**USE-CASE MODEL: ADDING DETAIL WITH OPERATION CONTRACTS**

**Introduction**

·       Contracts for operations define system behavior

·       Describe the outcome of executing system operation in terms of state changes to domain objects.

**Contracts**

·       Contracts describe detailed system behavior in terms of state changes to objects in the Domain Model, after a system operation has executed.

·       Contracts are defined for system operations

operations that the system offers in its public interface to handle incoming system events.

·       System operations can be identified by discovering these system events

|  |
| --- |
|  |
|  | http://csis.pace.edu/~marchese/CS616/Lec6/se_l6_files/image002.gif |

Figure: System operations handle input system events.

·       Entire set of system operations, across all use cases, defines the public system interface

o      UML - the system as a whole can be represented by a class.

**Example Contract: enteritem**

**Contract C02: enteritem**

|  |  |
| --- | --- |
| **Operation:****Cross References:****Preconditions:**  | enterltem(itemlD: ItemID, quantity: integer)Use Cases: Process SaleThere is a sale underway.  |
| **Postconditions:** | — A SalesLineltem 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 ProductSpecification, based on itemlD match (association formed). |

**Contract Sections - Template**

A description of each section in a contract is shown in the following schema.

|  |  |
| --- | --- |
| **Operation:** | Name of operation, and parameters  |
| **Cross References:** | (optional) 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 will not be tested within the logic of this operation, are assumed to be true, and are non-trivial assumptions the reader should know were made.  |
| **Postconditions:** | The state of objects in the Domain Model after completion of the operation. Discussed in detail in a following section.  |

**Postconditions – General Discussion**

·       Postconditions are declarations about the Domain Model objects that are true when the operation has finished

·       Domain Model state changes include:

·       Instance creation and deletion.

·       Attribute modification.

·       Associations formed and broken

·       Advantage of Postconditions

o      Describes state changes required of a system operation without having to describe *how*they are to be achieved.

o      e.g. postconditions:

— A SalesLineltem 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 ProductSpecification, based on itemlD match (association formed).

* No comment is made about how a *SalesLineltem*instance is created, or associated with a *Sale.*

·       Express postconditions in the past tense, to emphasize they are declarations about a state change in the past.

1.     For example:

·    (better) A *SalesLineltem*was created.

·    (worse) Create a *SalesLineltem.*

**Postconditions –enterItem Postconditions Discussion**

*Instance Creation and Deletion*

After the *itemlD*and *quantity*of an item have been entered, the new object *SalesLineltem*is created:

·         A *SalesLineltem*instance *sli*was created (instance creation).

*Attribute Modification*

After the itemlD and quantity of an item have been entered by the cashier, the *quantity*of the *SalesLineltem*should have become equal to the *quantity*parameter:

*·         sli.quantity*became *quantity*(attribute modification).

*Associations Formed and Broken*

After the *itemlD*and *quantity*of an item have been entered by the cashier, the new *SalesLineltem*should have been related to its *Sale,*and related to its *ProductSpecification*:

·                 *sli*was associated with the current *Sale*(association formed).

·                 *sli*was associated with a *ProductSpecification,*based on *itemlD*match (association formed).

·       **When Are Contracts Useful ?**

Use cases are main repository of project’s requirements.

Situations where details and complexity of required state changes are awkward to capture in use cases.

e.g. airline reservation system and the system operation *addNewReservation.*

**Guidelines for creating Contracts**

1.       Identify system operations from the SSDs.

2.       Construct a contract for system operations that are complex or subtle in their results, or which are not clear in the use case

3.               Describe postconditions, using the following categories:

o        instance creation and deletion
o        attribute modification
o        associations formed and broken

·                 State postconditions in a declarative, passive past tense form to emphasize the declaration of a state change rather than a design of how it is going to be achieved.

o      e.g.

o   (better) A *SalesLineltem***was**created.

o   (worse) Create a *SalesLineltem.*

·                 Establish a memory between existing objects or those newly created by defining the forming of an association.

o      e.g.,

§       Not enough that new *SalesLineltem*instance is created when the *enterltem*operation occurs.

§       After the operation is complete, it should be true that the newly created instance was associated with *Sale*:

The *SalesLineltem*was associated with the *Sale*(association formed).

**NextGen POS Example: Contracts**

*System Operations of Process Sale*

**Contract COl: inakeNewSale**

|  |  |
| --- | --- |
| **Operation:****Cross References:****Preconditions:**  | MakeNewSale()Use Cases: Process Salenone  |
| **Postconditions:**  | — A Sale instance s was created (instance creation).— s was associated with the Register (association formed).— Attributes of s were initialized.  |

**Contract CO2: enterltem**

|  |  |
| --- | --- |
| **Operation:****Cross References:****Preconditions:**  | enterltem(itemlD : ItemID, quantity: integer)Use Cases: Process SaleThere is a sale underway.  |
| **Postconditions:**  | — *A SalesLineltem instance sli was created (instance creation).*—       *s/i was associated with the current Sale (association tormed).*—       *sli.quantity became quantity (attribute moditication).*— *sli was associated with a ProductSpecification, based on itemlD match (association tormed).* |

**Contract C03: endSale**

|  |  |
| --- | --- |
| **Operation:****Cross References:****Preconditions:**  | endSale()Use Cases: Process SaleThere is a sale underway |
| **Postconditions:** | - Sale.isComplete became true (attribute modification) |

**Contract C04: makePayment**

|  |  |
| --- | --- |
| **Operation:****Cross References:****Preconditions:**  | makePayment( amount: Money)Use Cases: Process SaleThere is a sale underway.  |
| **Postconditions:**  | —       *A Payment instance p was created (instance creation).*—       *p.amount Tendered became amount (attribute moditication).*—       *p was associated with the current Sale (association tormed).*—       *The current Sale was associated with the Store (association tormed); (to add it to the historical log ot completed sales)*  |

**Contracts, Operations, and the UML**

·       UML formally defines **operations**:

An operation is a specification of a transformation or query that an object may be called to execute

·       A **method**(in the UML) is an implementation of an operation.

·       UML operation has a:

1.     **signature**(name and parameters)

2.     **operation specification -** describes the effects produced by executing the operation (postconditions).

·       Contracts applied to operations at any level of granularity:

o      e.g.

§       the public operations (or interface) of a subsystem

§       an abstract class

**Operation Contracts Within the UP**

*Phases*

**Inception**—— No - too detailed.

**Elaboration**——

·       Most contracts written during elaboration, when most use cases are written.

·       Only write contracts for the most complex and subtle system operations.