Business Informatics Group

Vienna University of Technology

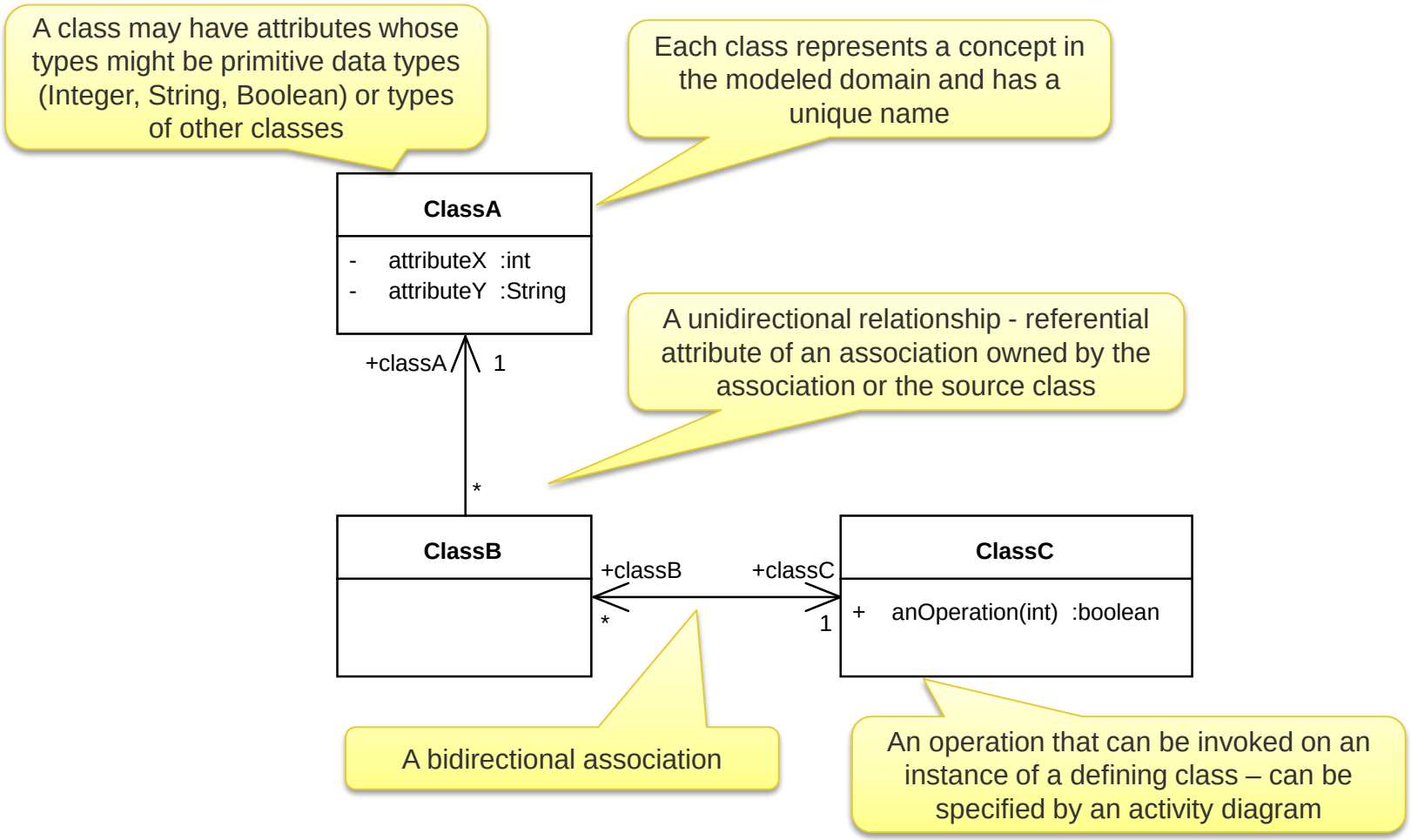
Introduction to fUML and Test Language

Business Informatics Group

Institute of Software Technology and Interactive Systems
Vienna University of Technology

