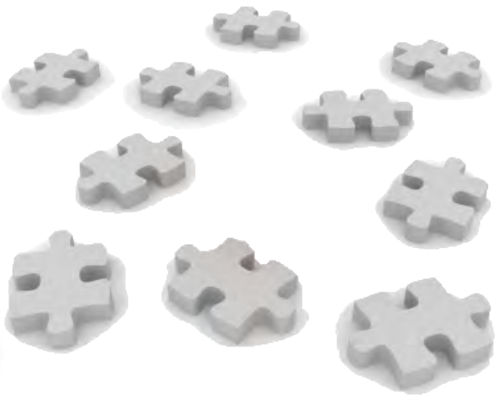
Reusable Aspect Models

Jörg Kienzle
(contributions from Jacques Klein, Wisam Al Abed, Omar Alam, Matthias Schöttle, Valentin Bonnet, Engin Yildirim, Abir Ayed)
Software Engineering Laboratory
School of Computer Science, McGill University

Overview

- Aspect-Oriented Modelling
- Reusable Aspect Models
  - Overview
  - Usage and Customization Interface
    - Observer Design Concern
  - Aspect Hierarchies and Instantiations
    - ZeroToMany
  - TouchRAM
  - Instantiations
    - depends
    - extends
  - Case Studies
    - Workflow Execution Engine
    - AspectOPTIMA

Aspect-Oriented Modelling



- Define new features that allow the modularization of crosscutting concerns at the modelling level

➡ A modeller can reason about one concern individually and later on separately on concern composition

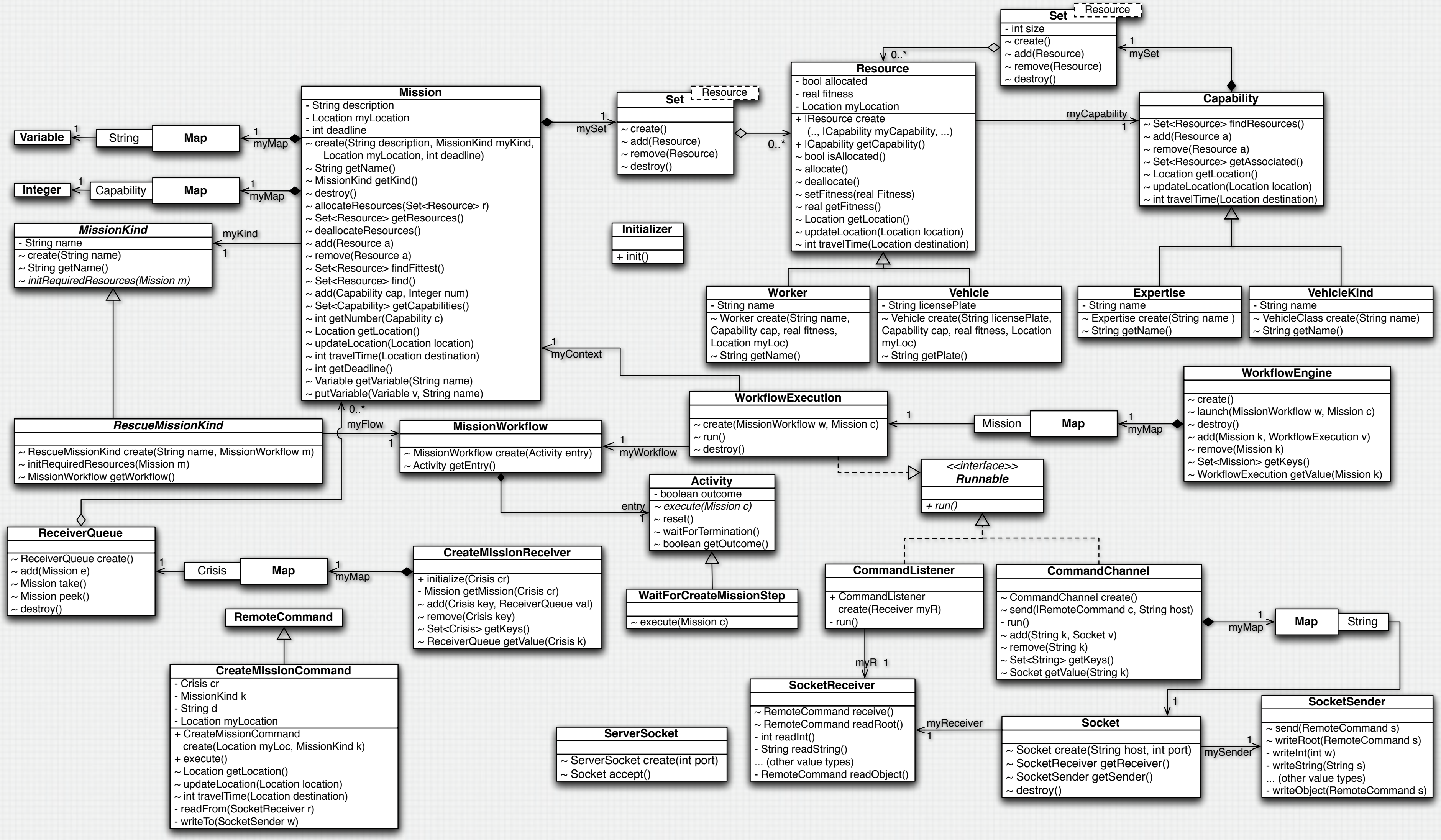
- Define a model weaving algorithm to create final application model
    - For model checking, code generation, simulation / debugging purpose

Multi-View Modelling

- Allows developers to describe a (software) system from multiple points of view
  - Structural views vs. behavioural views
- Allows developers to use multiple modelling notations / formalisms
  - Makes it possible for a modeler to use the most appropriate formalism to express the facet of the system in focus
- Challenges
  - Scalability of one view
  - Consistency between views
  - Model reuse

Aspect-Orientation can help us with these challenges!

Scalability Problem Example



Set Resource
- int size
~ create()
~ add(Resource)
~ remove(Resource)
~ destroy()

Mission
- String description
- Location myLocation
- int deadline
~ create(String description, MissionKind myKind, Location myLocation, int deadline)
~ String getName()
~ MissionKind getKind()
~ destroy()
~ allocateResources(Set<Resource> r)
~ Set<Resource> getResources()
~ deallocateResources()
~ add(Resource a)
~ remove(Resource a)
~ Set<Resource> findFittest()
~ Set<Resource> find()
~ add(Capability cap, Integer num)
~ Set<Capability> getCapabilities()
~ int getNumber(Capability c)
~ Location getLocation()
~ updateLocation(Location location)
~ int travelTime(Location destination)
~ int getDeadline()
~ Variable getVariable(String name)
~ putVariable(Variable v, String name)

Set Resource
~ create()
~ add(Resource)
~ remove(Resource)
~ destroy()

Resource
- bool allocated
- real fitness
- Location myLocation
+ IResource create (.., ICapability myCapability, ...)
+ ICapability getCapability()
~ bool isAllocated()
~ allocate()
~ deallocate()
~ setFitness(real Fitness)
~ real getFitness()
~ Location getLocation()
~ updateLocation(Location location)
~ int travelTime(Location destination)

Capability
~ Set<Resource> findResources()
~ add(Resource a)
~ remove(Resource a)
~ Set<Resource> getAssociated()
~ Location getLocation()
~ updateLocation(Location location)
~ int travelTime(Location destination)

Variable — String — **Map**

Integer — Capability — **Map**

MissionKind
- String name
~ create(String name)
~ String getName()
~ *initRequiredResources(Mission m)*

Initializer
+ init()

Worker
- String name
~ Worker create(String name, Capability cap, real fitness, Location myLoc)
~ String getName()

Vehicle
- String licensePlate
~ Vehicle create(String licensePlate, Capability cap, real fitness, Location myLoc)
~ String getPlate()

Expertise
- String name
~ Expertise create(String name)
~ String getName()

VehicleKind
- String name
~ VehicleClass create(String name)
~ String getName()

WorkflowEngine
~ create()
~ launch(MissionWorkflow w, Mission c)
~ destroy()
~ add(Mission k, WorkflowExecution v)
~ remove(Mission k)
~ Set<Mission> getKeys()
~ WorkflowExecution getValue(Mission k)

WorkflowExecution
~ create(MissionWorkflow w, Mission c)
~ run()
~ destroy()

Mission — **Map**

RescueMissionKind
~ RescueMissionKind create(String name, MissionWorkflow m)
~ initRequiredResources(Mission m)
~ MissionWorkflow getWorkflow()

MissionWorkflow
~ MissionWorkflow create(Activity entry)
~ Activity getEntry()

<<interface>>
Runnable
+ *run()*

Activity
- boolean outcome
~ *execute(Mission c)*
~ reset()
~ waitForTermination()
~ boolean getOutcome()

WaitForCreateMissionStep
~ execute(Mission c)

ReceiverQueue
~ ReceiverQueue create()
~ add(Mission e)
~ Mission take()
~ Mission peek()
~ destroy()

Crisis — **Map**

CreateMissionReceiver
+ initialize(Crisis cr)
- Mission getMission(Crisis cr)
~ add(Crisis key, ReceiverQueue val)
~ remove(Crisis key)
~ Set<Crisis> getKeys()
~ ReceiverQueue getValue(Crisis k)

RemoteCommand

CreateMissionCommand
- Crisis cr
- MissionKind k
- String d
- Location myLocation
+ CreateMissionCommand create(Location myLoc, MissionKind k)
+ execute()
~ Location getLocation()
~ updateLocation(Location location)
~ int travelTime(Location destination)
- readFrom(SocketReceiver r)
- writeTo(SocketSender w)

CommandListener
+ CommandListener create(Receiver myR)
- run()

CommandChannel
~ CommandChannel create()
~ send(IRemoteCommand c, String host)
- run()
~ add(String k, Socket v)
~ remove(String k)
~ Set<String> getKeys()
~ Socket getValue(String k)

Map — String

SocketReceiver
~ RemoteCommand receive()
~ RemoteCommand readRoot()
- int readInt()
- String readString()
... (other value types)
- RemoteCommand readObject()

ServerSocket
~ ServerSocket create(int port)
~ Socket accept()

Socket
~ Socket create(String host, int port)
~ SocketReceiver getReceiver()
~ SocketSender getSender()
~ destroy()

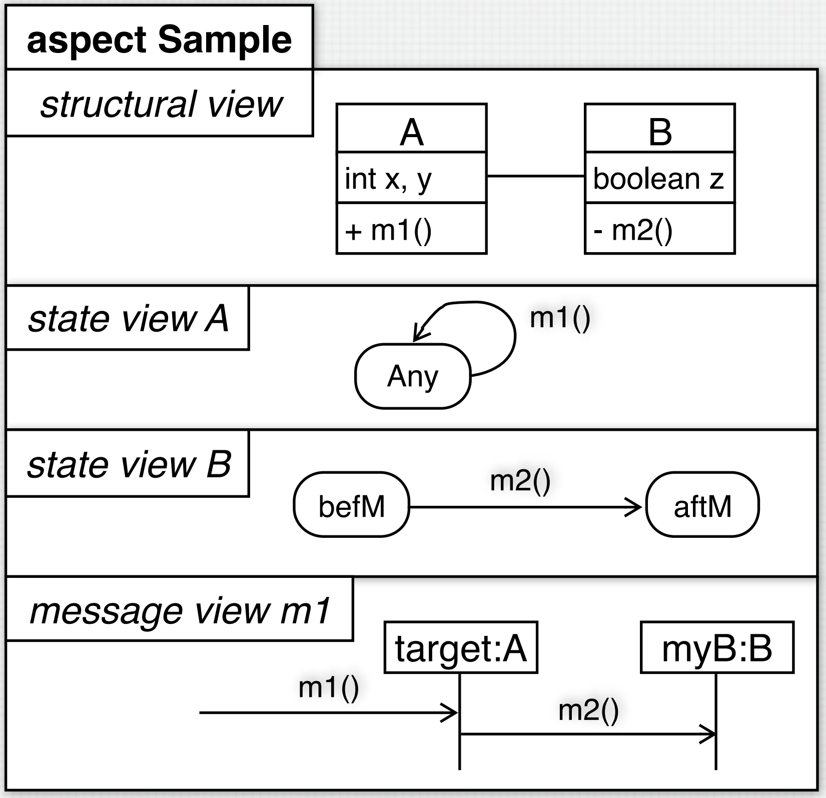
SocketSender
~ send(RemoteCommand s)
~ writeRoot(RemoteCommand s)
- writeInt(int w)
- writeString(String s)
... (other value types)
- writeObject(RemoteCommand s)

Reusable Aspect Models

- Modelling technique that exploits aspect-oriented technology for creating reusable Software Design Models
- A RAM model groups together the structure and behaviour related to a specific design concern in a UML package
  - Structural view: Class diagrams to capture structural design properties
  - State view: State diagrams to capture object invocation protocols
  - Message view: Sequence diagrams to capture object interactions
- Defines usage and customization interfaces for models

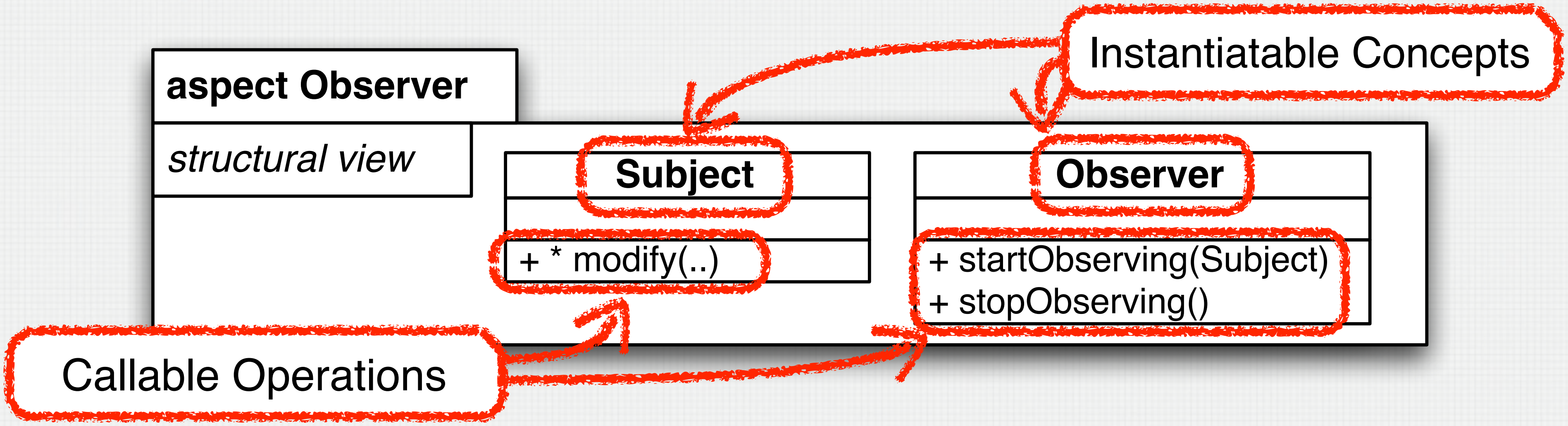
A RAM Model

- Aspect package groups structure and behavioural models related to a concern
  - One structural view
  - One state view for each class defined in structural view
  - At least one message view for each public method defined in structural view



Usage Interface

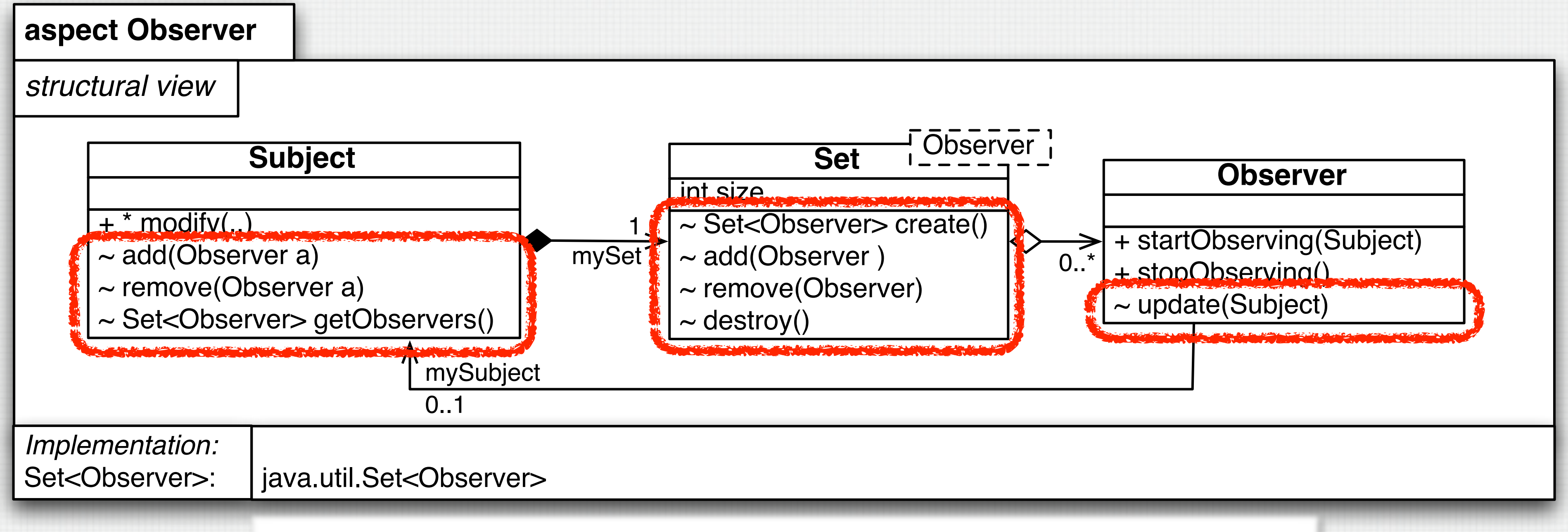
- RAM aspect models define a usage interface that details
  - The concepts relevant for users of the design concern that the aspect encapsulates (i.e., the classes that can be instantiated)
  - The operations provided by the aspect (visibility modifier +) that can be invoked (by entities in other models)



Example: Observer Design Pattern

Internal Structural Design

- Structural view (one for the entire aspect):
  - An observer has a reference to the subject it is observing
  - A subject has an internal set of observers
  - Internal methods: add, remove, getObservers, update (visibility modifier ~)
  - Implementation class: Set<Observer>



aspect Observer

structural view

Subject

+ * modify(..)
~ add(Observer a)
~ remove(Observer a)
~ Set<Observer> getObservers()

1
mySet

Set
Observer

int size
~ Set<Observer> create()
~ add(Observer)
~ remove(Observer)
~ destroy()

0..*

Observer

+ startObserving(Subject)
+ stopObserving()
~ update(Subject)

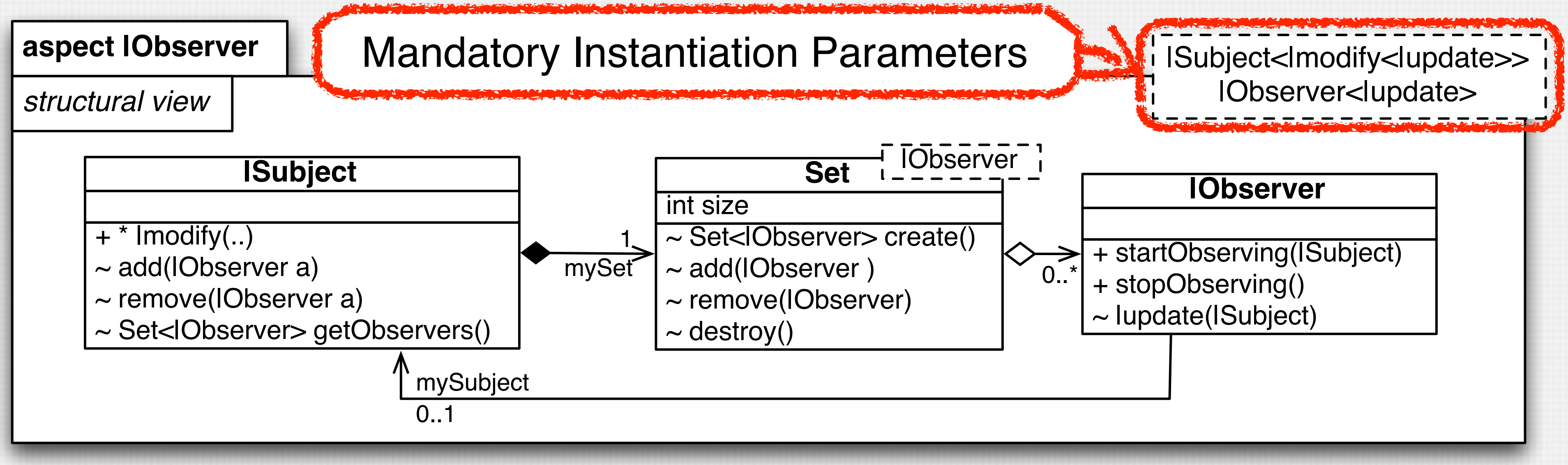
mySubject
0..1

Implementation:
Set<Observer>: java.util.Set<Observer>

Example: Observer Design Pattern

Customization Interface

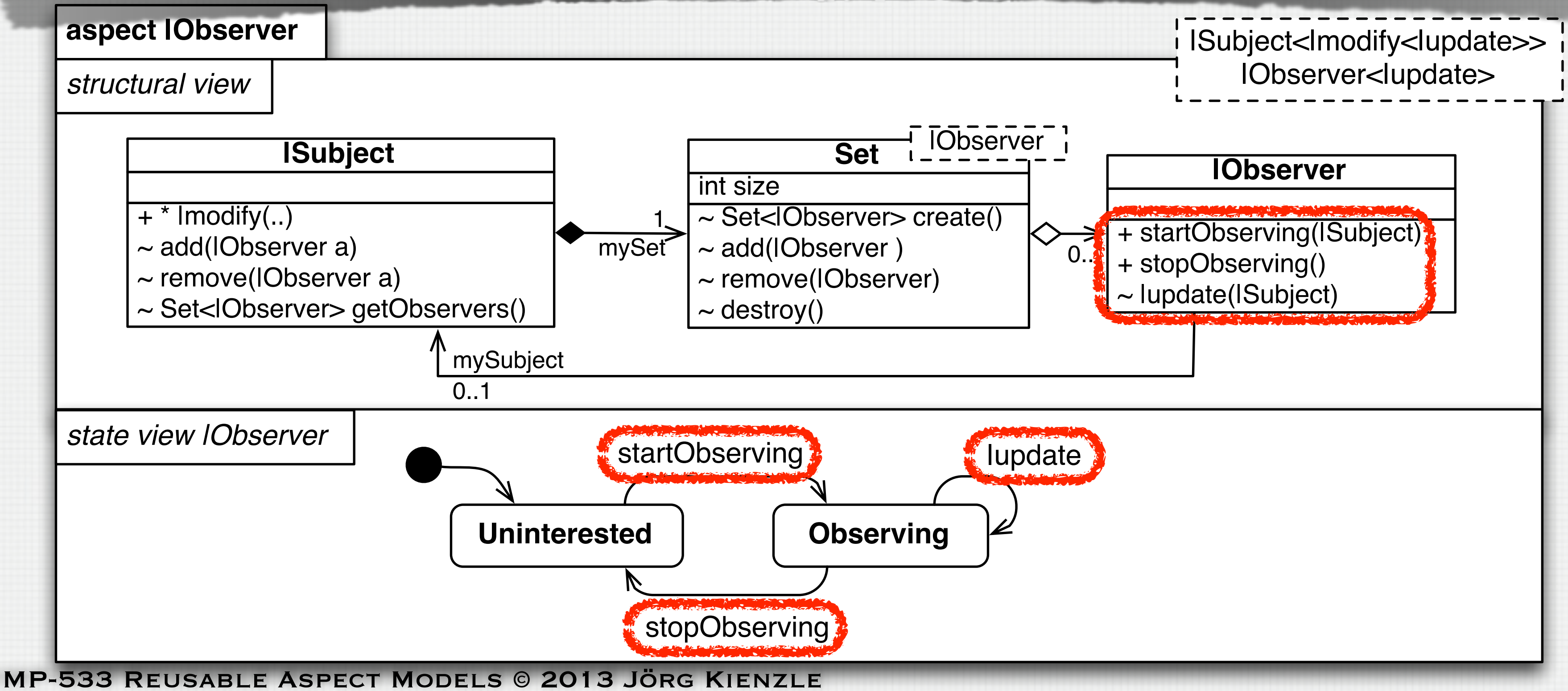
- RAM aspect models define a customization interface for reuse
- It details
  - The classes that are incomplete (and must be customized) (prefix |)
  - The operations that are incomplete (and must be customized) (prefix |)



Example: Observer Design Pattern

Internal Protocol: State Views

- Describes interaction protocol of instances
  - Each method must have at least one corresponding transition
- Observer Example
  - An Observer instance only receives update operation calls when it is observing a Subject



aspect IObserver

ISubject<Imodify<Iupdate>>
IObserver<Iupdate>

structural view

ISubject
+ * Imodify(..)
~ add(IObserver a)
~ remove(IObserver a)
~ Set<IObserver> getObservers()

1
mySet

Set
IObserver
int size
~ Set<IObserver> create()
~ add(IObserver)
~ remove(IObserver)
~ destroy()

0..

