Ch. 5.2: Part-B Advanced Behavioral Specification in UML

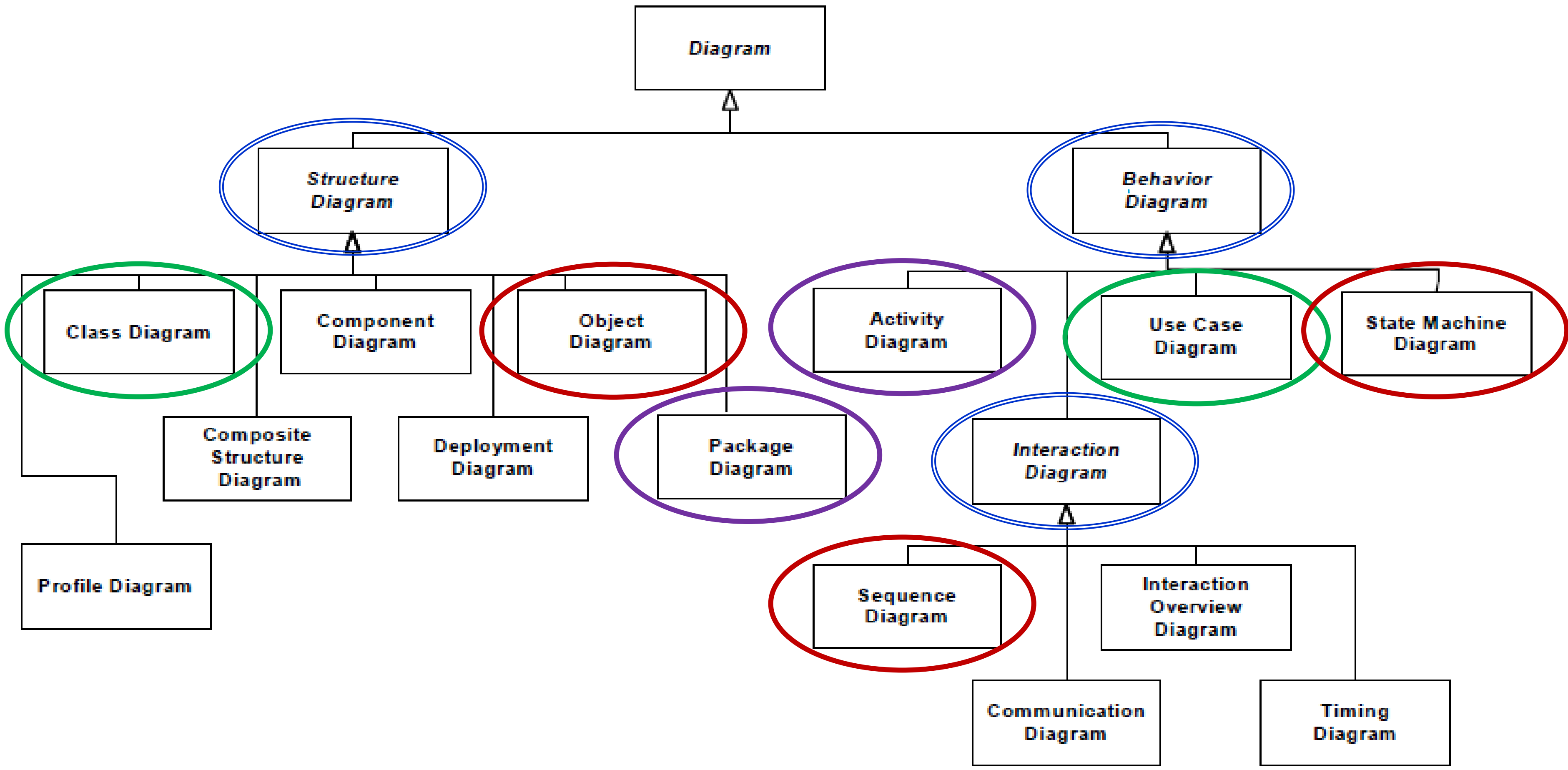
H.S. Sarjoughian

CSE 460: Software Analysis and Design

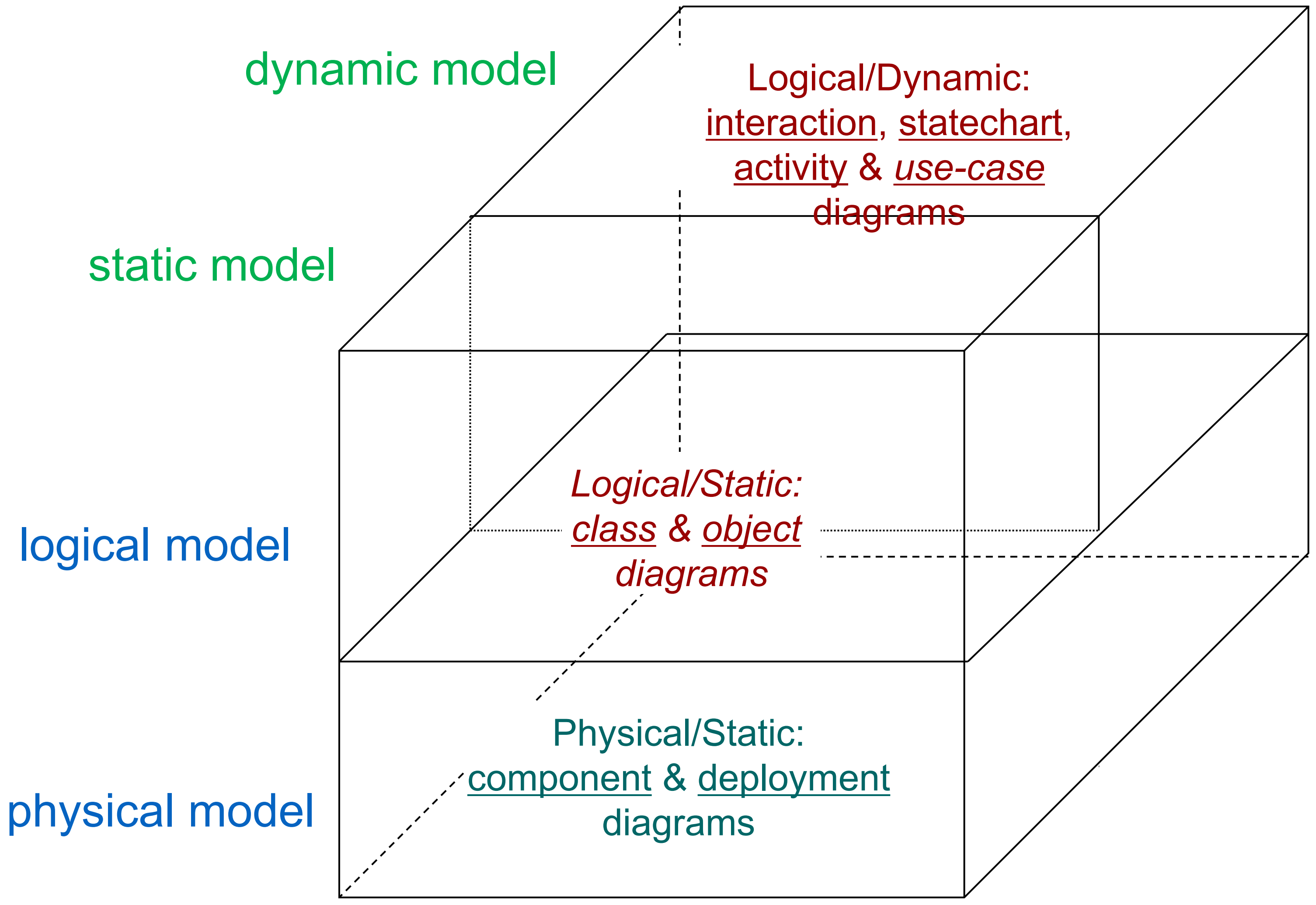
School of Computing, Informatics and Decision Systems Engineering
Fulton Schools of Engineering

Arizona State University, Tempe, AZ, USA

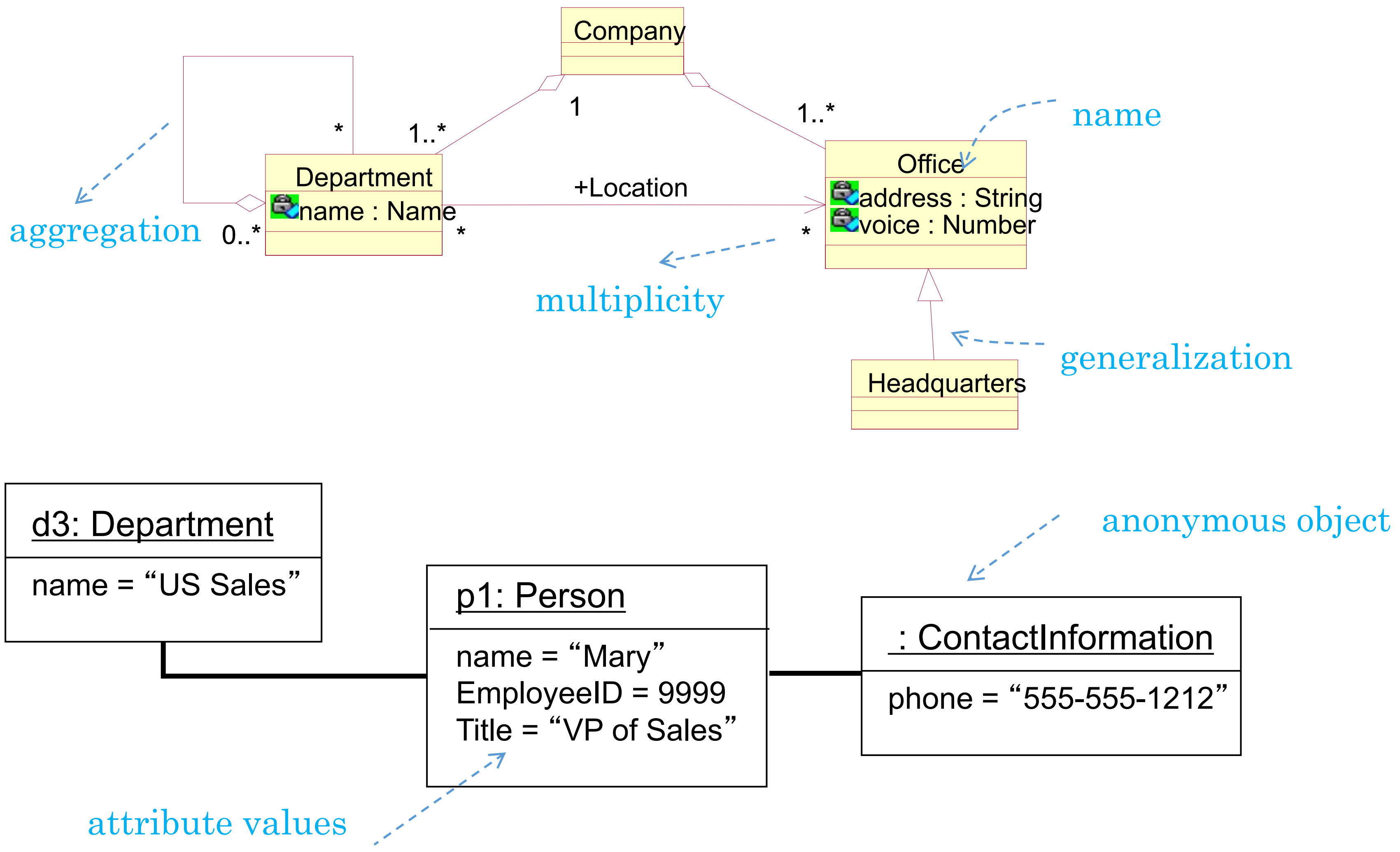
UML Languages



Models and Views of Object-Oriented Development


dynamic model
static model
logical model
physical model
Logical/Dynamic: interaction, statechart, activity & use-case diagrams
Logical/Static: class & object diagrams
Physical/Static: component & deployment diagrams

Class and Object Snippet Diagrams



Interactions

- **An interaction is a behavior that comprises a set of messages exchanged among a set of objects to accomplish a specific task for a given scenario**
- **Dynamic aspect of interactions is specified as flows of data and control** – *data and control can be simple or complex involving*
  - branching
  - looping
  - recursion
  - concurrency
- An interaction can be modeled in two ways:
  - time ordering of messages
  - sequencing of messages in the context of some structural organization
- Well-structured interactions are understandable, simple, efficient, and adaptable.