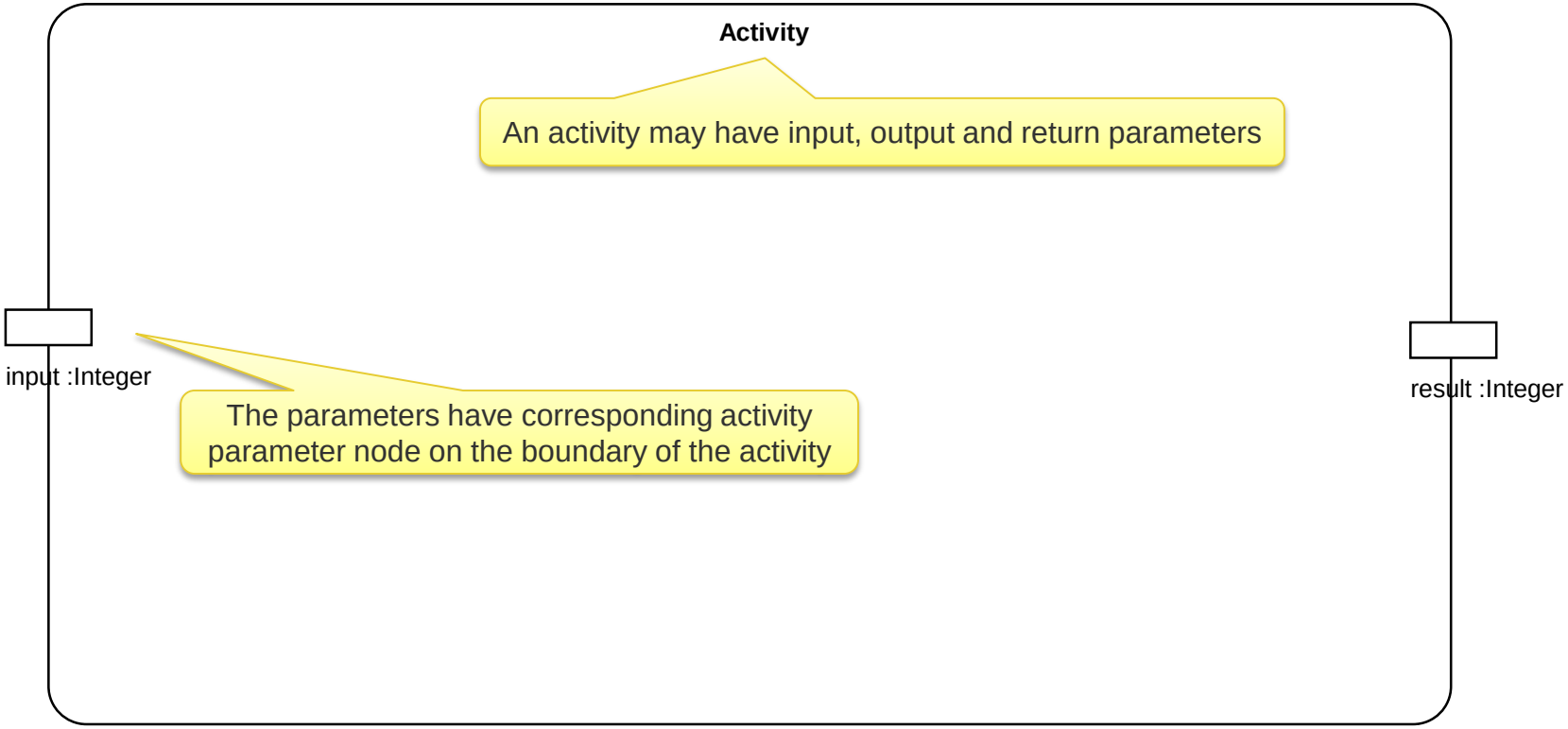
Favoritenstraße 9-11/188-3, 1040 Vienna, Austria
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896
office@big.tuwien.ac.at, www.big.tuwien.ac.at