IObserver
+ startObserving(ISubject)
+ stopObserving()
~ Iupdate(ISubject)

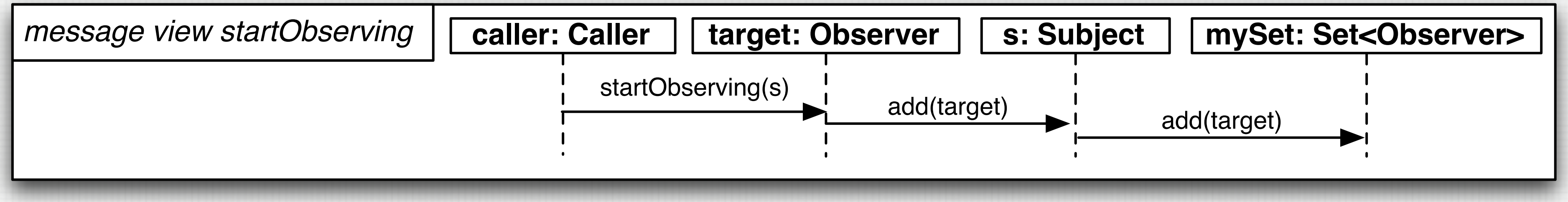
mySubject
0..1

state view IObserver

startObserving
Iupdate
Uninterested
Observing
stopObserving

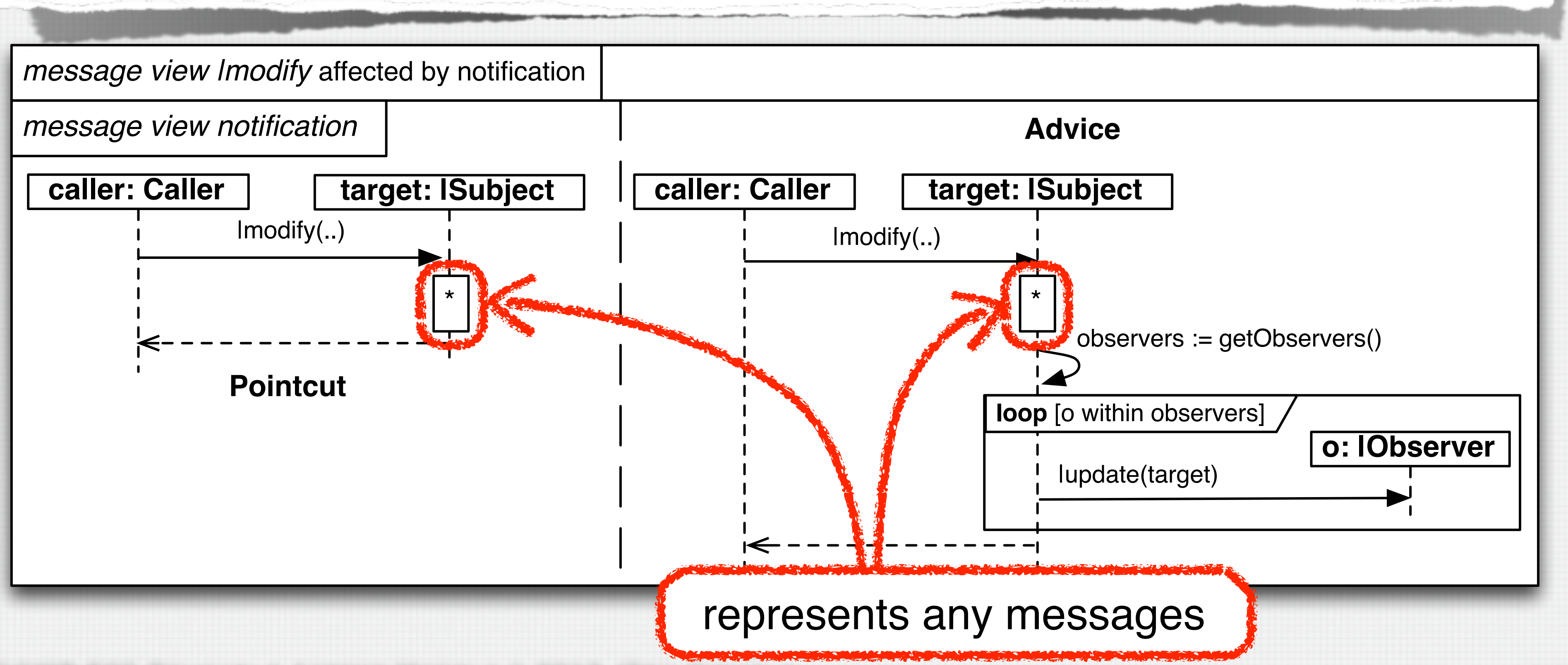
Internal Behavioural Design

- Message view (one for each public operation)
  - Describes message exchange between objects of the aspect that implement desired behaviour for each public operation
  - Simple message view defines operation behaviour
- Observer Example
  - The observer asks the subject to add the former into the set of observers



Internal Behavioural Design

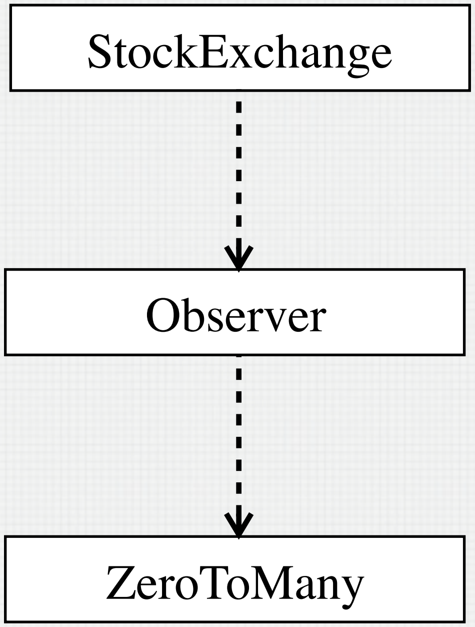
- Message view (one for each public operation)
  - Aspect message view extends operation behaviour
- Observer Example
  - Whenever Imodify is called on a Subject, the Iupdate operations of all associated Observer instances are called



Reuse with Aspect Hierarchies

- Aspect hierarchies allow a modeller to modularize crosscutting concerns at different levels of abstraction
- Aspects providing complex functionality can reuse aspects providing lower-level functionality

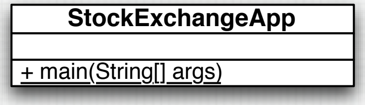
- Observer Example
  - StockExchange (re)uses Observer Aspect to notify the StockWindow of modifications to the Stock object
  - Observer (re)uses the ZeroToMany Aspect to establish a zero-to-many association between the subject and the associated observers

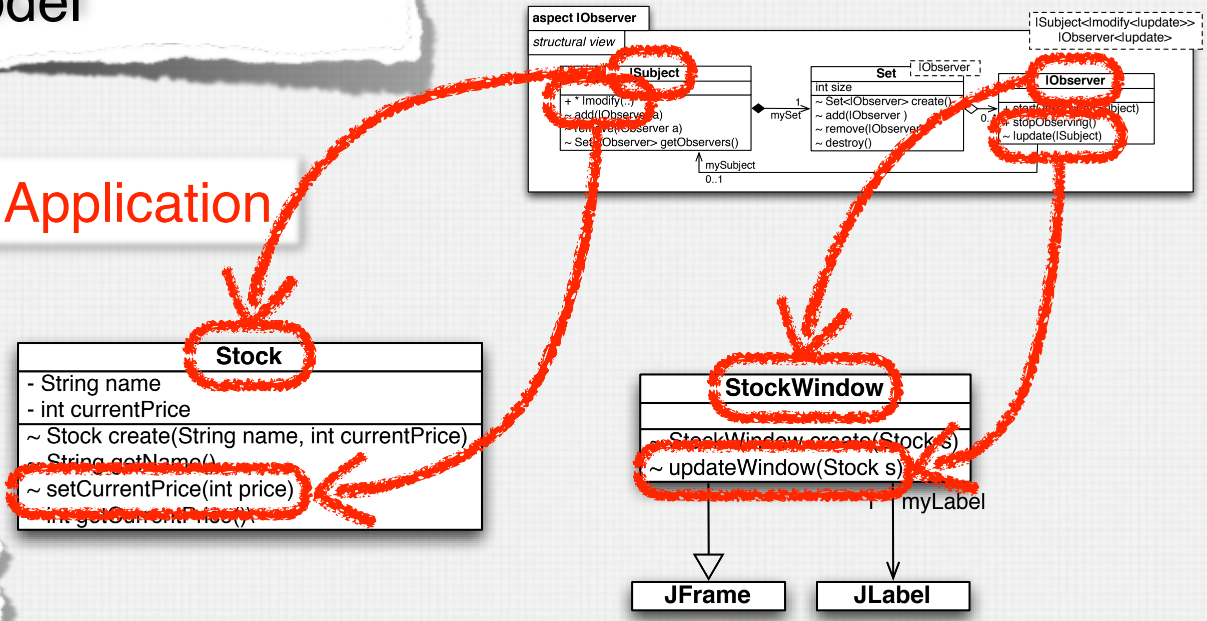


Reuse Through Instantiation

- To reuse an existing aspect model, and instantiation directive maps model elements from its customization interface to the other model

Example: Stock Exchange Application





- All mandatory parameters must be mapped
- Any element can be mapped

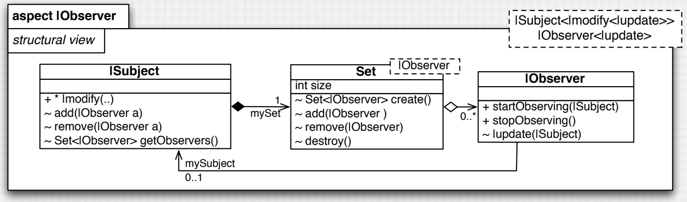
Stock is the Subject

StockWindow is the Observer

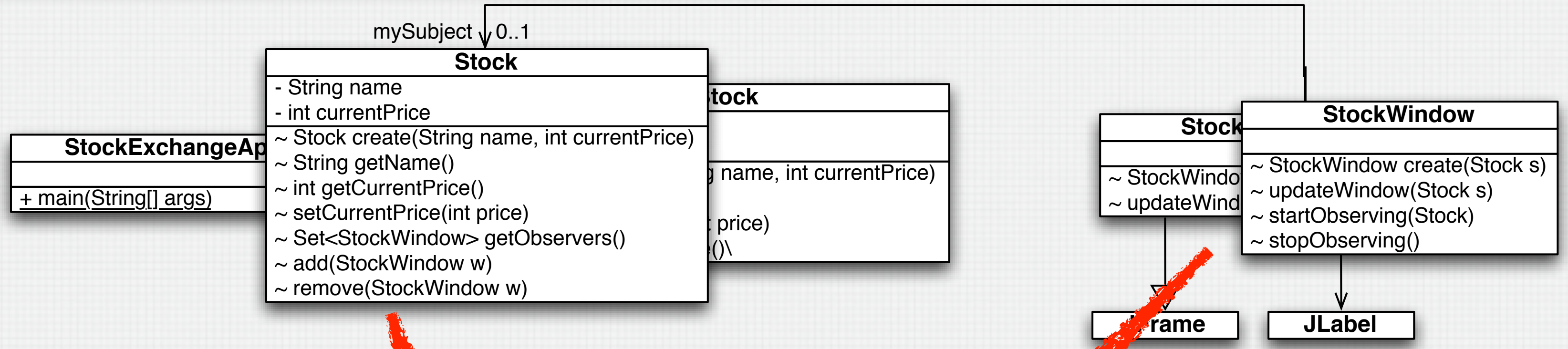
Class Composition

- The Aspect Weaver composes the structure of the aspect model with the application model according to the mapping

Class Diagram Composition [France et al.]

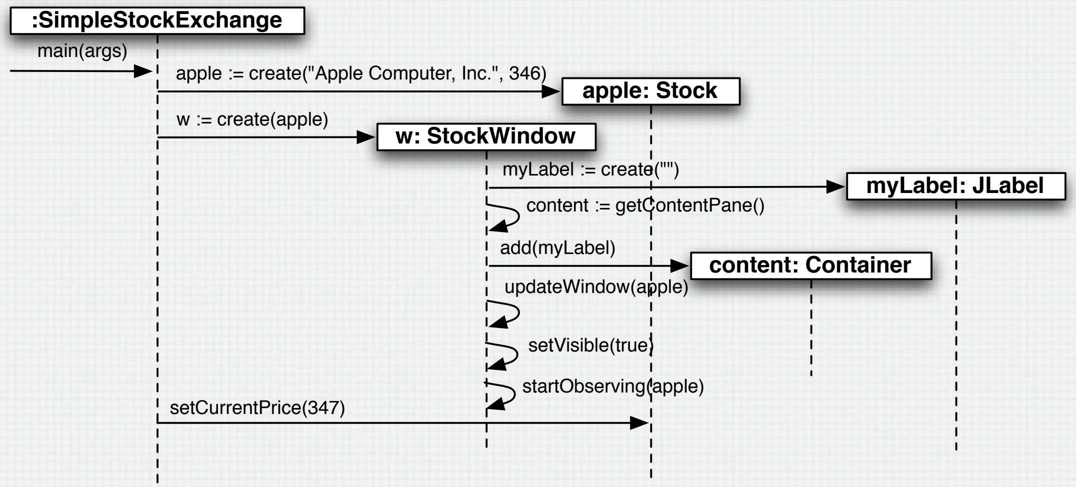


Example: Stock Exchange Application



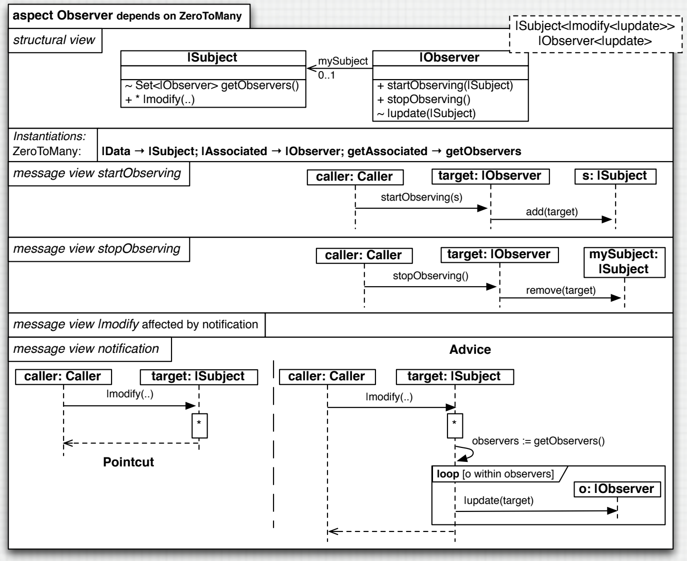
Interface of Observer is hidden

Sequence Diagram Weaving

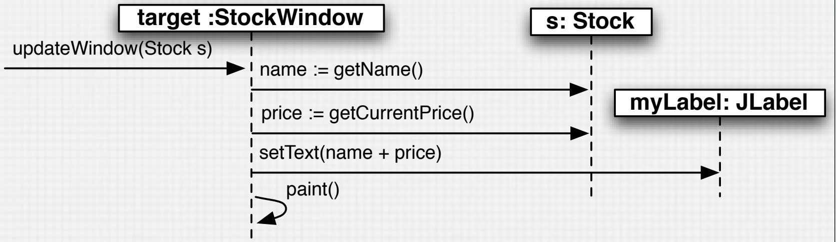


:SimpleStockExchange
main(args)
apple := create("Apple Computer, Inc.", 346)
apple: Stock
w := create(apple)
w: StockWindow
myLabel := create("")
myLabel: JLabel
content := getContentPane()
add(myLabel)
content: Container
updateWindow(apple)
setVisible(true)
startObserving(apple)
setCurrentPrice(347)

Sequence Diagram
Weaving [Klein et al.]

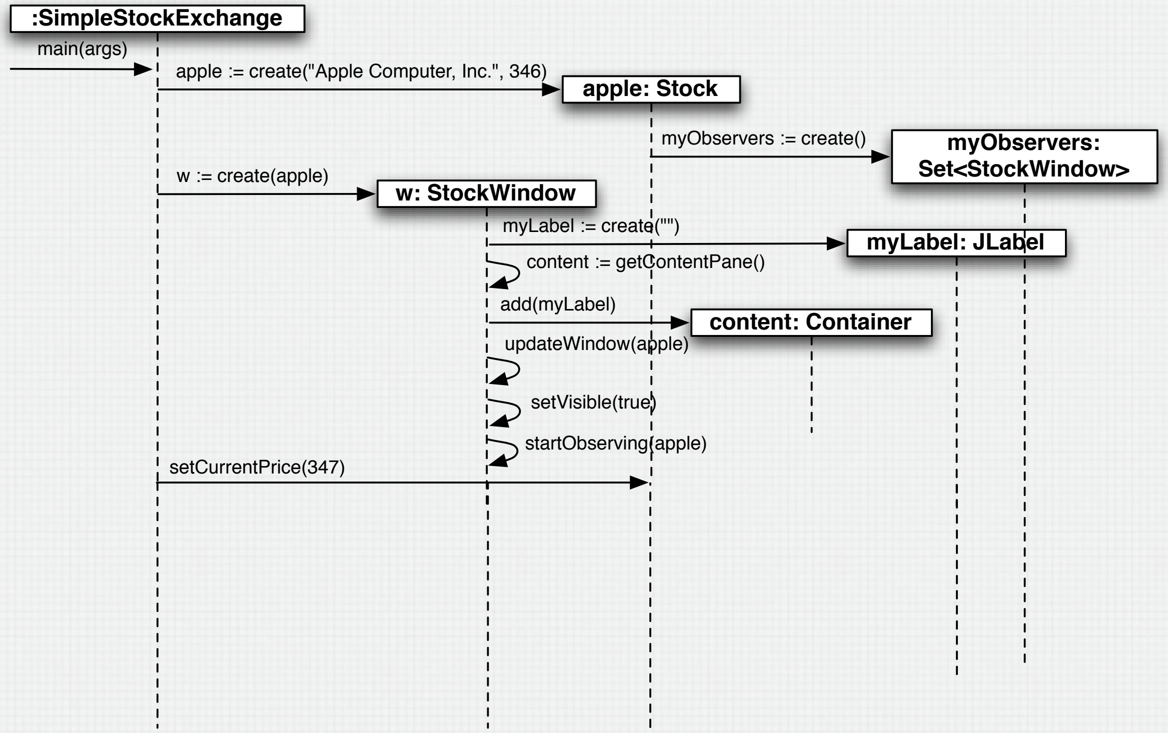


aspect Observer depends on ZeroToMany
ISubject<Imodify<Iupdate>>
IObserver<Iupdate>
structural view
ISubject
~ Set<IObserver> getObservers()
+ * Imodify(..)
mySubject
0..1
IObserver
+ startObserving(ISubject)
+ stopObserving()
~ Iupdate(ISubject)
Instantiations:
ZeroToMany:
IData → ISubject; IAssociated → IObserver; getAssociated → getObservers
message view startObserving
caller: Caller
target: IObserver
s: ISubject
startObserving(s)
add(target)
message view stopObserving
caller: Caller
target: IObserver
mySubject: ISubject
stopObserving()
remove(target)
message view Imodify affected by notification
message view notification
caller: Caller
target: ISubject
Imodify(..)
*
Pointcut
Advice
caller: Caller
target: ISubject
Imodify(..)
*
observers := getObservers()
loop [o within observers]
o: IObserver
Iupdate(target)

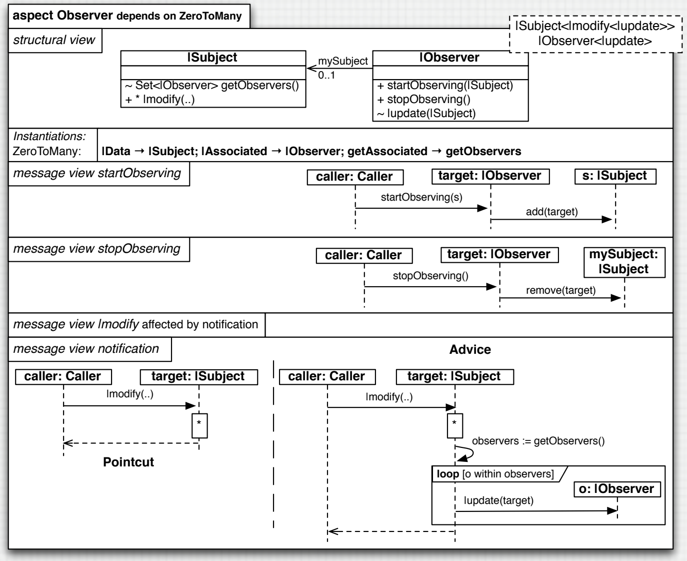


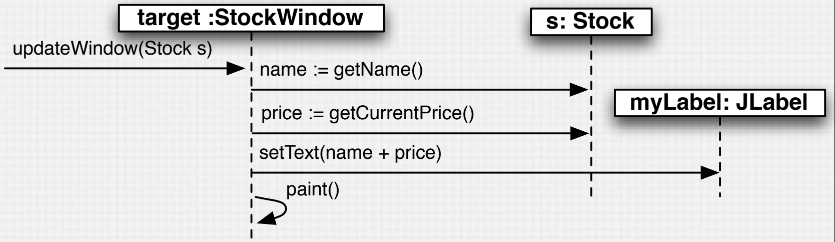
target :StockWindow
s: Stock
updateWindow(Stock s)
name := getName()
price := getCurrentPrice()
myLabel: JLabel
setText(name + price)
paint()

Sequence Diagram Weaving

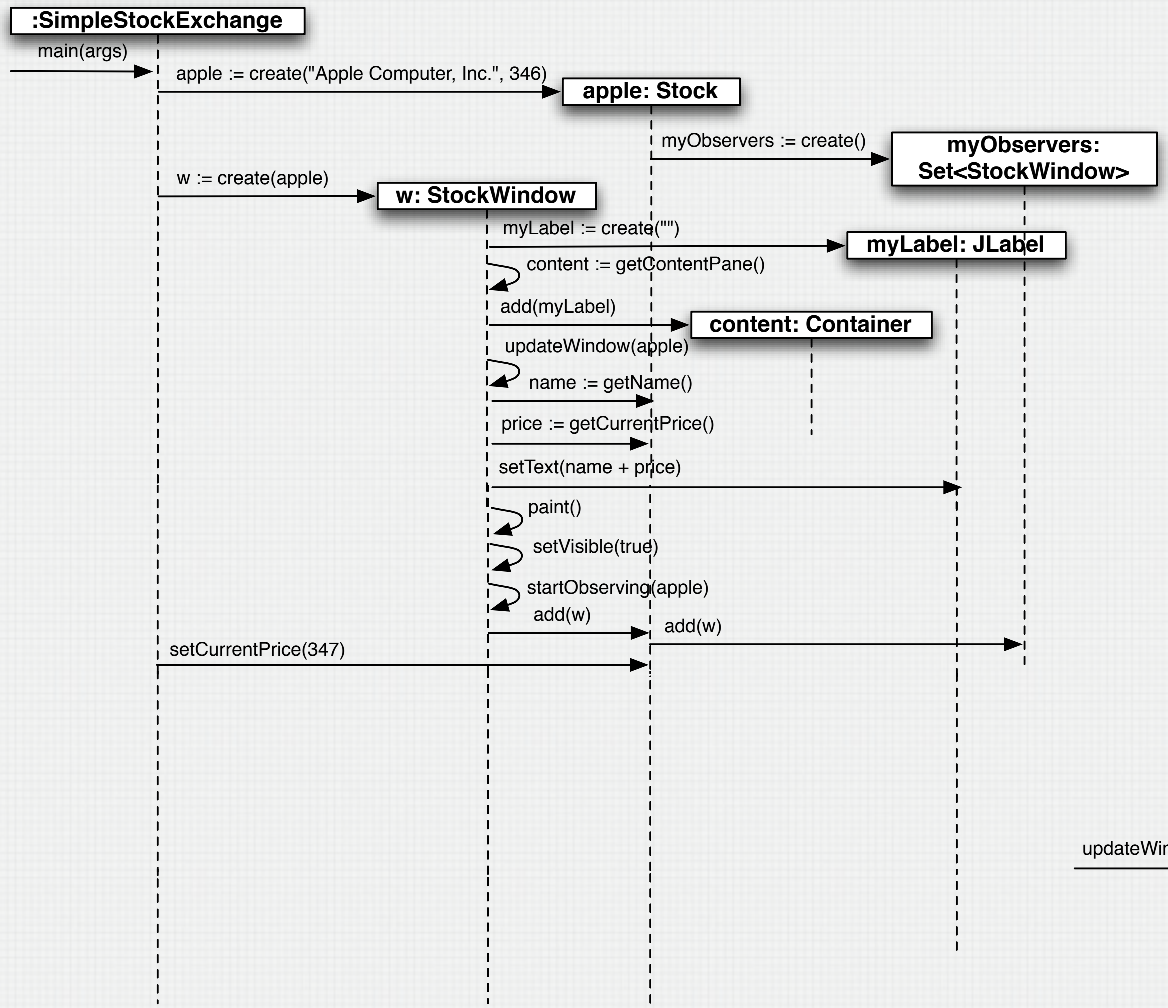


Sequence Diagram Weaving [Klein et al.]

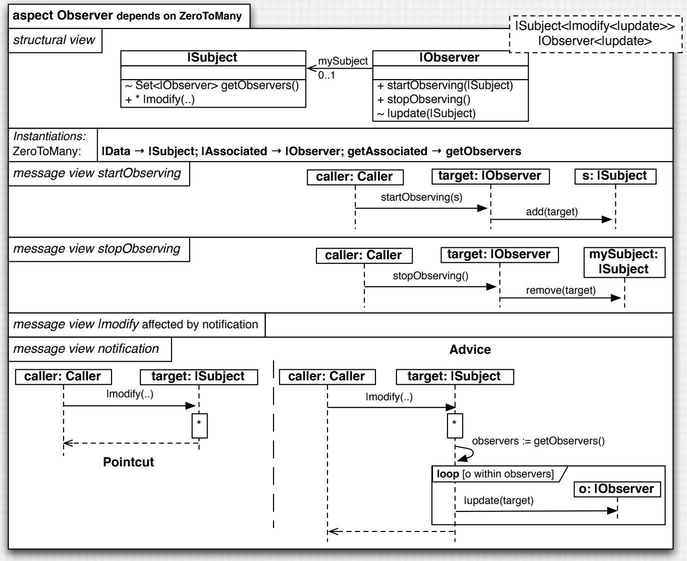


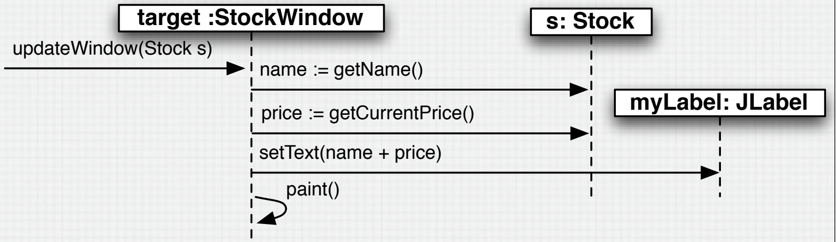


Sequence Diagram Weaving



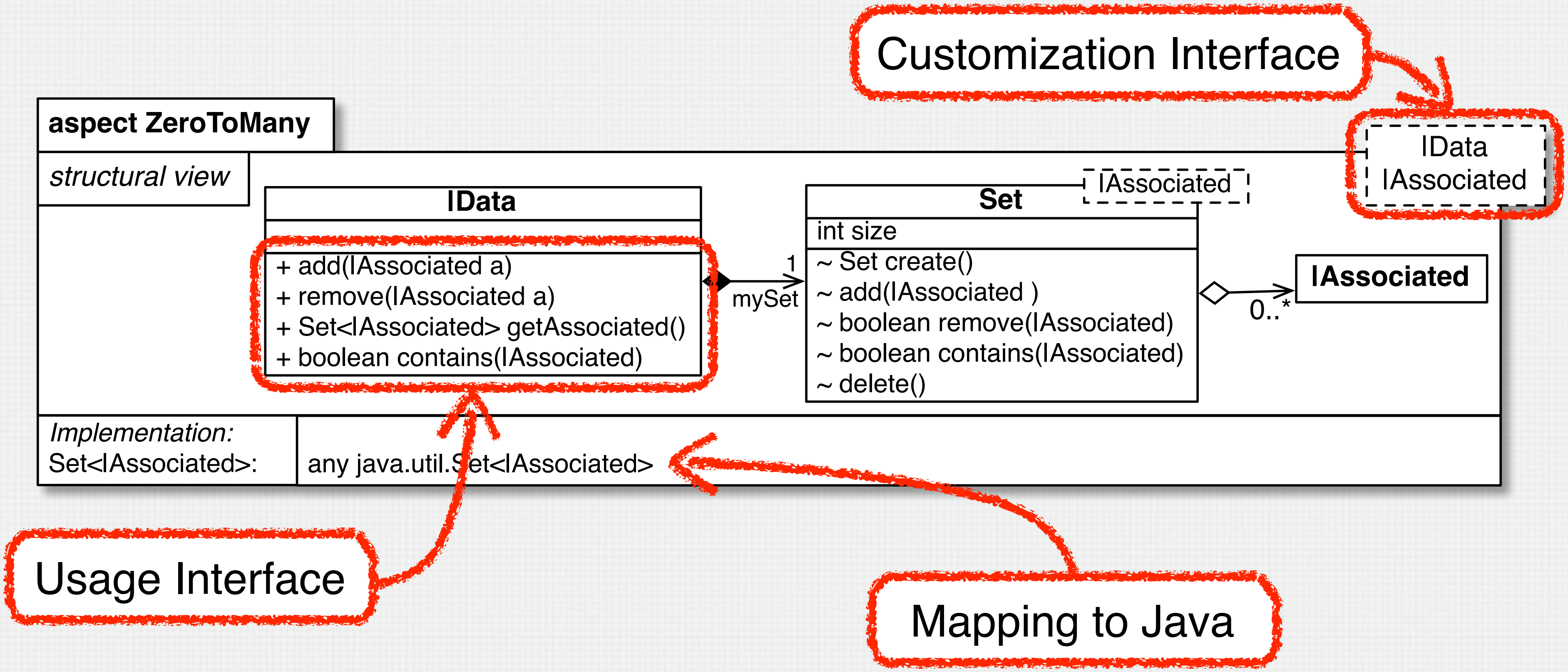
Sequence Diagram
Weaving [Klein et al.]