Links and Messages

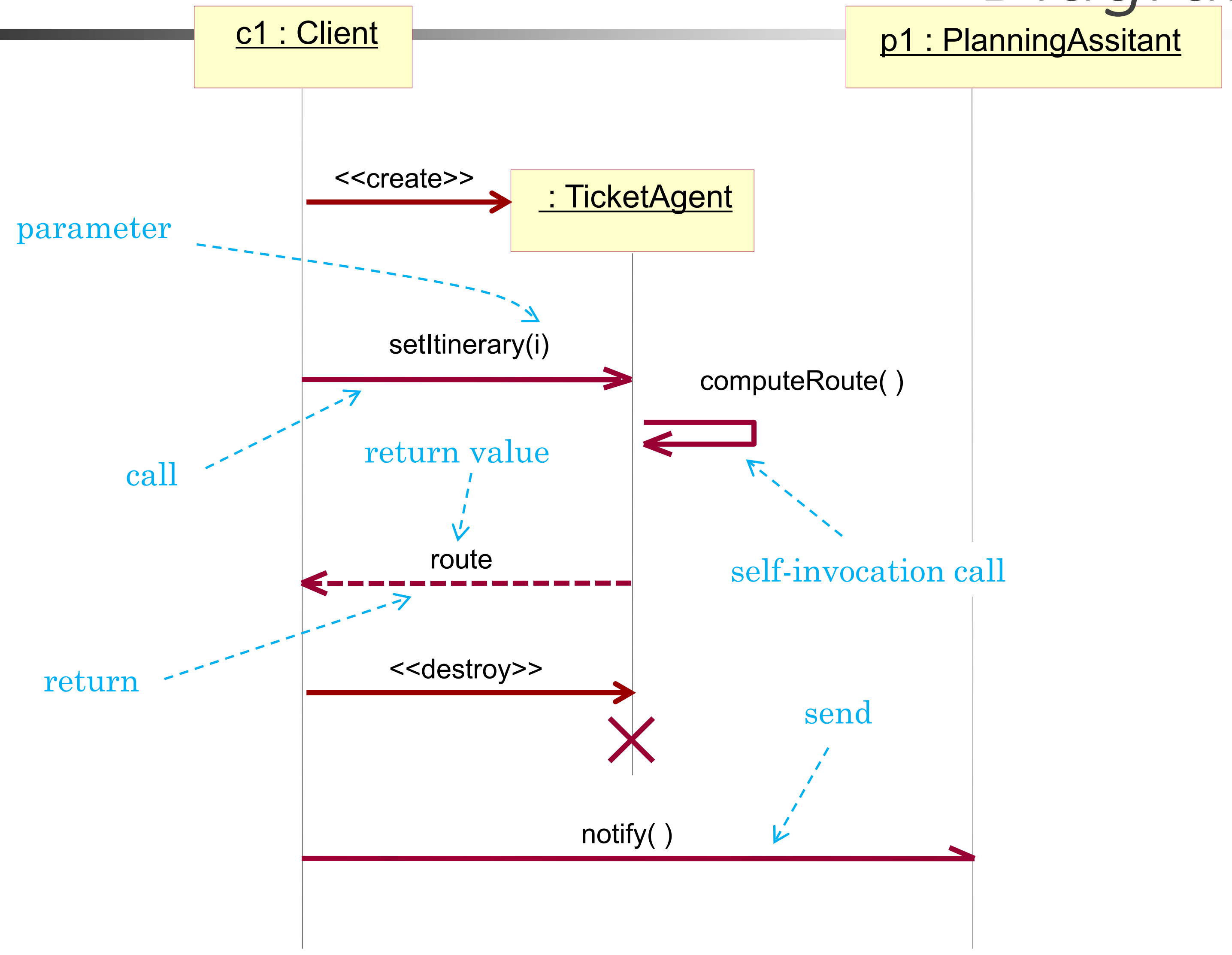
- **Link**
  - A link is an instance of an association – it is a semantic connection between two objects
  - Some standard stereotypes for a link are
    - Association: specifies the corresponding object is visible by association
    - Global: specifies that the corresponding object is visible since it is in an enclosing scope
    - Local: specifies that the corresponding object is visible since it is in a local scope
- **Message**
  - A message specifies communication among objects; there is the expectation that upon a message dispatch, some activity will occur – e.g., an operation is invoked and a change in state takes place
  - Objects interact with one another using messages – objects send messages to other objects resulting in invocation of operations

Messages (cont.)

- **Kinds of Messages**
  - **Call:** invokes an operation on an object (an object may send a message to itself – local- or self-invocation)
  - **Return:** returns some value to the caller
  - **Send:** sends a signal to an object
  - **Create:** creates an object
  - **Destroy:** destroys an object

- **Kinds of constraints for objects**
  - **New:** specifies object is created during execution of the enclosing interaction
  - **Destroyed:** specifies object is destroyed prior to the completion of the execution of the enclosing interaction
  - **Transient:** specifies object is created during execution of the enclosing interaction and destroyed before the completion of the interaction

A Ticketing System Snippet: Sequence Diagram



Interaction Diagrams

- Interaction diagrams are useful for modeling dynamic aspects of systems

- An interaction diagram contains a set of **objects**, **links**, and **messages** interchanged among objects. Interaction diagrams may contain notes and constraints