Classes and Attributes



Activities and Parameters

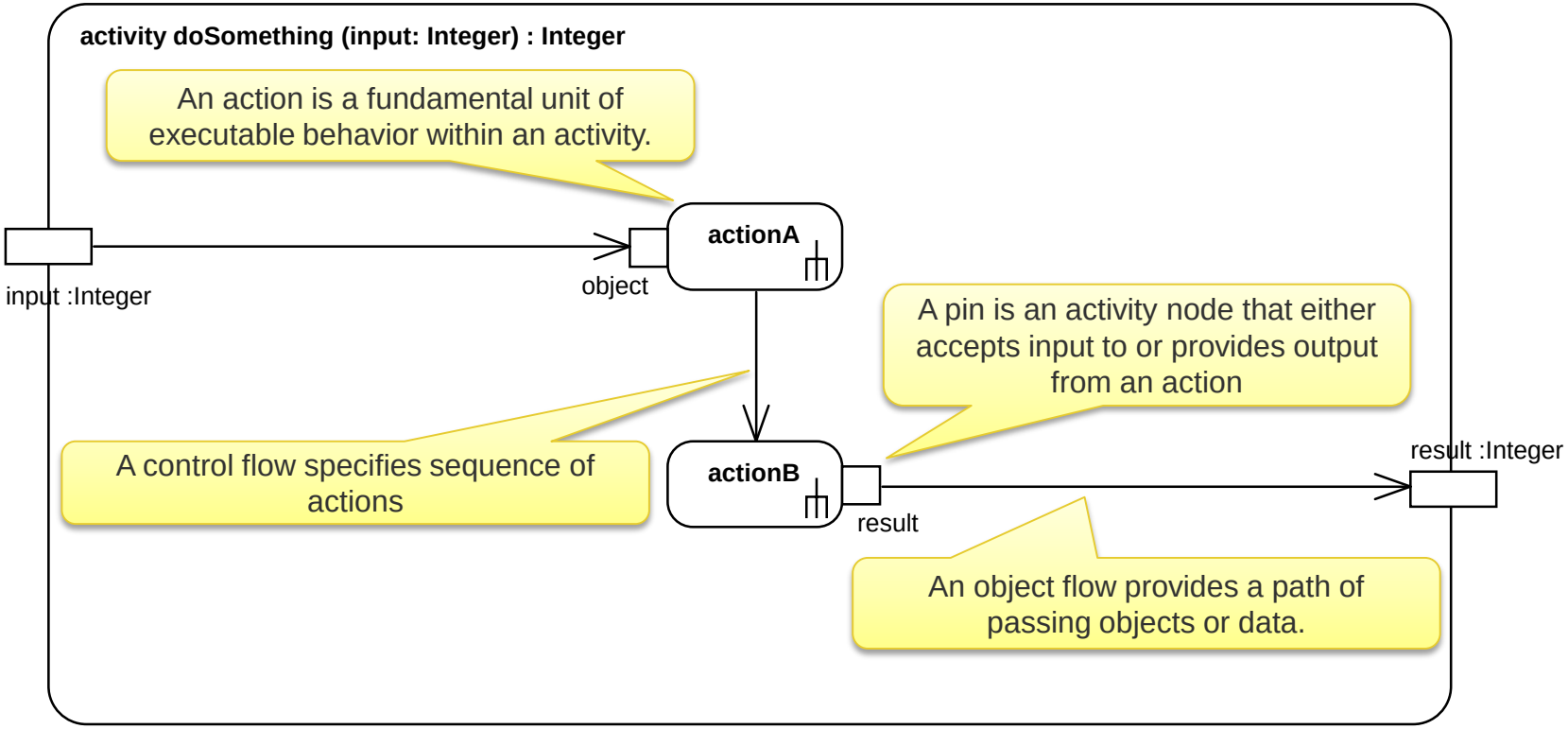
An *activity* is a specification of behavior as the coordinated execution of subordinate *actions,* using a control and data flow model.