ZeroToMany Aspect

- Implements a zero-to-many association between a IData object and many IAssociated objects



Customization Interface

IData
IAssociated

aspect ZeroToMany

structural view

IData

+ add(IAssociated a)
+ remove(IAssociated a)
+ Set<IAssociated> getAssociated()
+ boolean contains(IAssociated)

1
mySet

Set

IAssociated

int size

~ Set create()
~ add(IAssociated)
~ boolean remove(IAssociated)
~ boolean contains(IAssociated)
~ delete()

0..*

IAssociated

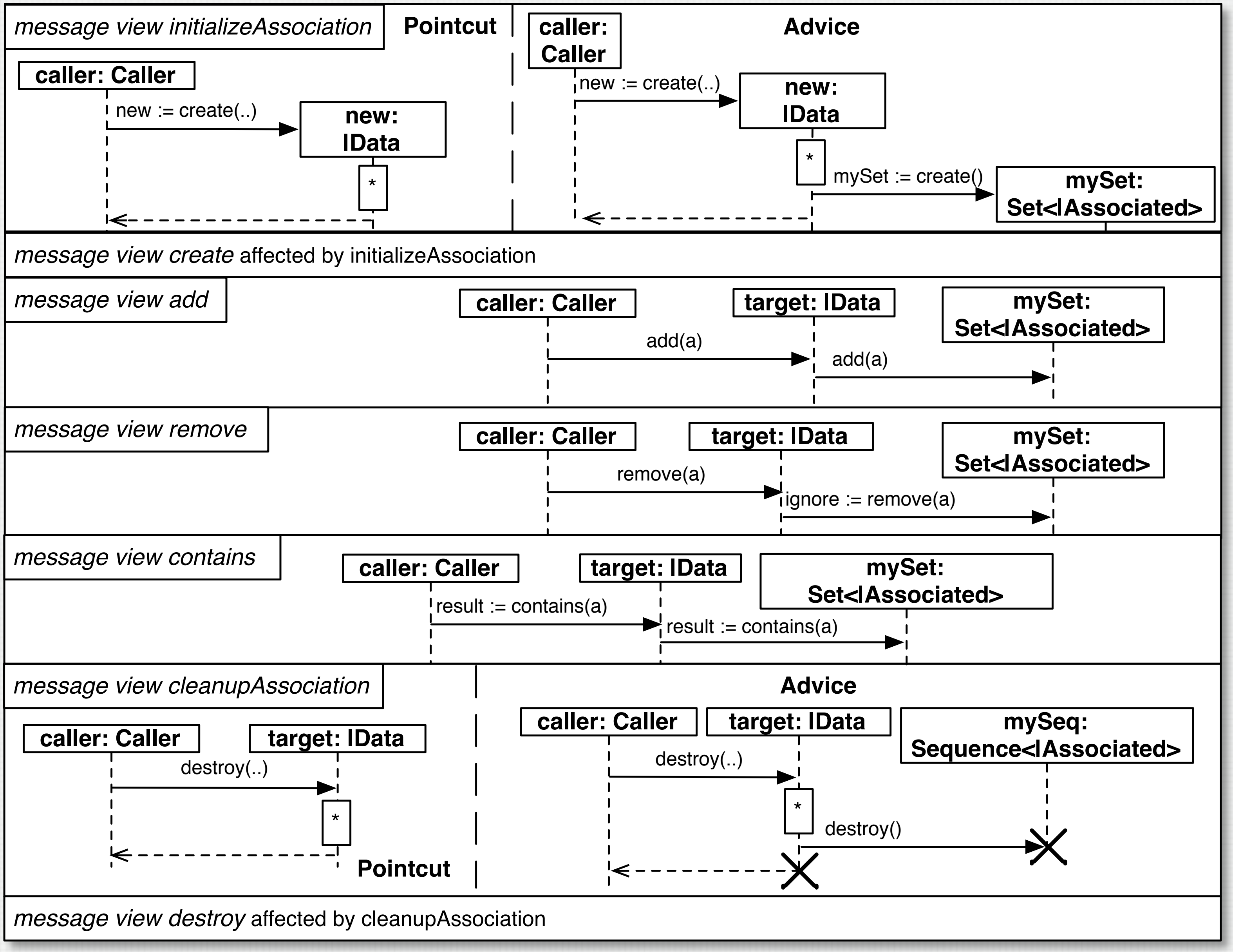
Implementation:
Set<IAssociated>:

any java.util.Set<IAssociated>

Usage Interface

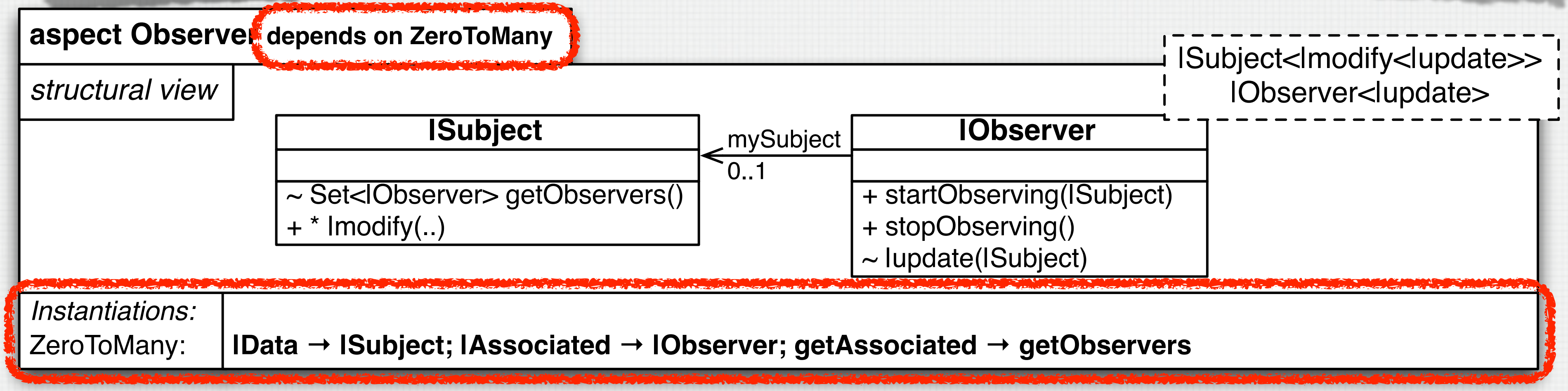
Mapping to Java

ZeroToMany Behaviour



Observer Reusing ZeroToMany

- To reuse ZeroToMany, the Observer Aspect must:
  - Declare a dependency on ZeroToMany
  - Instantiate ZeroToMany explicitly by mapping all mandatory instantiation parameters to model elements in Observer

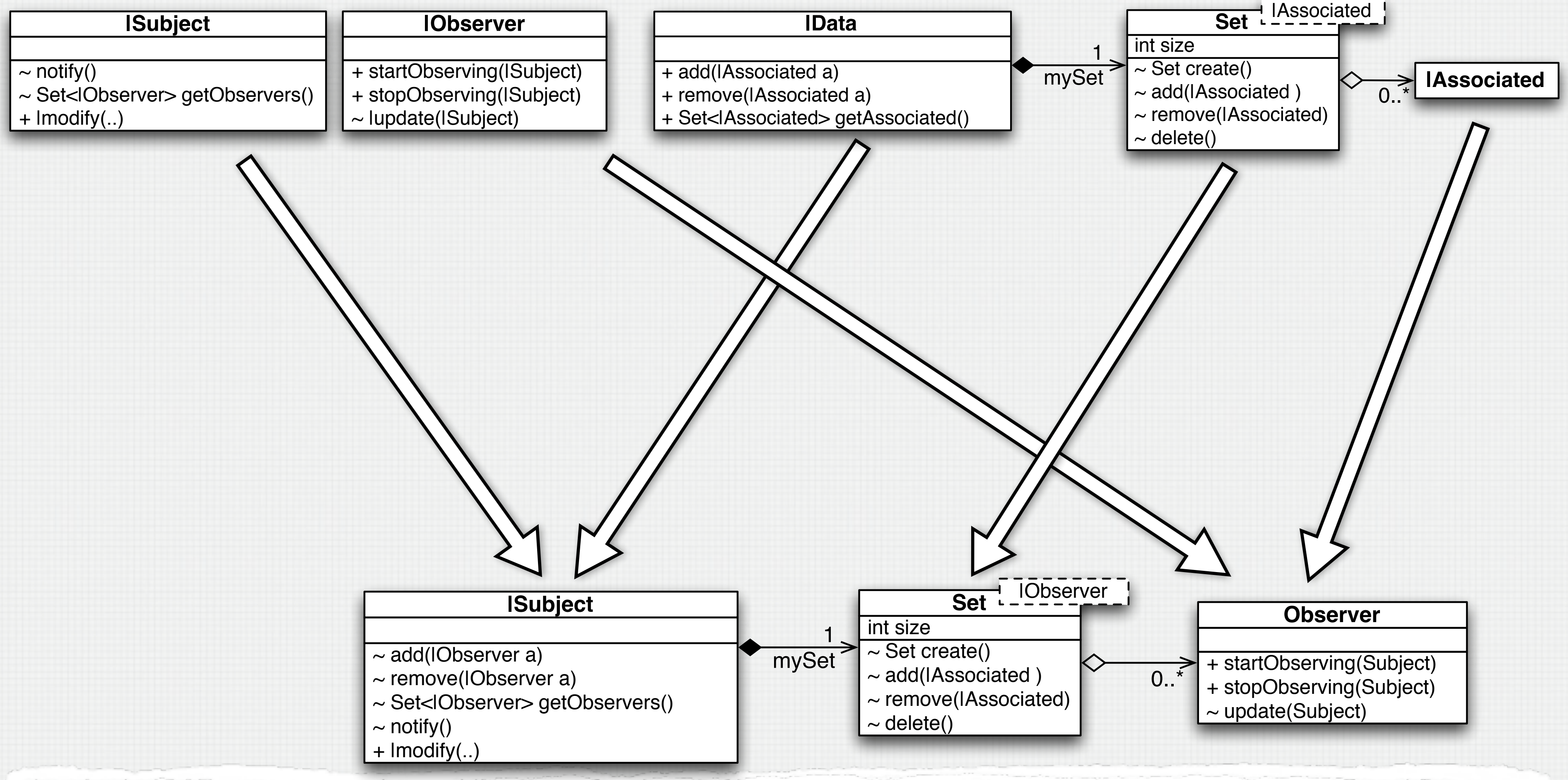


aspect Observer depends on ZeroToMany
structural view
ISubject<Imodify<Iupdate>>
IObserver<Iupdate>
ISubject
~ Set<IObserver> getObservers()
+ * Imodify(..)
mySubject
0..1
IObserver
+ startObserving(ISubject)
+ stopObserving()
~ Iupdate(ISubject)
Instantiations:
ZeroToMany:
IData → ISubject; IAssociated → IObserver; getAssociated → getObservers

- Non-mandatory model elements can also be mapped
  - The getObservers operation reuses the behaviour of getAssociated, or, in other words, getAssociated is “reexposed” as getObservers

Structural View Weaving

Instantiations:	
ZeroToMany:	**IData → ISubject; IAssociated → IObserver; getAssociated → getObservers**



ISubject
~ notify()
~ Set<IObserver> getObservers()
+ Imodify(..)

IObserver
+ startObserving(ISubject)
+ stopObserving(ISubject)
~ Iupdate(ISubject)

IData
+ add(IAssociated a)
+ remove(IAssociated a)
+ Set<IAssociated> getAssociated()

1
mySet

Set IAssociated
int size
~ Set create()
~ add(IAssociated)
~ remove(IAssociated)
~ delete()

0..*

IAssociated

ISubject
~ add(IObserver a)
~ remove(IObserver a)
~ Set<IObserver> getObservers()
~ notify()
+ Imodify(..)

1
mySet

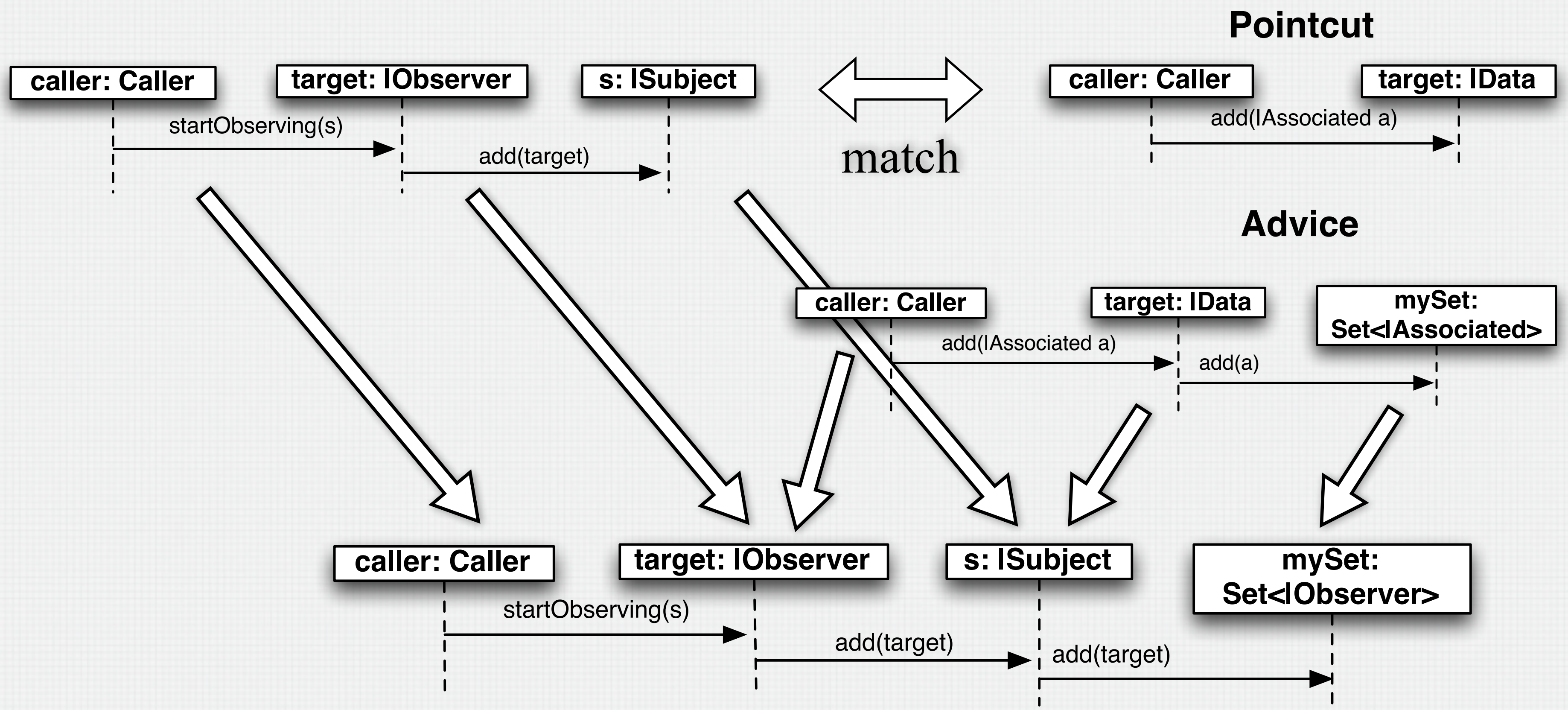
Set IObserver
int size
~ Set create()
~ add(IAssociated)
~ remove(IAssociated)
~ delete()

0..*

Observer
+ startObserving(Subject)
+ stopObserving(Subject)
~ update(Subject)

- Weaving of Class Diagrams is performed by merging classes, attributes, operations and associations, according to instantiation directives

Message View Weaving



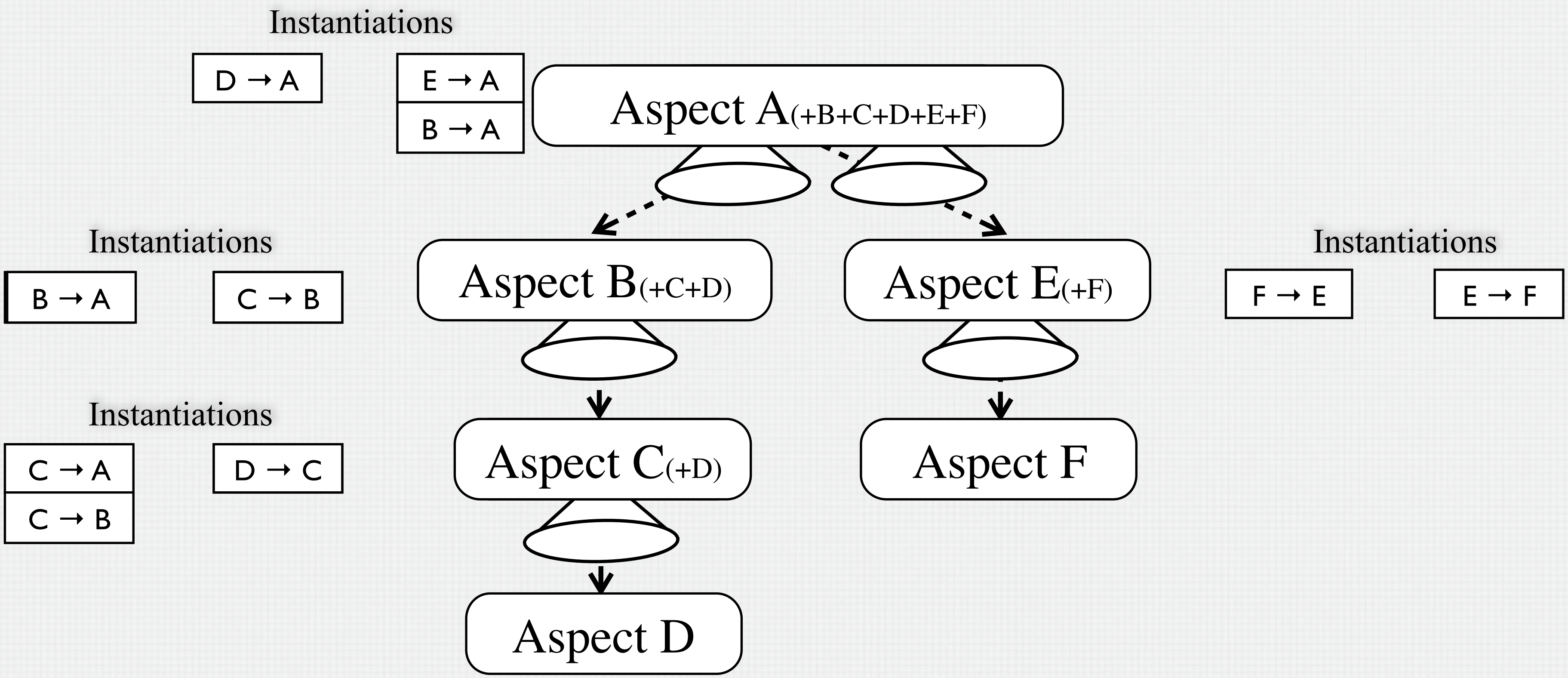
- Weaving of Sequence Diagrams is performed by pattern matching the pointcut lifelines and messages in the base sequence diagram according to the instantiation directives, and then replacing all found occurrences with the advice sequence diagram

Generating the "Woven" Model

- Model weaver creates an independent aspect model by weaving the lower-level aspects into the higher-level aspect
  - The independent aspect model includes all the structure and behaviour defined by the lower-level aspects
- Why is it interesting to generate an independent aspect model or "woven model"?
  - During Design: Looking at parts of the woven model allows a modeler to better understand the structure or behaviour of the result of the weaving
  - Debugging: The woven model can be model-checked, or executed
  - During Weaving: The performance of the weaver is increased if the high-level aspect is applied several times
  - Generation of the final application model, and ultimately code

Recursive Weaving Algorithm

- Recursively traverse the aspect dependency tree, weaving lower-level aspects into higher-level aspects according to instantiation directives



Consistency Checks

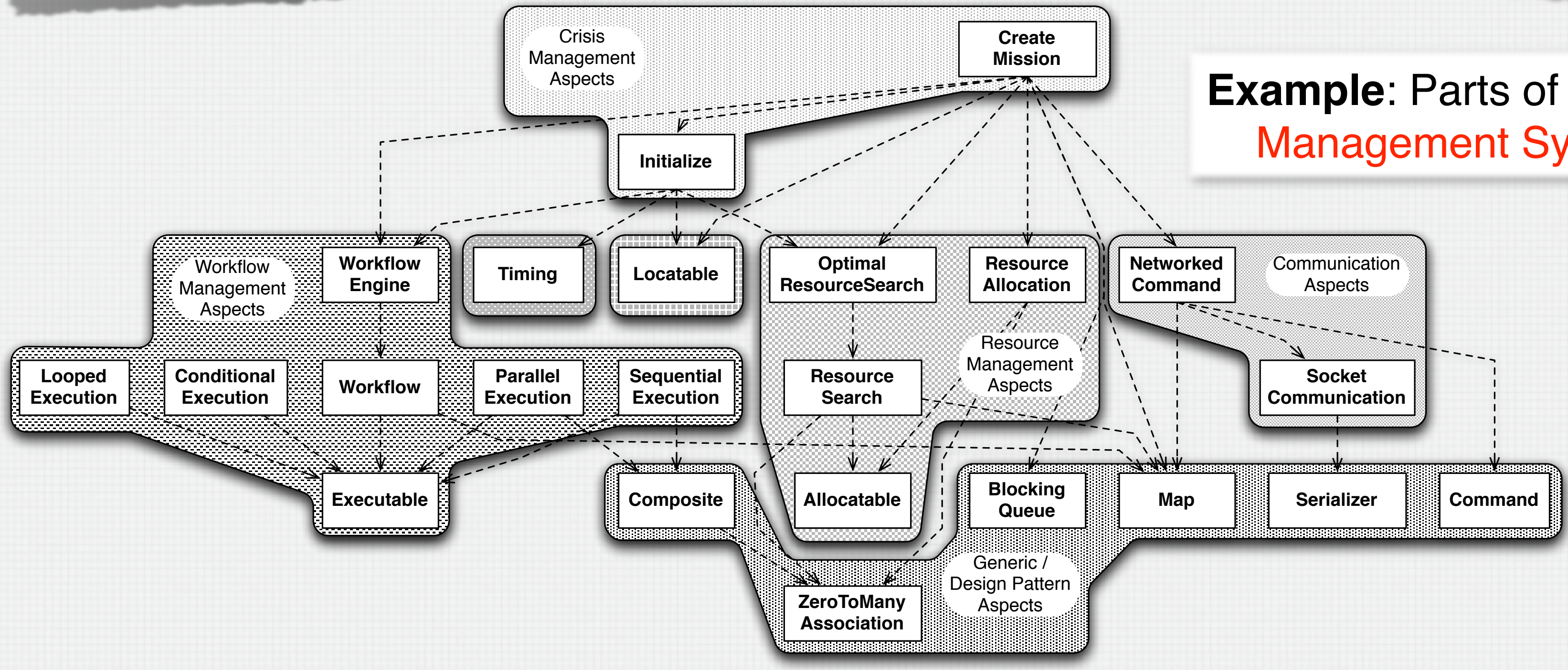
- When creating the aspect model, consistency checks are performed among views
  - Standard consistency checks
    - E.g., an operation can only be invoked in a message view if it is defined in the structural view)
  - Conformance between the message views and the state views (model checking)
- Before (during) the weaving, consistency checks are performed among the directives defined at different levels
  - For state views, instantiations and default instantiations specified in a higher-level aspect must designate the same state or sub states of the mappings specified in a lower-level aspect.
  - For message views, instantiations must be identical or more specific than default instantiations
- After executing the weaving, consistency checks are performed among the woven views
  - Conformance of the partial sequences of messages defined for each object instance (i.e., life line) in the woven sequence diagram with the protocol defined in the state view of the corresponding class (model checking)

RAM Tool: TouchRAM

- Download TouchRAM
  - http://www.cs.mcgill.ca/~joerg/SEL/TouchRAM.html
  - System requirements: Windows/Linux/MacOS running Java 1.5, 3D graphics
- Intuitive editing using multi-touch gestures
  - Support for simple (and advanced) gestures
- Significant speedup for
  - Navigating big models
  - Moving / rearranging classes
  - Establishing mappings between design concerns
- Simultaneous support for multi-touch (TUIO) as well as mouse / keyboard input
- Currently Support
  - Full Support for Structural Views
  - Display Support for Message Views

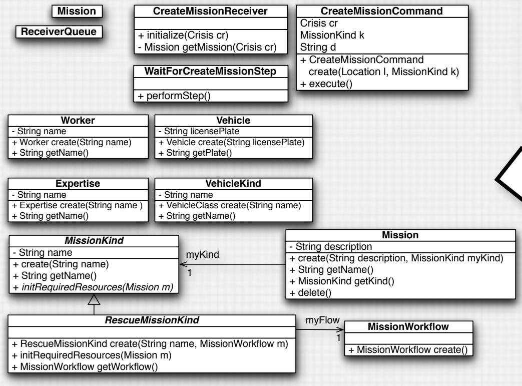
Design using RAM

- RAM targeted at Incremental Software Design
  - Final Application Model is formed by combining many (small) Reusable Aspect Models
  - Hierarchy of Aspects (Dependencies / Conflict Resolution)
  - Reuse of Aspects



Example: Parts of a Crisis Management System

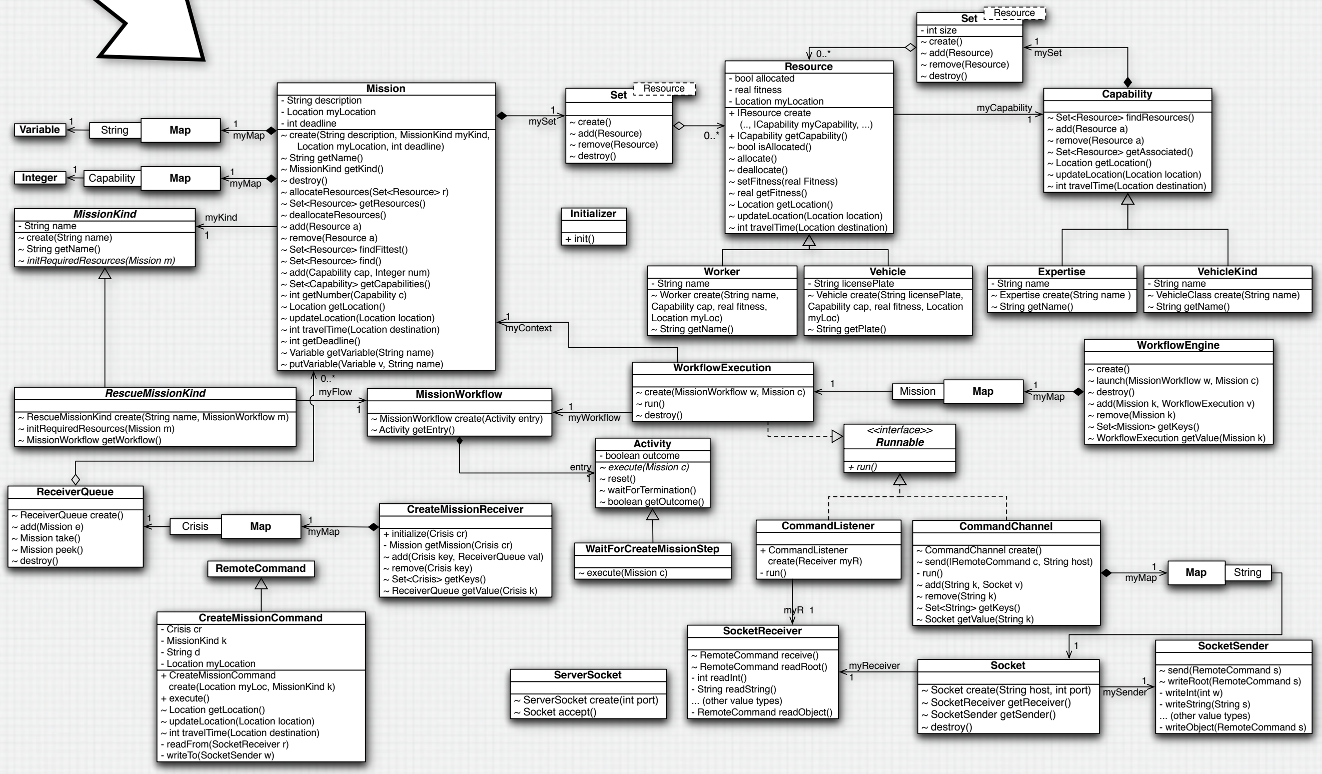
Scalability



High-level (Domain) Model

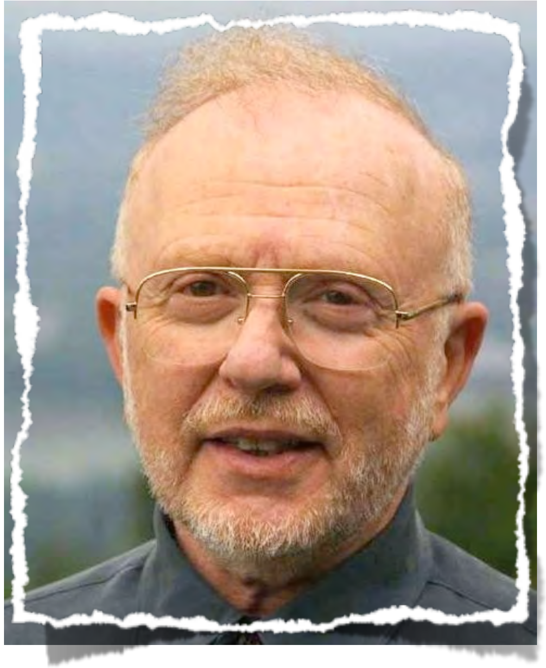
Example: Parts of a Crisis Management System