- There are two equivalent kinds of interaction diagrams
  - **Sequence diagram –** emphasizes the time ordering of messages
  - **Communication (Collaboration) diagram –** emphasizes the structural organization of the objects that send and receive messages

- Interaction diagrams model **concrete** instances of classes, components, and nodes, and messages dispatched among them

Sequence Diagram

- Emphasizes time ordering of messages
  - **Object lifeline** – shows existence of an object's lifetime over a period of time
    - depicted as dashed line
    - creation and destruction of an object is specified using keywords "create" and "destroy"
  - **Control** – shows how an object controls its subordinate object
    - depicted as rectangular overlaid over an object's lifeline
    - control can also be recursive – includes itself

- ***Multiple sequence diagrams*** might be necessary to show alternative paths or exception conditions

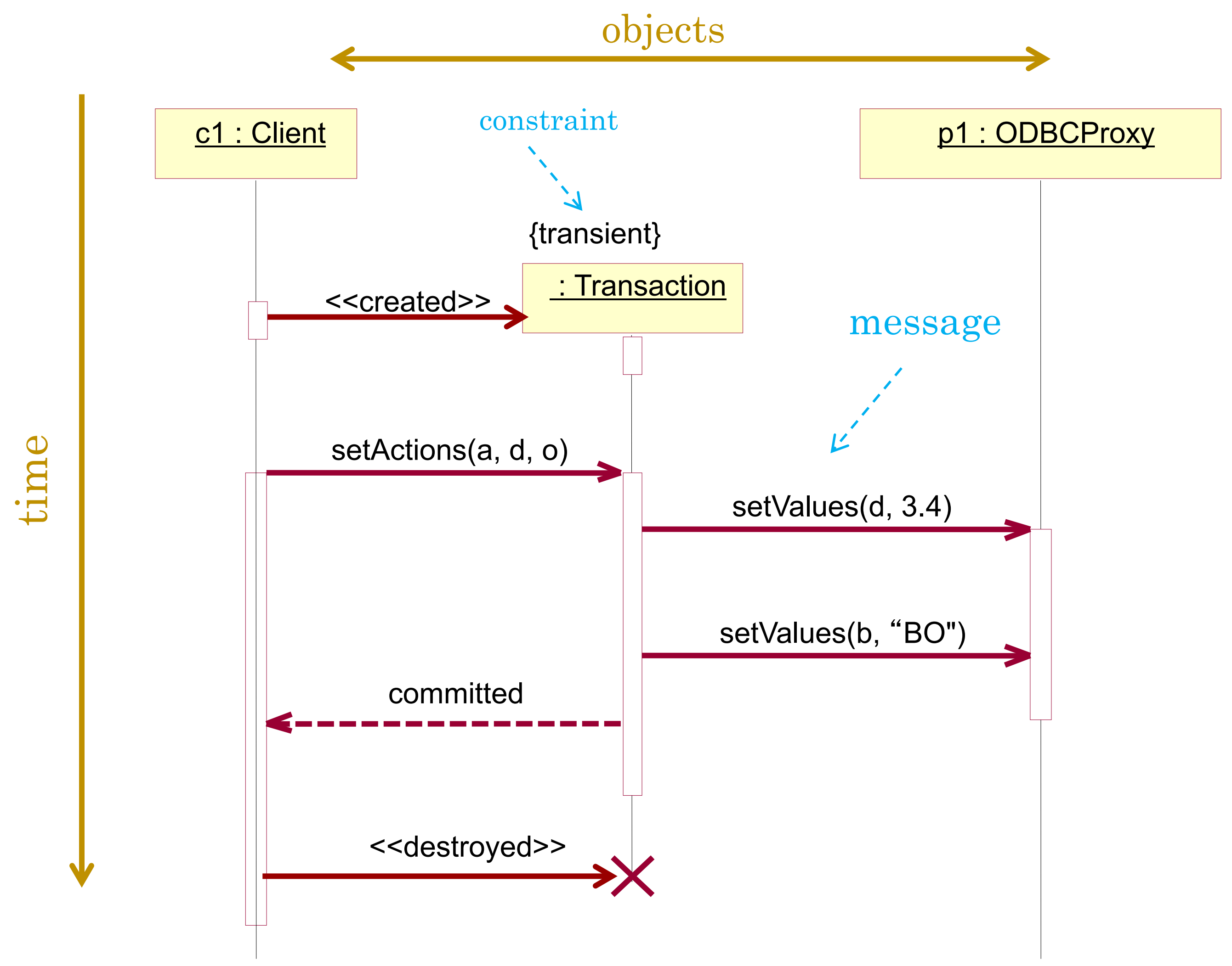
Modeling Sequence Diagrams

- Determine the context for the interaction – it can be for a system, subsystem, operation, or one scenario of a use-case or collaboration
- Determine the objects which have specific roles in the interaction. There may be transient objects (those that are created and destroyed during an interaction)
- Lay out objects from left to right
  - place the objects on top of the diagram space
  - place the more important objects to the left and their neighboring objects to their right
- Draw the lifeline for each object including transient ones
  - objects usually persist through the entire interaction period
  - explicitly indicate birth and death of transient objects