***Example taken from**: "Programming in UML: An Introduction to fUML and Alf", Tutorial for the OMG Executable UML Information Day Presented by Ed Seidewitz, 22 March 2011*

Actions and Flows

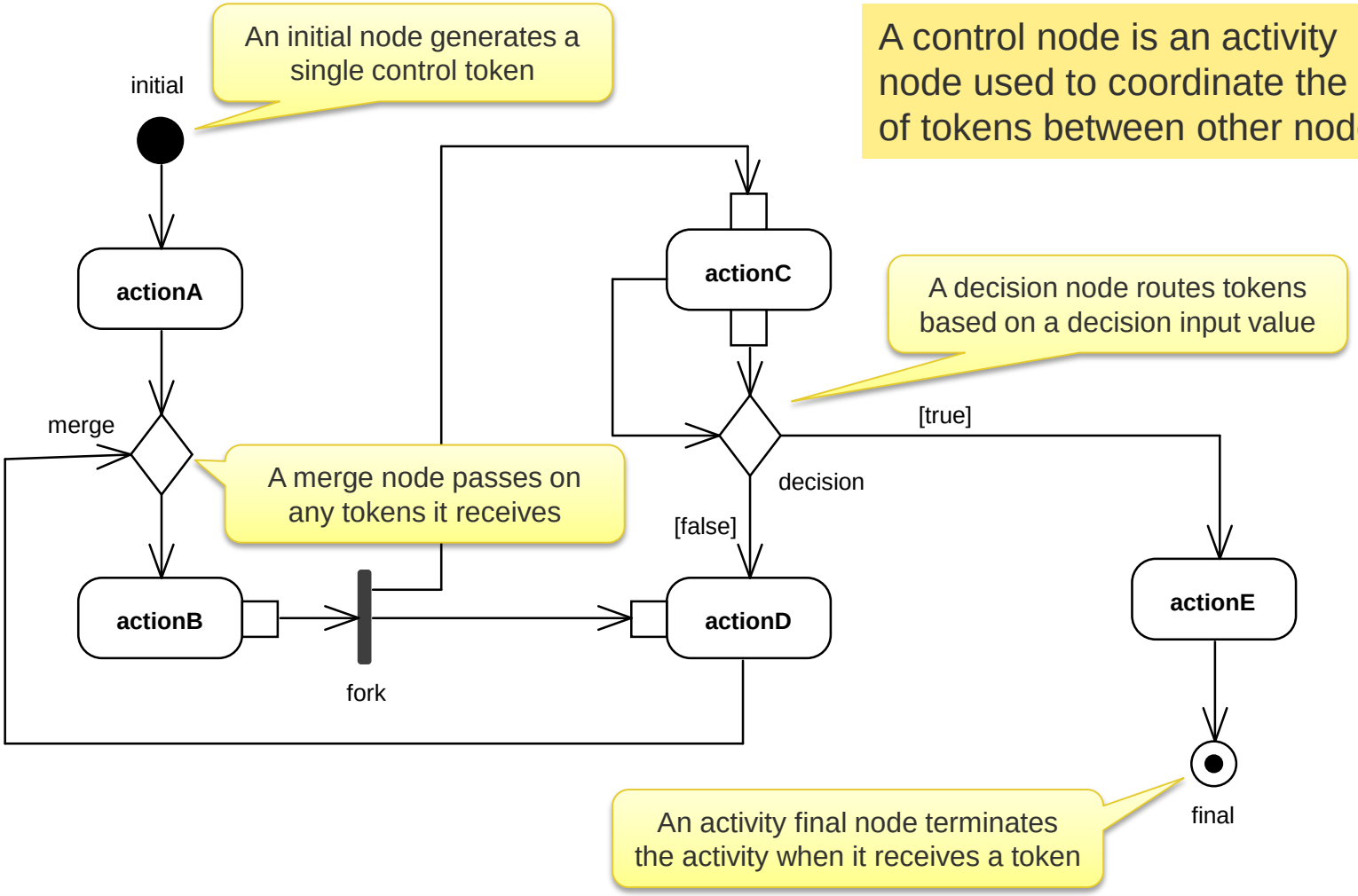
An *activity diagram* is a graph structure consisting of activity nodes connected by activity edges.



