Lecture 14: Interaction Diagrams

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2005

Goals for this Lecture

- Introduce the notion of *interactions* within OO designs
- Review the UML Notation for Interaction diagrams

Behavioral Modeling

- Interactions and Interaction diagrams allow the dynamic behavior of a system to be modeled
  - Class diagrams allow the static structure of a system to be modeled
- UML has two diagrams for interactions
  - sequence diagrams
  - collaboration diagrams

Interactions

- OO systems do not sit idle
  - their objects are constantly interacting with each other by sending messages
- Formally, an interaction is
  - a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose

Interaction Diagrams

- Interaction diagrams provide a notation for specifying interactions, including notations for
  - objects, links, messages, and sequencing
- Interaction diagrams allow analysis of
  - the flow of messages in a system over time
  - the structural relationships between objects and how messages are passed within that structure
- Interaction diagrams can be applied to
  - classes, operations, components, use cases, etc.

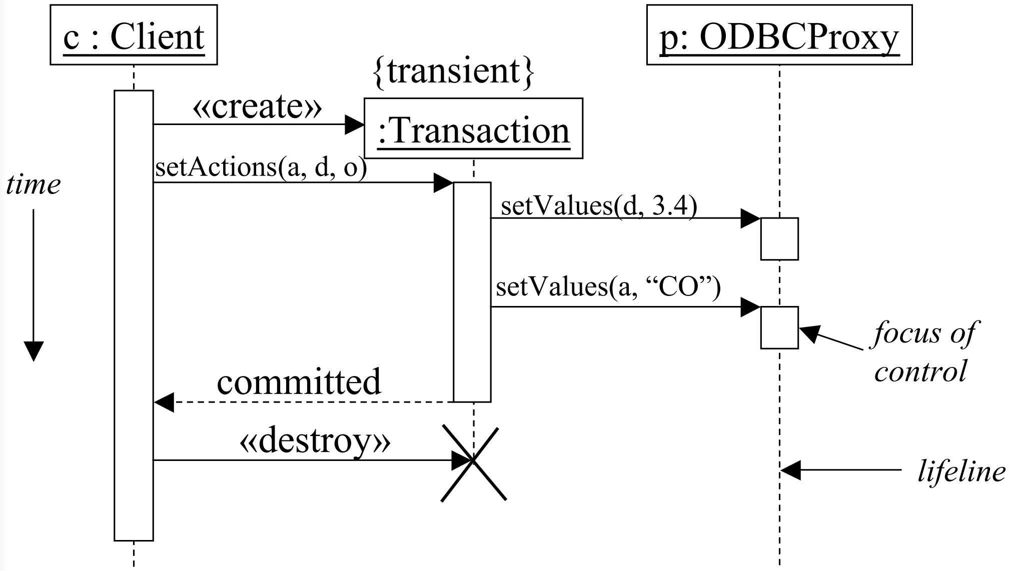
Quick Overview of Concepts

- Objects - instances of classes
- Links - instances of associations
- Messages - a request made by one object on another object; a message can only be sent across a link
- Sequencing - messages can be sequenced by time; as we shall see, message order can be indicated via numbers or via a top-to-bottom order
  - a sequence is valid only for a particular thread;
  - UML can specify synchronization across threads using a variety of constructs; we will see these in action soon!

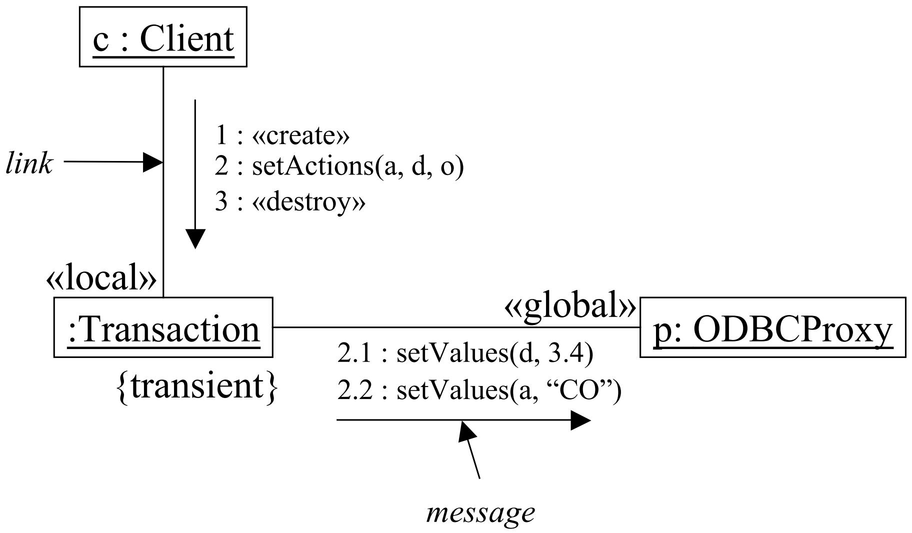
Interaction Diagrams

- Two types of interaction diagrams
  - – sequence diagrams
    - useful for modeling messages over time
  - – collaboration diagrams
    - useful for modeling messages across object structures

