Introduction to the Unified Modeling Language (UML)

Martin Gogolla

University of Bremen, Computer Science Department
Database Systems Group, http://www.db.informatik.uni-bremen.de

1. Background

Informal Description of UML

Explanation 1 (Unified Modeling Language)

- is a general purpose visual modeling language
- is designed to specify visualize, construct, and document the artifacts of software systems as well as for business modeling and other non-software systems
- is simple and powerful
- is based on a small number of core concepts that most object-oriented developers can easily learn and apply
- core concepts can be combined and extended so that expert object modelers can define large and complex systems across a wide range of domains

Explanation 2 (Primary goals in design of UML)

- provide users a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models
- provide extensibility and specialization mechanisms to extend the core concepts
- be independent of particular programming languages and development processes
- provide a formal basis for understanding the modeling language
- encourage growth of OO tools market
- support higher-level development concepts such as collaborations, frameworks, patterns, and components
- integrate best practices

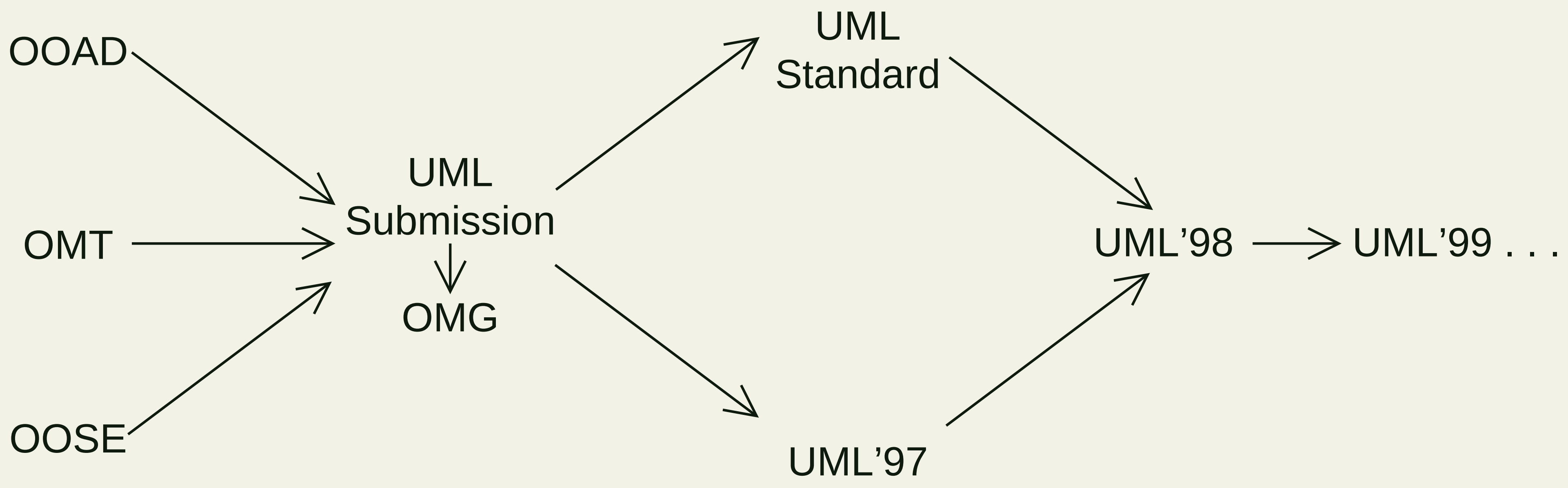
Explanation 3 (Benefits of using UML for formal methods)

- less time for explaining the notation
- document signatures and axioms with UML diagrams
- document the described system structure and behavior with UML diagrams
- think about formal specifications in a different language
- use Object Constraint Language OCL as a vehicle for transporting other notations

Benefits of using UML for formal methods (cont’d)

- tunable to special needs because it is a language independent from a software process model
- speak the same language as people in industrial applications (with the obvious danger that the two parties talk about different things $\longrightarrow$ use only a minmal UML subset and explain the semantics)
- way of bringing formal methods and industrial applications closer together

