Interaction Diagrams

How do groups (societies) of objects interact?

Classes and Associations Objects and Links
- objects are instances of classes
- links are instances of associations

Interaction Diagrams (IDs) come in two closely related (isomorphic) forms:
- sequence diagrams
- collaboration diagrams

Both describe the interactions between objects, and, hence, the dynamic behavior of a system.
Messages, Links and Sequencing

```
t: AirTrafficPlanner

1: getPositionAtTime(t)

message

sequence number

1.1: getLastCheckpoint()

p: FlightPlan

link

object

object
```

Interaction Diagrams – cont.

- Typically an ID captures the behavior of a single use case

- All interactions are based on the concept of "message" (Call, event, signal) sent from one object to another (along an association declared in the corresponding class diagram)

- The ID shows a number of example objects (concrete or prototypical) and the messages that are passed between them
  - concrete object --- > real world instance
  - prototypical object --- > prototype of real object.
Some Definitions

- An interaction is a behavior that comprises a set of messages exchanged between a set of objects within a context to accomplish a purpose;
- A message is a specification of a communication between objects that conveys information with the expectation that activity will ensue;
- Most often messages involve the invocation of an operation or the sending of a signal;
- Messages may also encompass the creation or destruction of other objects;
- Interactions are typically found in the collaboration of objects that exist in a system or subsystem as a whole (context).

Messages, Links and Sequencing

```
t: AirTrafficPlanner
1: getPositionAtTime(t)
p: FlightPlan
1.1: getLastCheckpoint()
```
Objects, Roles and Links

- The objects that participate in an interaction are either
  - concrete things, or
  - prototypical things
- As a concrete thing an object represents something in the real world (a specific window)
- As a prototypical thing an object represents any instance of the class (any window)
- A link is a semantic connection between objects, an instance of an association

Associations and Links

- Class `Person` with operations `+setCompensation(s: Salary)` and `+assign(d: Department)`
- Class `Company` with an association `1..*` to `*` `Person`
- A link labeled `assign(development)` connects a `Person` to a `Company`
- The `Person` is a named object
- The `Company` is an anonymous object
Links

A link specifies a path along which one object can dispatch a message
• to another object, or
• the same object

Standard stereotypes of links;
• self – object is visible because it is the dispatcher of the operation
• global – object is visible because it is in an enclosing scope
• local – object is visible because it is in a local scope
• association – object is visible by association
• parameter – object is visible because it is a parameter.

Messages

A message is the specification of a communication among objects;
The receipt of a message instance may be considered an instance of an event;
The action that results is the execution of a computational procedure that may change the state of the receiving object

Kinds of actions:
• call – invokes an operation of an object;
• return – returns a value to the caller;
• send – sends a signal to an object;
• create – creates an instance
• destroy – destroy an instance (suicide is legal)
Messages

- c: Client
- p: PlanningAssistant
- : TicketAgent
- create
- setItinerary(i)
- calculateRoute()
- notify()
- destroy

Notation

- Sending a message - (procedure invocation)
  - check()

- Return - (usually inferred)

- Asynchronous message - non-blocking invocation
  - perform()

- Return from concurrent process / thread
Sequencing

- An object passes a message to another object (in effect, delegating some action to the receiver);
- The receiving object may in turn send a message to another object, which might send a message to yet another object, and so on;
- This stream of messages forms a sequence;
- The start of every sequence is rooted in some process or thread;
- Each process or thread within a system defines a distinct flow of control;
- Within each flow, messages are ordered in sequence by time;
- The sequence of messages can be explicitly modeled by sequence numbers - [Root]n.m.k: message( . . ) .

Procedural Sequence

```
: View
  2: clickAt(p)
  2.1: I = findAt(p)

: Controller
  2.2: putRecentPick(I)

: Cache
```

sequence number
message
nested flow of control
nested flow of control
Modeling Flow of Control

- Set the context for the interaction, whether it is the system as a whole, a subsystem, a class, or an individual operation;
- Identify the objects which play a role, set their initial properties (attributes, state, role);
- If the model emphasizes the
  - structural organization of the objects, identify the links and specify their nature (collaboration)
  - time order, specify the messages that pass from object to object (sequence diagram)
- To convey the necessary detail of the interactions, adorn each object with its state and role, include parameters and return values.
Flow of Control by Organization

Flow of Control by Time
Well-structured Interactions

- Are simple and should encompass only those objects that work together to carry out some behavior bigger than the sum of all these elements;
- Have a clear context and may represent the interaction of objects in the context of an operation, a class, a subsystem, or a whole system;
- Are efficient and carry out the behavior with an optimal balance of time and resources;
- Are adaptable and elements of an interaction, that are likely to change should be isolated so that they can be easily modified (loose coupling);
- Are understandable and should be straightforward, involving no hacks, hidden side effects, or obscure semantics.

