Application Analysis

Prof. Kirtankumar Rathod

Department of Computer Science

Indus University

Application Interaction Model

- Most domain models are static and operations are unimportant, because a domain as a whole usually doesn't do anything.
- The focus of domain modeling is on building a model of intrinsic concepts.

Application Interaction Model

You can construct an application interaction model with the following steps.

- Determine the system boundary. [13.1.1]
- Find actors. [13.1.2]
- Find use cases. [13.1.3]
- Find initial and final events. [13.1.4]
- Prepare normal scenarios. [13.1.5]
- Add variation and exception scenarios. [13.1.6]
- Find external events. [13.1.7]
- Prepare activity diagrams for complex use cases. [13.1.8]

Organize actors and use cases. [13.1.9]

- Check against the domain class model. [13.1.10]

Determining the System Boundary

- During analysis, you determine the purpose of the system and the view that it presents to its actors. During design, you can change the internal implementation of the system as long as you maintain the external behavior.
- Usually, you should not consider humans as part of a system, unless you are modeling a human organization, such as a business or a government department.
- Humans are actors that must interact with the system, but their actions are not under the control of the system.
- However, you must allow for human error in your system.

Finding Actors

- Once you determine the system boundary, you must identify the external objects that interact directly with the system. These are its actors. Actors include humans, external devices, and other software systems. The important thing about actors is that they are not under control of the application, and you must consider them to be somewhat unpredictable.

Finding Use Cases

- For each actor, list the fundamentally different ways in which the actor uses the system.
- Each of these ways is a use case. The use cases partition the functionality of a system into a small number of discrete units, and all system behavior must fall under some use case.
- You may have trouble deciding where to place some piece of marginal behavior.
- Keep in mind that there are always borderline cases when making partitions; just make a decision even if it is somewhat arbitrary.

Finding Initial and Final Events

- Use cases partition system functionality into discrete pieces and show the actors that are involved with each piece, but they do not show the behavior clearly.
- To understand behavior, you must understand the execution sequences that cover each use case.
- You can start by finding the events that initiate each use case. Determine which actor initiates the use case and define the event that it sends to the system.

Preparing Normal Scenarios

- For each use case, prepare one or more typical dialogs to get a feel for expected system behavior.
- These scenarios illustrate the major interactions, external display formats, and information exchanges.
- A scenario is a sequence of events among a set of interacting objects.
- Think in terms of sample interactions, rather than trying to write down the general case directly.
- This will help you ensure that important steps are not overlooked and that the overall flow of interaction is smooth and correct.

Adding Variation and Exception Scenarios

- After you have prepared typical scenarios, consider "special" cases, such as omitted input, maximum and minimum values, and repeated values.
- Then consider error cases, including invalid values and failures to respond. For many interactive applications, error handling is the most difficult part of development.
- If possible, allow the user to abort an operation or roll back to a well-defined starting point at each step.
- Finally consider various other kinds of interactions that can be overlaid on basic interactions, such as help requests and status queries.

Finding External Events

- Examine the scenarios to find all external events-include all inputs, decisions, interrupts, and interactions to or from users or external devices.
- An event can trigger effects for a target object. Internal computation steps are not events, except for computations that interact withhe external world.
- Use scenarios to find normal events, but don't forget unusual events and error conditions.

Preparing Activity Diagramsfor Complex UseCases

- Sequence diagrams capture the dialog and interplay between actors, but they do not clearly show alternatives and decisions.
- For example, you need one sequence diagram for the main flow of interaction and additional sequence diagrams for each error and decision point.
- Activity diagrams let you consolidate all this behavior by documenting forks and merges in the control flow.
- It is certainly appropriate to use activity diagrams to document business logic during analysis, but do not use them as an excuse to begin implementation.

Organizing Actors and Use Cases

- The next step is to organize use cases with relationships (include, extend, and generalization).
- This is especially helpful for large and complex systems.
- As with the class and state models, we defer organization until the base use cases are in place.
- Otherwise, there is too much of a risk of distorting the structure to match preconceived notions.

Checking Against the Domain Class Model

- At this point, the application and domain models should be mostly consistent.
- The actors, use cases, and scenarios are all based on classes and concepts from the domain model.
- Recall that one of the steps in constructing the domain class model is to test access paths. In reality, such testing is a first attempt at use cases.
- Cross check the application and domain models to ensure that there are no inconsistencies.
- Examine the scenarios and make sure that the domain model has all the necessary data. Also make sure that the domain model covers all event parameters.

Sequence Diagram

Prof. Kirtankumar Rathod

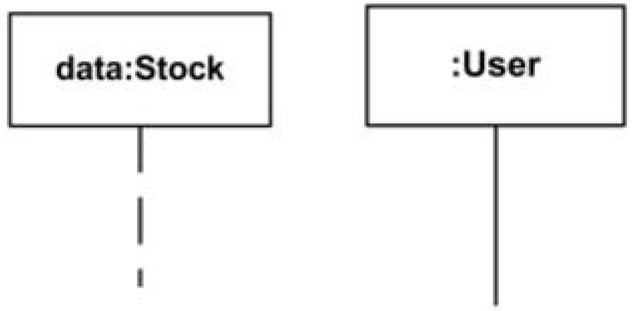
Department of Computer Science

Indus University

What is Sequence Diagram?