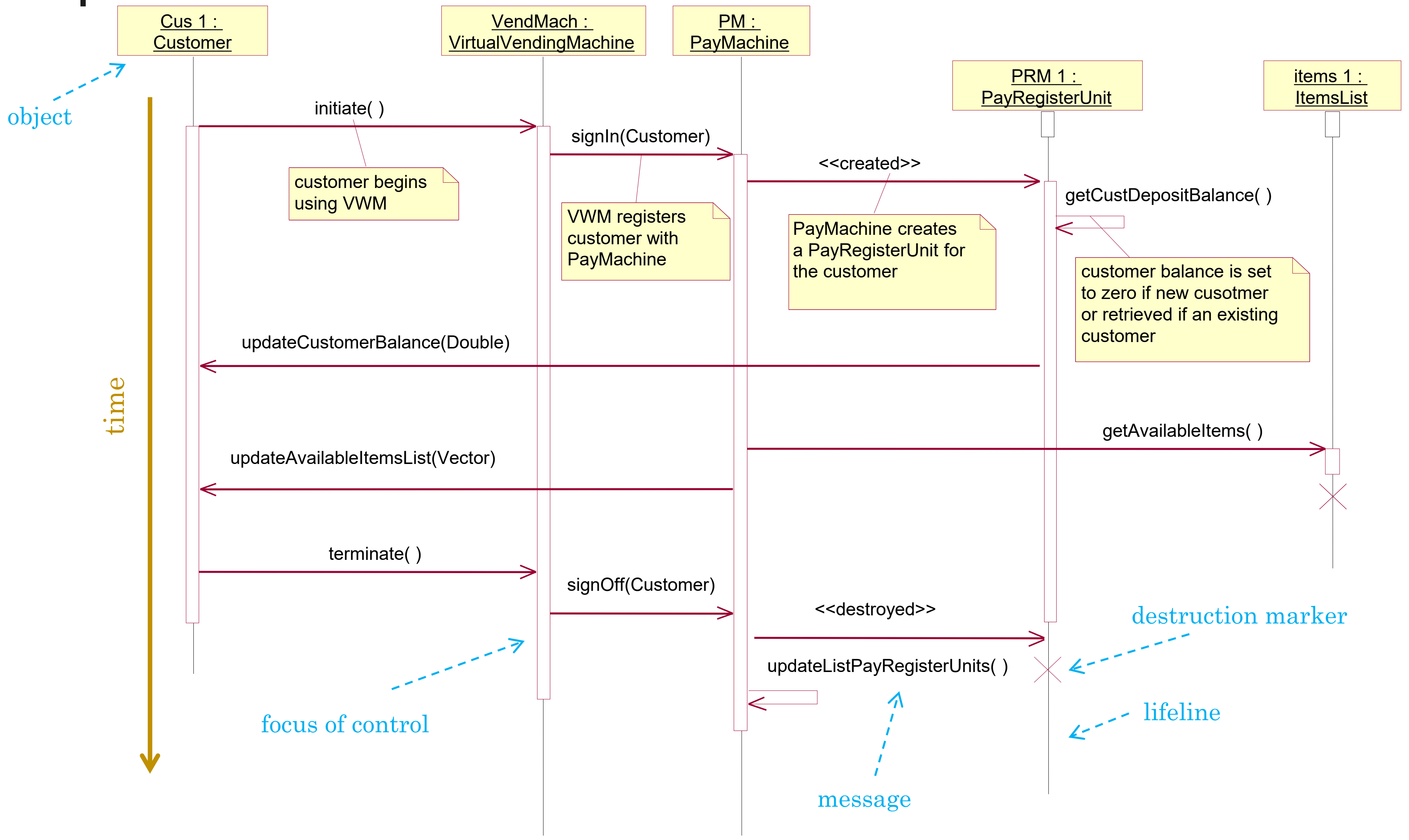
Modeling Sequence Diagrams (cont.)

- Starting with the message that initiates the interaction, lay out each subsequent message from top to bottom between the lifelines. Show properties of each message as necessary to illustrate the semantics of the interactions
- Specify each object's lifeline with focus of control – helps with visualizing nesting of messages or the points in time when actual computations take place
- Specify time and space constraints as necessary
- Include pre- and post-conditions to each message – provide higher degree of formal specification for flow of control