Development of UML



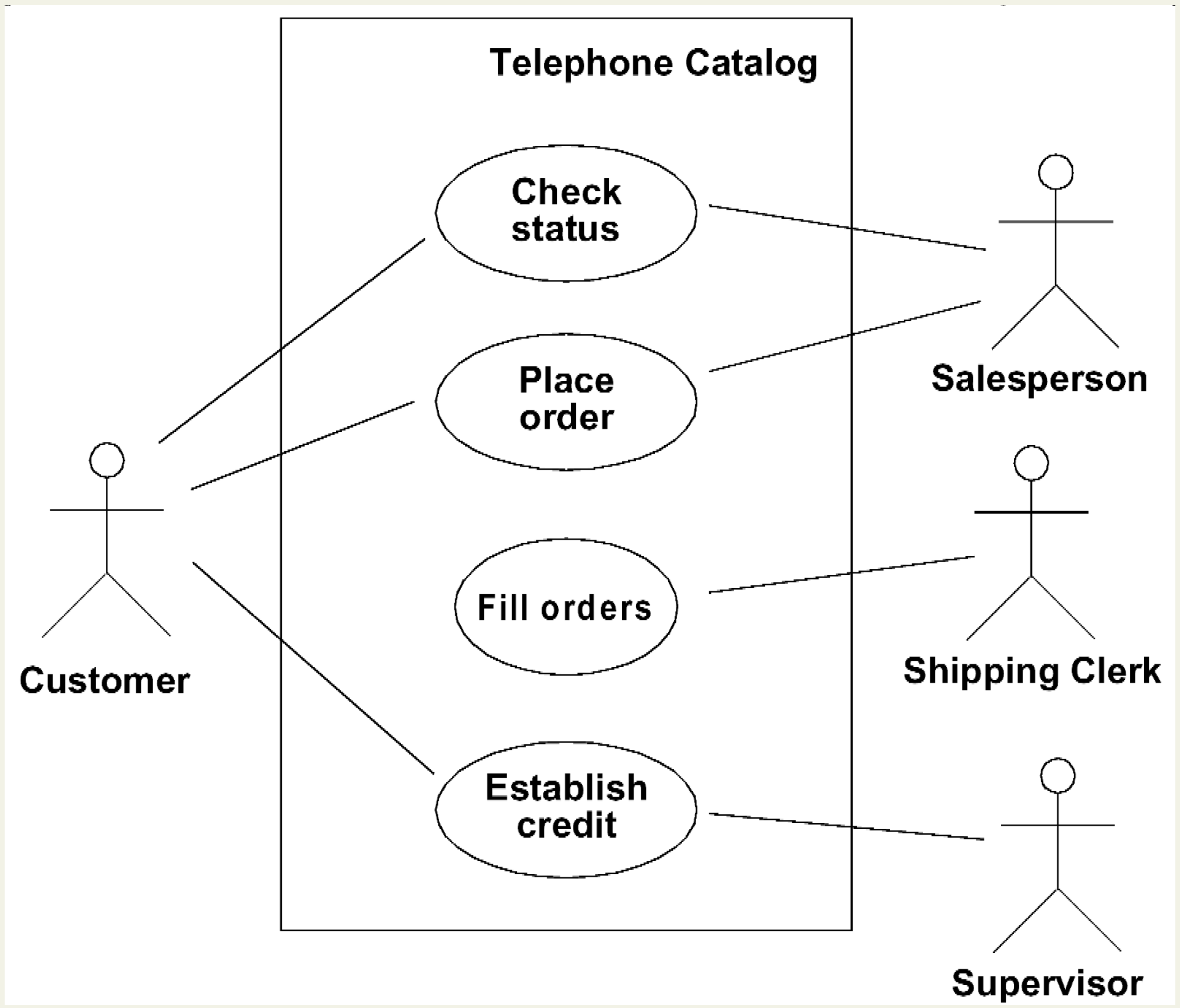
2. UML by Example

2.1 Nine Diagram Forms

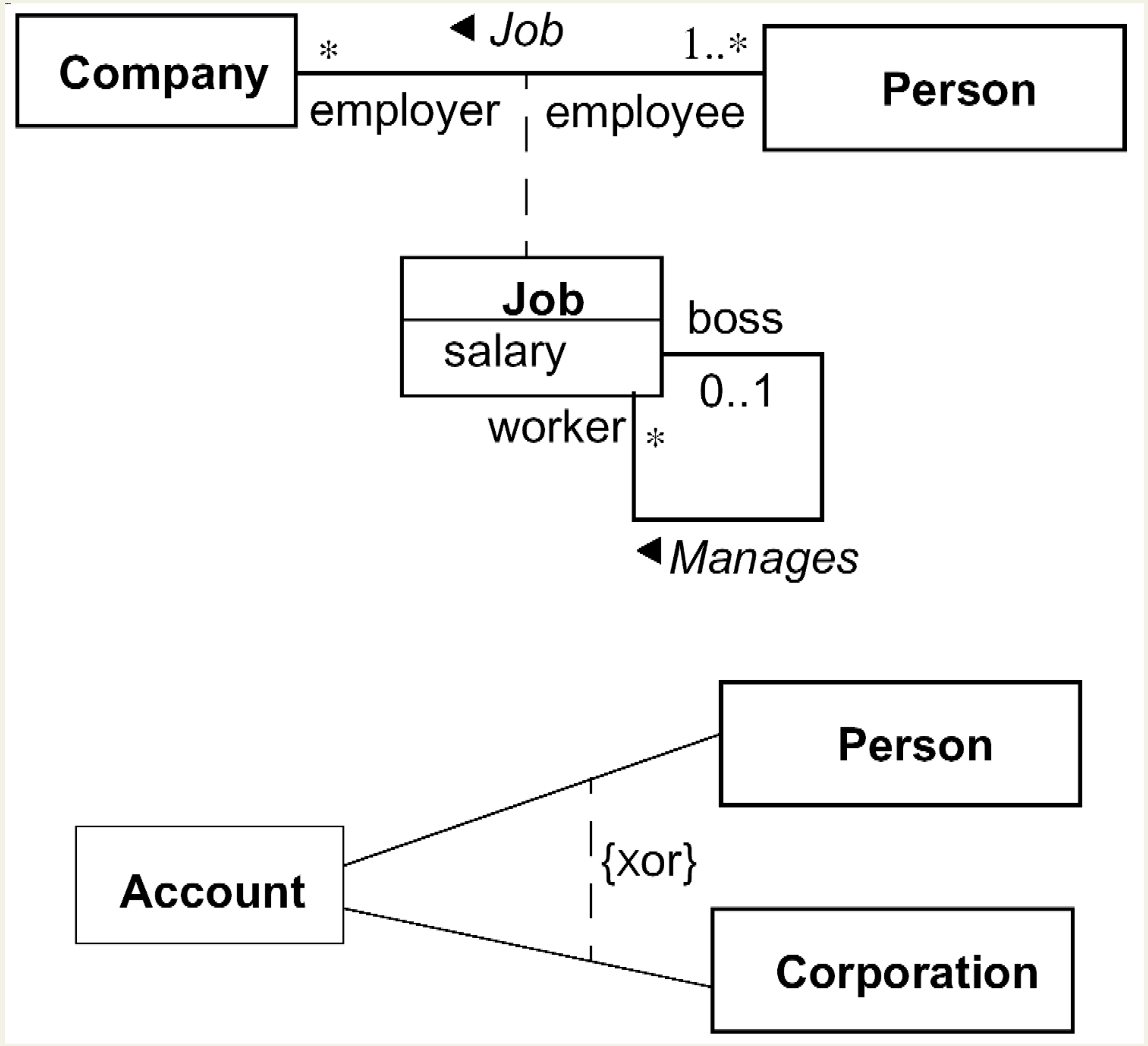
following the Notation Guide

- Static structure diagrams
  - (1) Class and (2) Object diagrams
- (3) Use case diagrams
- Interaction diagrams
  - (4) Sequence and (5) Collaboration diagrams
- (6) Statechart diagrams
- (7) Activity diagrams
- Implementation diagrams
  - (8) Component and (9) Deployment diagrams

