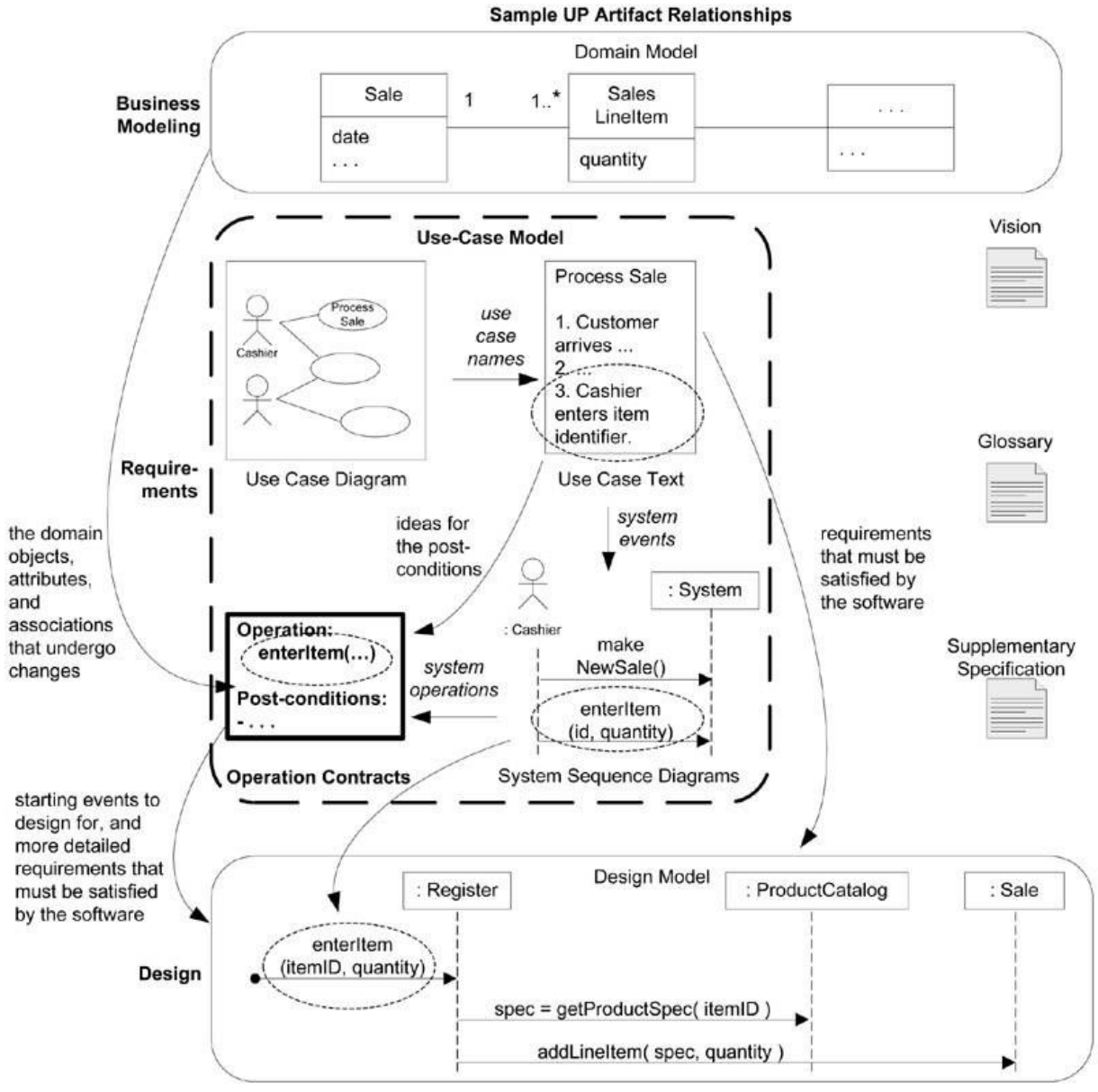
Object-Oriented Software Analysis and Design

School of Computer Science
University of Windsor

Operation Contracts (contd.)