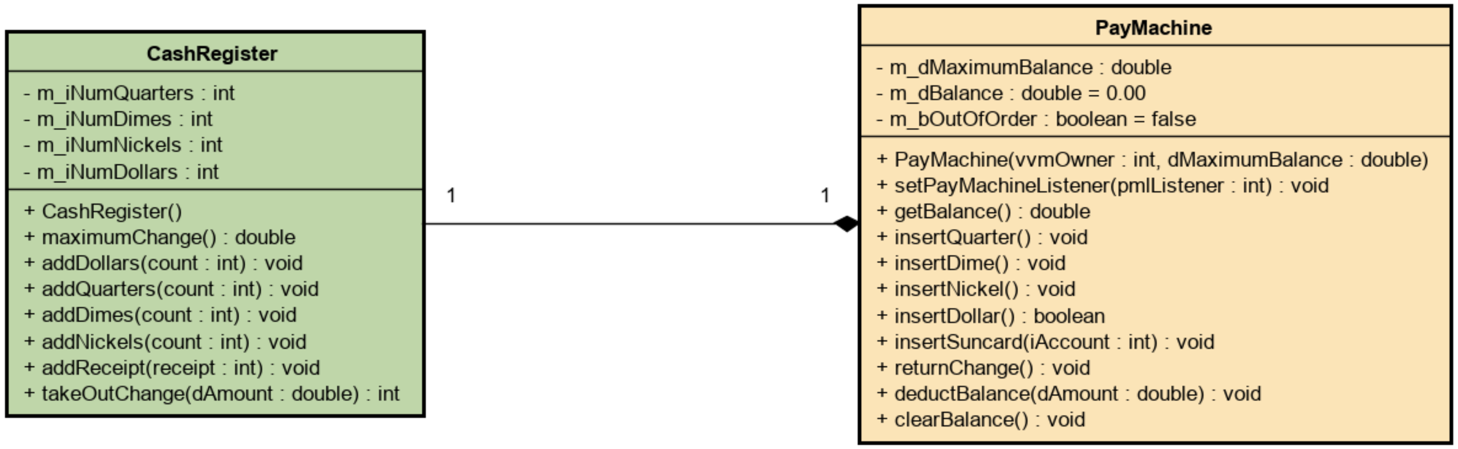
Sequence Diagram Examples



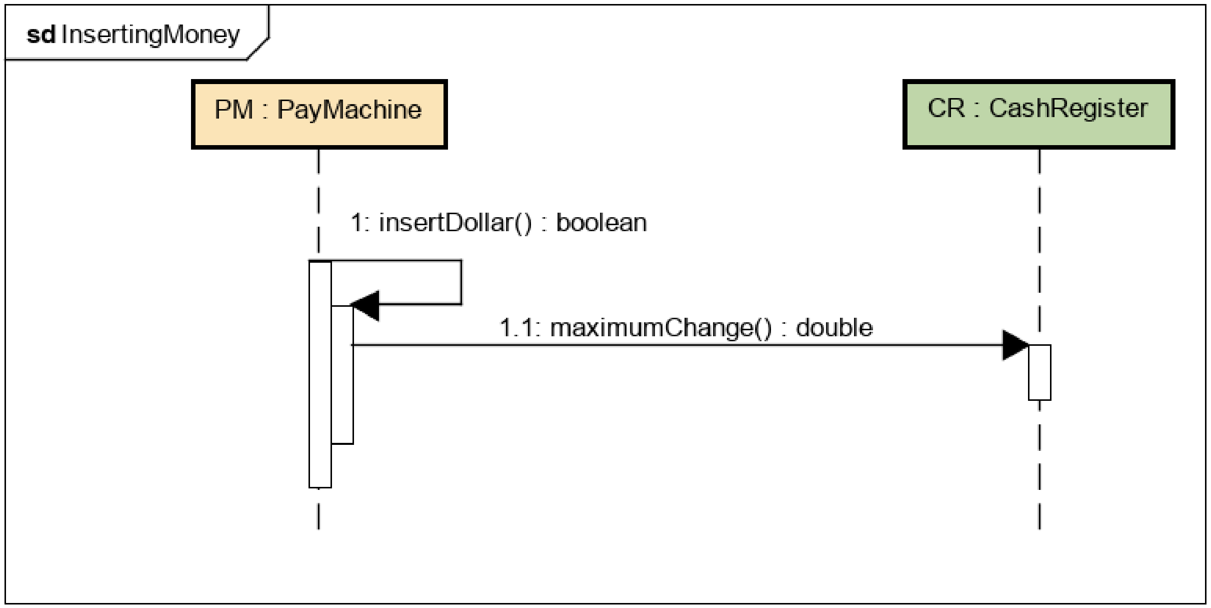
Sequence Diagram Examples (cont.)


Cus 1 : Customer
VendMach : VirtualVendingMachine
PM : PayMachine
PRM 1 : PayRegisterUnit
items 1 : ItemsList
object
time
initiate( )
customer begins using VWM
signIn(Customer)
VWM registers customer with PayMachine
<<created>>
PayMachine creates a PayRegisterUnit for the customer
getCustDepositBalance( )
customer balance is set to zero if new cusotmer or retrieved if an existing customer
updateCustomerBalance(Double)
getAvailableItems( )
updateAvailableItemsList(Vector)
terminate( )
signOff(Customer)
<<destroyed>>
destruction marker
updateListPayRegisterUnits( )
lifeline
focus of control
message

A Snippet of the Virtual Vending System Class Diagram with Implementation

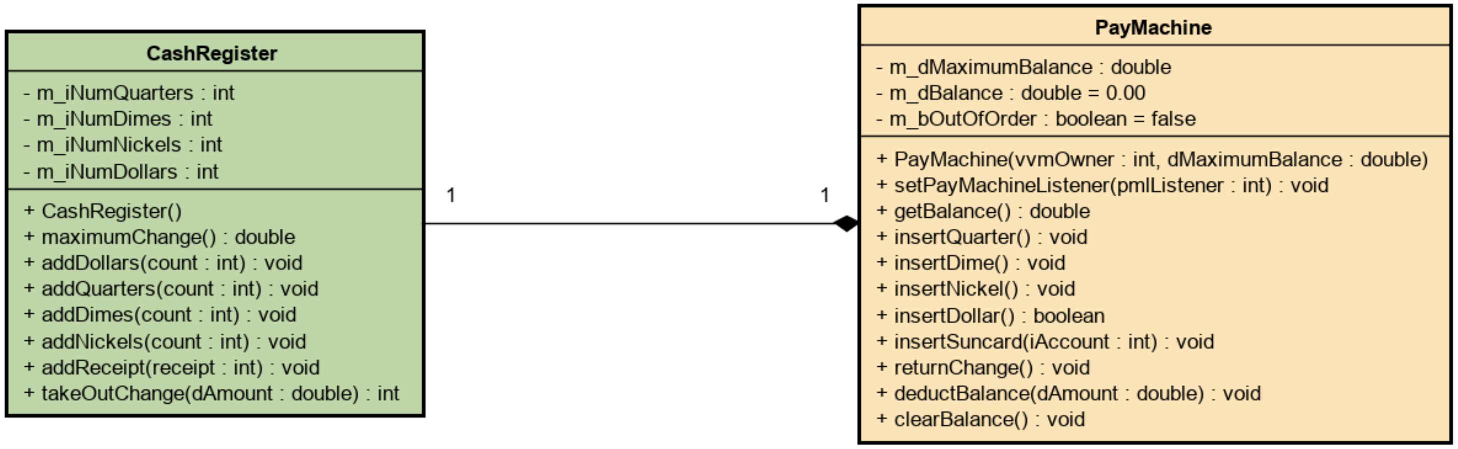

CashRegister
- m_iNumQuarters : int
- m_iNumDimes : int
- m_iNumNickels : int
- m_iNumDollars : int
+ CashRegister()
+ maximumChange() : double
+ addDollars(count : int) : void
+ addQuarters(count : int) : void
+ addDimes(count : int) : void
+ addNickels(count : int) : void
+ addReceipt(receipt : int) : void
+ takeOutChange(dAmount : double) : int
1
1
PayMachine
- m_dMaximumBalance : double
- m_dBalance : double = 0.00
- m_bOutOfOrder : boolean = false
+ PayMachine(vvmOwner : int, dMaximumBalance : double)
+ setPayMachineListener(pmlListener : int) : void
+ getBalance() : double
+ insertQuarter() : void
+ insertDime() : void
+ insertNickel() : void
+ insertDollar() : boolean
+ insertSuncard(iAccount : int) : void
+ returnChange() : void
+ deductBalance(dAmount : double) : void
+ clearBalance() : void

```
public double maximumChange(){
  double m_dN
  
  if (m_iNumN
    return 0;
  …
}
```



```
NewBalance){
```