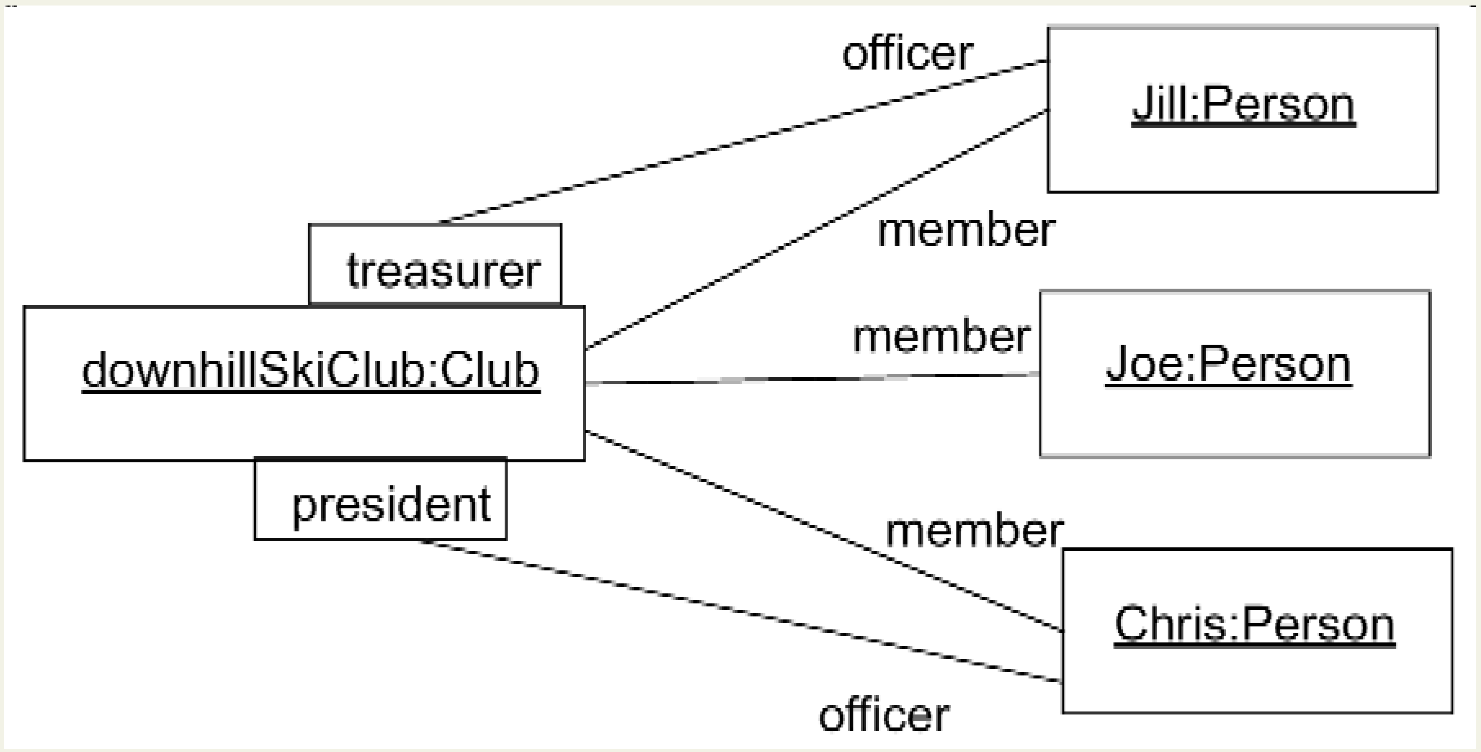
Use Case Diagram



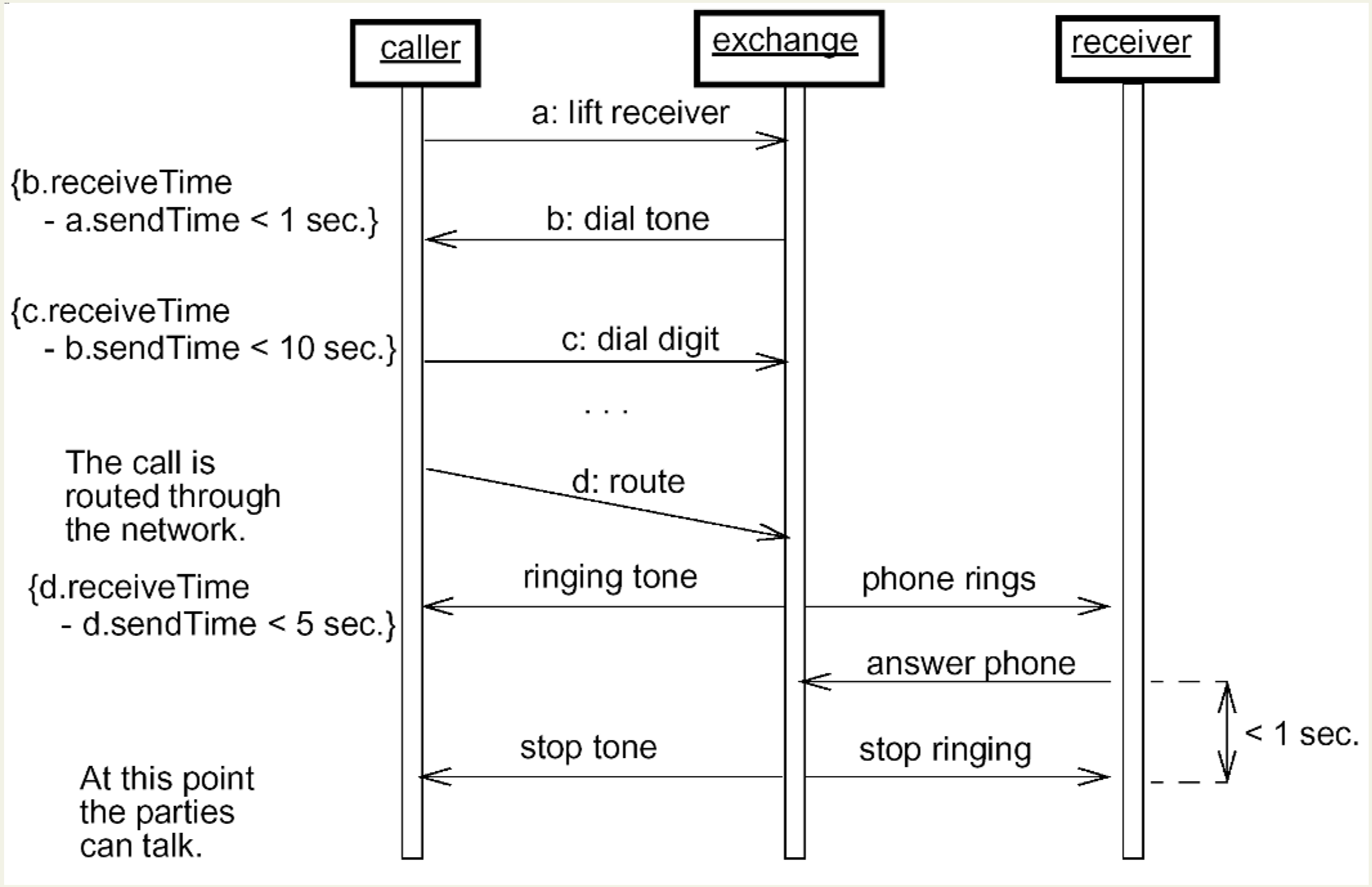
Class Diagram



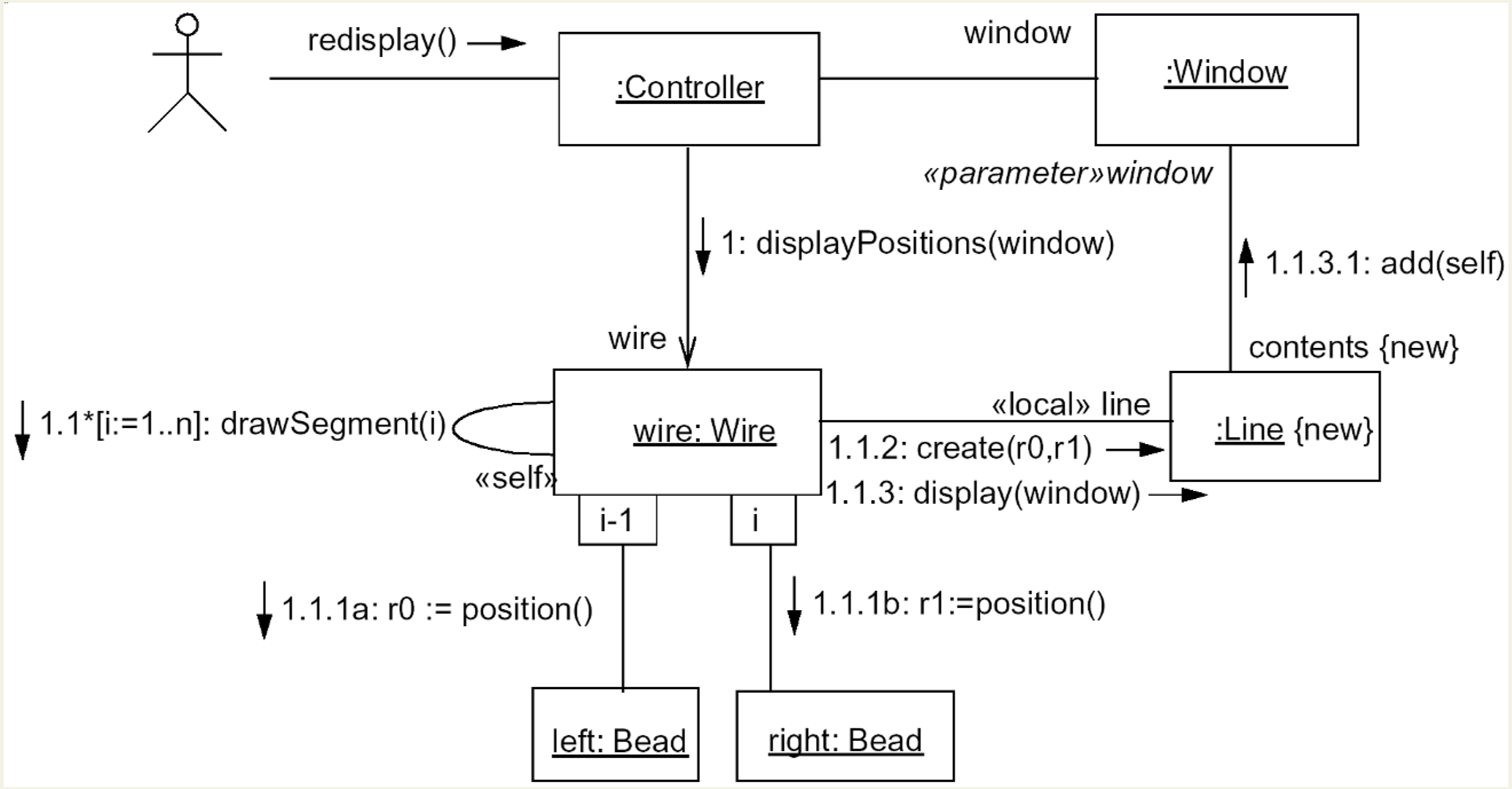
Object Diagram



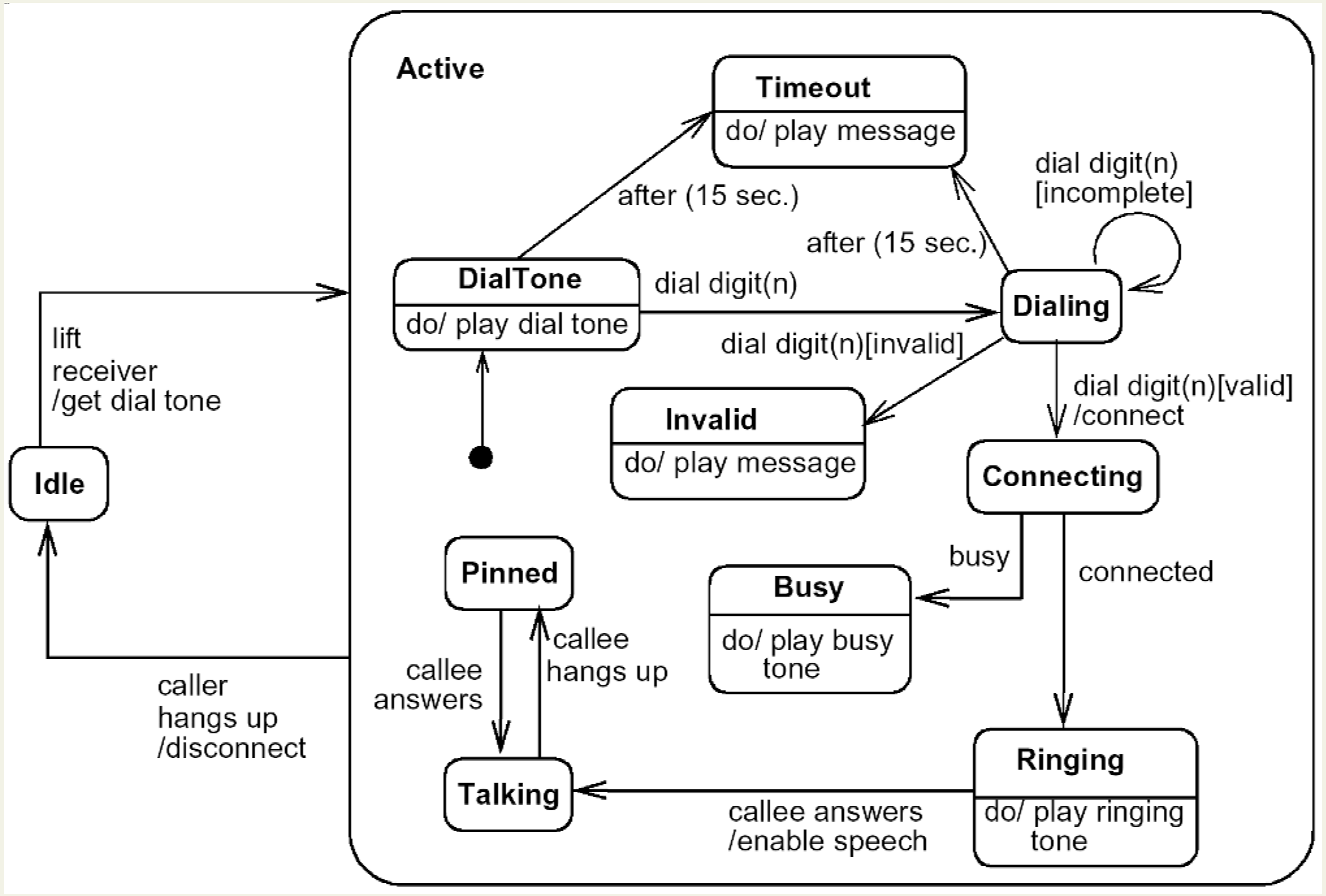
Sequence Diagram


caller
exchange
receiver
a: lift receiver
{b.receiveTime - a.sendTime < 1 sec.}
b: dial tone
{c.receiveTime - b.sendTime < 10 sec.}
c: dial digit
. . .
The call is routed through the network.
d: route
{d.receiveTime - d.sendTime < 5 sec.}
ringing tone
phone rings
answer phone
< 1 sec.
stop tone
stop ringing
At this point the parties can talk.