When combined with (Reusable) Aspects for Networking, Resource Allocation, Concurrency, Workflow Execution, Design Patterns


Variable
1
String
Map
1
myMap
Integer
1
Capability
Map
1
myMap
MissionKind
- String name
~ create(String name)
~ String getName()
~ initRequiredResources(Mission m)
myKind
1
Mission
- String description
- Location myLocation
- int deadline
~ create(String description, MissionKind myKind, Location myLocation, int deadline)
~ String getName()
~ MissionKind getKind()
~ destroy()
~ allocateResources(Set<Resource> r)
~ Set<Resource> getResources()
~ deallocateResources()
~ add(Resource a)
~ remove(Resource a)
~ Set<Resource> findFittest()
~ Set<Resource> find()
~ add(Capability cap, Integer num)
~ Set<Capability> getCapabilities()
~ int getNumber(Capability c)
~ Location getLocation()
~ updateLocation(Location location)
~ int travelTime(Location destination)
~ int getDeadline()
~ Variable getVariable(String name)
~ putVariable(Variable v, String name)
1
mySet
Set
Resource
~ create()
~ add(Resource)
~ remove(Resource)
~ destroy()
0..*
Resource
- bool allocated
- real fitness
- Location myLocation
+ IResource create (.., ICapability myCapability, ...)
+ ICapability getCapability()
~ bool isAllocated()
~ allocate()
~ deallocate()
~ setFitness(real Fitness)
~ real getFitness()
~ Location getLocation()
~ updateLocation(Location location)
~ int travelTime(Location destination)
0..*
Set
Resource
- int size
~ create()
~ add(Resource)
~ remove(Resource)
~ destroy()
1
mySet
myCapability
1
Capability
~ Set<Resource> findResources()
~ add(Resource a)
~ remove(Resource a)
~ Set<Resource> getAssociated()
~ Location getLocation()
~ updateLocation(Location location)
~ int travelTime(Location destination)
Initializer
+ init()
Worker
- String name
~ Worker create(String name, Capability cap, real fitness, Location myLoc)
~ String getName()
Vehicle
- String licensePlate
~ Vehicle create(String licensePlate, Capability cap, real fitness, Location myLoc)
~ String getPlate()
Expertise
- String name
~ Expertise create(String name)
~ String getName()
VehicleKind
- String name
~ VehicleClass create(String name)
~ String getName()
1
myContext
WorkflowExecution
~ create(MissionWorkflow w, Mission c)
~ run()
~ destroy()
1
Mission
Map
1
myMap
WorkflowEngine
~ create()
~ launch(MissionWorkflow w, Mission c)
~ destroy()
~ add(Mission k, WorkflowExecution v)
~ remove(Mission k)
~ Set<Mission> getKeys()
~ WorkflowExecution getValue(Mission k)
RescueMissionKind
~ RescueMissionKind create(String name, MissionWorkflow m)
~ initRequiredResources(Mission m)
~ MissionWorkflow getWorkflow()
0..*
myFlow
1
MissionWorkflow
~ MissionWorkflow create(Activity entry)
~ Activity getEntry()
1
myWorkflow
<<interface>>
Runnable
+ run()
Activity
- boolean outcome
~ execute(Mission c)
~ reset()
~ waitForTermination()
~ boolean getOutcome()
entry
1
ReceiverQueue
~ ReceiverQueue create()
~ add(Mission e)
~ Mission take()
~ Mission peek()
~ destroy()
1
Crisis
Map
1
myMap
CreateMissionReceiver
+ initialize(Crisis cr)
- Mission getMission(Crisis cr)
~ add(Crisis key, ReceiverQueue val)
~ remove(Crisis key)
~ Set<Crisis> getKeys()
~ ReceiverQueue getValue(Crisis k)
WaitForCreateMissionStep
~ execute(Mission c)
CommandListener
+ CommandListener create(Receiver myR)
- run()
CommandChannel
~ CommandChannel create()
~ send(IRemoteCommand c, String host)
- run()
~ add(String k, Socket v)
~ remove(String k)
~ Set<String> getKeys()
~ Socket getValue(String k)
1
myMap
Map
String
RemoteCommand
CreateMissionCommand
- Crisis cr
- MissionKind k
- String d
- Location myLocation
+ CreateMissionCommand create(Location myLoc, MissionKind k)
+ execute()
~ Location getLocation()
~ updateLocation(Location location)
~ int travelTime(Location destination)
- readFrom(SocketReceiver r)
- writeTo(SocketSender w)
myR 1
SocketReceiver
~ RemoteCommand receive()
~ RemoteCommand readRoot()
- int readInt()
- String readString()
... (other value types)
- RemoteCommand readObject()
ServerSocket
~ ServerSocket create(int port)
~ Socket accept()
myReceiver
1
1
Socket
~ Socket create(String host, int port)
~ SocketReceiver getReceiver()
~ SocketSender getSender()
~ destroy()
1
mySender
SocketSender
~ send(RemoteCommand s)
~ writeRoot(RemoteCommand s)
- writeInt(int w)
- writeString(String s)
... (other value types)
- writeObject(RemoteCommand s)

Information Hiding

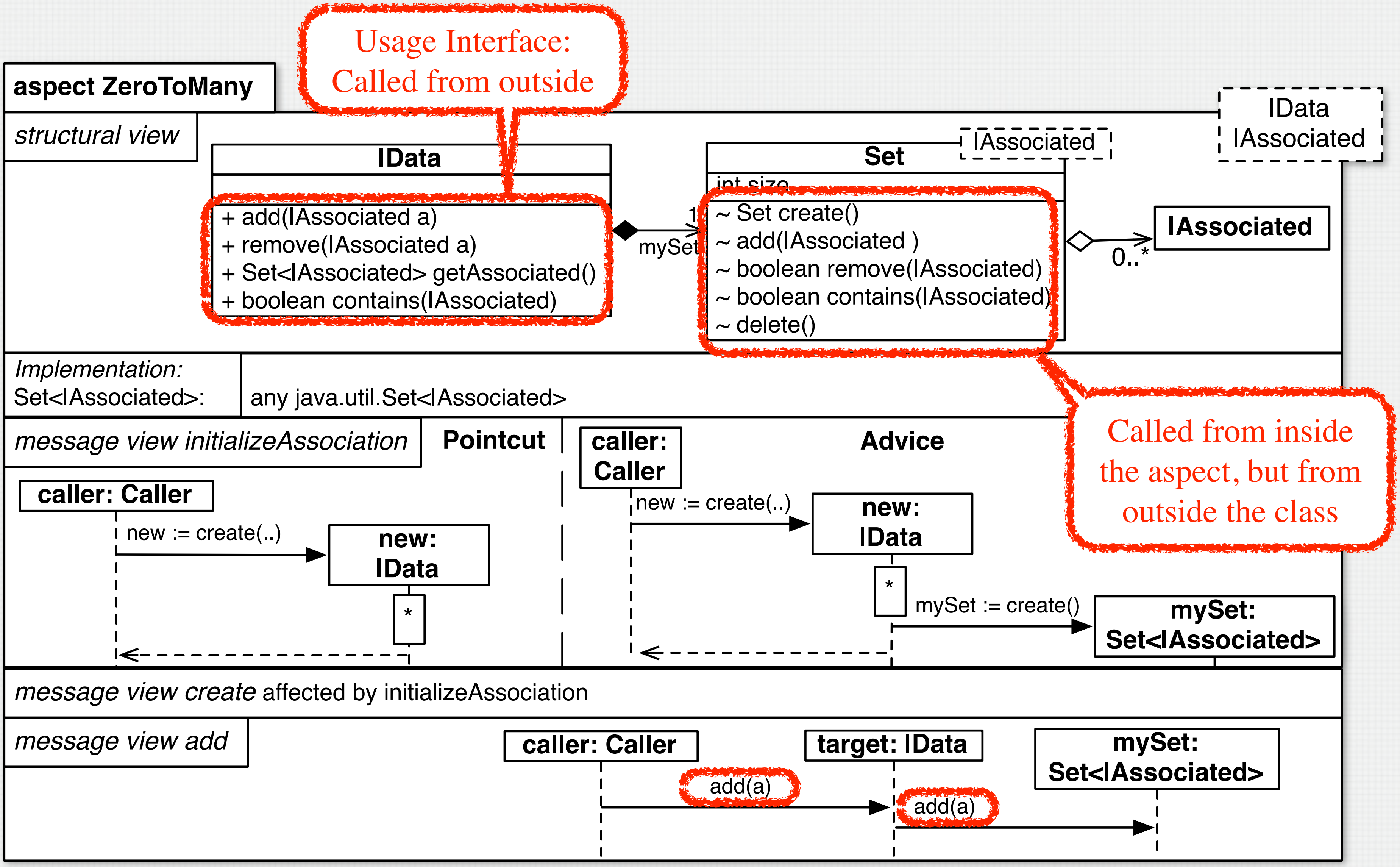
- Introduced in famous paper by Parnas in 1972
- Modules
  - Units of decomposition
- Encapsulation
  - Modules have well-defined boundaries
- Information Hiding
  - Decide what to put in the module's interface
  - Hide design decisions that are likely to change inside the module
- Object-orientation
  - Modules are classes
  - Interface includes attributes and operations
    - Public, visible to the outside
    - Protected, visible to the class and to subclasses
    - Private, visible to the class only



RAM Information Hiding

- Aspect-orientation
  - New dimension of modularization
  - Aspects encapsulate crosscutting concerns
- RAM Aspects have
  - Public usage and customization interfaces
    - Usage interface: Defines what is visible to everyone, in particular to the outside, i.e. to model elements that are not within the aspect package
      - Modifier “+”
    - Customization interface: Declares what entities need to be specialized / completed when the aspect is reused
  - An Intra-aspect interface
    - Defines what is visible within an aspect, i.e. what is visible to model elements that pertain to the same crosscutting concern
      - Modifier “~”
  - Class interface
    - Defines what is visible to only to model elements within the boundaries of the class (private) or subclasses (protected)
      - Modifier “-” or “#”
- As a result, RAM aspects can hide design decisions that crosscut the class boundaries

ZeroToManyAssociation Aspect



Usage Interface: Called from outside

aspect ZeroToMany

IData
IAssociated

structural view

IData

+ add(IAssociated a)
+ remove(IAssociated a)
+ Set<IAssociated> getAssociated()
+ boolean contains(IAssociated)

1
mySet

Set

IAssociated

int size

~ Set create()
~ add(IAssociated)
~ boolean remove(IAssociated)
~ boolean contains(IAssociated)
~ delete()

0..*

IAssociated

Implementation: Set<IAssociated>: any java.util.Set<IAssociated>

Called from inside the aspect, but from outside the class

message view initializeAssociation

Pointcut

caller: Caller

new := create(..)

new: IData

*

Advice

caller: Caller

new := create(..)

new: IData

*

mySet := create()

mySet: Set<IAssociated>

message view create affected by initializeAssociation

message view add

caller: Caller

target: IData

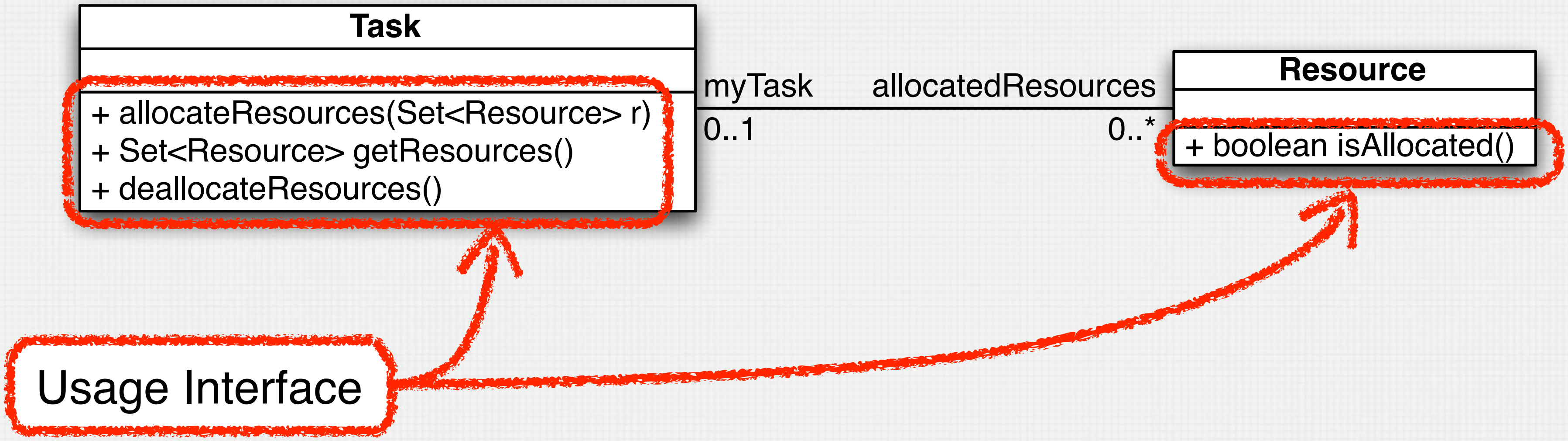
mySet: Set<IAssociated>

add(a)

add(a)

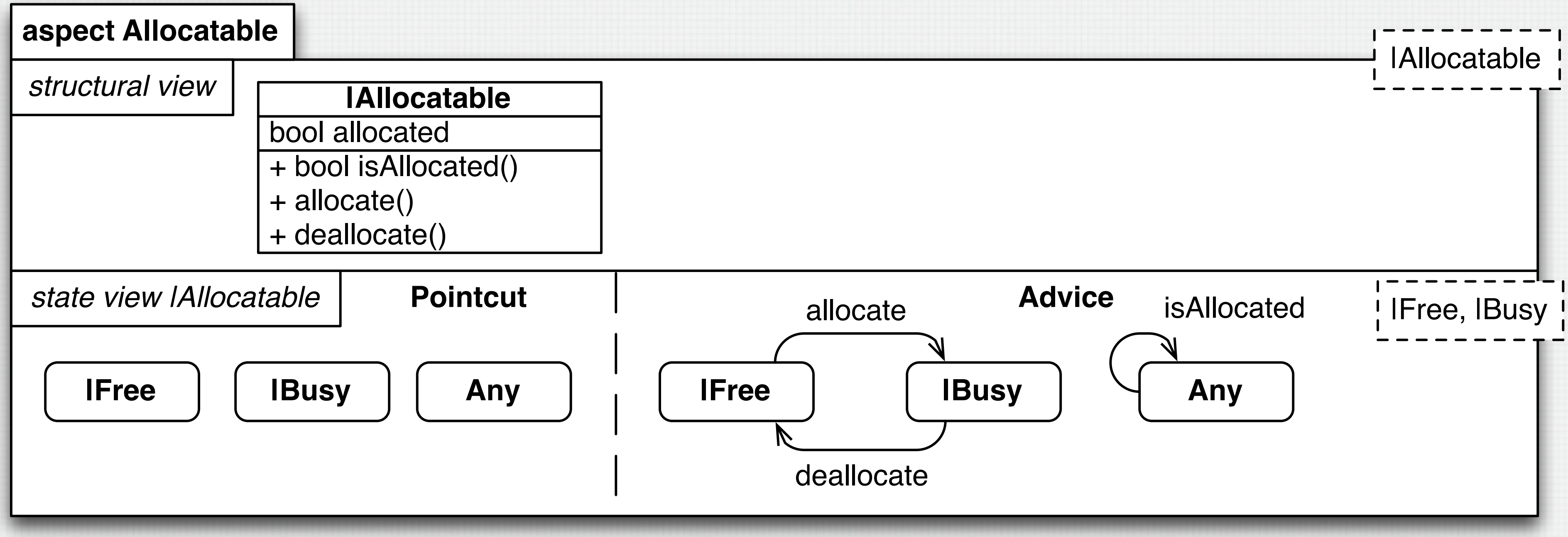
Resource Allocation

- We want to model the concern of resource allocation
  - Two types of entities: resources and tasks
  - Resources can be allocated to a task
    - A task can have many resources
    - A resource can only be allocated to one task at a given time
  - Desired behaviour
    - Allocating a set of resources to a task
    - Querying a resource to determine if it is free or not
    - Obtaining the set of resources allocated to a task



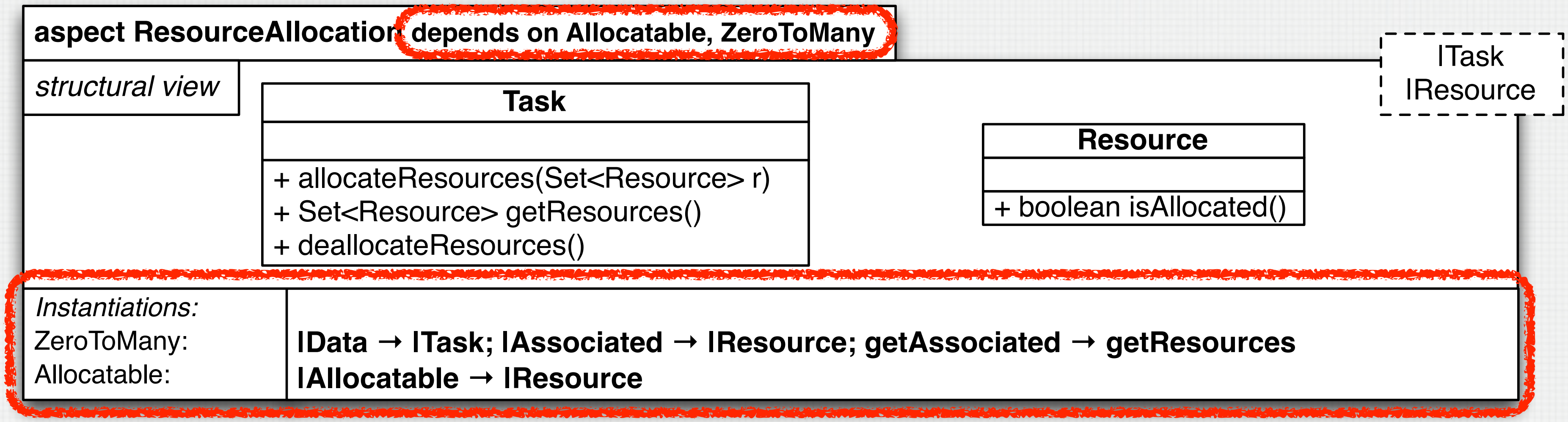
Allocatable

- Allocatable models the situation in which an entity can be allocated or free



ResourceAllocation

- ResourceAllocation is itself an aspect
  - The Resource and Task entities are both incomplete classes
  - ZeroToMany and Allocatable are reused



aspect ResourceAllocation depends on Allocatable, ZeroToMany

ITask
IResource

structural view

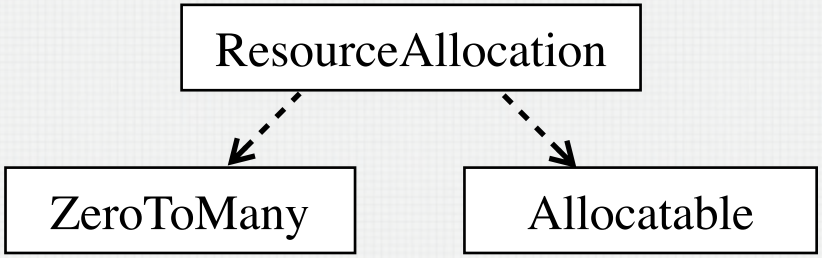
Task

+ allocateResources(Set<Resource> r)
+ Set<Resource> getResources()
+ deallocateResources()

Resource

+ boolean isAllocated()

Instantiations:
ZeroToMany: **IData → ITask; IAssociated → IResource; getAssociated → getResources**
Allocatable: **IAllocatable → IResource**



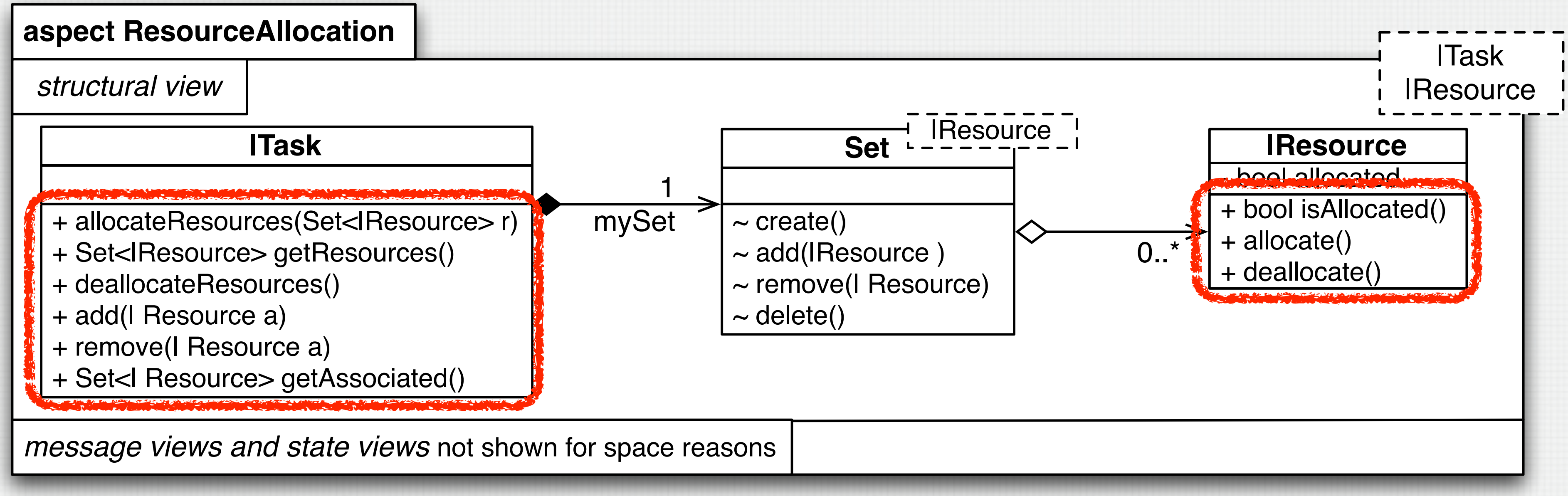
ResourceAllocation

ZeroToMany

Allocatable

Woven ResourceAllocation

- “Straightforward” merging does not yield a correctly encapsulated aspect model
  - Potential for inconsistent use



aspect ResourceAllocation

structural view

ITask
IResource

ITask

+ allocateResources(Set<IResource> r)
+ Set<IResource> getResources()
+ deallocateResources()
+ add(I Resource a)
+ remove(I Resource a)
+ Set<I Resource> getAssociated()

1
mySet

Set
IResource

~ create()
~ add(IResource)
~ remove(I Resource)
~ delete()

0..*

IResource
bool allocated

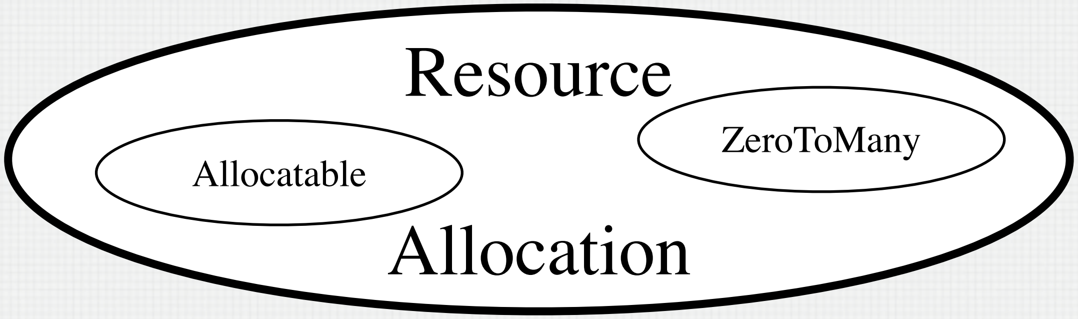
+ bool isAllocated()
+ allocate()
+ deallocate()

message views and state views not shown for space reasons

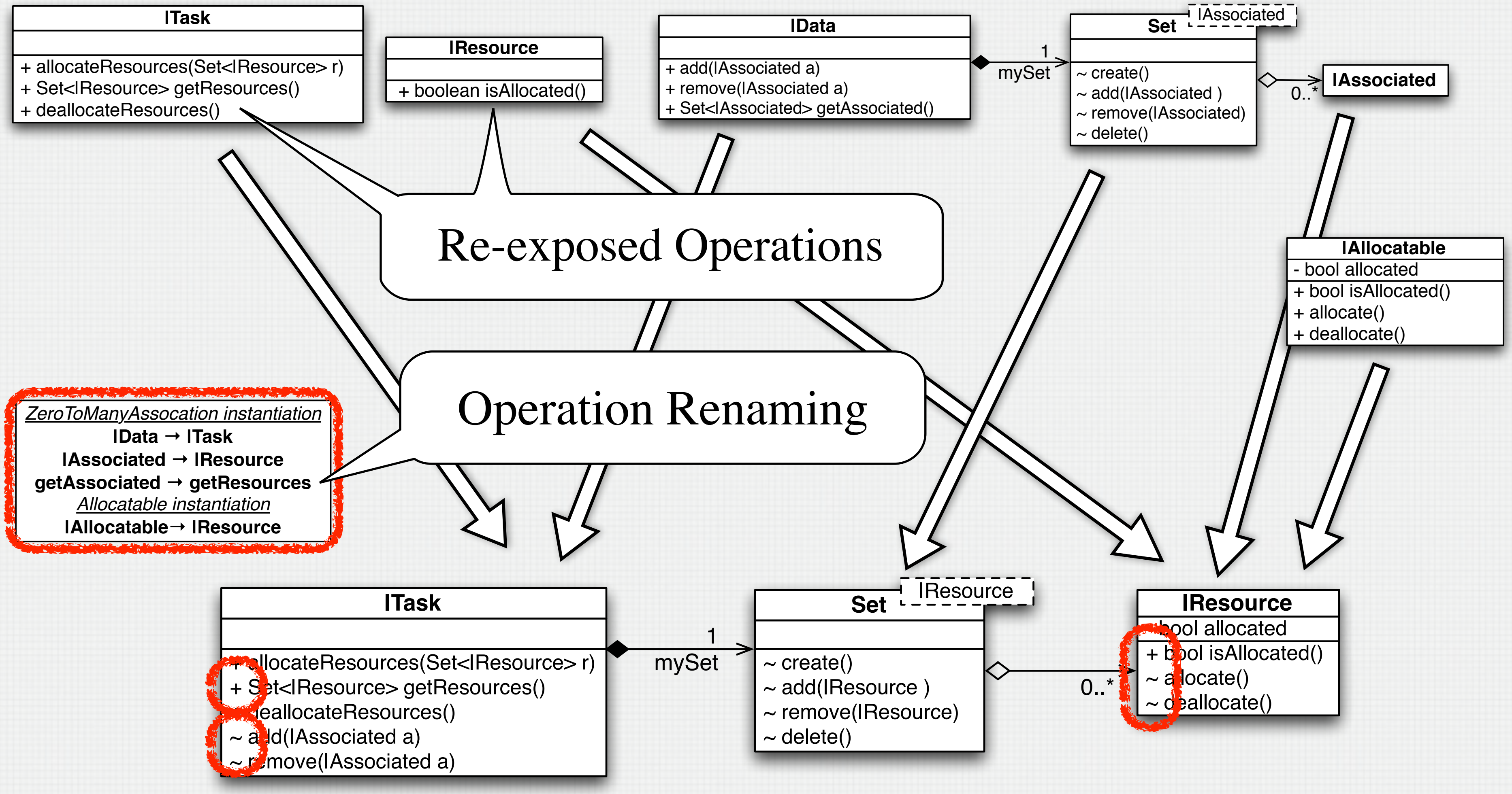
A call to add could allocate a resource to a task without calling the resource’s allocate operation or vice versa!