Example: Sequence Diagram



Example: Collaboration Diagram



c : Client

link

1 : «create»
2 : setActions(a, d, o)
3 : «destroy»

«local»

:Transaction

{transient}

«global»

p: ODBCProxy

2.1 : setValues(d, 3.4)
2.2 : setValues(a, “CO”)

message

Semantic Equivalence

- These examples are semantically equivalent
  - you can convert one diagram into the other with (almost) no loss of information
  - however, each view tends to stress different details
    - for instance, the sequence diagram shows method return information, while the collaboration diagram contains information on how the objects are linked

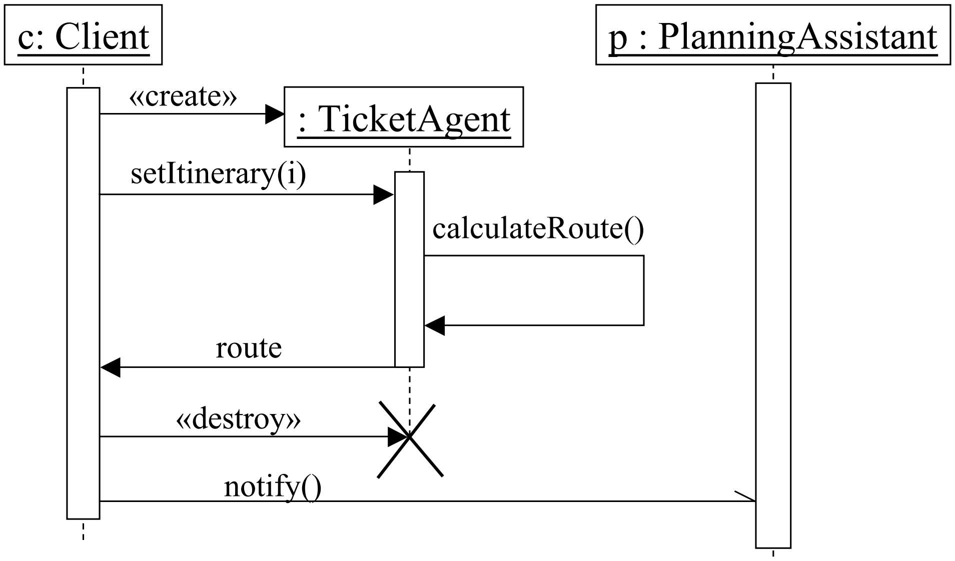
Details: Links

- A link is a path along which a message can be sent; there are different types of links
  - **association**: the link is present due to a class association
  - **self**: an object can send a message to itself
  - **global**: a link to an object is possible because the object exists in an enclosing scope
  - **local**: a link to an object is possible because the object exists in a local scope
  - **parameter**: a link to the object is possible because the object was passed as a parameter

Details: Messages

- A message is a request for action or a query for information
- UML supports several pre-defined message types
  - **call**: invokes an operation on an object
  - **return**: returns a value to the caller
  - **send**: sends a signal to an object
  - **create**: creates an object
  - **destroy**: destroys an object