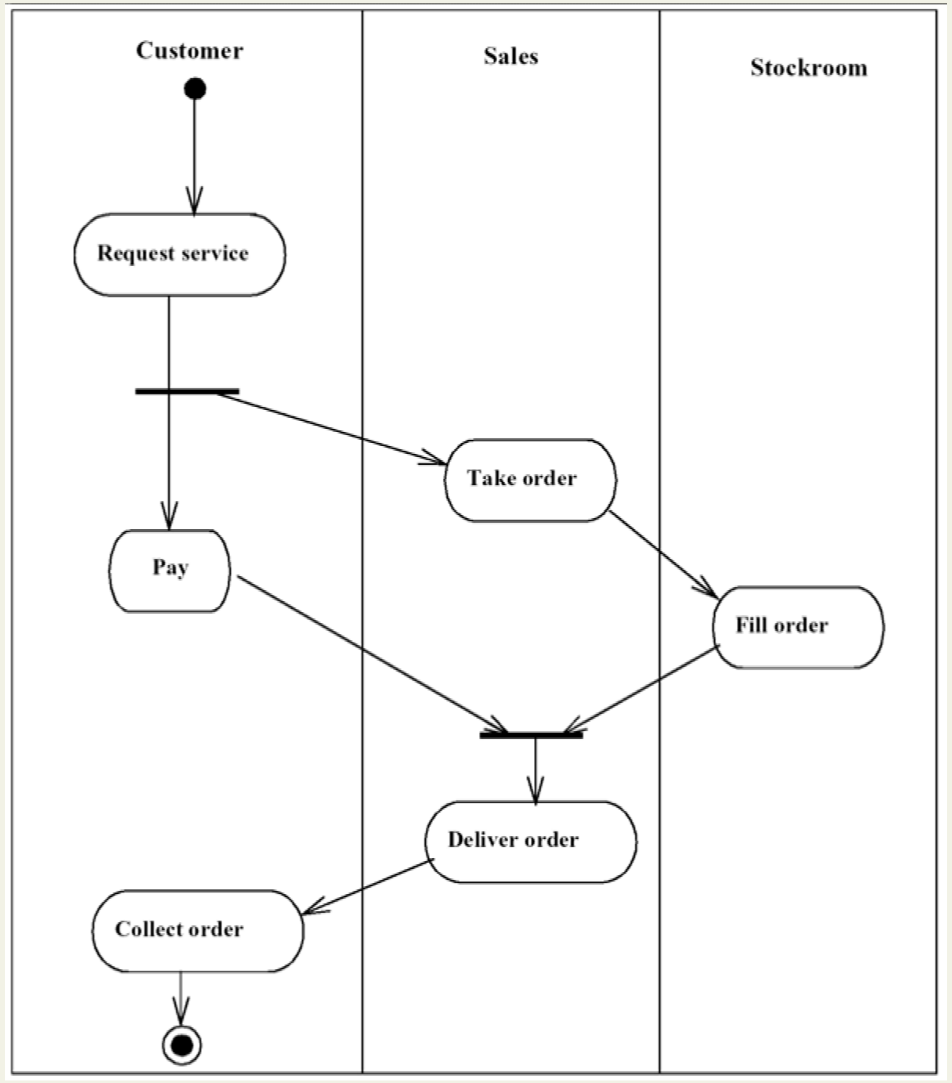
Collaboration Diagram



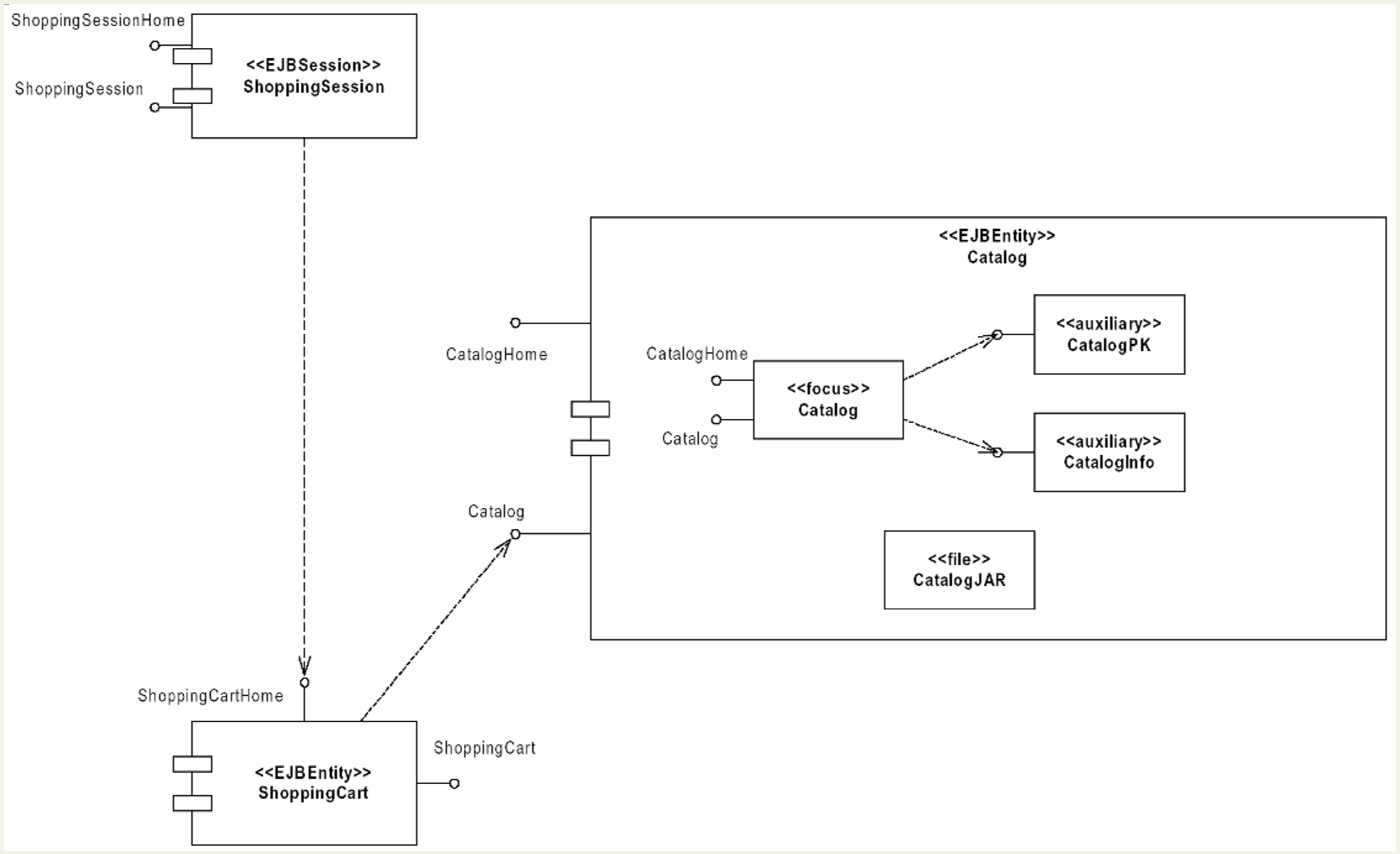
Statechart Diagram



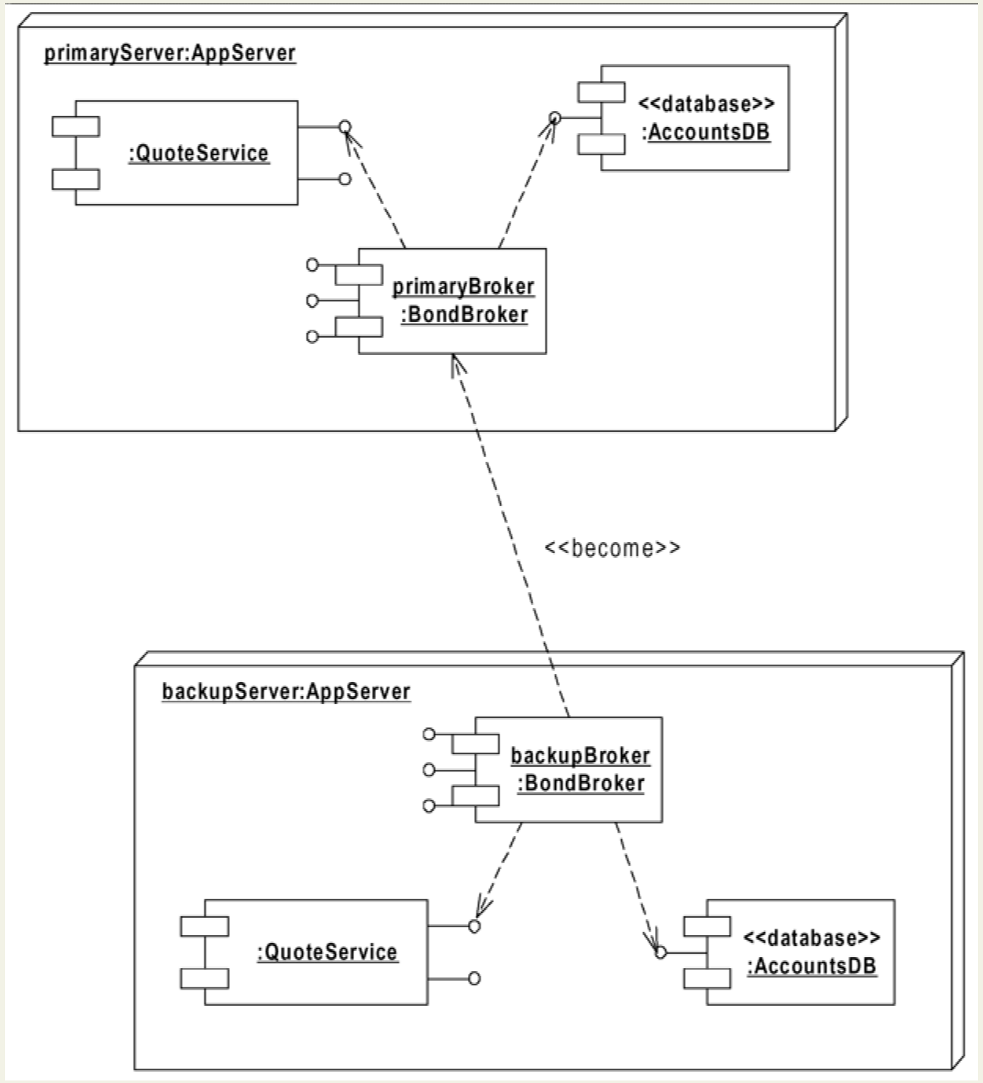
Activity Diagram



Component Diagram



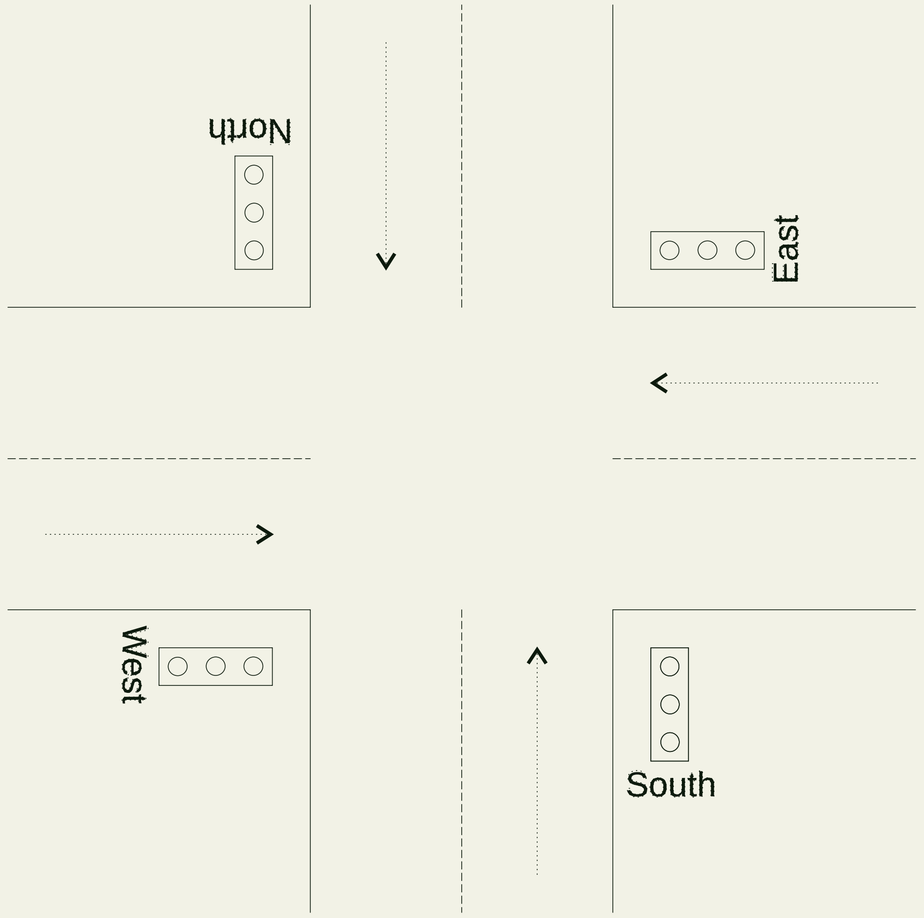
Deployment Diagram



2.2 TROLL light

- intended to specify an information system as a community existing and interacting objects
- intends to coherently describe structure and behaviour of objects
- incorporates ideas from semantic data models, process theory, and abstract data types
- has set-theoretic semantics based on transition systems
- is supported by an animation system with web-based user interface

Crossing System



```
TEMPLATE          TrafficLight
DATA TYPES        Int, Bool;
ATTRIBUTES        Phase : int;
                  DERIVED Red, Yellow, Green : bool;
EVENTS            BIRTH createLight(initPhase:int);
                  switchLight;
CONSTRAINTS       1<=Phase AND Phase<=3;
VALUATION         [createLight(initPhase)] Phase=initPhase;
                  [switchLight] Phase=(Phase MOD 3)+1;
DERIVATION        Red    = (Phase=1);
                  Yellow = (Phase=3);
                  Green  = (Phase=2 OR Phase=3);
BEHAVIOR          PROCESS TrafficLight =
                              ( createLight -> LightLife );
                  PROCESS LightLife =
                              ( switchLight -> LightLife );
END TEMPLATE
```

```
TEMPLATE          Control
DATA TYPES        Int;
TEMPLATES         TrafficLight;
SUBOBJECTS        West, East, North, South : trafficLight;
ATTRIBUTES        Phase : int;
EVENTS            BIRTH createControl;
                  createLights;
                  switchControl;
VALUATION         [createControl] Phase=4;
                  [switchControl] Phase=(Phase MOD 4)+1;
CONSTRAINTS       1<=Phase AND Phase<=4;
                  West.Phase=East.Phase;
                  North.Phase=South.Phase;
                  West.Phase<>North.Phase;
                  NOT(West.Green AND North.Green);
                  NOT(West.Red AND North.Red);
```