Depends On

- When an aspect internally depends on functionality of another aspect
  - The weaver automatically changes visibility of model elements from public to intra-aspect when weaving an aspect into a target model
  - The default policy is therefore to hide everything, i.e. the usage interface of the lower-level aspects are not visible to the outside
  - Selectively, a modeller can explicitly re-expose operations from a low-level aspect in its public interface
    - Possible to change the name of the operation in order to reflect the new semantic meaning
- Supports top-down and bottom-up modelling that crosses abstraction levels



“Depends On” Weaving



Re-exposed Operations

Operation Renaming

ZeroToManyAssocation instantiation

IData → ITask

IAssociated → IResource

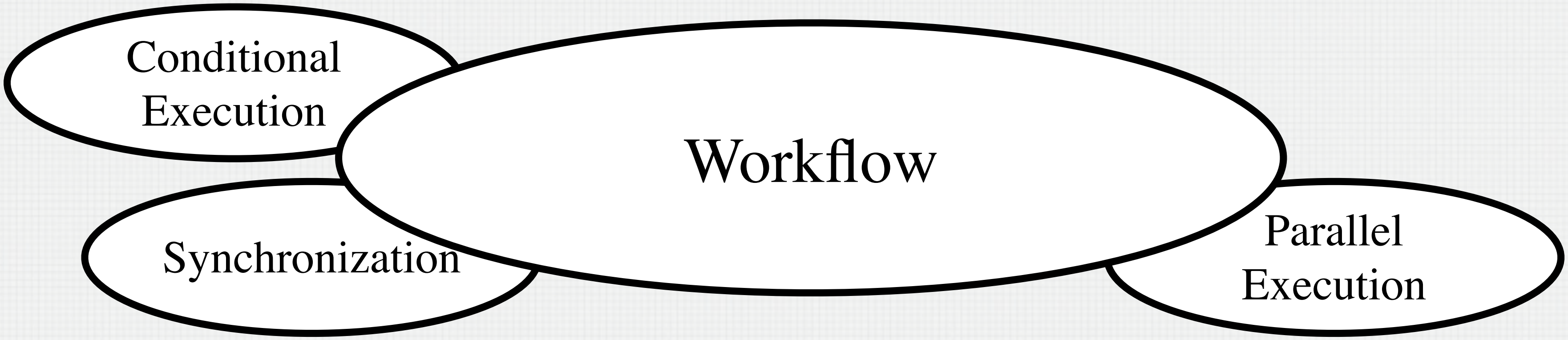
getAssociated → getResources

Allocatable instantiation

IAllocatable→ IResource

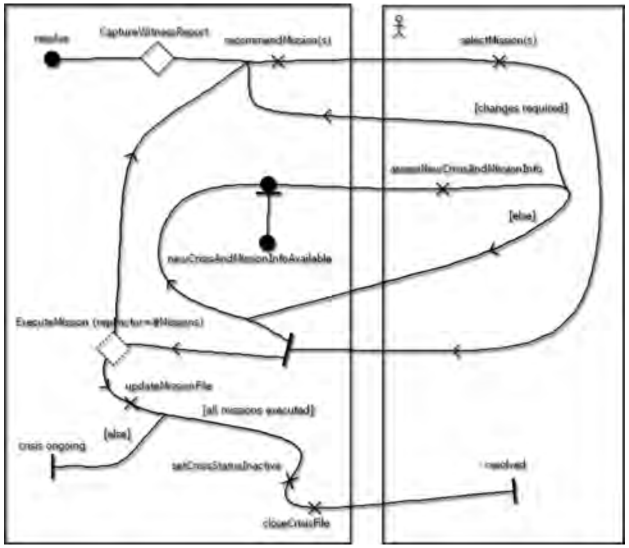
Extends

- When an aspect extends functionality of another aspect
  - By default, the weaver merges all classes / attributes that have the same name / all operations with the same signature
  - The weaver keeps the visibility of model elements of the extended aspect
  - The resulting aspect model interface is the union of both interfaces
- Supports incremental modelling at the same level of abstraction
- Supports modelling of optional add-ons to design concerns



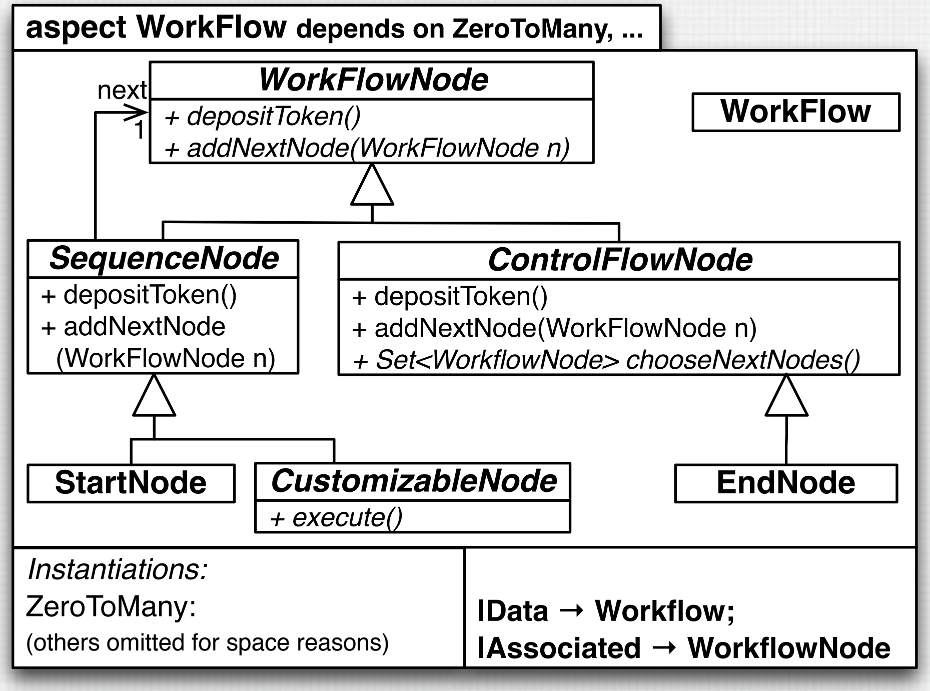
Workflows

- Workflow
  - A set of operations/activities that need to be completed in a certain order to fulfill a certain goal/task
- Modelling Notations for Specifying Workflows
  - UML Activity Diagrams
  - User Requirements Notation (URN)
- We applied RAM to design a Workflow Execution Middleware that supports the execution of URN models
  - Only parts of the structural model are presented here



RAM Incremental WF Design

- Workflow aspect defines simple, sequential workflows
  - A Workflow defines Workflow Nodes, which can either be SequenceNodes or ControlFlowNodes
  - StartNodes and CustomizableNodes are SequenceNodes, and EndNodes are ControlFlowNodes
- A WorkFlow has a set of workflow nodes associated with it
  - Structure provided by the lower-level aspect ZeroToMany



aspect WorkFlow depends on ZeroToMany, ...

WorkFlowNode
+ depositToken()
+ addNextNode(WorkFlowNode n)

next
1

WorkFlow

SequenceNode
+ depositToken()
+ addNextNode (WorkFlowNode n)

ControlFlowNode
+ depositToken()
+ addNextNode(WorkFlowNode n)
+ Set<WorkflowNode> chooseNextNodes()

StartNode

CustomizableNode
+ execute()

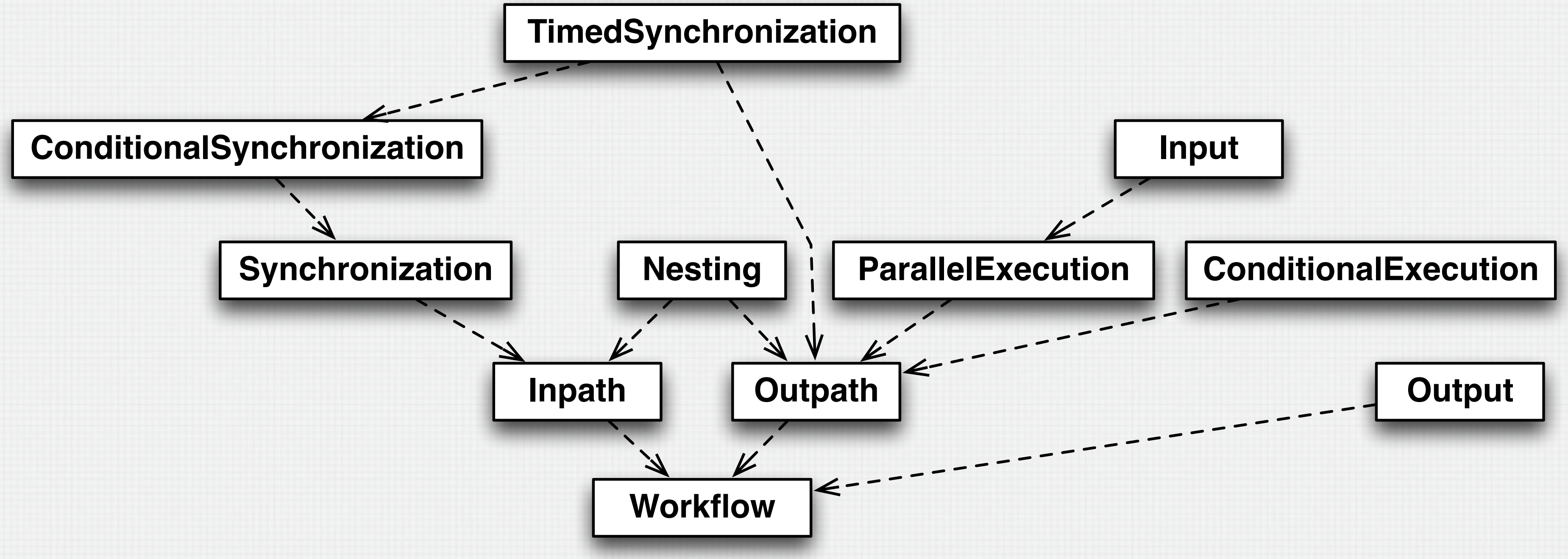
EndNode

Instantiations:
ZeroToMany:
(others omitted for space reasons)

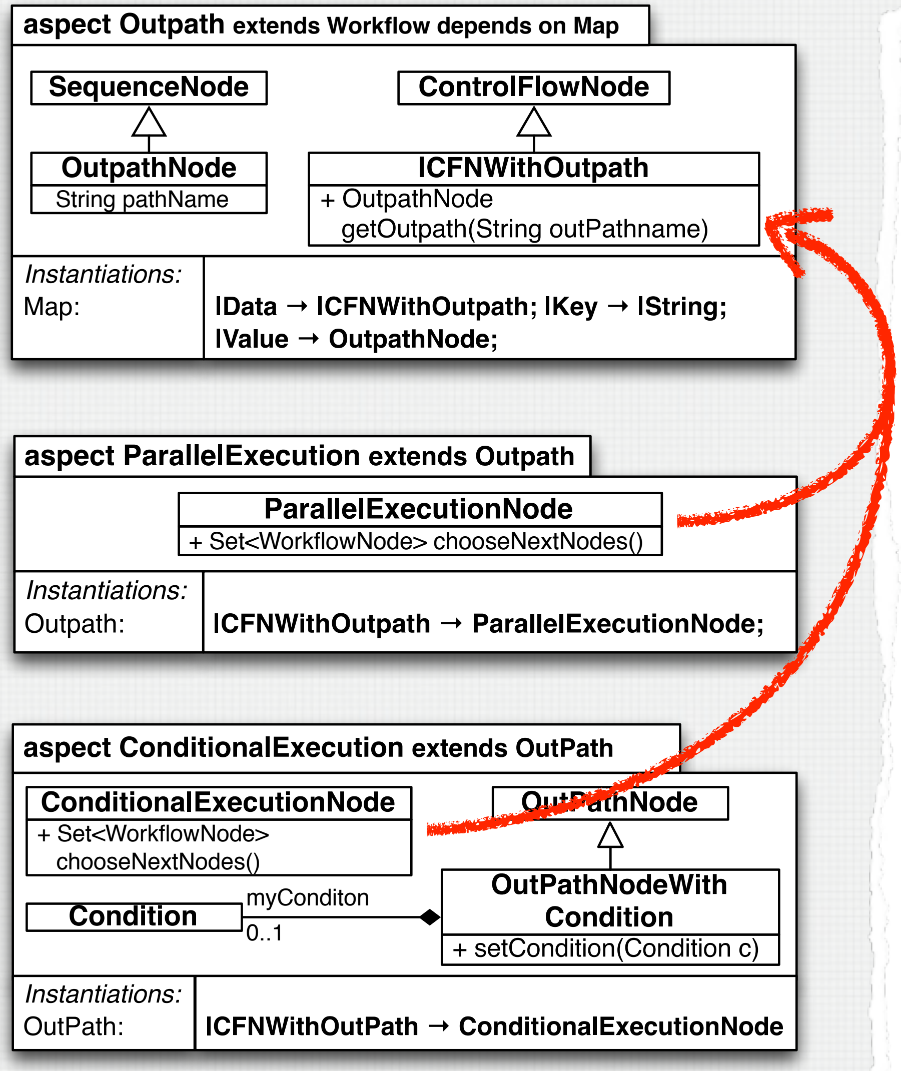
IData → Workflow;
IAssociated → WorkflowNode

Incremental Modelling

- Enables decomposition of big software design models into models of manageable size
- Enables specification of designs with multiple variations and software product lines



Workflow Extensions



aspect Outpath extends Workflow depends on Map

SequenceNode

ControlFlowNode

OutpathNode

String pathName

ICFNWithOutpath

+ OutpathNode getOutpath(String outPathname)

Instantiations:

Map:

IData → ICFNWithOutpath; IKey → IString; IValue → OutpathNode;

aspect ParallelExecution extends Outpath

ParallelExecutionNode

+ Set<WorkflowNode> chooseNextNodes()

Instantiations:

Outpath:

ICFNWithOutpath → ParallelExecutionNode;

aspect ConditionalExecution extends OutPath

ConditionalExecutionNode

+ Set<WorkflowNode> chooseNextNodes()

OutPathNode

Condition

myConditon

0..1

OutPathNodeWith Condition

+ setCondition(Condition c)

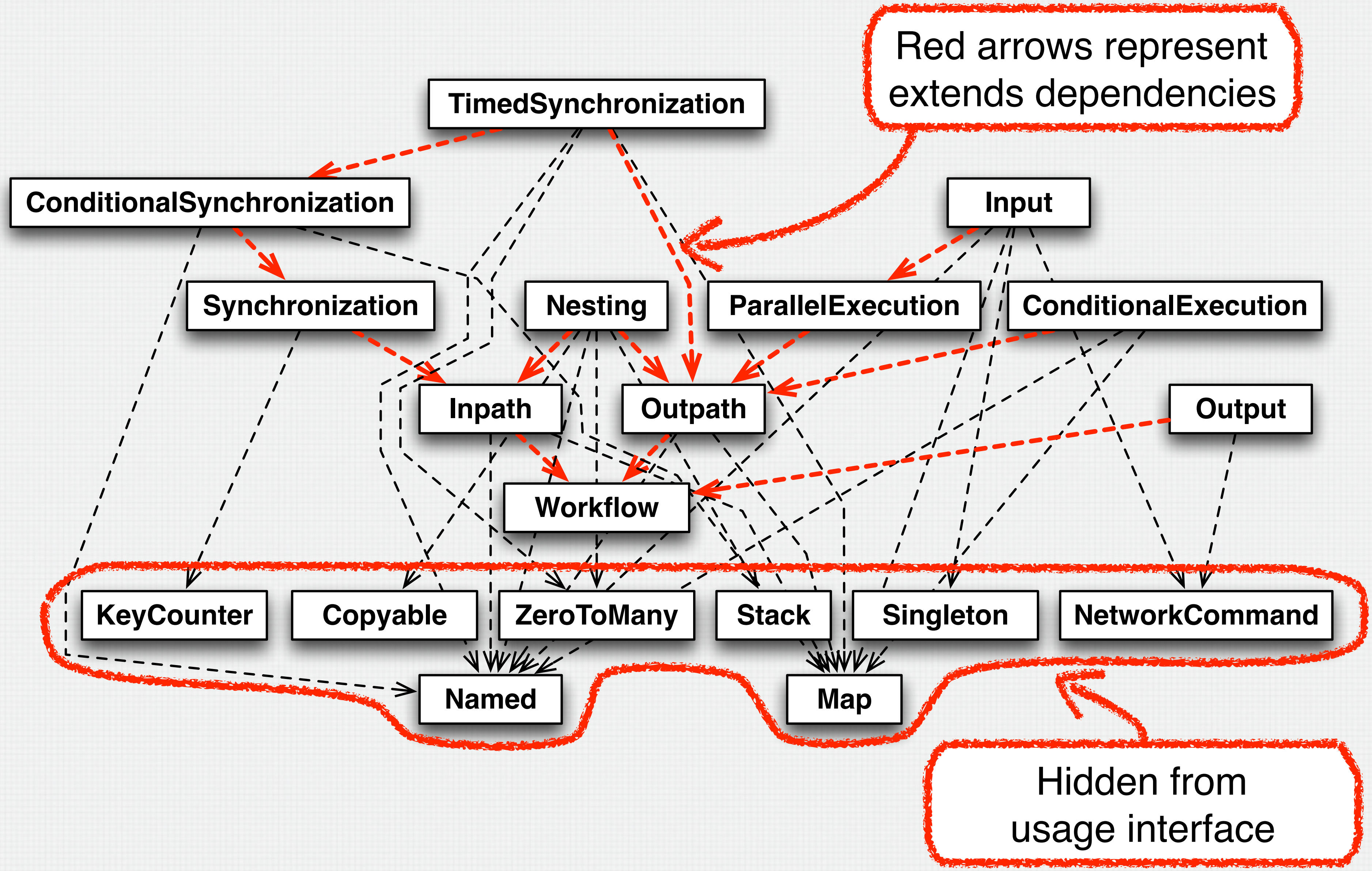
Instantiations:

OutPath:

ICFNWithOutPath → ConditionalExecutionNode

- Outpath encapsulates structure needed by control flow nodes with multiple named outgoing paths
  - ICFNWithOutpath is a mandatory instantiation parameter
  - Map is used to lookup outgoing paths by name
- ParallelExecution and ConditionalExecution extend Outpath
  - Instantiation directives map ParallelExecutionNode and ConditionalExecutionNode to ICFNWithOutpath

Internal Design

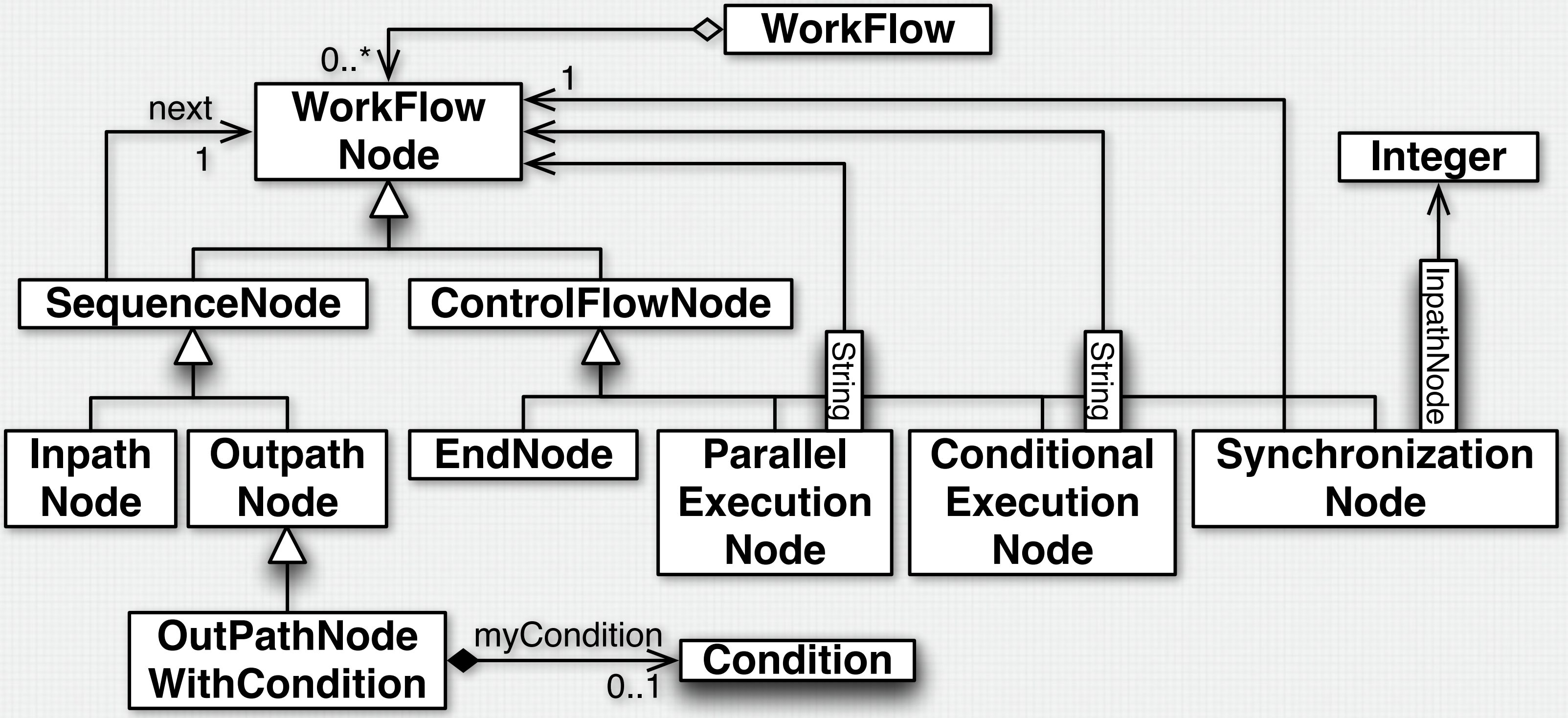


Red arrows represent extends dependencies

Hidden from usage interface

Woven Model

- For example, when choosing Workflow with ParallelExecution, ConditionalExecution and Synchronization



Aspect Case Study: Transactions

- A transaction groups together a set of operations on data objects, guaranteeing the ACID properties
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Concurrency Control (Isolation, Consistency)

- Prevent transactions from seeing intermediate, possibly inconsistent state
- Pessimistic vs. Optimistic
- Strict vs. Semantic-based Concurrency Control
  - Strict Concurrency Control Conflict Table

	Read(y)	Write(y)	Update(y)
Read(x)	No	Yes	Yes
Write(x)	Yes	Yes	Yes
Update(x)	Yes	Yes	Yes

Recovery (Atomicity, Durability, Consistency)

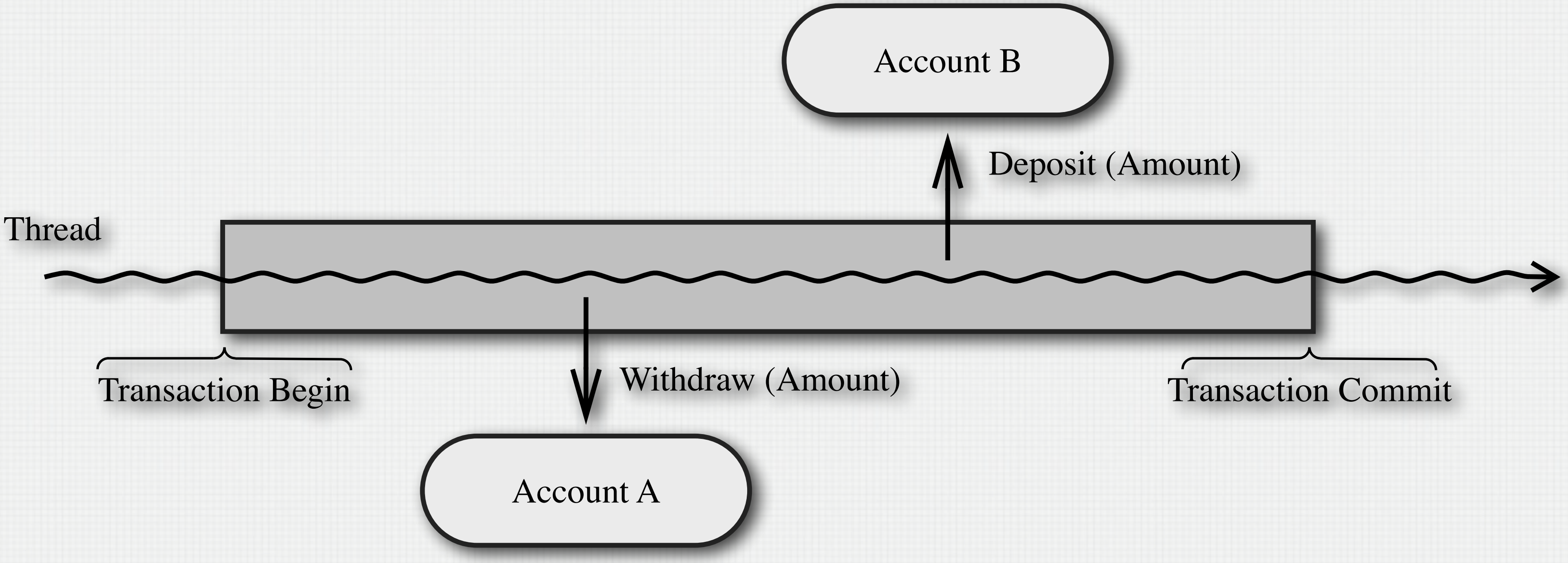
- On transaction abort
  - Undo the changes made on behalf of the transaction
- Snapshot-based recovery
  - In-place update
    - All operations are executed on one main copy of the state of a transactional object
    - Make a backup copy (or snapshot) before modifying
  - Deferred update
    - Each modifying transaction operates on a different copy
    - Upon transaction commit, the changes are propagated to the main copy
- Durability requires to save all committed changes to *stable storage*