***Example taken from**: "Programming in UML: An Introduction to fUML and Alf", Tutorial for the OMG Executable UML Information Day Presented by Ed Seidewitz, 22 March 2011*

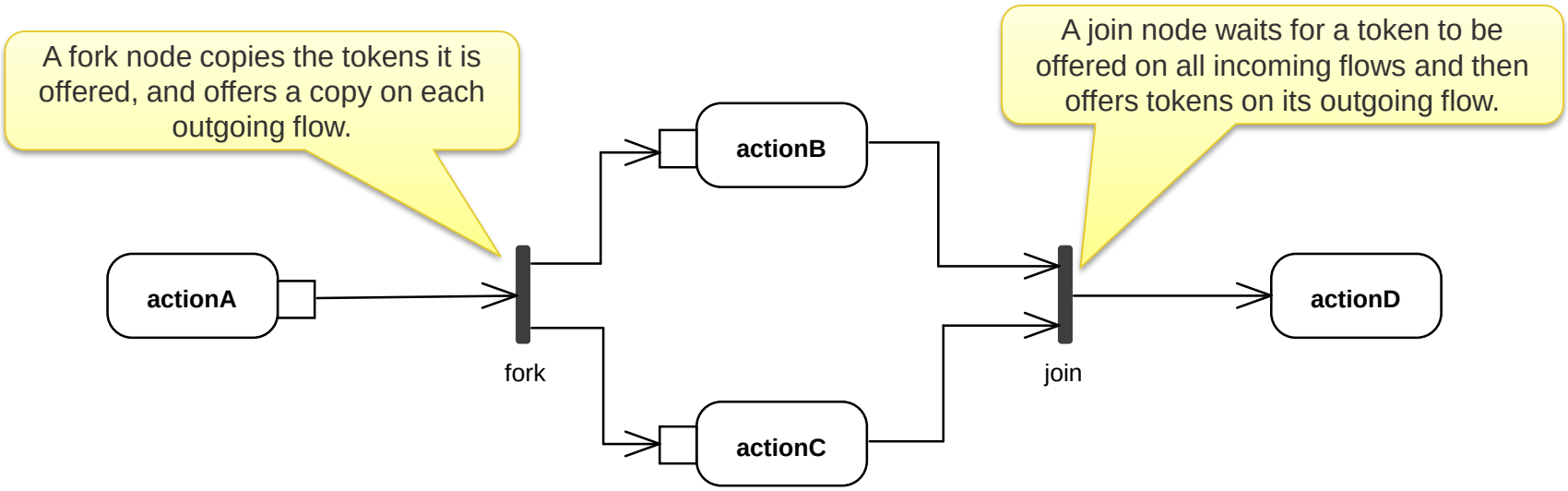
Control Nodes

A control node is an activity node used to coordinate the flow of tokens between other nodes.



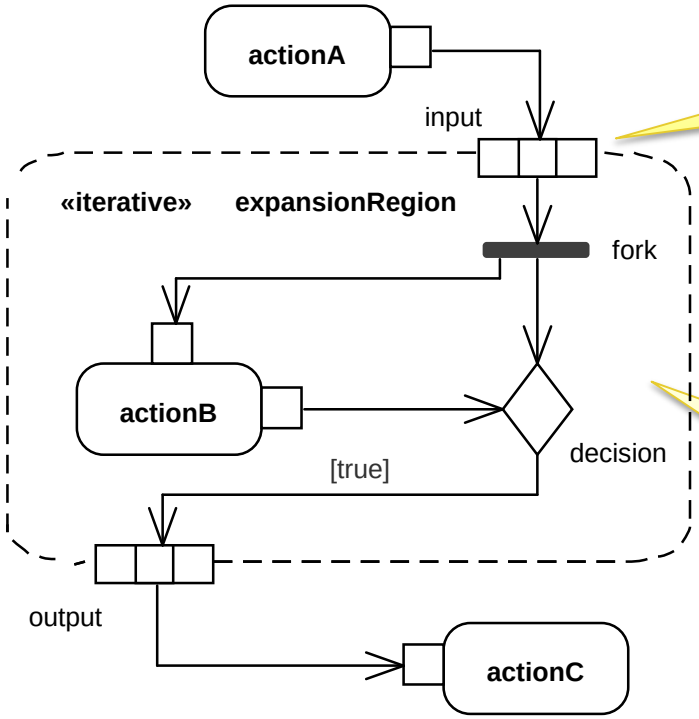
***Example taken from**: "Programming in UML: An Introduction to fUML and Alf", Tutorial for the OMG Executable UML Information Day Presented by Ed Seidewitz, 22 March 2011*