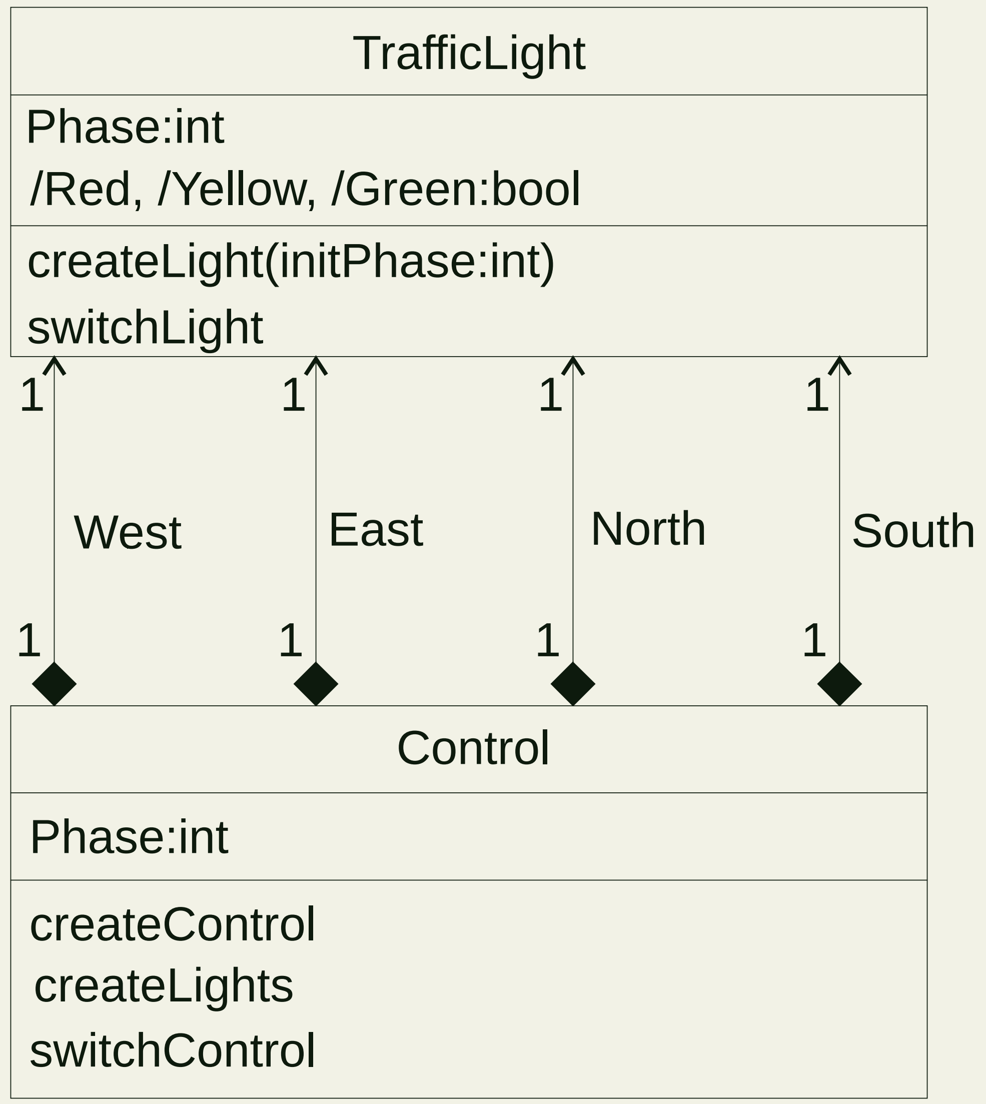
```
INTERACTION  createLights >> West.createLight(3),
                             East.createLight(3),
                             North.createLight(1),
                             South.createLight(1);
             {Phase=4 OR Phase=2} switchControl >>
                             West.switchLight,
                             North.switchLight;
             {Phase=1} switchControl >> North.switchLight;
             {Phase=3} switchControl >> West.switchLight;
             West.switchLight >> East.switchLight;
             North.switchLight >> South.switchLight;
BEHAVIOR     PROCESS Control =
                          (createControl -> CreateLights);
             PROCESS CreateLights =
                          (createLights -> ControlLife);
             PROCESS ControlLife =
                          (switchControl -> ControlLife);
END TEMPLATE
```

2.3 UML Diagrams for TROLL light Example

To follow

- examples from crossing scenario for the nine diagram forms
- demonstrating UML diagram variety and emphasizing different aspects
- description of general goal of respective diagram form
- quotations taken from UML Notation Guide

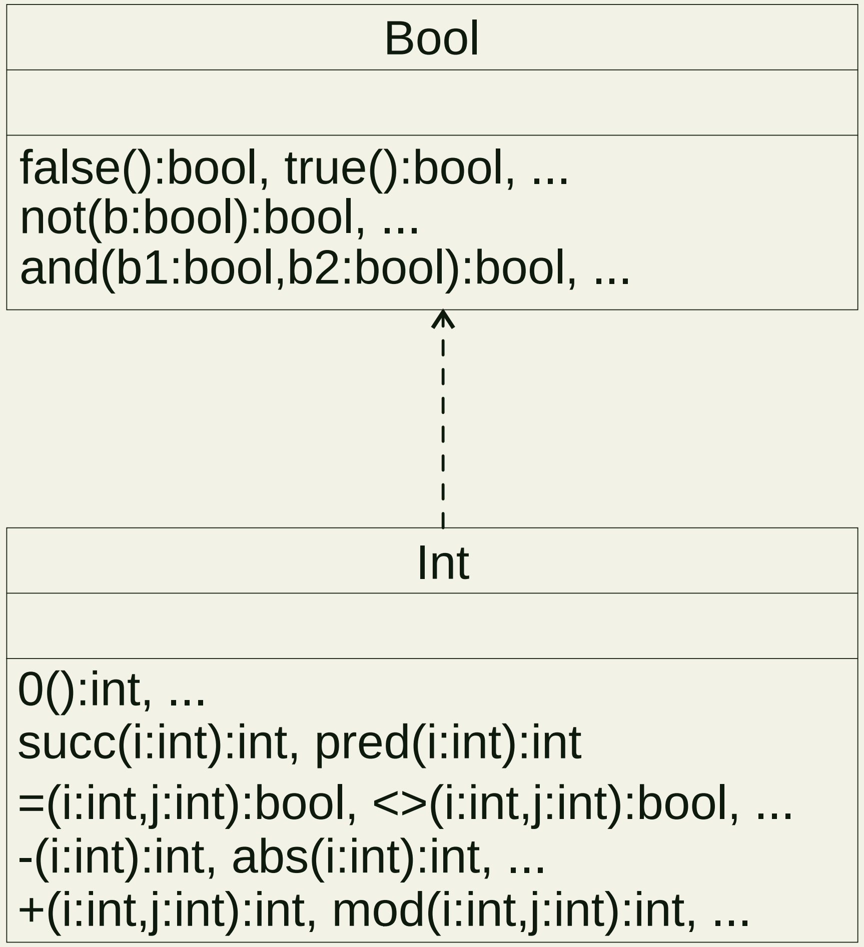
Class Diagram for Object Types



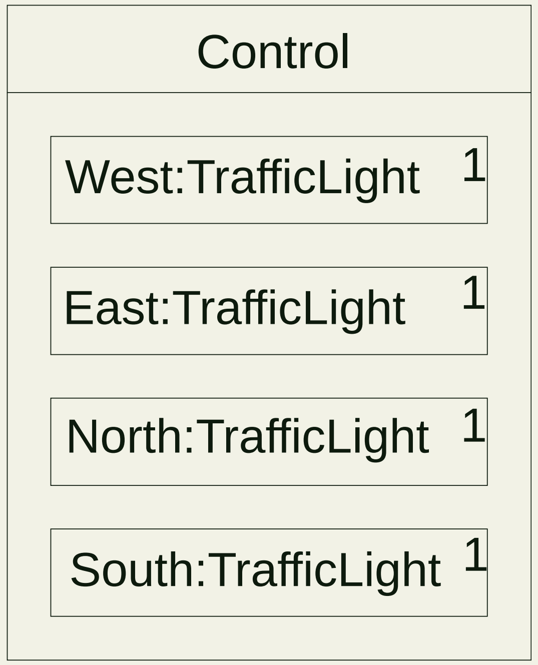
Explanation 4 (Class diagrams)

- show static structure of the model
- in particular, things that exist (such as classes and types), their internal structure, and their relationships to other things
- do not show temporal information
- although they may contain reified occurences of things that have or things that describe temporal behavior
- are graphs of classifier elements connected by their various static relationships