At Run-Time

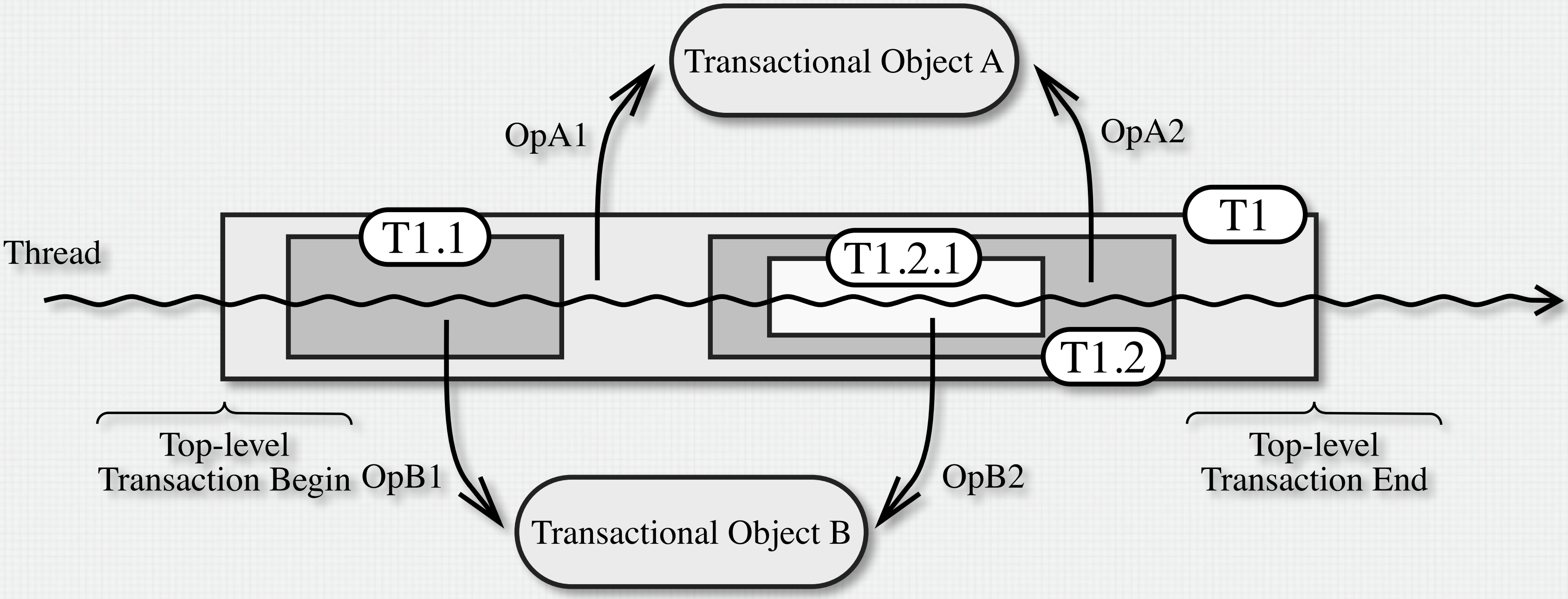
- Each time a method is invoked on a transactional object, the following actions must be taken:

1. Concurrency Control Prologue
2. Recovery Prologue
3. Method Execution
4. Recovery Epilogue
5. Concurrency Control Epilogue

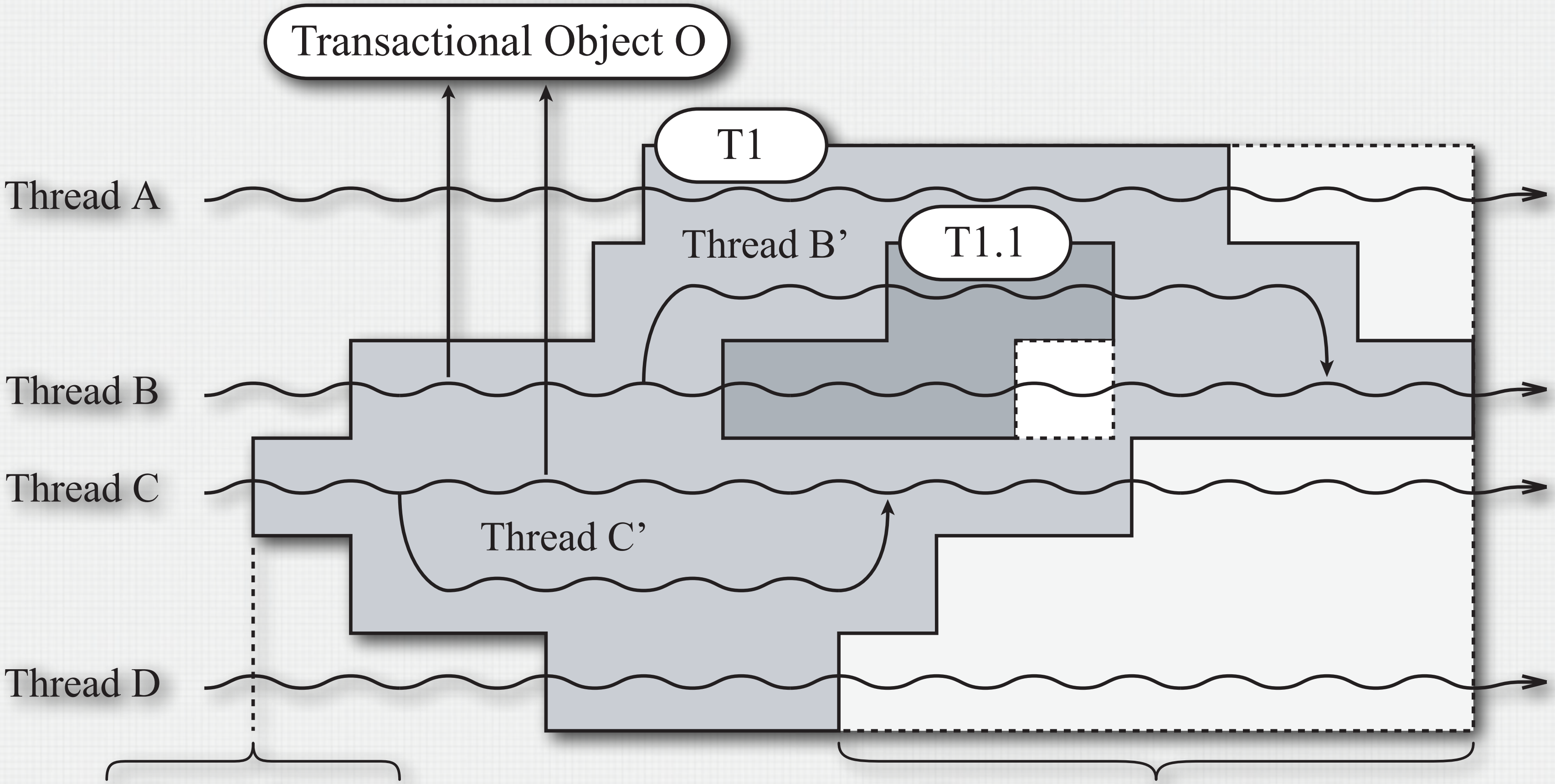
Flat Transactions



Nested Transactions



Open Multithreaded Transactions



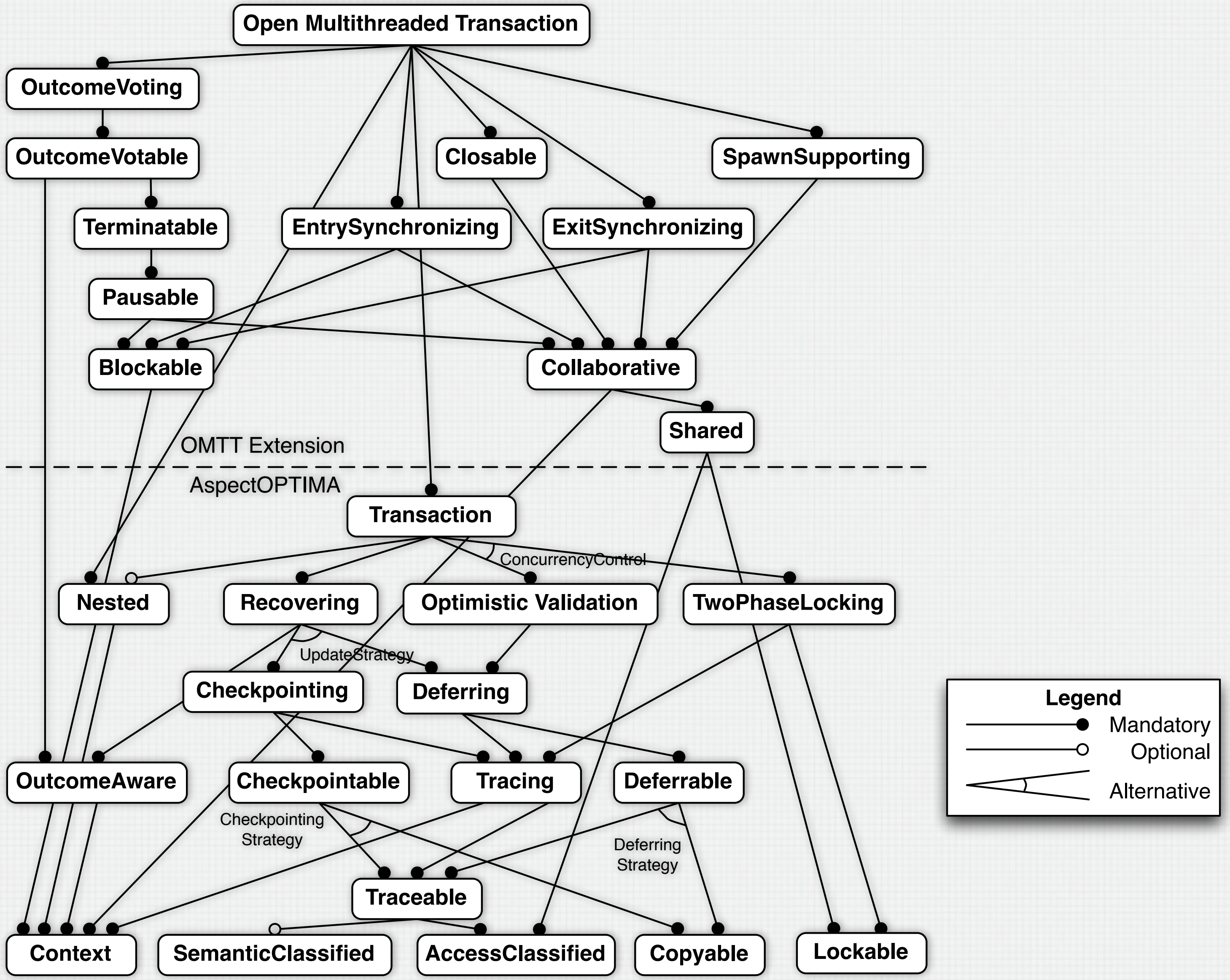
Thread C starts the transaction

Threads are blocked until the outcome of the transaction is known

AspectOptima Case Study

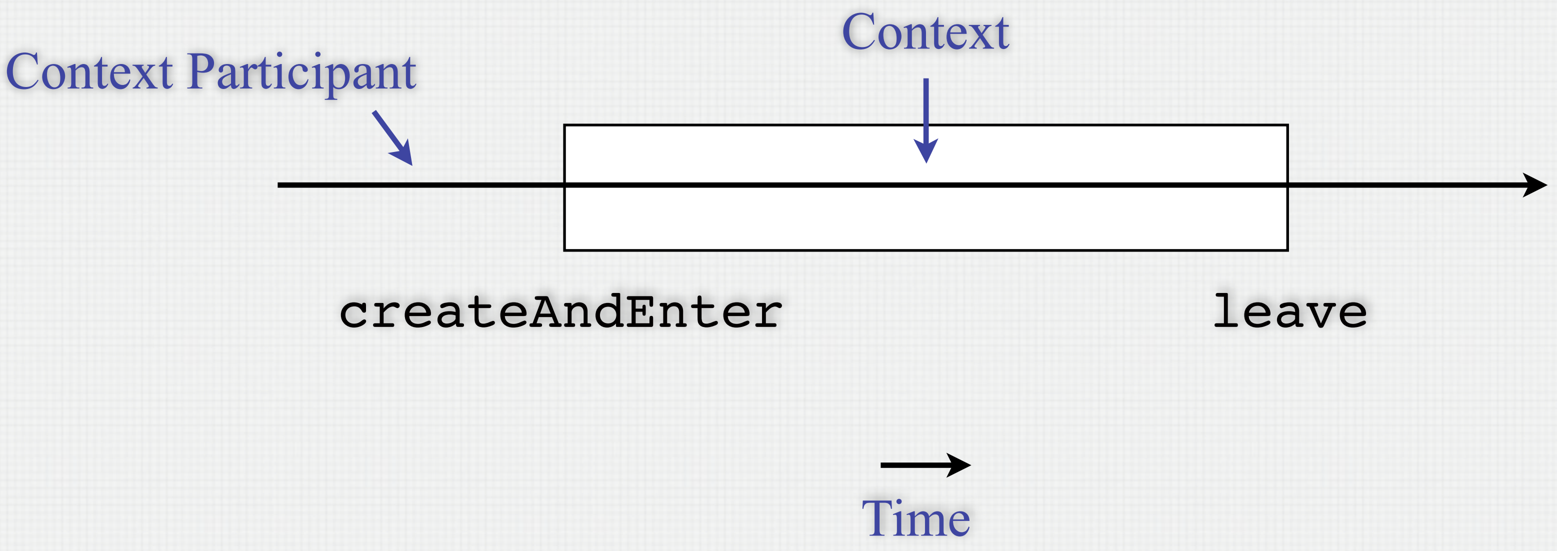
- Observations
  - Concurrency control and recovery are separate concerns at a higher level of abstraction
    - At the implementation level, the two concerns are tightly coupled
  - Most transaction models are related, i.e. they share common concepts
- Challenge
  - Is it possible to define many individually reusable aspects that, when put together in different ways, can implement various transaction models, concurrency control and recovery strategies?

AspectOptima Feature Diagram

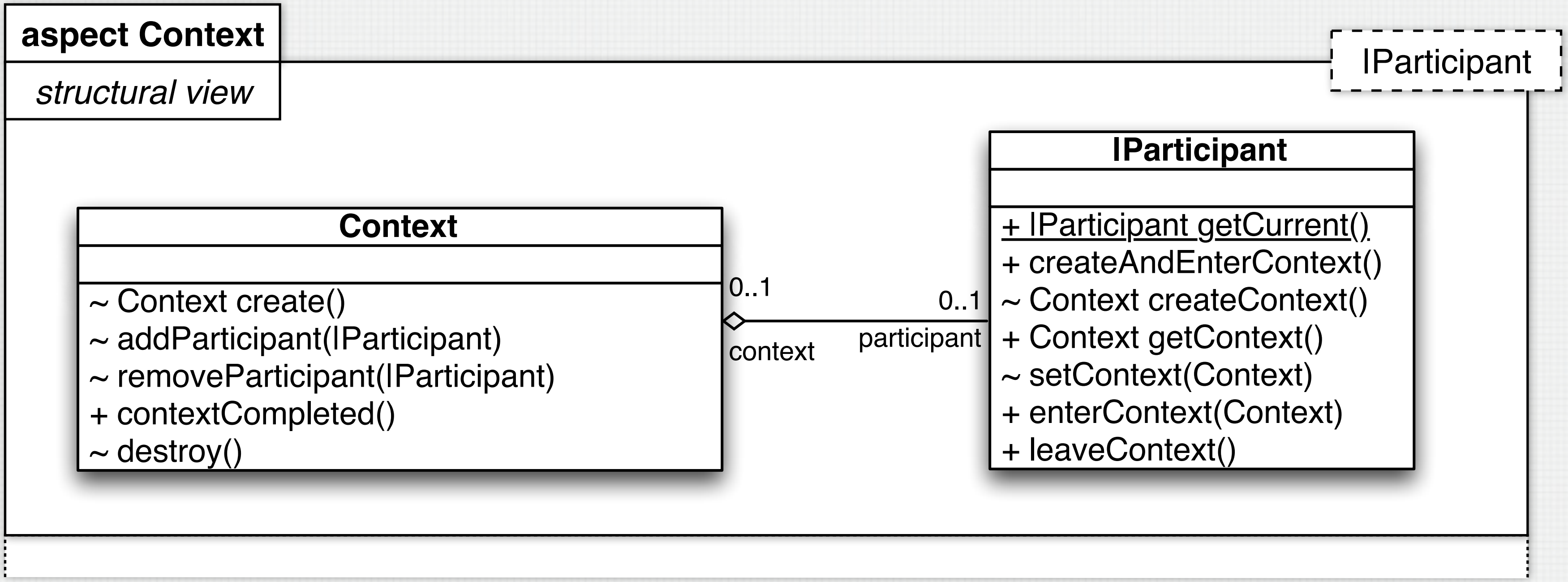


AspectOptima Core: ExecutionContext

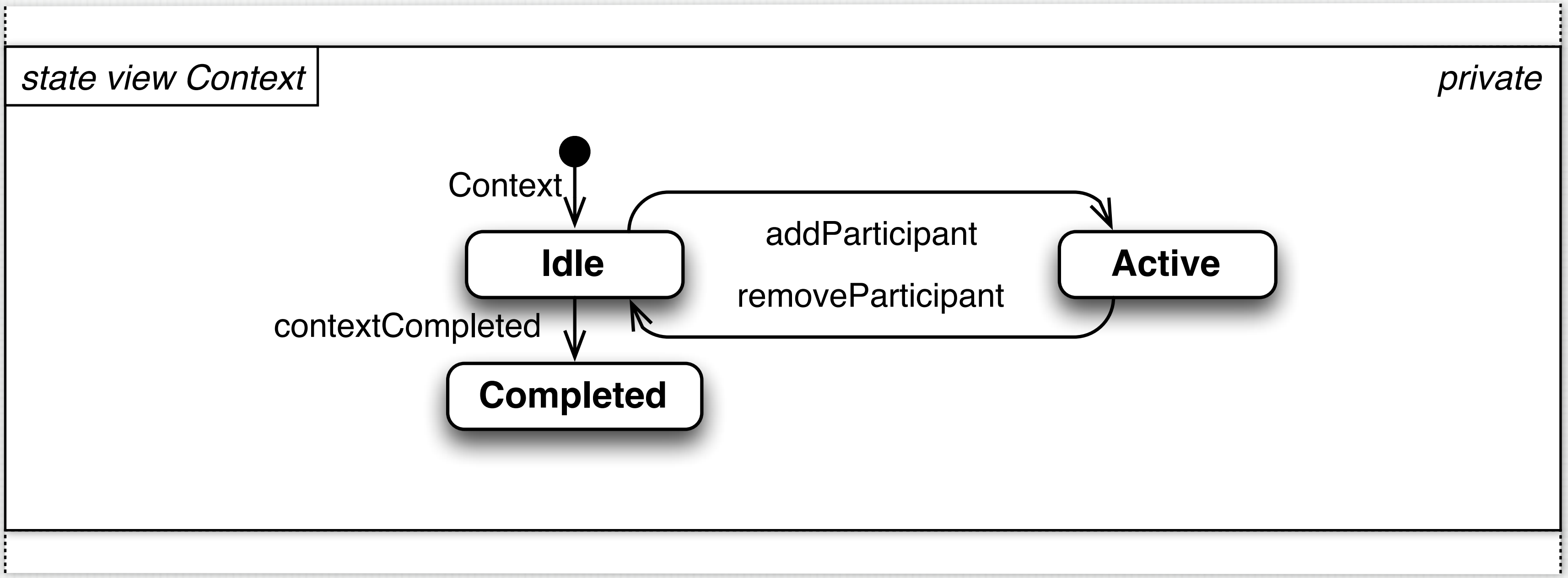
- A process / thread of computation can *create and enter* a *context*
- Once it is inside, the thread is a *context participant*
- A context participant can *leave* the context at any time



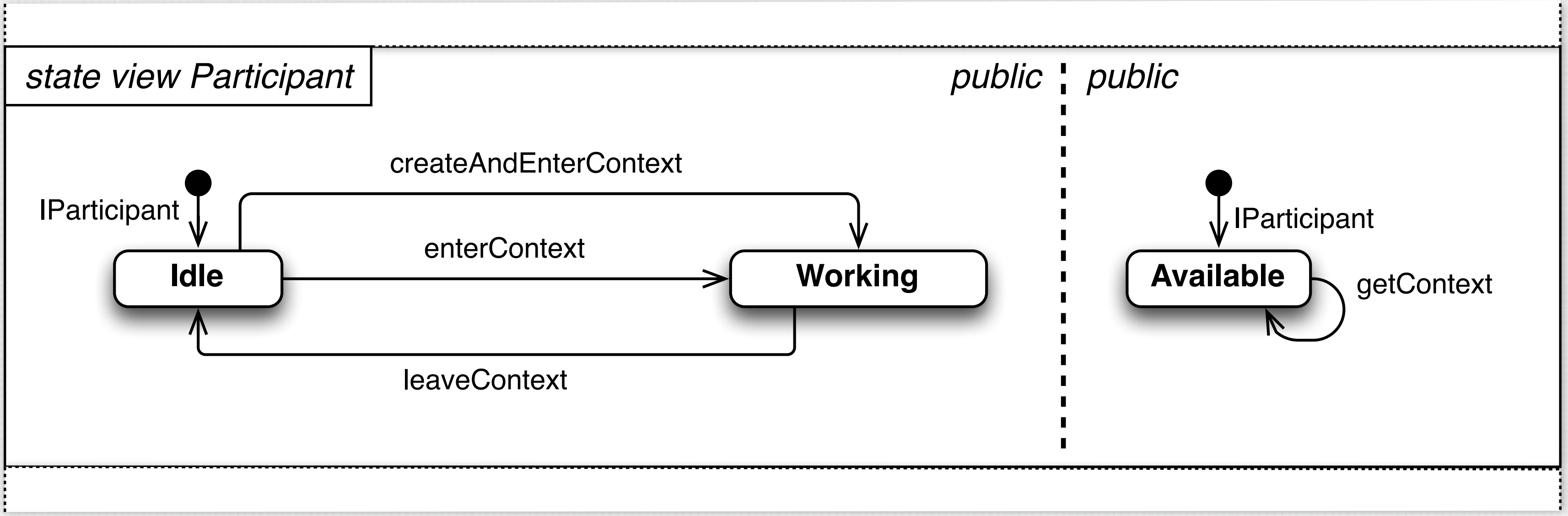
Context Aspect Structural View



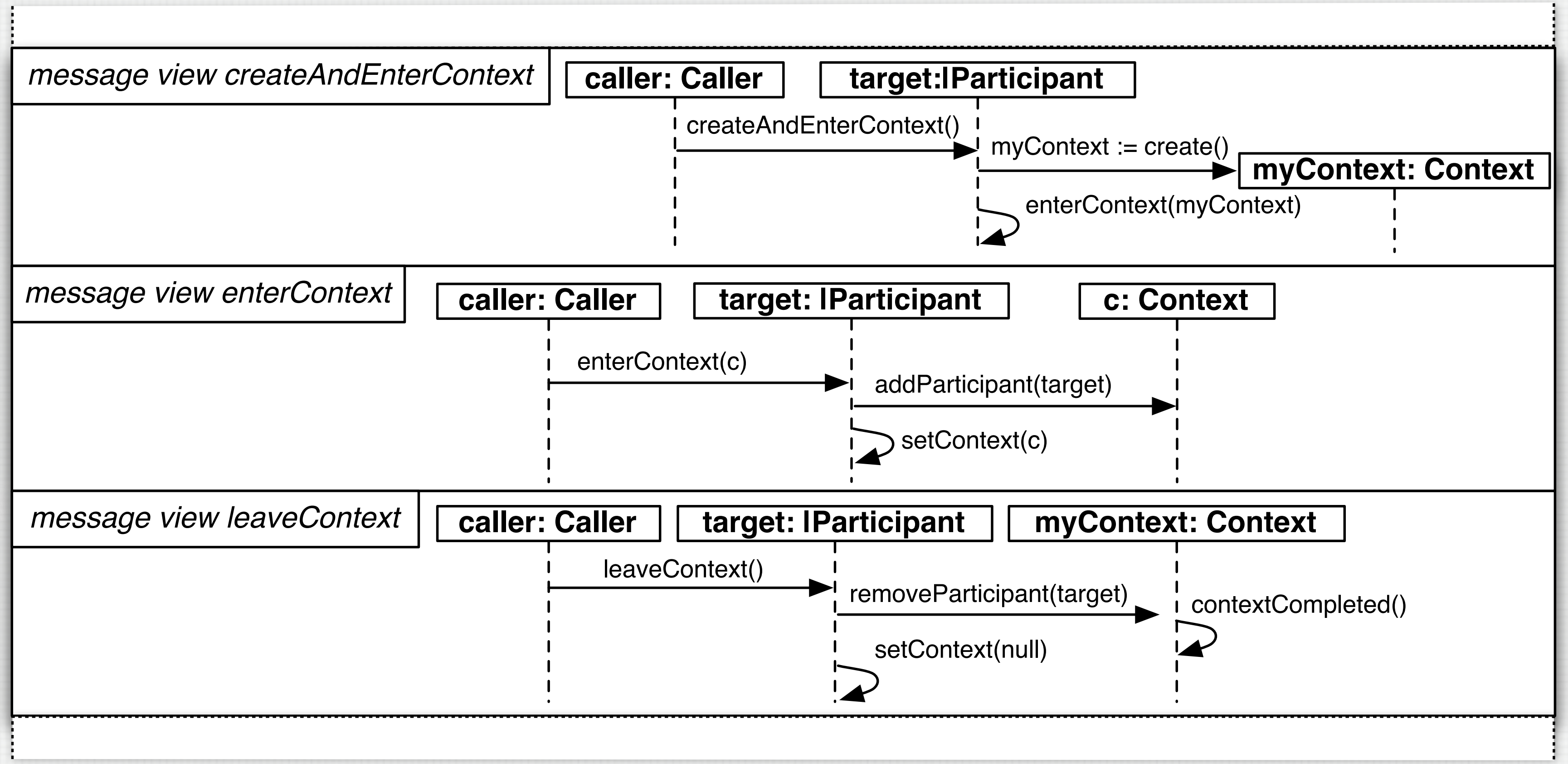
ExecutionContext Aspect State View (1)



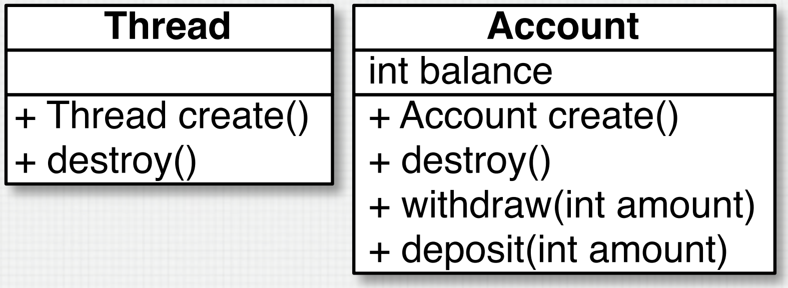
ExecutionContext Aspect State View (2)



Context Aspect Message View



Base Model: A Banking Application



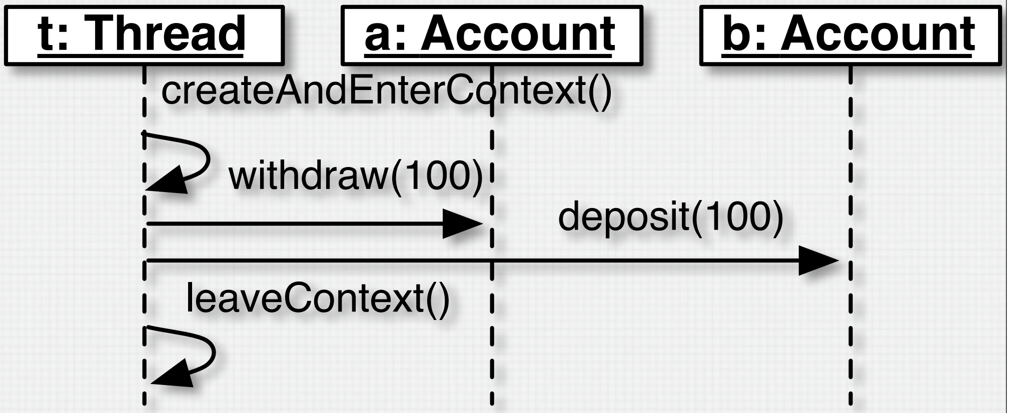
Thread

+ Thread create()
+ destroy()