Fork and Join Nodes



***Example taken from**: "Programming in UML: An Introduction to fUML and Alf", Tutorial for the OMG Executable UML Information Day Presented by Ed Seidewitz, 22 March 2011*

Expansion regions

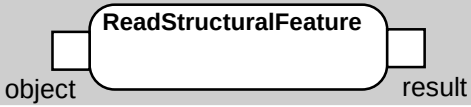


An expansion region is used to apply subordinate actions on all members of an input collection

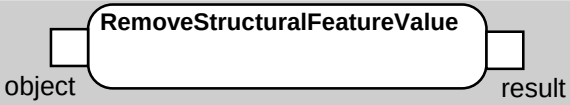
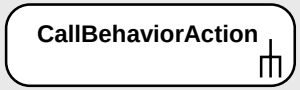
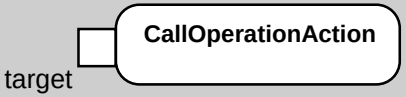
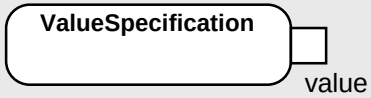
An iterative expansion region applies nested behavior sequentially to all collection elements
It can be also parallel

***Example taken from**: "Programming in UML: An Introduction to fUML and Alf", Tutorial for the OMG Executable UML Information Day Presented by Ed Seidewitz, 22 March 2011*

fUML Actions Table

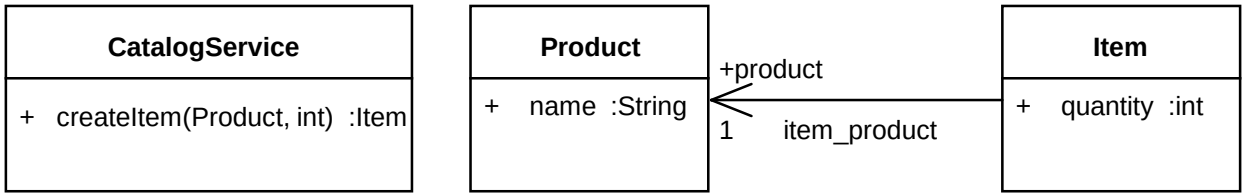
Action	Description
	Creates an instance of a specified class.
	Destroys the instance provided to the *object* input pin
	Retrieves the instance which was set as context of the owner activity
	Retrieves all existing instances of a specified class
	Retrieves a value of a specified structural feature of an instance provided to object input pin
	Adds a new value to the specified structural feature of an instance provided to object input pin

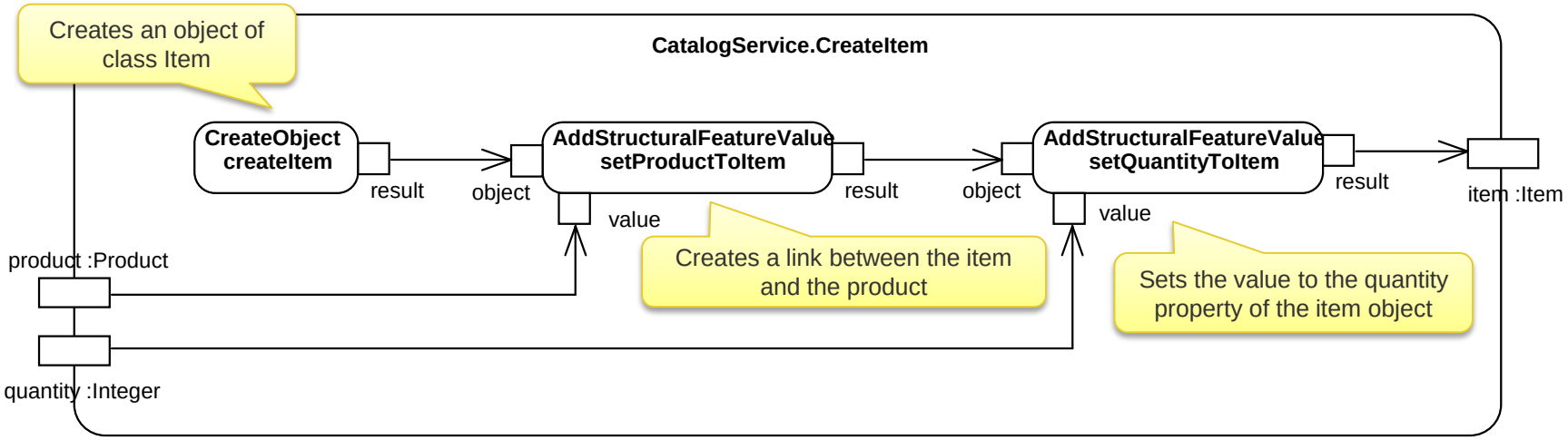
fUML Actions Table Ctd.