A Snippet of the Virtual Vending System Class Diagram with Implementation


CashRegister
- m_iNumQuarters : int
- m_iNumDimes : int
- m_iNumNickels : int
- m_iNumDollars : int
+ CashRegister()
+ maximumChange() : double
+ addDollars(count : int) : void
+ addQuarters(count : int) : void
+ addDimes(count : int) : void
+ addNickels(count : int) : void
+ addReceipt(receipt : int) : void
+ takeOutChange(dAmount : double) : int
1
1
PayMachine
- m_dMaximumBalance : double
- m_dBalance : double = 0.00
- m_bOutOfOrder : boolean = false
+ PayMachine(vvmOwner : int, dMaximumBalance : double)
+ setPayMachineListener(pmlListener : int) : void
+ getBalance() : double
+ insertQuarter() : void
+ insertDime() : void
+ insertNickel() : void
+ insertDollar() : boolean
+ insertSuncard(iAccount : int) : void
+ returnChange() : void
+ deductBalance(dAmount : double) : void
+ clearBalance() : void

```
public double maximumChange(){
  double m_dNickelsAndDimes = 0.10*m_iNumDimes + 0.05*m_iNumNickels;

  if (m_iNumNickels == 0)
    return 0;
  …
}
```

```
public boolean insertDollar(){
  double dNewBalance = m_dBalance + 1.00;

  if (dNewBalance <= m_dMaximumBalance &&
     m_bOutOfOrder == false){
    if (m_crRegister.maximumChange() >= dNewBalance){
     …
    }
  }
  return false;
}
```

Communication Diagram

- Emphasizes the organization of the objects that participate in an interaction
  - **Path** – specifies a link between two objects
    - Path stereotypes can be used to indicate how one object is "linked" to another – e.g., a link can have stereotype "local" at one end showing that the designated object is local to the other object
  - **Sequence number** – specifies the time order of messages
    - Numbering scheme is monotonic – 1, 2, …
    - Dewey numbering scheme is used for nesting – e.g., for depth 2, we have 1.1, 1.2, …
- Interactions can have simple form (sequential) as well as complex forms (iterations and branching)

Semantic Equivalency*:** sequence and communication diagrams are semantically equivalent ***– one can be derived from the other; these diagrams, however, do not contain the same visual information and can be at different levels of specifications w.r.t. their formal degrees of specificity

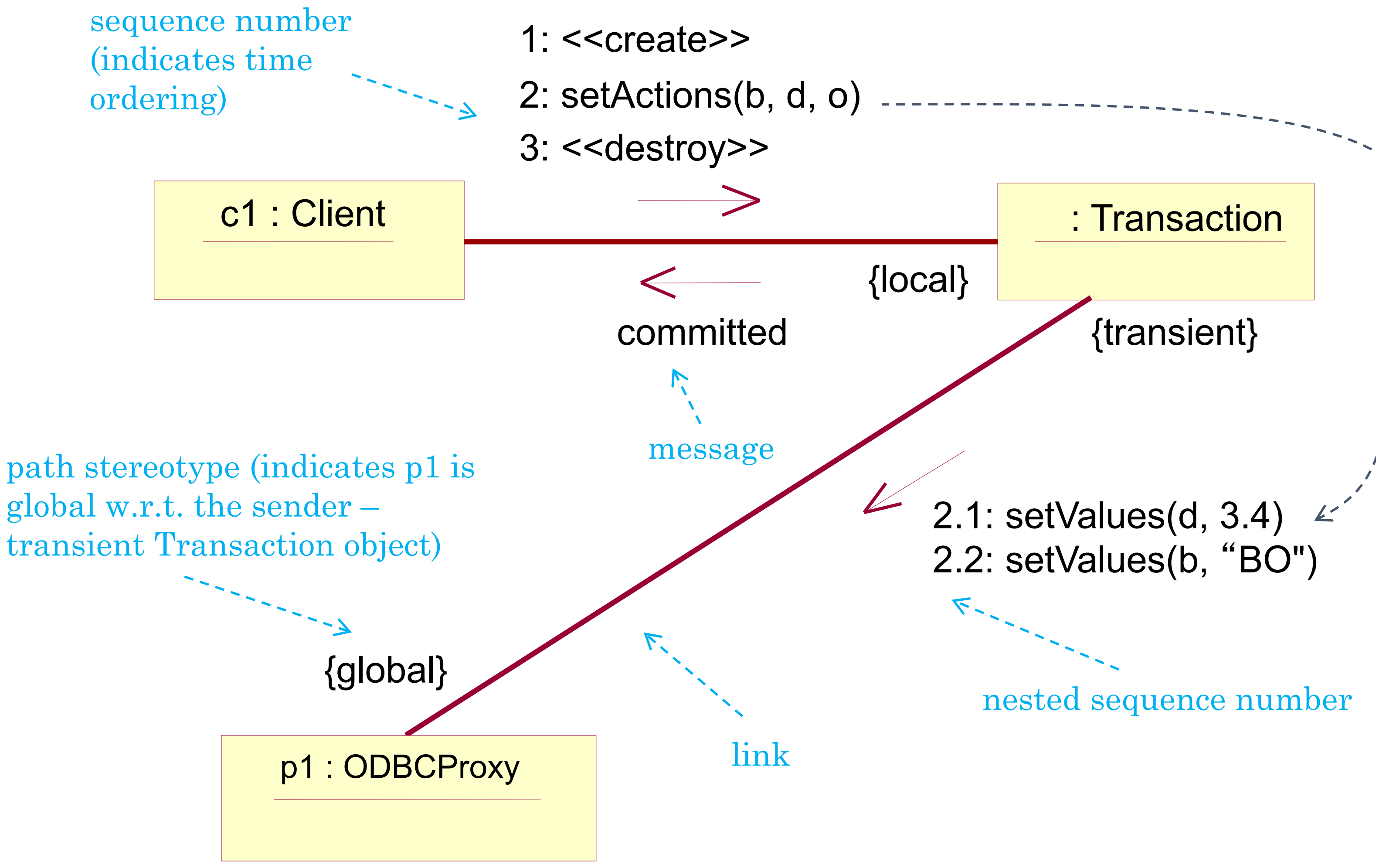
Modeling Communication Diagrams

- Determine the context for the interaction – it can be for a system, subsystem, operation, or one scenario of a use-case or collaboration
- Determine the objects which have specific roles in the interaction. There may be transient objects (those that are created and destroyed during an interaction)
- Lay objects as vertices in a graph
  - place the more important objects in the center of the diagram and their neighboring objects to the outside
  - if attribute values or role of an object changes in a significant way, include a duplicate object in the diagram – use stereotype become or copy .