Account

int balance

+ Account create()
+ destroy()
+ withdraw(int amount)
+ deposit(int amount)



t: Thread

a: Account

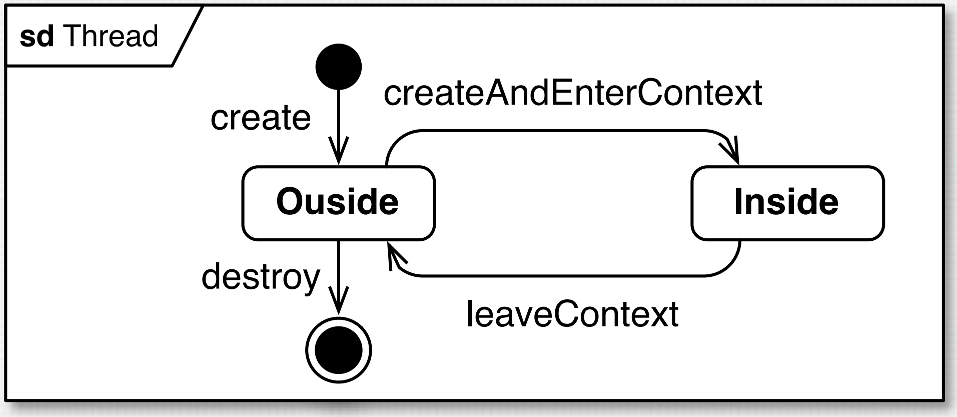
b: Account

createAndEnterContext()

withdraw(100)

deposit(100)

leaveContext()



sd Thread

create

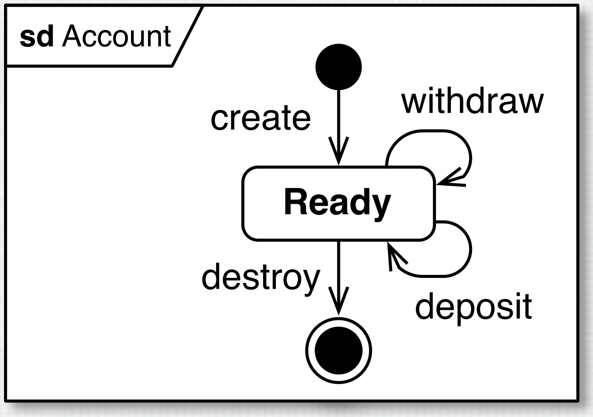
createAndEnterContext

Ouside

Inside

destroy

leaveContext



sd Account

create

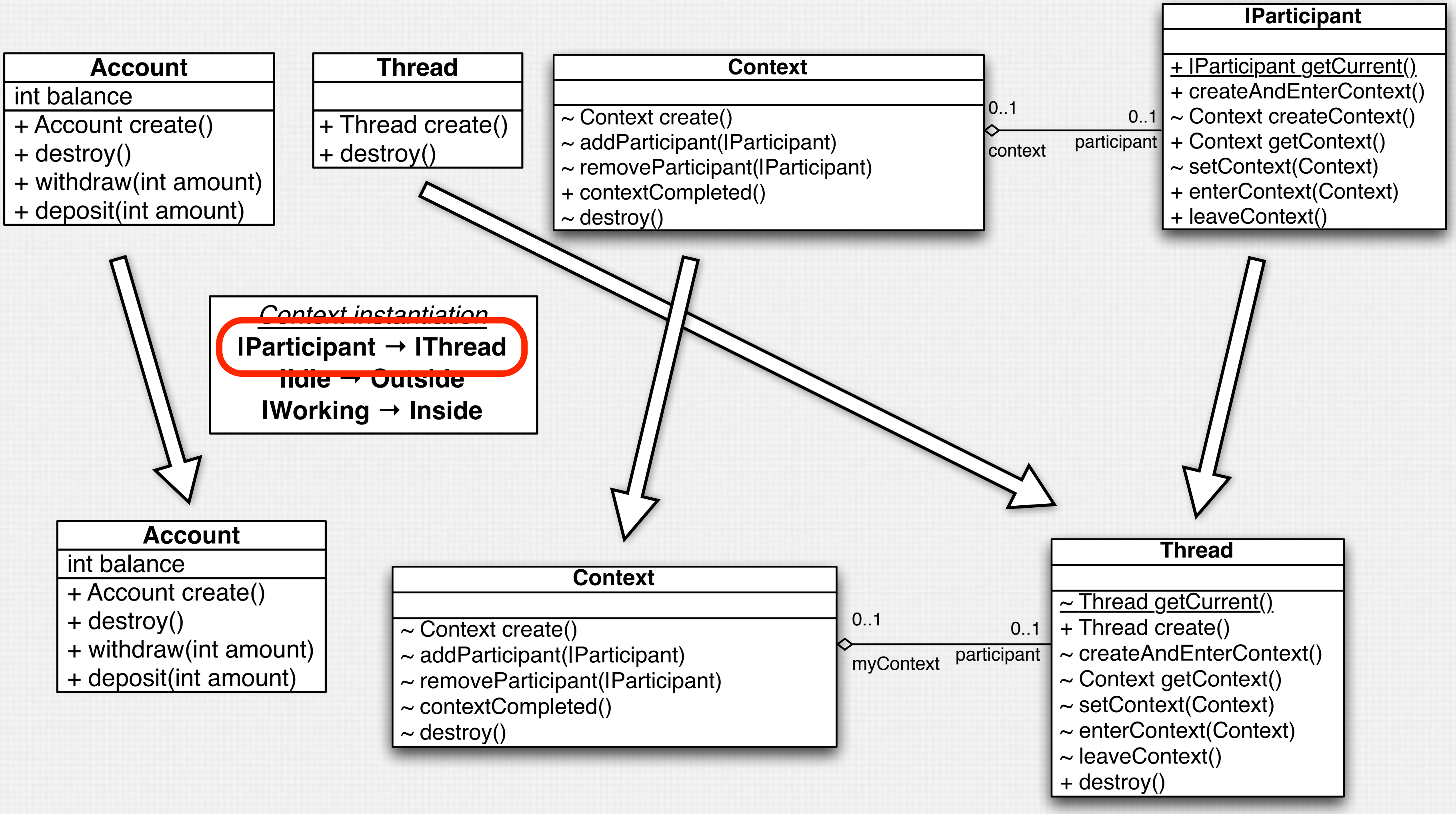
withdraw

Ready

destroy

deposit

Structural View Weaving



Account
int balance
+ Account create()
+ destroy()
+ withdraw(int amount)
+ deposit(int amount)

Thread
+ Thread create()
+ destroy()

Context
~ Context create()
~ addParticipant(IParticipant)
~ removeParticipant(IParticipant)
+ contextCompleted()
~ destroy()

IParticipant
+ IParticipant getCurrent()
+ createAndEnterContext()
~ Context createContext()
+ Context getContext()
~ setContext(Context)
+ enterContext(Context)
+ leaveContext()

Context instantiation
IParticipant → IThread
IIdle → Outside
IWorking → Inside

Account
int balance
+ Account create()
+ destroy()
+ withdraw(int amount)
+ deposit(int amount)

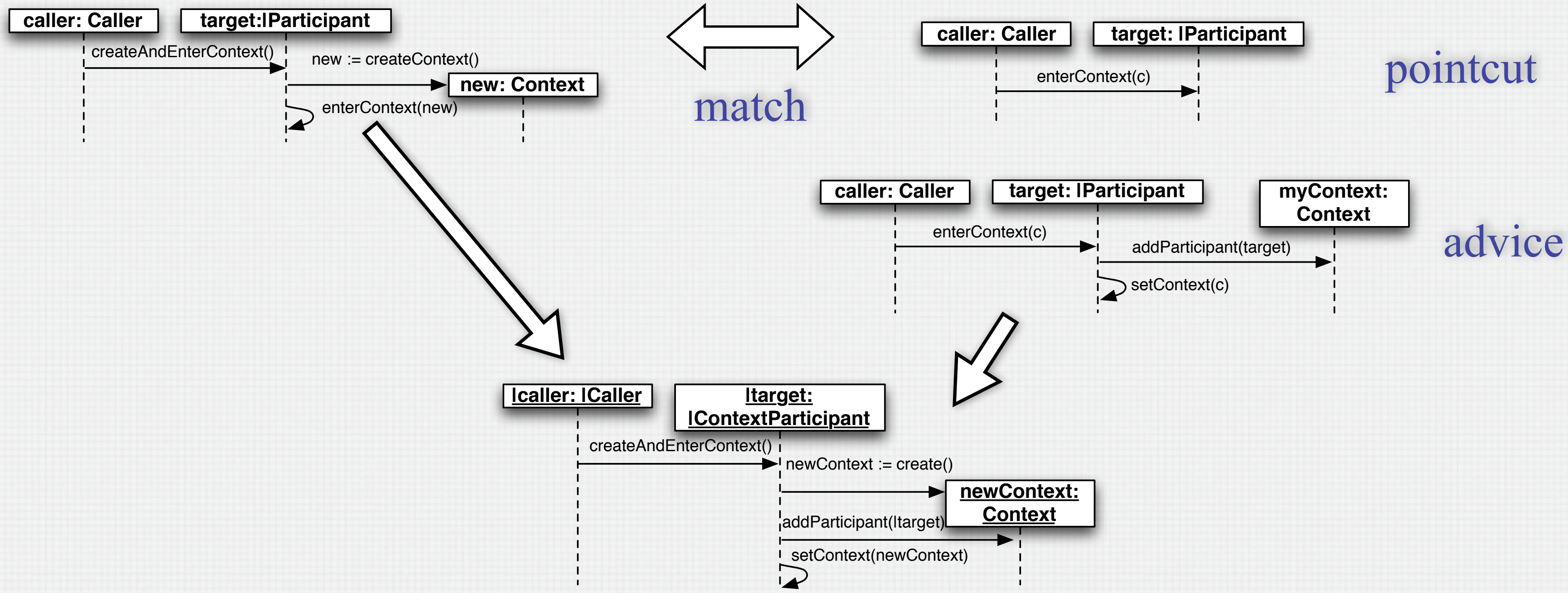
Context
~ Context create()
~ addParticipant(IParticipant)
~ removeParticipant(IParticipant)
~ contextCompleted()
~ destroy()

Thread
~ Thread getCurrent()
+ Thread create()
~ createAndEnterContext()
~ Context getContext()
~ setContext(Context)
~ enterContext(Context)
~ leaveContext()
+ destroy()

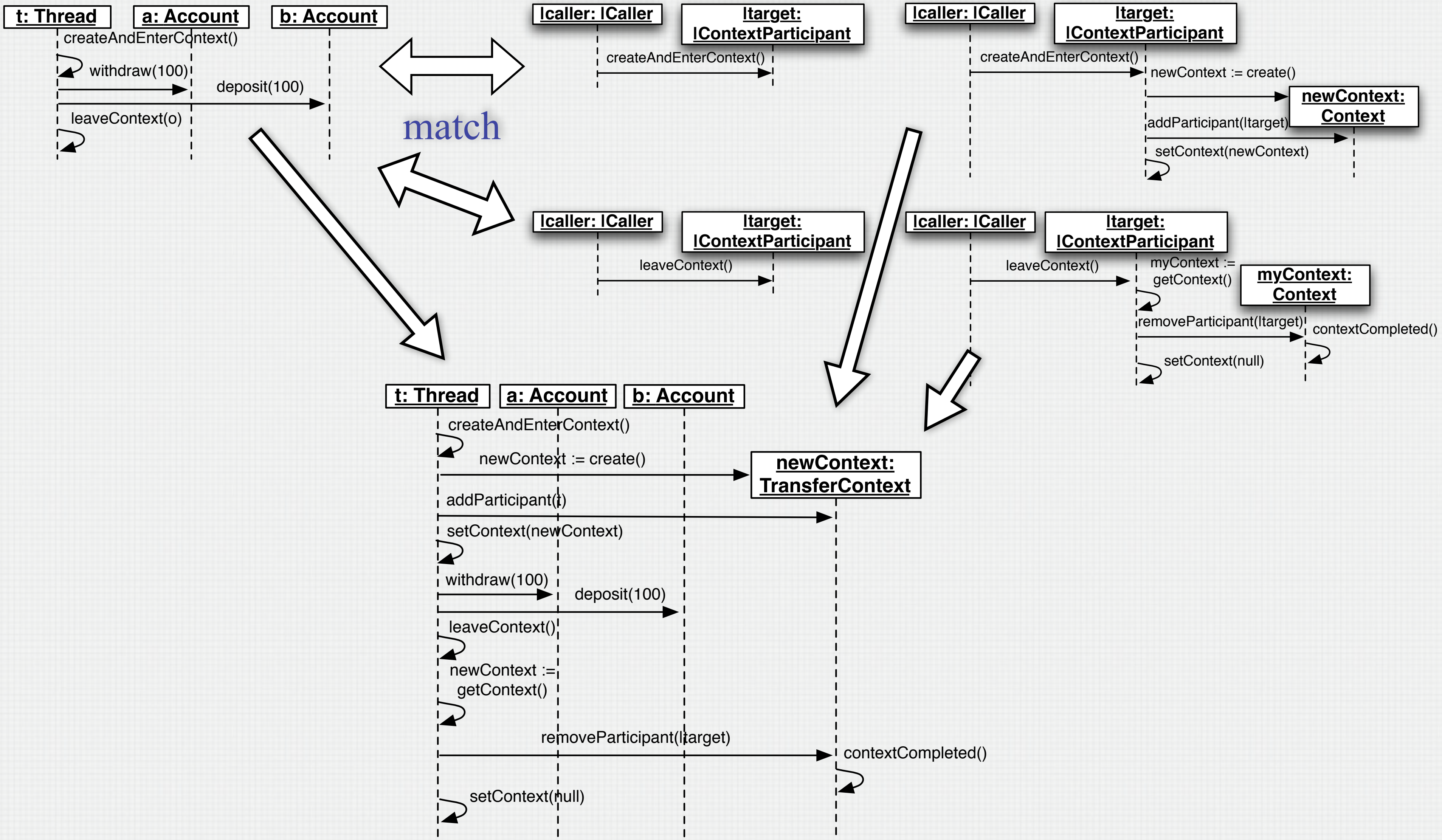
Message View Weaving (1)

- The base model sequence diagram is affected by createAndEnterContext and leaveContext
  - createAndEnterContext in turn is affected by enterContext

1. Weave enterContext into advice of createAndEnterContext
2. Weave createAndEnterContext into base sequence diagram



Message View Weaving (2)



t: Thread
a: Account
b: Account
createAndEnterContext()
withdraw(100)
deposit(100)
leaveContext(o)

match

Icaller: ICaller
Itarget: IContextParticipant
createAndEnterContext()

Icaller: ICaller
Itarget: IContextParticipant
createAndEnterContext()
newContext := create()
newContext: Context
addParticipant(Itarget)
setContext(newContext)

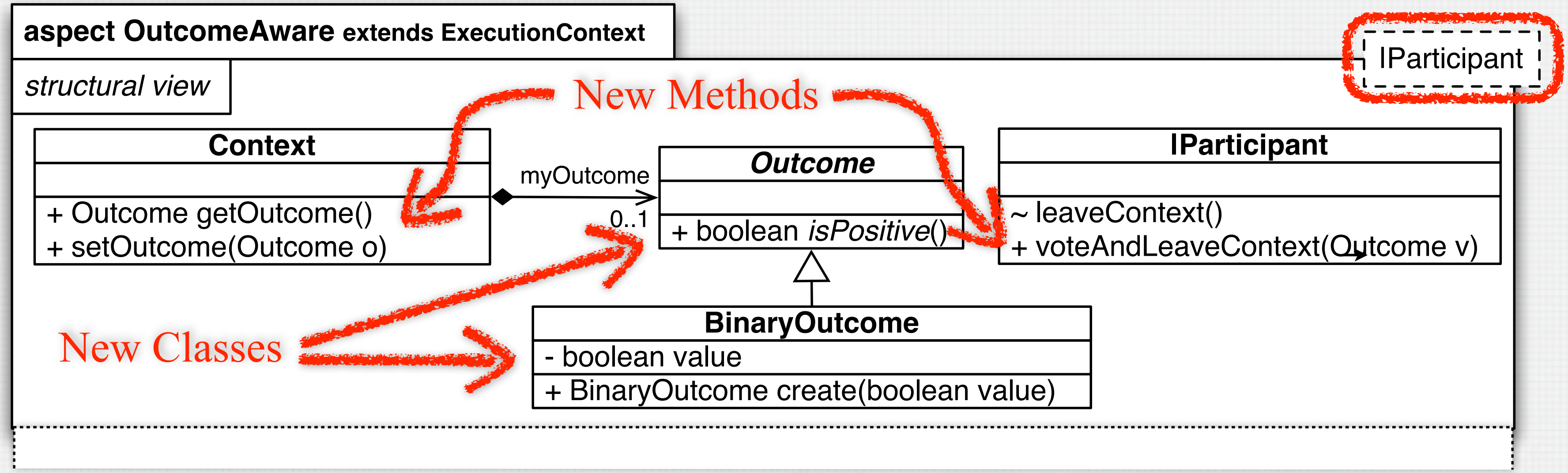
Icaller: ICaller
Itarget: IContextParticipant
leaveContext()

Icaller: ICaller
Itarget: IContextParticipant
leaveContext()
myContext := getContext()
myContext: Context
removeParticipant(Itarget)
contextCompleted()
setContext(null)

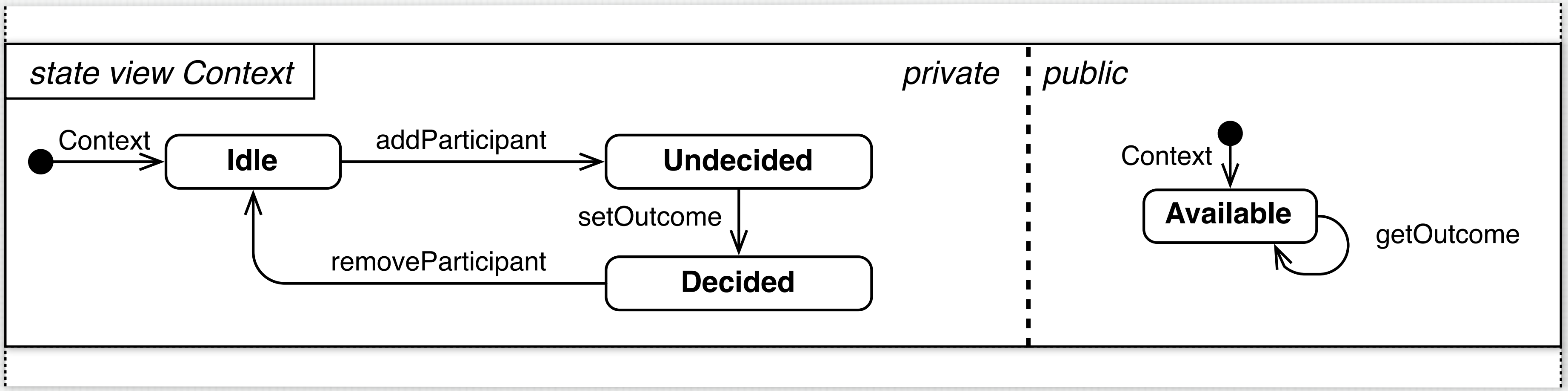
t: Thread
a: Account
b: Account
createAndEnterContext()
newContext := create()
newContext: TransferContext
addParticipant(t)
setContext(newContext)
withdraw(100)
deposit(100)
leaveContext()
newContext := getContext()
removeParticipant(Itarget)
contextCompleted()
setContext(null)

OutcomeAware Structural View

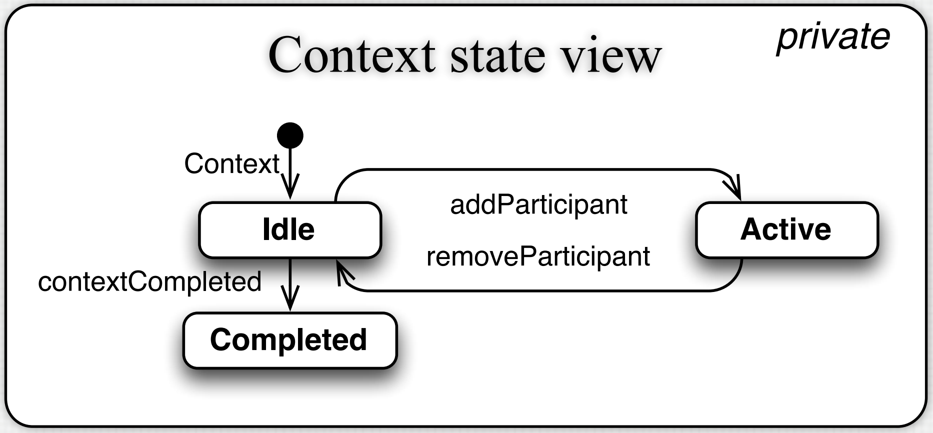
- OutcomeAware adds the notion of successful or unsuccessful completion to a context



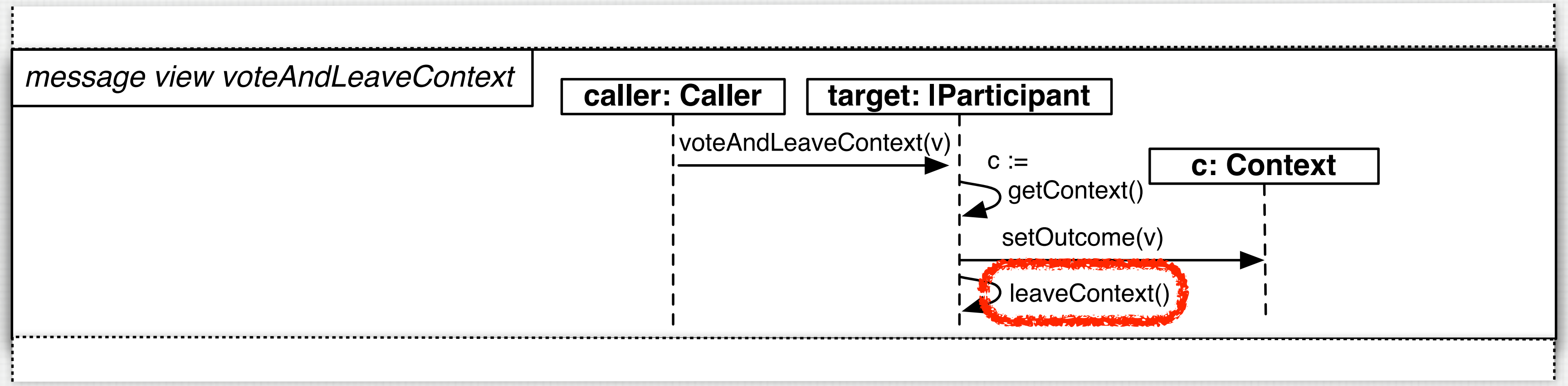
OutcomeAware State Views



- CSP || Composition
  - A transition t is accepted iff all machines that have t in their alphabet accept it
  - Machines that do not have t in their alphabet are ignored



OutcomeAware Message View



message view voteAndLeaveContext

caller: Caller

target: IParticipant

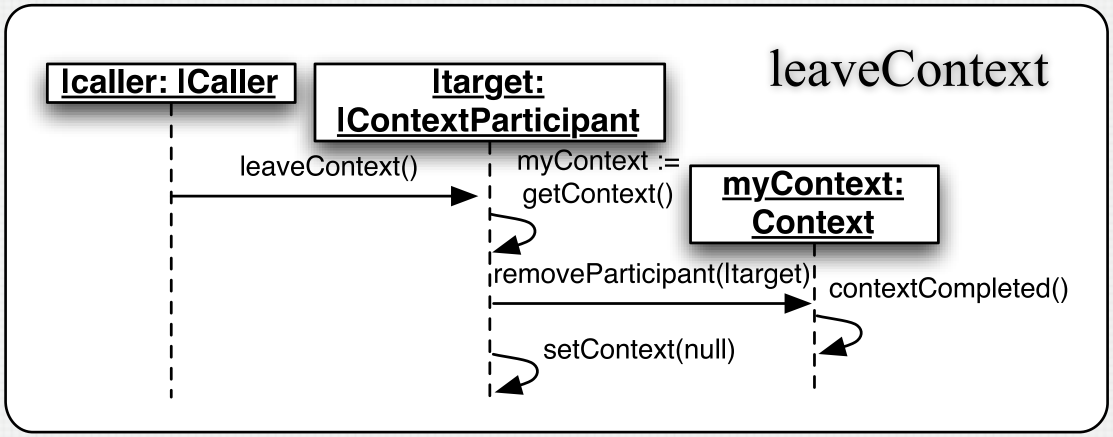
voteAndLeaveContext(v)

c := getContext()

c: Context

setOutcome(v)

leaveContext()



leaveContext

Icaller: ICaller

Itarget: IContextParticipant

leaveContext()

myContext := getContext()

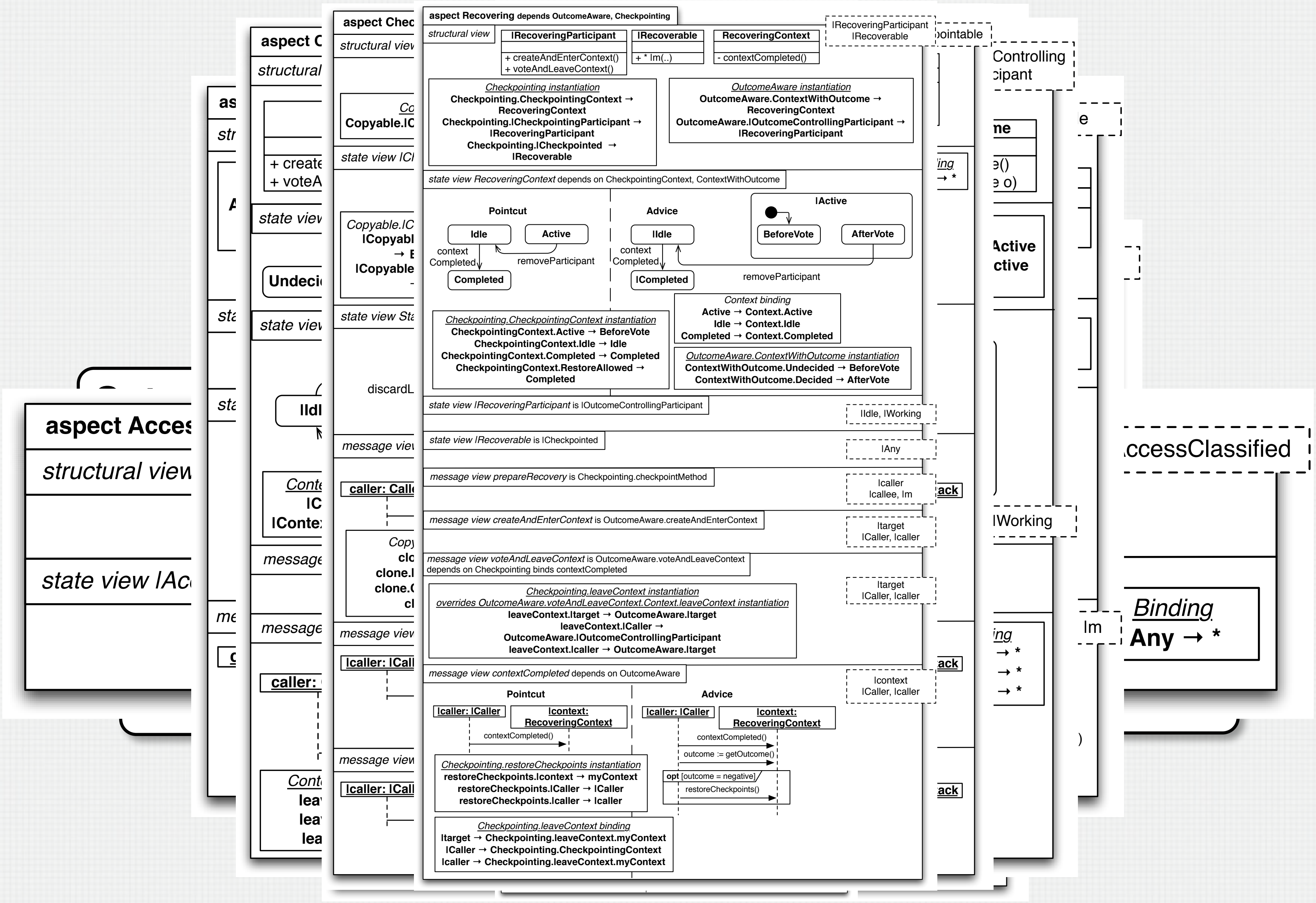
myContext: Context

removeParticipant(Itarget)