Operation Contracts

- Use cases or system features are the main ways in the UP to describe system behavior, and are usually sufficient.
- Sometimes a more detailed or precise description of system behavior has value.
- **Operation contracts** use a **pre- and post-condition form** to describe detailed changes to objects in a domain model, as the result of a system operation.
- A domain model is the most common OOA model, but **operation contracts** and **state models** can also be useful OOA-related artifacts.

Operation Contracts (contd.)

- Operation contracts may be considered part of the **UP Use-Case Model** because they provide more analysis detail on the effect of the system operations implied in the use cases.

- The prime inputs to the contracts are:
  1. the system operations identified in SSDs (such as `enterItem`),
  2. the domain model, and
  3. domain insight from experts.

- *The contracts can in turn serve as input to the object design, as they describe changes that are likely required in the software objects or database.*

An Example: Operation Contracts

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)*.

The categorizations such as "*(instance creation)*" are a learning aid, not properly part of the contract.

Definition: What are the Sections of a Contract?

Operation:	Name of operation, and parameters
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:	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.

Definition: What is a System Operation?

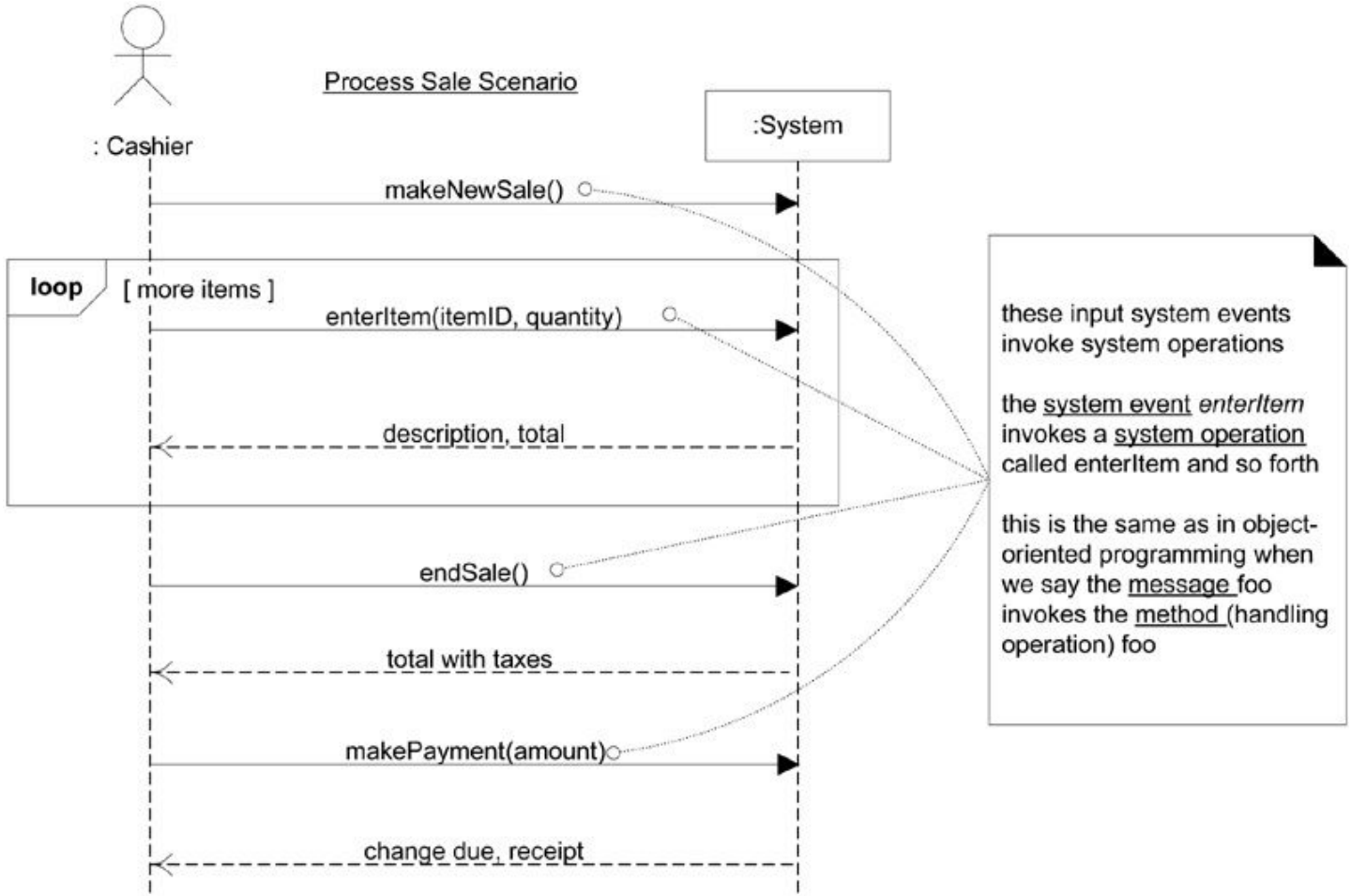


Figure: *SSD. System operations handle input system events.*

Definition: What is a System Operation? (contd.)

- Operation contracts may be defined for system operations–operations that the system as a black box component offers in its public interface.
- *System operations can be identified while sketching SSDs*.
- To be more precise, the SSDs show system events or I/O messages relative to the system.

Definition: What is a System Operation? (contd.)

- Input system events imply the system has system operations to handle the events, just as an OO message (a kind of event or signal) is handled by an OO method (a kind of operation).
- The entire set of system operations, across all use cases, defines the public **system interface**, viewing 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.

Definition: Postconditions

- The postconditions describe **changes in the state of objects** in the domain model. Domain model state changes include
  - instances created/deleted
  - associations formed or broken, and