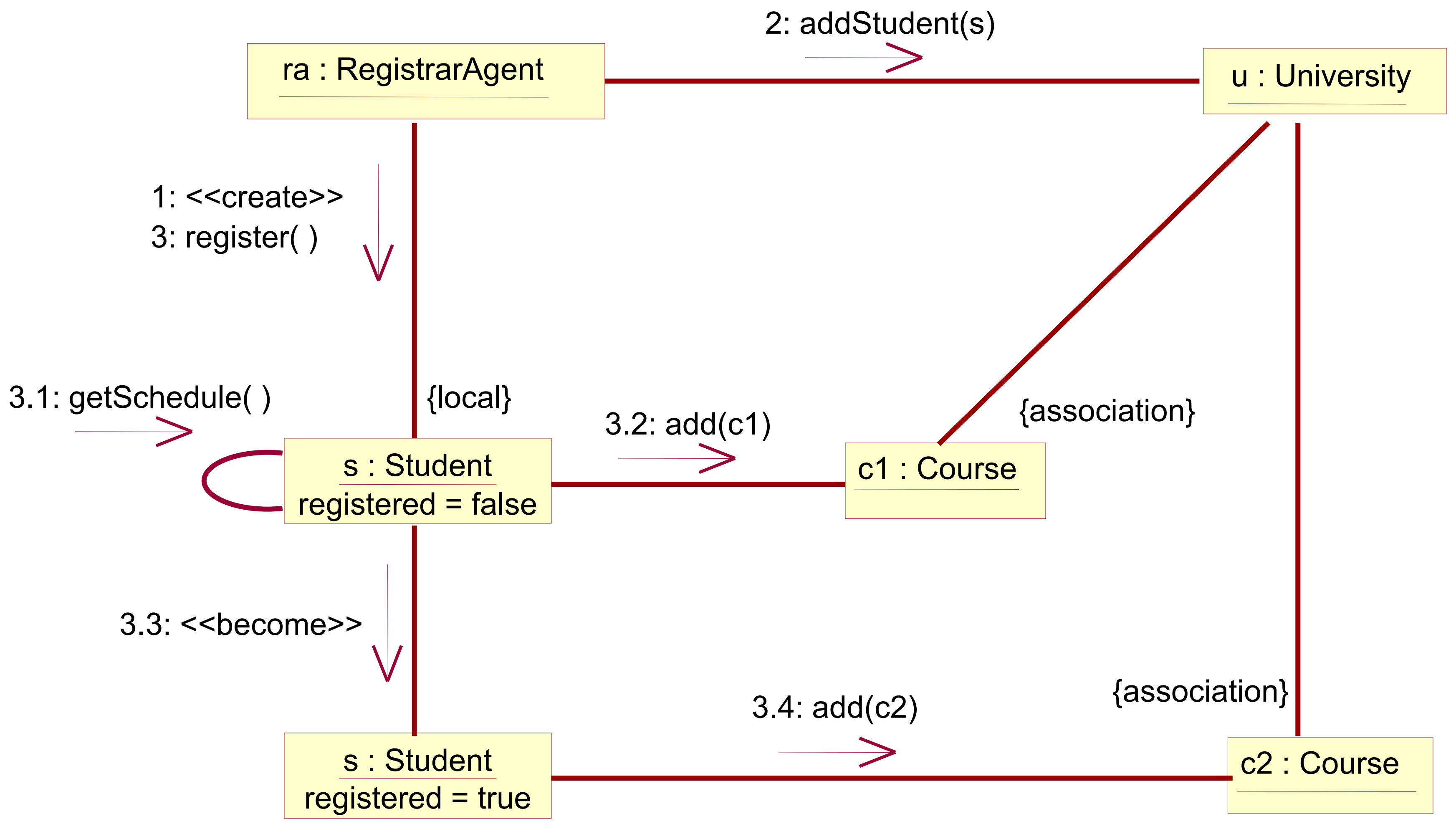
Modeling Communication Diagrams (cont.)

- Specify the links among objects along which messages are passed
    - starting with the message that initiates the interaction, attach each subsequent message to the appropriate link and assigning its sequence number. Use Dewey decimal numbering scheme
    - adorn links with suitable stereotypes local and global to explicitly specify how objects relate to one another
- As necessary adorn messages with time and space constraints
- Include pre- and post-conditions to each message – provide higher degree of formal specification for flow of control

Communication Diagrams



Communication Diagrams (cont.)



Attributes of a Well-Structured Interaction Diagram

- Focuses on communicating one aspect of a system's dynamics process view
- Contains only the essential elements – e.g., key objects, links, and messages are included
- Provides details consistent with its level of abstraction – only includes those adornments that are key to understanding
- Is not so minimalist that it misinforms the user about important semantics

Visualization: same kinds of hints (e.g., choosing a name that communicates the purpose of the interaction diagram and minimizing lines that cross) are applicable for creating interaction diagrams that are visually appropriate

References

- *Object-Oriented Analysis and Design with Applications, 3rd Edition, G. Booch, et. al. Addison Wesley, 2007*
- *OMG Unified Modeling Language Specification, http://www.omg.org/spec/, 2015*
- *The Unified Modeling Language User Guide, G. Booch, J. Rumbaugh, I. Jacobson, Addison Wesley Object Technology Series, 1999*