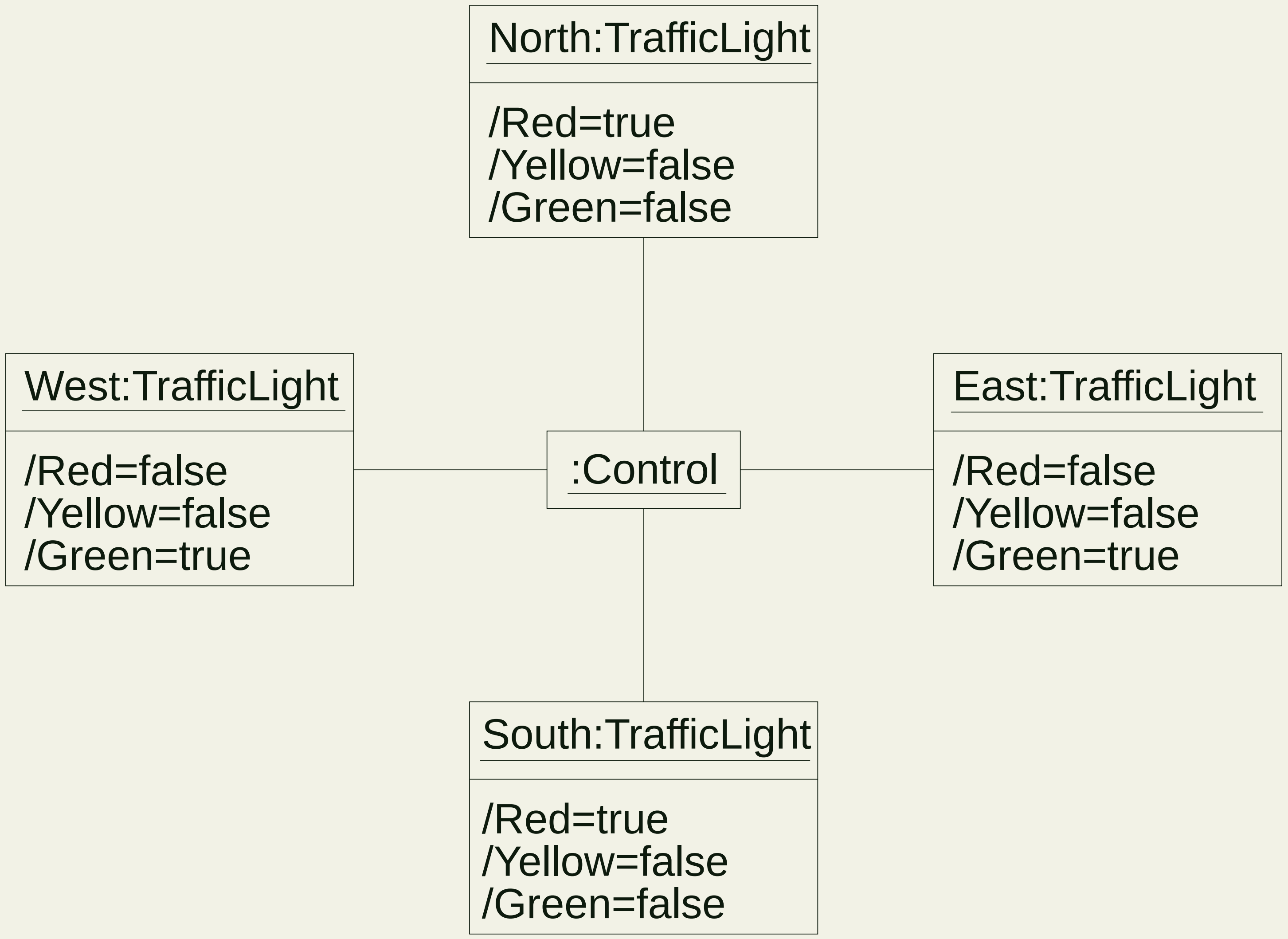
Class Diagram for Data Types


Bool
false():bool, true():bool, ...
not(b:bool):bool, ...
and(b1:bool,b2:bool):bool, ...
Int
0():int, ...
succ(i:int):int, pred(i:int):int
=(i:int,j:int):bool, <>(i:int,j:int):bool, ...
-(i:int):int, abs(i:int):int, ...
+(i:int,j:int):int, mod(i:int,j:int):int, ...

Class Diagram with Graphical Nesting for Composition



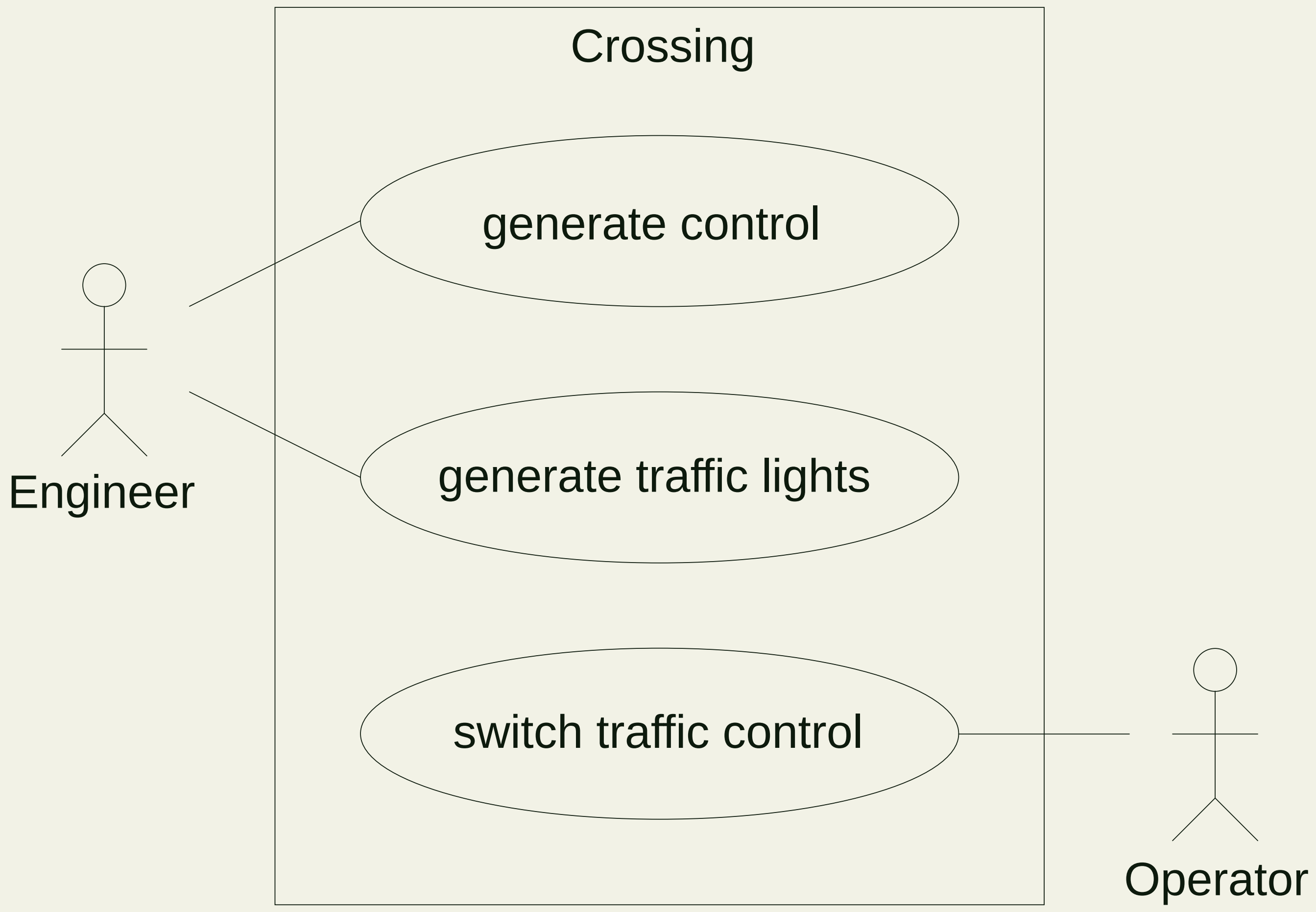
Object Diagram



Explanation 5 (Object diagrams)

- show instances compatible with a particular class diagram
- are graphs of instances, including objects and data values
- are instances of a class diagram (see Expl. 3)
- show a snapshot of detailed state of a system at a point in time