- Sequence diagram is the most common kind of Interaction diagram, which focuses on the message interchange between a number of lifelines.
- Sequence diagram is a UML behavior diagram.
- Sequence diagram depicts the inter-object behavior of a system, ordered by time.
- The major components of a Sequence Diagram are:
    - Lifeline
    - Messages
    - Interaction Fragments

- Lifeline is an element which represents an individual participant in the interaction.
- Lifelines represent only one interacting entity.
- A lifeline is shown using a symbol that consists of a rectangle forming its "head" followed by a vertical line (which may be dashed) that represents the lifetime of the participant.



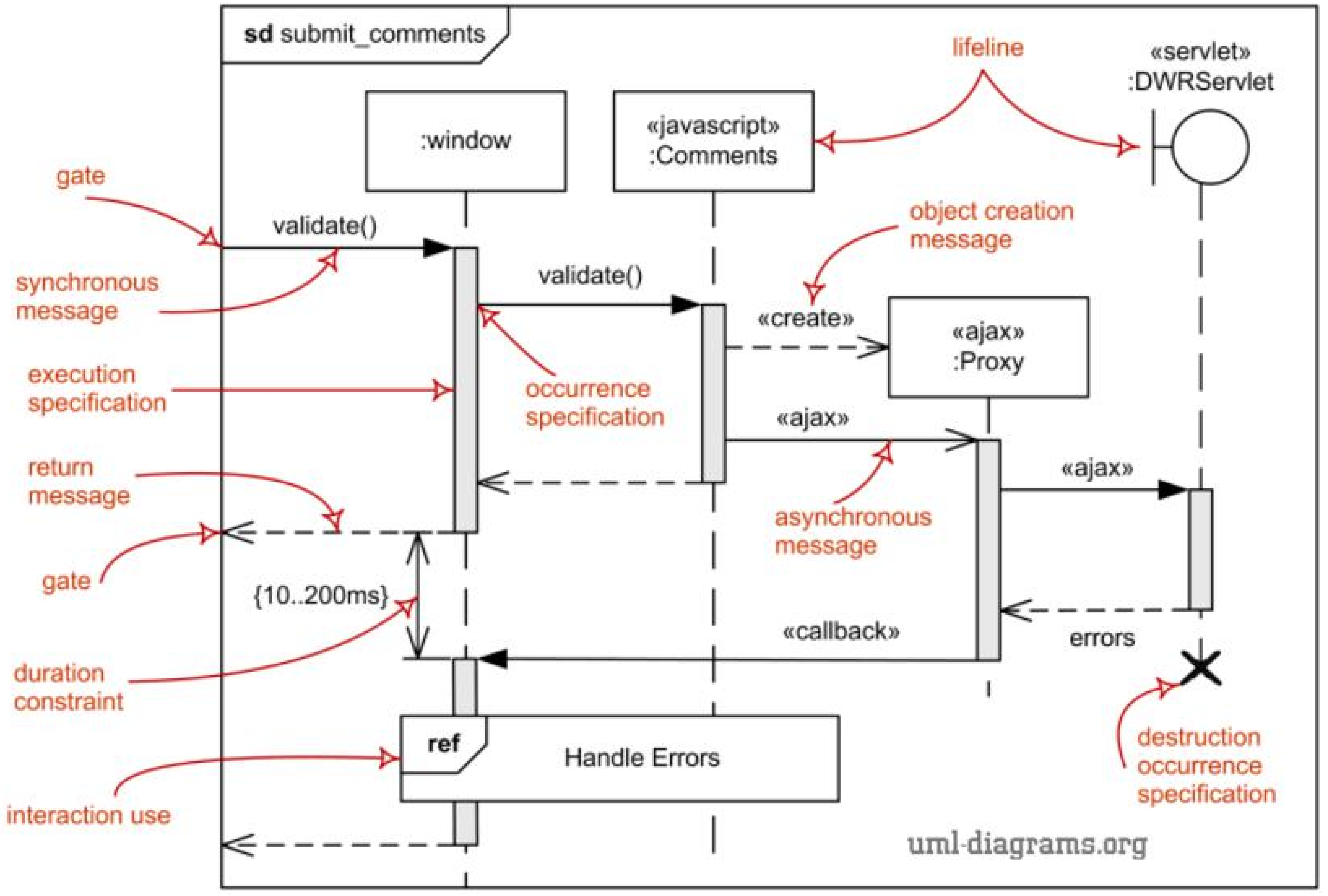
- Message is an element that defines one specific kind of communication between lifelines of an interaction.
- There are 2 major types of message in Sequence Diagram.
  - Messages by Action Type
  - Messages by Presence of Events

  - Message by Action Type: A message reflects either an operation call and start of execution or a sending and reception of a signal.
  - Message by Presence of Events: A message depends on whether message send event and receive events are present

The various types of Messages by Action type are:

- synchronous call
- asynchronous call / signal
- create
- delete
- reply

- The various types of Messages by Presence of Events are:
  - complete message
    - The semantics of a complete message is the trace
    - <sendEvent, receiveEvent>
    - Both sendEvent and receiveEvent are present
  - lost message
  - found message
  - unknown message (default) { both sendEvent and
  - receiveEvent are absent (should not appear)


sd submit_comments
lifeline
«servlet» :DWRServlet
:window
«javascript» :Comments
gate
validate()
synchronous message
validate()
object creation message
«create»
«ajax» :Proxy
execution specification
occurrence specification
«ajax»
return message
«ajax»
asynchronous message
gate
{10..200ms}
errors
«callback»
duration constraint
ref
Handle Errors
destruction occurrence specification
interaction use
uml-diagrams.org