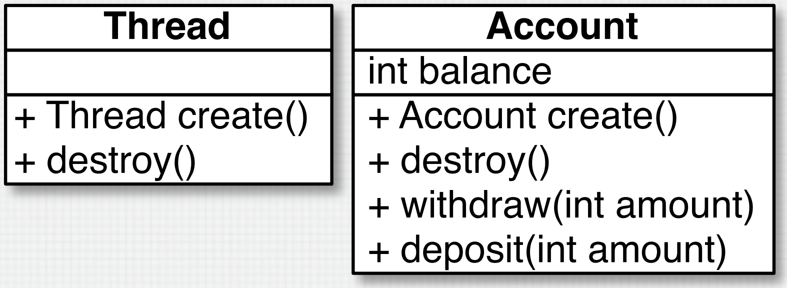
contextCompleted()

setContext(null)

Complex Example: Recovering


aspect Recovering depends OutcomeAware, Checkpointing
structural view
IRecoveringParticipant
+ createAndEnterContext()
+ voteAndLeaveContext()
IRecoverable
+ * Im(..)
RecoveringContext
- contextCompleted()
IRecoveringParticipant
IRecoverable
Checkpointing instantiation
Checkpointing.CheckpointingContext → RecoveringContext
Checkpointing.ICheckpointingParticipant → IRecoveringParticipant
Checkpointing.ICheckpointed → IRecoverable
OutcomeAware instantiation
OutcomeAware.ContextWithOutcome → RecoveringContext
OutcomeAware.IOutcomeControllingParticipant → IRecoveringParticipant
state view RecoveringContext depends on CheckpointingContext, ContextWithOutcome
Pointcut
Idle
Active
context Completed
removeParticipant
Completed
Advice
IIdle
context Completed
ICompleted
IActive
BeforeVote
AfterVote
removeParticipant
Context binding
Active → Context.Active
Idle → Context.Idle
Completed → Context.Completed
Checkpointing.CheckpointingContext instantiation
CheckpointingContext.Active → BeforeVote
CheckpointingContext.Idle → Idle
CheckpointingContext.Completed → Completed
CheckpointingContext.RestoreAllowed → Completed
OutcomeAware.ContextWithOutcome instantiation
ContextWithOutcome.Undecided → BeforeVote
ContextWithOutcome.Decided → AfterVote
state view IRecoveringParticipant is IOutcomeControllingParticipant
IIdle, IWorking
state view IRecoverable is ICheckpointed
IAny
message view prepareRecovery is Checkpointing.checkpointMethod
Icaller
Icallee, Im
message view createAndEnterContext is OutcomeAware.createAndEnterContext
Itarget
ICaller, Icaller
message view voteAndLeaveContext is OutcomeAware.voteAndLeaveContext depends on Checkpointing binds contextCompleted
Itarget
ICaller, Icaller
Checkpointing.leaveContext instantiation
overrides OutcomeAware.voteAndLeaveContext.Context.leaveContext instantiation
leaveContext.Itarget → OutcomeAware.Itarget
leaveContext.ICaller → OutcomeAware.IOutcomeControllingParticipant
leaveContext.Icaller → OutcomeAware.Itarget
message view contextCompleted depends on OutcomeAware
Icontext
ICaller, Icaller
Pointcut
Icaller: ICaller
Icontext: RecoveringContext
contextCompleted()
Advice
Icaller: ICaller
Icontext: RecoveringContext
contextCompleted()
outcome := getOutcome()
opt [outcome = negative]
restoreCheckpoints()
Checkpointing.restoreCheckpoints instantiation
restoreCheckpoints.Icontext → myContext
restoreCheckpoints.ICaller → ICaller
restoreCheckpoints.Icaller → Icaller
Checkpointing.leaveContext binding
Itarget → Checkpointing.leaveContext.myContext
ICaller → Checkpointing.CheckpointingContext
Icaller → Checkpointing.leaveContext.myContext

Base Model: Structure View



Thread

+ Thread create()
+ destroy()

Account

int balance

+ Account create()
+ destroy()
+ withdraw(int amount)
+ deposit(int amount)

+

Recovering instantiation

Recovering.IRecoveringParticipant → Thread

Recovering.IRecoverable → Account

Recovering.IRecoveringParticipant instantiation

IRecoveringParticipant.IIdle → Outside

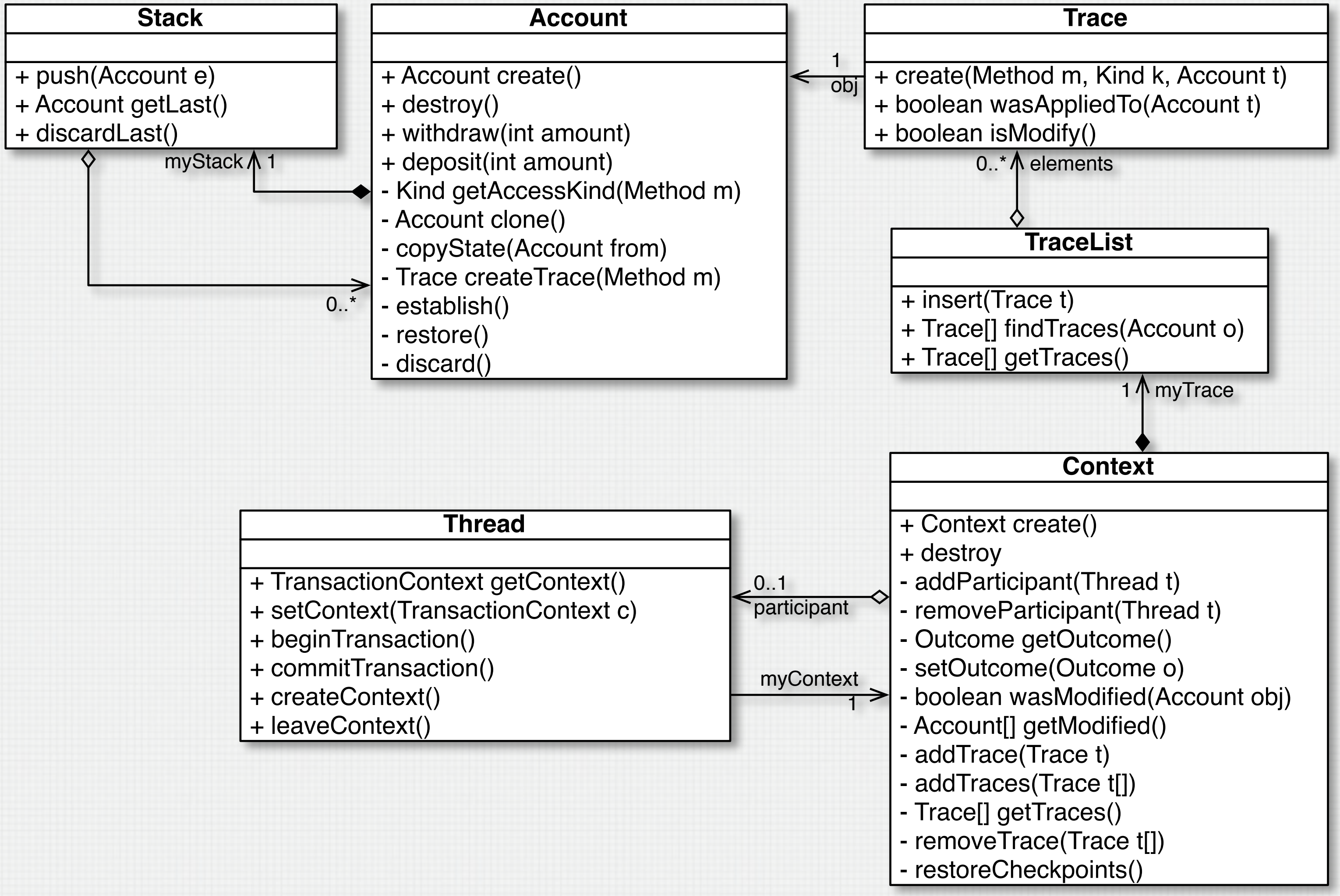
IRecoveringParticipant.IWorking → Inside

Recovering.IRecoverable instantiation

IRecoverable.BeforeM → Account.Ready

IRecoverable.AfterM → Account.Ready

Woven Model: Structure View



Stack

- + push(Account e)
- + Account getLast()
- + discardLast()

Account

- + Account create()
- + destroy()
- + withdraw(int amount)
- + deposit(int amount)
- - Kind getAccessKind(Method m)
- - Account clone()
- - copyState(Account from)
- - Trace createTrace(Method m)
- - establish()
- - restore()
- - discard()

Trace

- + create(Method m, Kind k, Account t)
- + boolean wasAppliedTo(Account t)
- + boolean isModify()

TraceList

- + insert(Trace t)
- + Trace[] findTraces(Account o)
- + Trace[] getTraces()

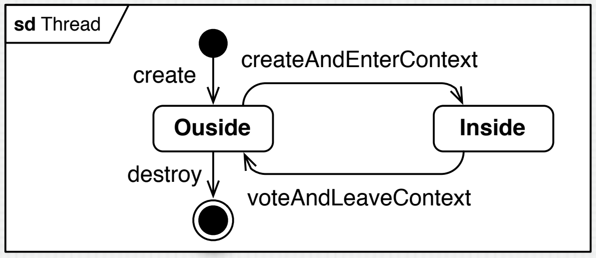
Context

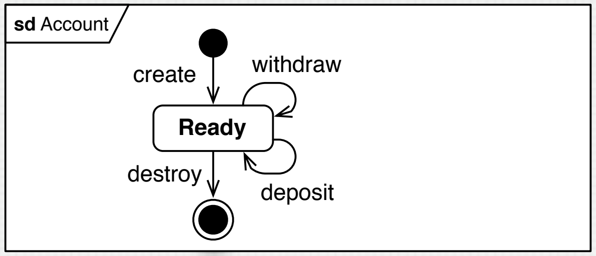
- + Context create()
- + destroy
- - addParticipant(Thread t)
- - removeParticipant(Thread t)
- - Outcome getOutcome()
- - setOutcome(Outcome o)
- - boolean wasModified(Account obj)
- - Account[] getModified()
- - addTrace(Trace t)
- - addTraces(Trace t[])
- - Trace[] getTraces()
- - removeTrace(Trace t[])
- - restoreCheckpoints()

Thread

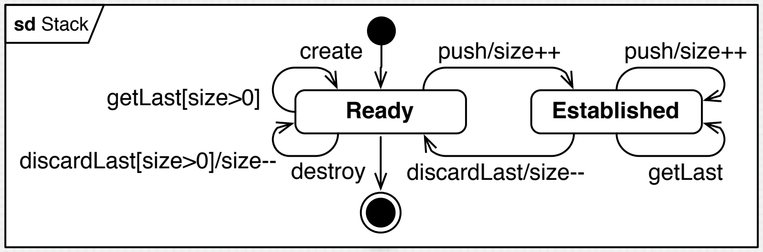
- + TransactionContext getContext()
- + setContext(TransactionContext c)
- + beginTransaction()
- + commitTransaction()
- + createContext()
- + leaveContext()

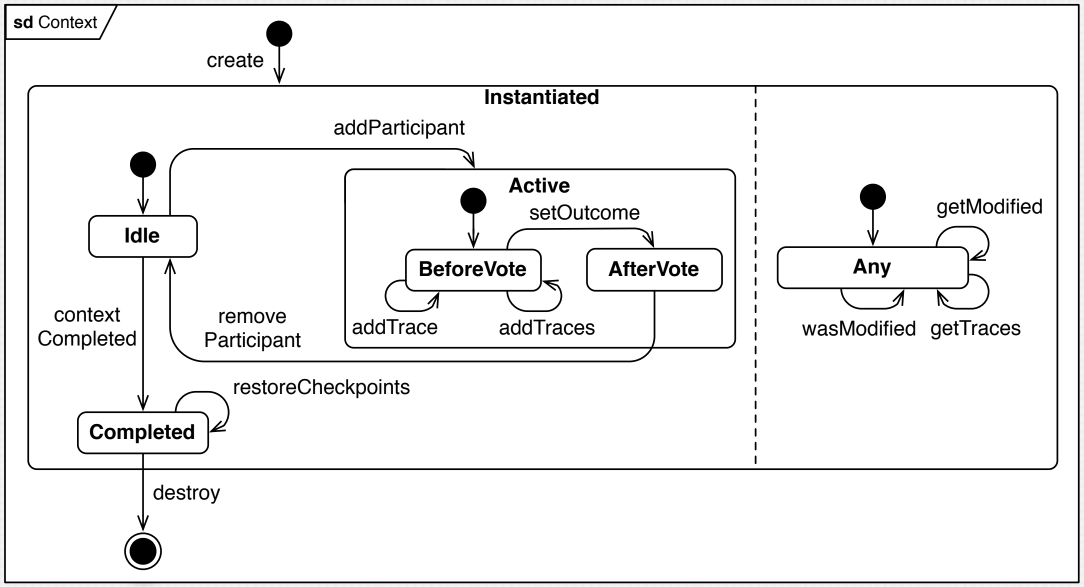
Base Model: State View

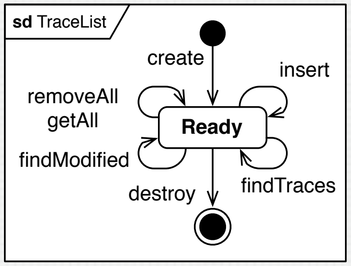


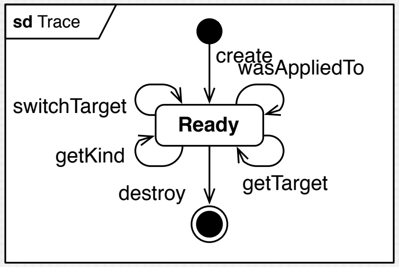


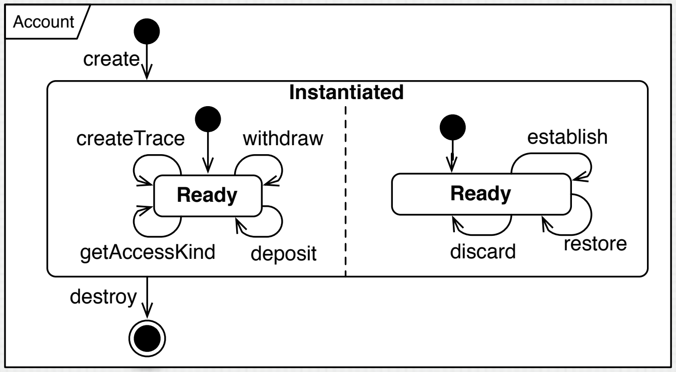
Woven Model: State View

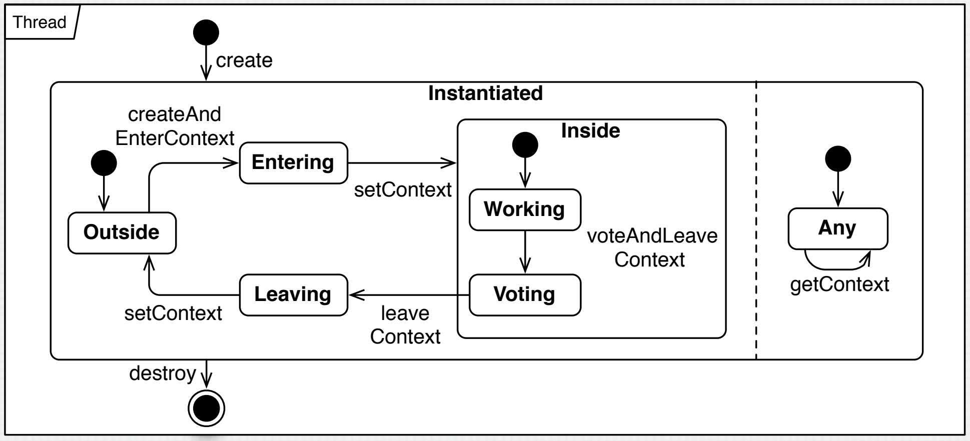




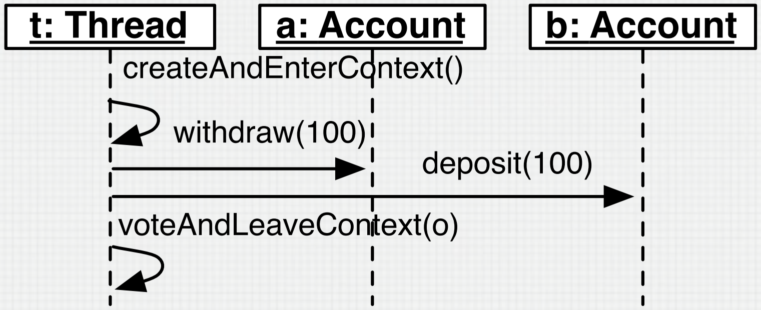








Base Model: Message View



+

Recovering instantiation

Recovering.IRecoveringParticipant → Thread

Recovering.IRecoverable → Account

Recovering.IRecoveringParticipant instantiation

IRecoveringParticipant.IIdle → Outside

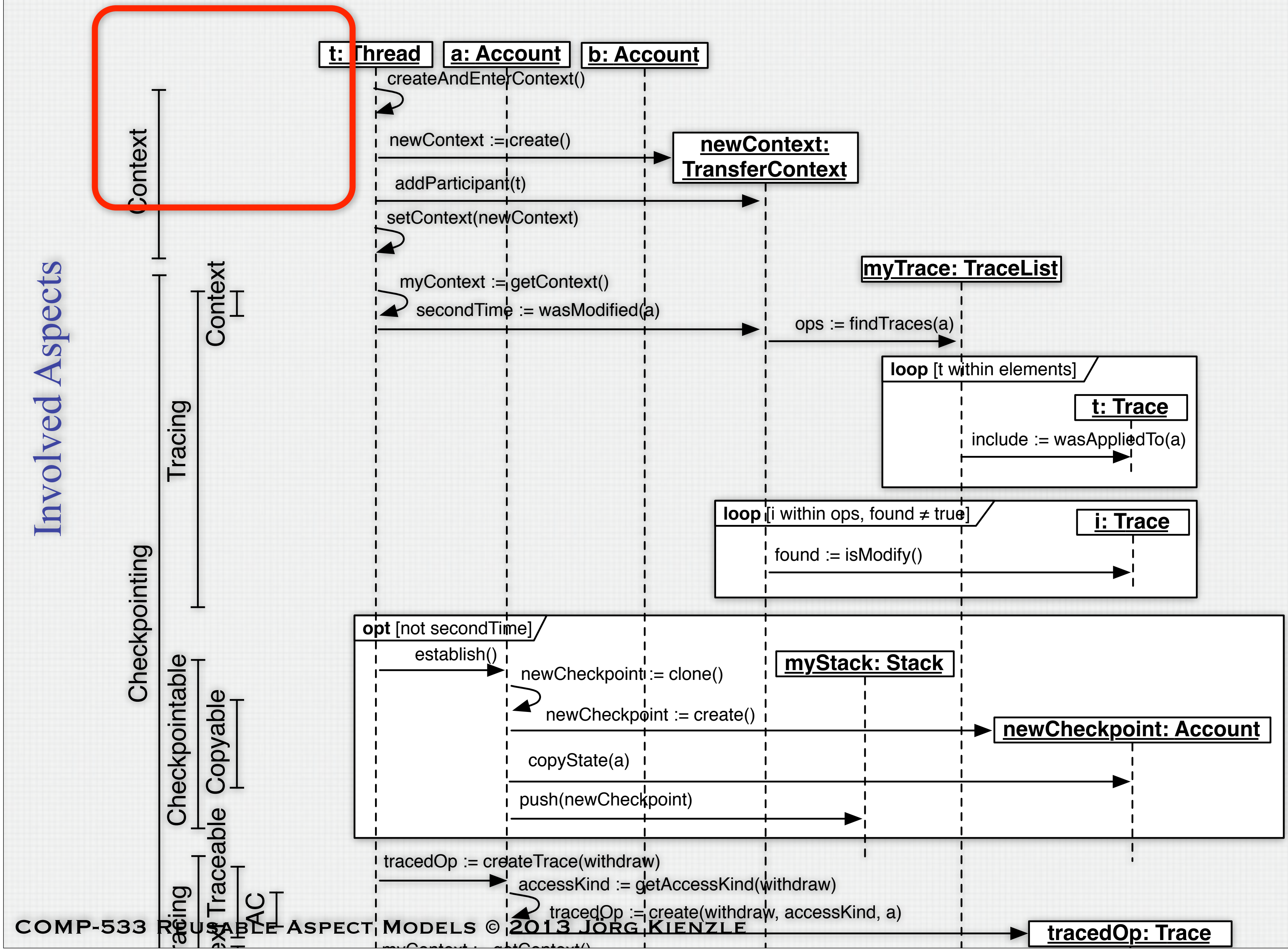
IRecoveringParticipant.IWorking → Inside

Recovering.IRecoverable instantiation

IRecoverable.BeforeM → Account.Ready

IRecoverable.AfterM → Account.Ready

Woven Model: Message View



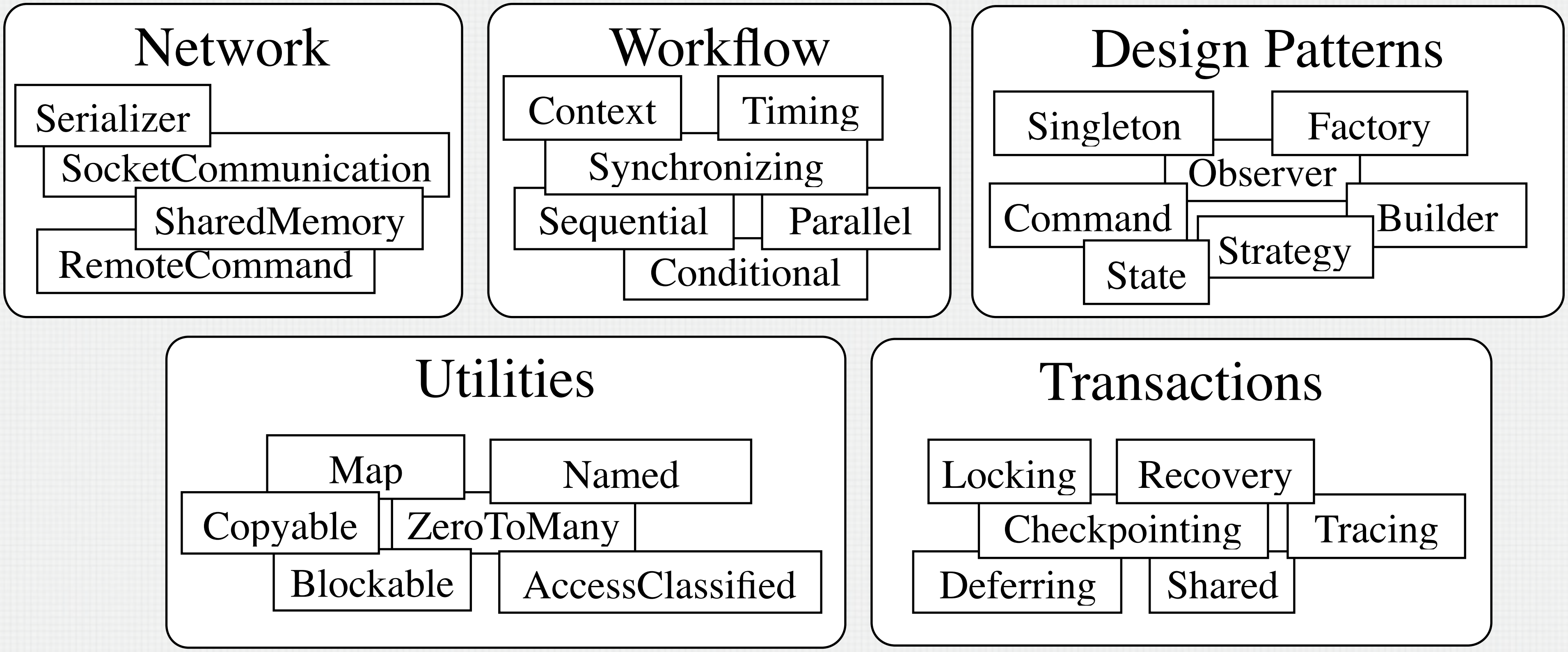
Summary

- RAM targeted at incremental software design
  - Final application model is formed by combining many (small) RAM Models
- A design concern model is a hierarchy of reusable aspect models
- Supports top-down and bottom-up modelling that crosses abstraction levels
  - A model higher in the hierarchy depends on a lower model to provide internal design structure and behaviour
    - The interface of the lower model is hidden
- Supports incremental modelling at the same level of abstraction
  - A model higher in the hierarchy extends the structure and behaviour of a lower model
    - The resulting aspect model interface is the union of both interfaces

- TouchRAM Tool
  - Fast navigation through aspect model hierarchies
  - Selective weaving of aspect models to collapse abstraction layers
  - Flattening of hierarchy to generate object-oriented final model

RAM Design Concern Library

- When designing your application, choose from a library of existing design concern models to solve common design problems



Network

Serializer

SocketCommunication

SharedMemory

RemoteCommand

Workflow

Context

Timing

Synchronizing

Sequential

Parallel

Conditional

Design Patterns

Singleton

Factory

Observer

Command

Builder

Strategy

State

Utilities

Map

Named

Copyable

ZeroToMany

Blockable

AccessClassified

Transactions

Locking

Recovery

Checkpointing

Tracing

Deferring

Shared

TouchRAM User Guide

	Tap	Double-Tap	Tap & Hold	Drag
Background	-	-	Create New Class	Pan / Scroll
Class	Select / deselect Class	-	-	Move Class
Class Name	-	Change Name	Toggle Partial	-
Field Name	-	Change Name	-	-
Field Type	-	Select Type	-	-
Operation Name	-	Change Name	Toggle Partial	-
Operation Type	-	Select Type	-	-
Operation Visibility	-	Select Visibility	-	-
Parameter Name	-	Change Name	-	-
Parameter Type	-	Select Type	-	-
Association End	-	Select Type + Delete Association	Toggle Navigability	-
Association Name	-	Change Name	-	-
Association Multiplicity	-	Change Multiplicity	-	-
Instantiated Aspect Name	-	Display Aspect	Weave Aspect	-
Instantiation Visibility	-	Toggle Visibility	-	-
Mapping	-	Select Model Element	Delete Mapping	-

If classes are selected	Tap	Double-Tap	Tap & Hold	Drag
Class	Select / deselect	-	Switch to “Edit” Mode	Move all selected classes
Other Class	Select / deselect	(If only a SINGLE class is selected) Create Association	(If only a SINGLE class is selected) Create Inheritance	Move all selected classes

Reusable Aspect Models

Download TouchRAM:

http://www.cs.mcgill.ca/~joerg/SEL/TouchRAM.html