System Sequence Diagrams and Contracts

Eunjee Song
Computer Science Department
Baylor University

Still doing... Requirements Analysis

- Focusing on the WHAT not the HOW
- System sequence diagrams and how they relate to use cases

System Sequence diagrams

- Describes in more detail a scenario in a use case
  - Created from the text of the use case
  - A kind of UML sequence diagram
  - SSD is a simplified version of SD
  - useful for requirements analysis
  - More general version of sequence diagrams: later, when talking about design
  - Example: Process Sale for POS system

Specifying required behavior using the UML

- Class models describe objects and their relationships
  - Behavior can be specified in terms of operation pre and postconditions, but behavior is not the primary focus of a class model
- Behavioral models at the requirements level
  - Interaction models: describe interactions between actors and the system
  - Contracts: Specify operations invoked by actors.
### Elements of SSD

- **Actors**
  - UML notation for an object
- **System events**
  - Cashier generates `makeNewSale`, `enterItem`, and `endSale` system events
- **(Optional) Information from the system back to the actors**
  - Item description, running total, etc.

### Different Kinds of Arrows

- **Synchronous flow**: e.g., Procedure call
- **Asynchronous flow**: Return

### SSDs and Use Cases for “Process Sale”

#### 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 presents total with taxes calculated.
5. System presents receipt.
6. System logs completed sale and sends sale info to the external **Accounting** system and to the external **Inventory** system.

### SSD for “Process Sale (cont)"

- **System** presents total with taxes. To determine taxes, System uses an external **Tax Calculator**.

### Payment SSDs

- **Cashier** enters cash amount tendered, and **System** presents change due
- **System** presents receipt
- **System** logs completed sale and sends sale info to the external **Accounting** system and to the external **Inventory** system.
SSD Sketches

Events and return values are abstractions
- Independent of mechanism & representation
- makePayment(amount)
  - Shows input info
  - Looks like a method call, but is really an abstraction of an event
- Name: should capture the intent
  - Avoid specifying implementation choices
    - enterItem(itemID) is better than scan(itemID)

Timeline for SSDs
- SSDs are created during elaboration
  - Clarify the major events that the system should be able to handle
  - Later we design objects to handle these events (object-oriented design)
- SSDs are created for some chosen scenarios from the current iteration
  - Happy path + frequent/complex alternatives.

Abstractions in SSDs
- Events and return values are abstractions
- makePayment(amount)
  - Shows input info
  - Looks like a method call, but is really an abstraction of an event
- Name: should capture the intent
  - Avoid specifying implementation choices
    - enterItem(itemID) is better than scan(itemID)

Contracts
- A system operation is a behavior that is invoked by a user
- A contract specifies a system operation in terms of pre- and postconditions
  - Precondition: what must be true about concepts represented in the domain model before the start of the operation
  - Postcondition: what must be true about concepts represented in the domain model after the operation has ended
    - Include instances created or deleted, new attribute values, links created or deleted

Contract Definition
- A system operation is a behavior that is invoked by a user
- A contract specifies a system operation in terms of pre- and postconditions
  - Precondition: what must be true about concepts represented in the domain model before the start of the operation
  - Postcondition: what must be true about concepts represented in the domain model after the operation has ended
    - Include instances created or deleted, new attribute values, links created or deleted

Contract Examples
- Stand-alone system
  - No external inventory system
- Adding items to the inventory
  - An instance of Item is created
  - The new instance is associated with the Store
- Removing items from the inventory
  - The association is destroyed and the Item instance is also destroyed
Contract Example 1 - makeNewSale

**Operation:** makeNewSale()

**Cross References:** Use Case – Process Sale

**Precondition:** An authenticated cashier, c, is logged-in at a Register, r.

**Postcondition:**
- A Sale instance, s, was created;
- Attributes of s were initialized.
- s was associated with r.
Contract Example 2 - enterItem

**Operation:** enterItem(itemId:ItemID, quantity:integer)

**Cross References:** Use Case – Process Sale

**Precondition:** current sale object, s, exists

**Postcondition:**
- A SalesLineItem instance, sli, was created;
- sli was associated with s
- sli.quantity = quantity
- sli was associated with a ProductDescription, based on itemID match

---

Contract Example - enterItem

**Precondition:** current sale object, s, exists

**Postcondition:**
- A SalesLineItem instance, sli, was created;
- sli was associated with s
- sli.quantity = quantity
- sli was associated with an Item, based on itemID match
Process Registration Use Case for the University Registration System

- A student arrives at the Registrar’s office in order to enroll in one or more classes. The clerk will access the terminal in order to initiate a new enrollment session (you may assume that they already have been authenticated by the system). Each enrollment session captures the date and time it was initiated. The clerk will proceed to enroll the student in each class requested. For each class, the clerk will enter the student name and identification number and the class identification number. In response to each entry, the system will display a description and a confirmation. At the end of the session, the system will display a confirmation of the procedure and the total amount of tuition fees due. The clerk will then initiate a payment of tuition fees and the system will respond with the change due and a receipt.
Domain Model (incomplete)

SSD for Process Registration
Operation Contract

Contract CO1: makeNewEnrollment
Operation: makeNewEnrollment ()
Cross References: Use Cases: Process Registration.
Pre-conditions:

Post-conditions:

Combined Fragment Types

- **Loop** (*loop*)
  - Optional guard: [<min>, <max>, <Boolean-expression>]
  - No guard means no specified limit

- **Alternatives** (*alt*)
  - choice of behaviors – at most one will execute
  - depends on the value of the guard (“else” guard supported)

- **Option** (*opt*)
  - Special case of alternative

- **Parallel** (*par*)
  - Concurrent (interleaved) sub-scenarios
Acknowledgements: This lecture material is prepared based on slides by Craig Larman and Dr. Robert France (CSU)