最主要的部分
  - This is the most important section. The state of objects in the Domain Model after completion of the operation. Discussed in detail in a following section.

What is a System Operation?

- Input system events imply the system has system operations to handle the events
- The entire set of system operations, across all use cases, defines the public **system interface**,
- View the system as a single component or class.
  - In the UML, the system as a whole can be represented as one object of a class named (for example) *System*.

What is a System Operation?



系统事件调用系统操作.

message

Method

系统

是(对于OO系统中的) message and method.

Contract 中需要的内容

Postconditions

- Describe changes in the state of objects in the domain model. 操作导致的新的状态.
- Changes include instances created, associations formed or broken, and attributes changed
  - They are not actions to be performed during the operation.
  - They are results, observations about the objects after the operation
- Instance deletion postconditions are most rare, deletion may break an association 删除实例很少作为后置条件.

The postconditions fall into these categories

- These postconditions are expressed in the context of the Domain Model objects.
- To summarize:
  - Instance creation and deletion.
  - Attribute change of value.
  - Associations (to be precise, UML *links*) formed and broken.

后置条件的三种形式

Why Postconditions?

- They aren't always necessary. Usually the effects of the operations are clear
- Support fine detail in defining the outcome of an operation
- Could do it in use cases, but would make them too bulky

How to Write a Postcondition?

- Express in the past tense; they are observations about state changes from the operation, not an action
- "A SalesLineItem was created"(better) vs. Create a SalesLineItem (worse)
- One way I do this is to start with the phrase: "After the operation, the following things will be true: " This avoids past tense and passive voice.

How Complete?

- Complete set not necessary
- Use only when they add clarity
- In the spirit of Agile Modeling, treat their creation
  - as an initial best guess,
  - with the understanding they will not be complete
  - and that "perfect" complete specifications are rarely possible or believable.

Example: *EnterItem* Postconditions

- A SalesLineItem instance *sli* was created
  - **Instance Creation and Deletion**
- *Sli.quantity* became *quantity*
  - **Attribute Modification**
- *Sli* was associated with the current *Sale*
  - **Associations Formed and Broken**
- *Sli* was associated with a *ProductDescription* based on *itemID* match
  - **Associations Formed and Broken**

Should We Update the Domain Model?

- It's common during the creation of the contracts to discover the need to record new conceptual classes, attributes, or associations in the domain model.
- Do not be limited to the prior definition of the domain model; enhance it as you make new discoveries while thinking through operation contracts.
- Update your domain model as needed

发现新词

完善领域模型.

在构建Contracts时，可能发现新的领域类，把它们加入到领域模型中.

When are contracts useful?

- When the level of detail would make the use case awkward or too detailed 用例太过细节时
- When the number of things that must be true at the end of an operation is large 操作效果太过庞大时

How to Create and Write Contracts?

- Identify the operations from System Sequence Diagrams
- For operations that are **complex** and perhaps **not clear** in the use case, construct a contract
- Categories:
  - Instance creation and deletion
  - Attribute modification
  - Associations formed and broken
- Do not forget to include the *forming of associations*

Example: NextGen POS Contracts 1

System Operations of the Process Sale Use Case

- **Contract CO1: makeNewSale**
  - Operation:makeNewSale()
  - Cross References: Use Cases: Process Sale
  - Preconditions: none
  - Postconditions:
    - A Sale instance s was created (instance creation).
    - s was associated with a Register (**association formed**).
    - Attributes of s were initialized.
- **Contract CO2: enterItem**
  - Operation: enterItem(itemID: ItemID, quantity: integer)
  - Cross References: Use Cases: Process Sale
  - Preconditions: There is a sale underway.
  - Postconditions:
    - A SalesLineItem instance sli was created (instance creation).
    - sli was associated with the current Sale (**association formed**).
    - sli.quantity became quantity (attribute modification).
    - sli was associated with a ProductDescription, based on itemID match (**association formed**).

- Keep it as light as possible, and avoid all artifacts unless they really add value.

Example: NextGen POS Contracts 2

- **Contract CO3: endSale**
  - Operation: endSale()
  - Cross References: Use Cases: Process Sale
  - Preconditions: There is a sale underway.
  - Postconditions:
    - Sale.isComplete became true (attribute modification).
- **Contract CO4: makePayment**
  - Operation: makePayment(amount: Money)
  - Cross References: Use Cases: Process Sale
  - Preconditions:There is a sale underway.
  - Postconditions:
    - A Payment instance p was created (instance creation).
    - p.amountTendered became amount (attribute modification).
    - p was associated with the current Sale (**association formed**).
    - The current Sale was associated with the Store (**association formed**); (to add it to the historical log of completed sales)

Changes to the POS Domain Model

- These contracts suggest that the completion of item entry to the sale is not yet represented in the domain model.
- The *endSale* specification modifies it, and it is probably a good idea later during design work for the *makePayment* operation to test it, to disallow payments until a sale is complete (meaning, no more items to add).
- One way to represent this information is with an *isComplete* attribute in the *Sale*:

UML Constraint objects

- In UML, an operation is associated with a set of UML **Constraint** objects classified as preconditions and postconditions that specify **the semantics of the operation**.
- In UML, contracts can be applied to operations at any level of granularity.
- The pre- and post-condition format can be informal in natural language perfectly acceptable in the UML, and desirable to be easily understood.
- But also associated with the UML is a formal, rigorous language called the Object Constraint Language (**OCL**), which can be used to express constraints of UML operations.
- Unless there is a compelling practical reason to require people to learn and use the OCL, keep things simple and use natural language.

Operation Contracts within the UP

- In the UML, operations exists at many levels, from System down to fine-grained classes, such as Sale.
- Operation contracts for the System level are part of the Use-Case Model.
- Inception phase
  - Contracts are not motivated during inception, they are too detailed.
- Elaboration phase
  - If used at all, most contracts will be written during elaboration, when most use cases are written.
  - **Only write contracts for the most complex and subtle system operations.**