Interaction Diagrams

- Come in two forms:
  - Sequence diagrams
  - Collaboration diagrams → Communication diagrams
- They are isomorphic - semantically equivalent
- But they do not explicitly visualize the same information!
- Use sequence diagrams to model the flow of control over time, emphasizing the passing of messages over time, which visualizes the dynamic behavior in the context of a use case scenario;
- Use collaboration diagrams to model the structural relationships between instances and to visualize complex iteration and branching control flow and multiple concurrent flows.
Sequence Diagram [table structure]

<table>
<thead>
<tr>
<th>c: Client</th>
<th>(transient)</th>
<th>p: ODBCProxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;create&gt;&gt;</td>
<td>: Transaction</td>
<td></td>
</tr>
<tr>
<td>setActions(a,d,o)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>setValues(d,3.4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>setValues(a,&quot;CO&quot;)</td>
<td></td>
</tr>
<tr>
<td>committed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>focus of control</td>
<td></td>
</tr>
<tr>
<td></td>
<td>focus of control</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lifeline</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;destroy&gt;&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Collaboration Diagram [graph structure]

<table>
<thead>
<tr>
<th>c: Client</th>
<th>object</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-&gt; vertex</td>
<td></td>
</tr>
<tr>
<td>1:&lt;&lt;create&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2: setActions(a,d,o)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3:&lt;&lt;destroy&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;local&gt;&gt;</td>
<td>link</td>
<td>arc</td>
</tr>
<tr>
<td></td>
<td>link</td>
<td></td>
</tr>
<tr>
<td>: Transaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(transient)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sequence</td>
<td></td>
</tr>
<tr>
<td>2.1: setValues(d,3.4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2: setValues(a,&quot;CO&quot;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;global&gt;&gt;</td>
<td>p: ODBCProxy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>path stereotype</td>
<td></td>
</tr>
<tr>
<td></td>
<td>message</td>
<td></td>
</tr>
</tbody>
</table>
Sequence and Collaboration Diagrams

- Sequence Diagrams have two features which distinguish them from collaboration diagrams:
  - object lifelines representing the existence of an object over a period of time
  - focus of control indicating the period of time when an object is performing an action (either directly or through a subordinate procedure)

- Collaboration Diagrams have two features which distinguish them from sequence diagrams:
  - path stereotypes to specify the link (local, global, parameter, self)
  - sequence numbers indicating the time order of the messages (1, 2, 3, ... or 2.3, 2.4, 3.1.2, 3.1.3, ...)

---

**Sequence and Collaboration Diagram**

```
c: Client
  <<create>>
  setActions(a,d,o)
  committed
  <<destroy>>

: Transaction
  (transient)
  setValues(d,3.4)
  setValues(a,'CO')

p: ODBCProxy
  2.1: setValues(d,3.4)
  2.2: setValues(a,'CO')
```

---

**Vertex / Arc**

- Client
  - <<local>>
    - 1: <<create>>
    - 2: setActions(a,d,o)
    - 3: <<destroy>>

- Transaction
  - (transient)

- ODBCProxy
  - <<global>>
    - 2.1: setValues(d,3.4)
    - 2.2: setValues(a,'CO')
Semantic Equivalence

- Sequence diagrams and collaboration diagrams derive from the same information in the UML metamodel;
- Hence, they are semantically equivalent and can be converted into the other without loss of information;
- But the diagrams do not explicitly visualize the same information! (e.g. <<global>> and <<local>>)
- To model flows of control over time, sequence diagrams should be used, emphasizing the passing of messages over time, which visualizes the dynamic behavior in the context of a use case scenario; sequence diagrams do a better job of visualizing simple iteration and branching;
- To model flows of control by organization emphasizing the structural relationships between instances and to visualize complex iteration and branching and multiple concurrent flows, collaboration diagrams should be used.

Example

Order entry system:

- Order Entry window sends a “prepare” message to an order;
- Order sends a “prepare” message to each orderLine on the Order;
- Each orderLine checks the given stockItem
  - IF ok – remove the quantity and create a deliveryItem
  - IF < reorderLevel THEN stockItem requests a reOrder.
Conditions, Branching and Iterations

Getting a value

\[ \text{hasStock} := \text{check()} \]

Checking a condition

\[ \text{[needsReorder]} \]

\[ \text{[hasStock]} \text{ remove()} \]

Iteration

\[ *[\text{for all orderLines}] \]

\[ *[i := 1 .. n] \text{ do()} \]
Example 5.2

Asynchronous Message

new a Transaction

new a Transaction Coordinator

new

new

new

a first Transaction Checker

a second Transaction Checker

ok

ok

all done?

all done?

valid

error

Figure 5-2: Concurrent Processes and Activations

Example 5.3

When a new Transaction is created...

...it creates a Coordinator to manage the checking.

The Coordinator creates a series of Checkers, one for each kind of check. These Checkers do their checks in separate processes.

If a given check fails, the Coordinator kills all other Checkers that are still running...

...and tells the Transaction that it is invalid.

Figure 5-3: Sequence Diagram: Check Failure
Example 5.4

![Collaboration Diagram with Simple Numbering](image)

*Figure 5-4: Collaboration Diagram with Simple Numbering*

Example 5.5

![Collaboration Diagram with Decimal Numbering](image)

*Figure 5-5: Collaboration Diagram with Decimal Numbering*
CRC – Class-Responsibility-Collaboration

- A great tool on the conceptual level, capture the purpose of a class
- Easy to change and rearrange

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order</td>
<td></td>
</tr>
<tr>
<td>Check if item is in stock</td>
<td>Order Line</td>
</tr>
<tr>
<td>Determine price</td>
<td></td>
</tr>
<tr>
<td>Check for valid payment</td>
<td>Customer</td>
</tr>
<tr>
<td>Dispatch to delivery address</td>
<td></td>
</tr>
</tbody>
</table>

End