Action	Description
	Removes a value of a specified structural feature of an instance provided to object input pin
	Invokes execution of a specified activity
	Invokes execution of a specified operation of a class (method of operation defined with an activity)
	Creates a specified value instance

Example activity: CreateItem of CatalogService

An activity specifying the behavior of the operation **createItem**() of the class CatalogService.





Test Suite

```
import petstore.data.*
import petstore.logic.*

scenario TestData[
    object productTD: Product{ Product.name='defaultProduct'; }
    object itemTD: Item{Item.quantity=10;}
    object catalogServiceTD: CatalogService{}
    link item_product {
            source item_product.item = itemTD;
            target item_product.product = productTD;
    }
]
```

An import statement used to refer to UML model elements

A scenario is composed of several objects and links between those objects, and can be used for providing input to the activities under test, and as expected result in state assertions

Test Case

If there is a **readSelf** action in the activity then *the context object* should be specified: e.g., **on** TestData.catalogServiceTD

```
test createItemTest activity CatalogService.CreateItem (product=TestData.productTD, quantity=10) {
    initialize TestData;

    assertOrder CreateItem.setProductToItem, CreateItem.setQuantityToItem, *;

    assertState always after action CreateItem.createItem until action CreateItem.setQuantityToItem {
            CreateItem.item != null;
            CreateItem.product = TestData.productTD;
    }
    finally {
            CreateItem.item::quantity = 2;
            check 'itemProductSize' on CreateItem.item;
    }
}
```

Assert **order of execution**: * and _

finally is a shorthand for checking the last state after complete execution of the activity

Temporal Quantifier	Description
always	Assertion should be true in all states of the defined period
immediately	Assertion should be true in the first state after/until an action is executed
sometimes	Assertion should be true in at least one state in the defined period
eventually	Assertion should become true and remain true until the end of the defined period

OCL expressions

Name of a package for which the constraints are defined

```
package data
    context Product
    inv defaultName: name = 'defaultProduct'

    context Item
    inv itemProductSize: product->size()=1

    context Order
    inv noEmptyOrderLines: orderLines->forAll(item->collect(product)<>null)
endpackage
```

Each constraint is defined within a **scope of an element** in the model

It's possible to specify **simple equality expressions** on attributes and links

OCL standard functions such as iteration, selection, filtering, etc. that can be used in expressions

OCL Quick Reference

Type	Values	Operators and Operations
Boolean	false, true	or, and, not, <>
Integer	…, -10, 0, 10, …	=, <>, <, >, <=, >=, +, -, *, /
String	‘Carmen’	=, <>, size(), concat(String)
OclAny		oclIsTypeOf(T), T.allInstances()
Collections		size(), forAll(exp), select(exp)..

Test Results

Test Suite Run: 29-09-2014 15:00:05
Activity: CatalogService.CreateItem
Activity input: product = productTD; quantity = 10;

BruteForce check:
Number of paths checked: 1
Number of paths failed: 1

Failed Path: createItem, setProductToItem, setQuantityToItem
Order specification: setProductToItem, setQuantityToItem, *

Matrix validation result: FAIL

State assertion: ALWAYS AFTER createItem UNTIL setQuantityToItem
State assertions checked: 2
State assertions failed: 0

State assertion: ALWAYS AFTER setQuantityToItem
State assertions checked: 1
State assertions failed: 1
    Expression: CreateItem.item::quantity = 2 / Actual was: 10