Details: Message Type Example



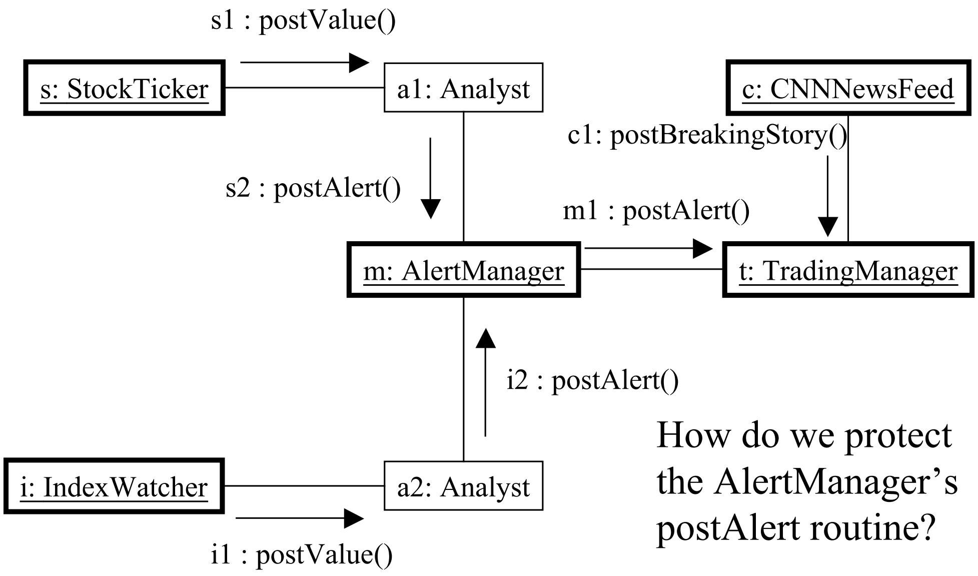
Iteration and Branching

- Interaction diagrams can support both iteration and branching
- Iteration is indicated with an asterick followed by an optional iteration expression, followed by the message name; “||” indicates parallel execution
  - * dialDigit()
  - * [i := 1..n] updateAccount(i)
  - * [i := 1..n] || q[i].calculateScore()
- Branching is indicated with a boolean condition that appears before the sequence number or message name
  - [x >= 0] doThis
  - [x < 0] doThat
- See examples in class [UML ref. manual, page. 529 and 530]

Synchronizing Flows of Control

- In the previous example, *notify()* was an asynchronous message between two different flows of control
- Normally, flows of control are associated with active objects (e.g. threads) and a designer must take care to sequence the interactions between distinct flows
  - In collaboration diagrams, you can label each flow of control with a different flow identifier
  - but that’s about it…interaction diagrams have weak notations for synchronizing flows

Example of Multiple Flows

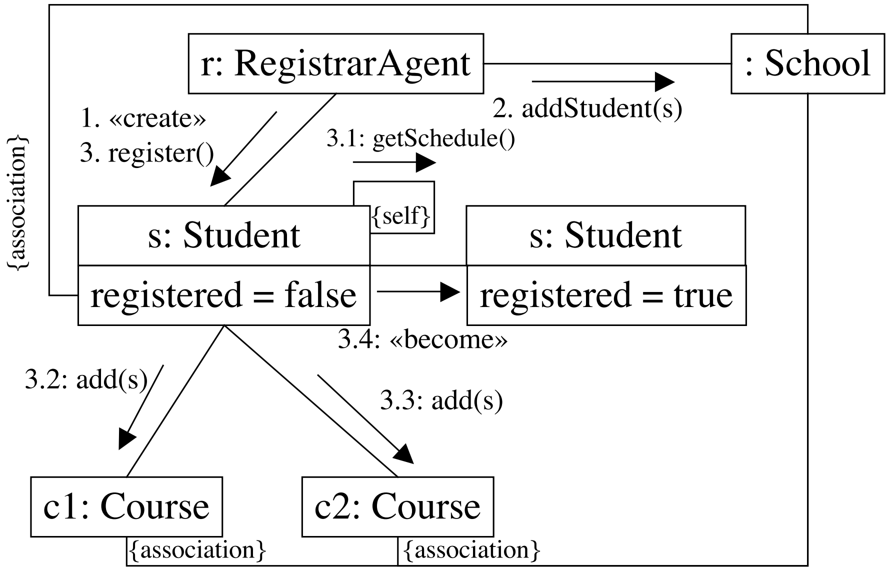


How do we protect the AlertManager’s postAlert routine?

Object Transformation

- Objects evolve over time, and this evolution can be explicitly captured in Interaction diagrams
  - In particular, the «become» stereotype is used to indicate that two objects in the same diagram are actually the same object at different points in time
  - Less commonly used, the «copy» stereotype can be used to indicate that an object is an exact copy of some other object; the copies can then evolve independently
- These stereotypes are typically used in collaboration diagrams; in sequence diagrams, object evolution is shown by redrawing a new version of the object lower on its lifeline

Object Transformation Example


r: RegistrarAgent
: School
2. addStudent(s)
1. «create»
3. register()
3.1: getSchedule()
{self}
{association}
s: Student
registered = false
s: Student
registered = true
3.4: «become»
3.2: add(s)
3.3: add(s)
c1: Course
c2: Course
{association}
{association}