  - attributes changed.

- **Postconditions are not actions to be performed during the operation**; rather, they are observations about the domain model objects that are true when the operation has finished after the smoke has cleared.

- The postconditions fall into these categories:
  - Instance creation and deletion.
  - Attribute change of value.
  - Associations formed and broken.

How are Postconditions Related to the Domain Model?

- These postconditions are expressed in the context of the Domain Model objects.
- What instances can be created?
  - those from the Domain Model.
- What associations can be formed?
  - those in the Domain Model

Why Postconditions?

- First, they aren't always necessary. But sometimes more detail and precision is useful. Contracts offer that.
- The postconditions **support fine-grained detail and precision in declaring what the outcome of the operation must be**.
  - It is also possible to express this level of detail in the use cases, but undesirable—they would be too verbose and low-level detailed.
- A contract is an excellent **tool of requirements analysis or OOA that describes in great detail the changes required by a system operation** (in terms of the domain model objects) without having to describe how they are to be achieved.
- In other words, the design can be deferred, and we can focus on the analysis of *what* must happen, rather than *how* it is to be accomplished.

Guideline: How to Write a Postcondition?

- Express postconditions in the **past tense** to emphasize they are observations about state changes that arose from an operation, not an action to happen. That's why they are called postconditions!

  For example:

  - (better) A `SalesLineItem` was created.

    rather than

  - (worse) Create a `SalesLineItem`, or, A `SalesLineItem` is created.

Guideline: How Complete Should Postconditions Be? Agile vs. Heavy Analysis

- Generating a complete and detailed set of postconditions for all system operations is not likely or necessary.
- 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.
- But understanding that light analysis is realistic and skillful doesn't mean to abandon a little investigation before programming that's the other extreme of misunderstanding.

Example: `enterItem` Postconditions

This example dissects the motivation for the postconditions of the enterItem system operation.

- Instance Creation and Deletion

  For example,

  After the `itemID` and quantity of an item have been entered, what new object should have been created?

  - A `SalesLineItem`. Thus:
  - A `SalesLineItem` instance sli was created (instance creation).
  - Note the naming of the instance. This name will simplify references to the new instance in other post-condition statements.

Example: `enterItem` Postconditions (contd.)

- Attribute Modification

  For example,

  After the `itemID` and quantity of an item have been entered by the cashier, what attributes of new or existing objects should have been modified?

  - The quantity of the `SalesLineItem` should have become equal to the quantity parameter. Thus:
  - sli.quantity became quantity (attribute modification).

Example: `enterItem` Postconditions (contd.)

- Associations Formed and Broken

  For example,

  - After the `itemID` and quantity of an item have been entered by the cashier, what associations between new or existing objects should have been formed or broken?
  - The new SalesLineItem should have been related to its Sale, and related to its ProductDescription. Thus:
    - sli was associated with the current Sale (association formed).
    - sli was associated with a ProductDescription, based on `itemID` match (association formed).

Guideline: Should We Update the Domain Model?

- In iterative and evolutionary methods (and reflecting the reality of software projects), all analysis and design artifacts are considered partial and imperfect, and evolve in response to new discoveries.

Guideline: When Are Contracts Useful?

- In the UP, **the use cases are the main repository of requirements for the project.** They may provide most or all of the detail necessary to know what to do in the design, in which case, contracts are not helpful. However, there are situations where **the details and complexity of required state changes are awkward or too detailed to capture in use cases**.

For example:

consider an airline reservation system and the system operation addNewReservation. The complexity is very high regarding all the domain objects that must be changed, created, and associated. These fine-grained details can be written up in the use case, but it will make it extremely detailed (for example, noting each attribute in all the objects that must change).

Guideline: When Are Contracts Useful? (contd.)

- Observe that the postcondition format offers and encourages a very precise, analytical language that supports detailed thoroughness.

- If developers can comfortably understand what to do without them, then avoid writing contracts.

Guideline: How to Create and Write Contracts

1. Identify system operations from the SSDs.

2. For system operations that are complex and perhaps subtle in their results, or which are not clear in the use case, construct a contract.

3. To describe the postconditions, use the following categories:
   - instance creation and deletion
   - attribute modification
   - associations formed and broken

Guideline: How to Create and Write Contracts (contd.)

- As mentioned, write the postconditions in a declarative, passive past tense form (was . . .) to emphasize the observation of a change rather than a design of how it is going to be achieved.

  For example,

  - (better) A `SalesLineItem` was created.
  - (worse) Create a `SalesLineItem`.

- Remember to establish an association between existing objects or those newly created.

  For example,

  it is not enough that a new `SalesLineItem` instance is created when the `enterItem` operation occurs. After the operation is complete, it should also be true that the newly created instance was associated with `Sale`; thus: The `SalesLineItem` was associated with the `Sale` (association formed).

Guideline: How to Create and Write Contracts (contd.)

What's the Most Common Mistake?

- The most common problem is forgetting to include the **forming of associations**.
  - Particularly when new instances are created, it is very likely that associations to several objects need be established. Don't forget!

Example-1: NextGen POS Contracts-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.

Example-2: NextGen POS Contracts-System Operations of the *Process Sale* Use Case (contd.)

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).

Example-3 & 4: NextGen POS Contracts-System Operations of the *Process Sale* Use Case (contd.)

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

- There is at least one point suggested by these contracts that is not yet represented in the domain model: completion of item entry to the sale.

- 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`:

Operation Contracts Within the UP

- Inception
  - Contracts are not motivated during inception—they are too detailed.
- Elaboration
  - 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.*

It's Quiz Time

1. Operation contracts may be considered part of the UP Business Modeling (True or False)

2. The postconditions support fine-grained detail and precision in declaring what the outcome of the operation must be. (True or False)

3. "Identify system operations from the SSDs" is one of the advices to create contracts. (True or False)