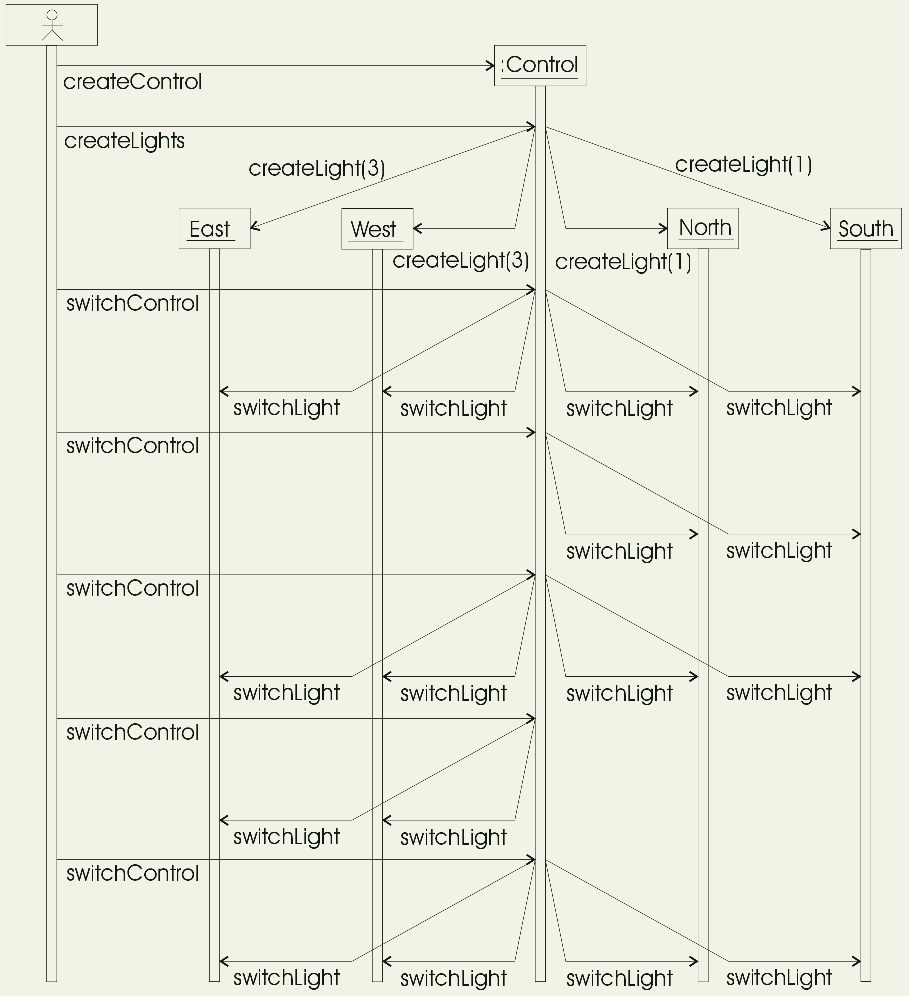
Use Case Diagram



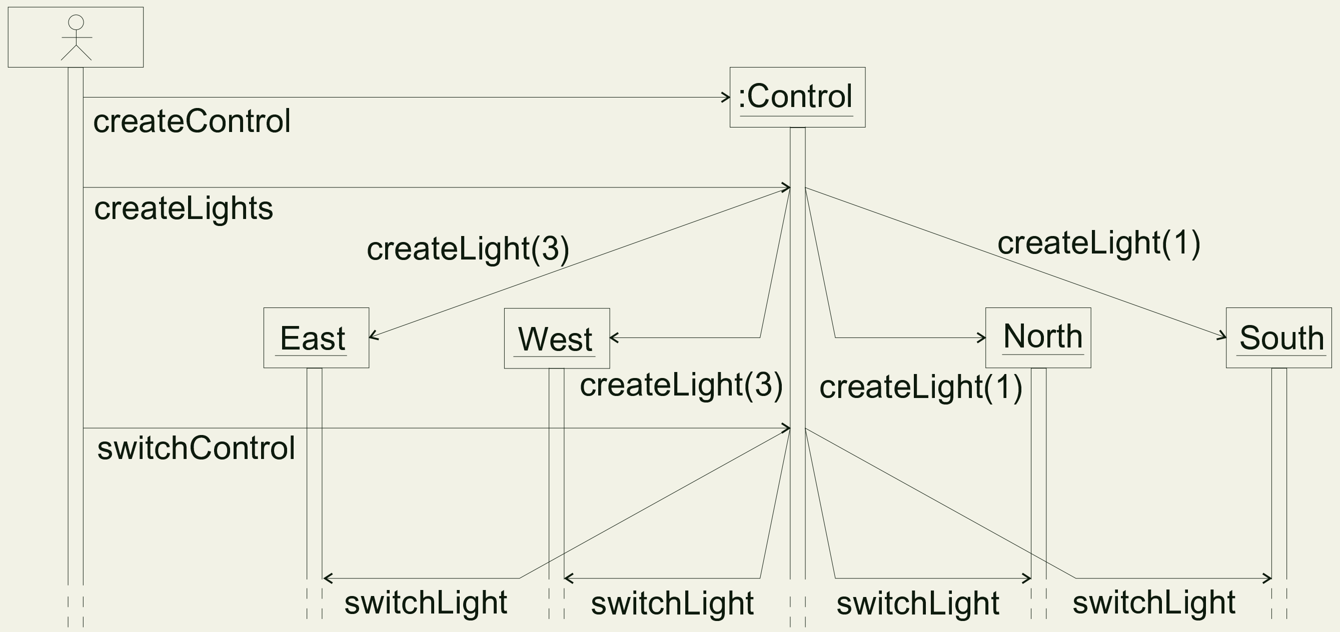
Explanation 6 (Use case diagrams)

- show relationship among actors and use cases within a system
- represent functionality of a system or a class as manifested to external interactors
- are graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between actors and use cases, and generalizations among use cases
- show coherent units of functionality provided by a system or class as manifested by sequences of messages exchanged among the system and one or more outside interactors (called actors) together with actions performed by the system

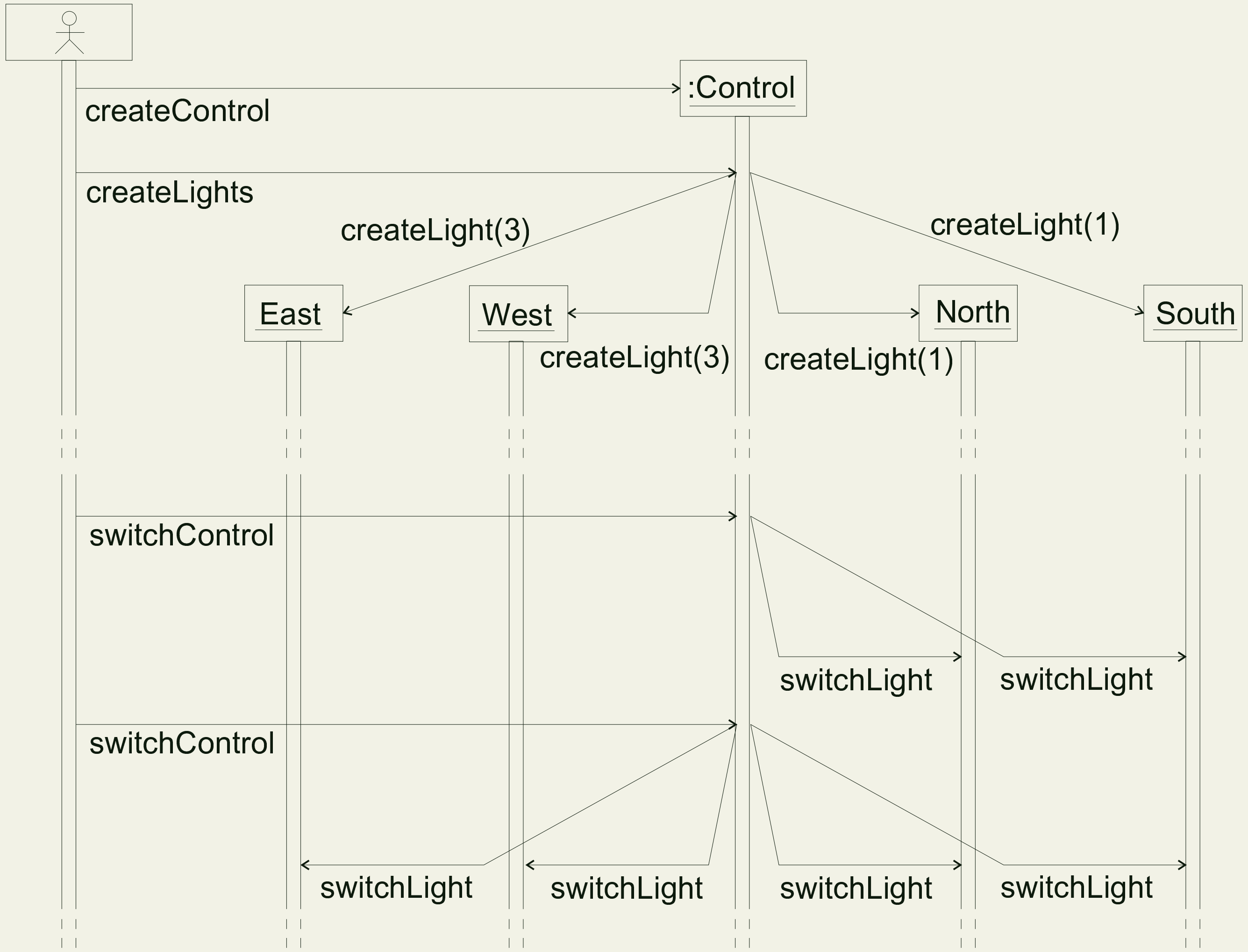
Sequence Diagram



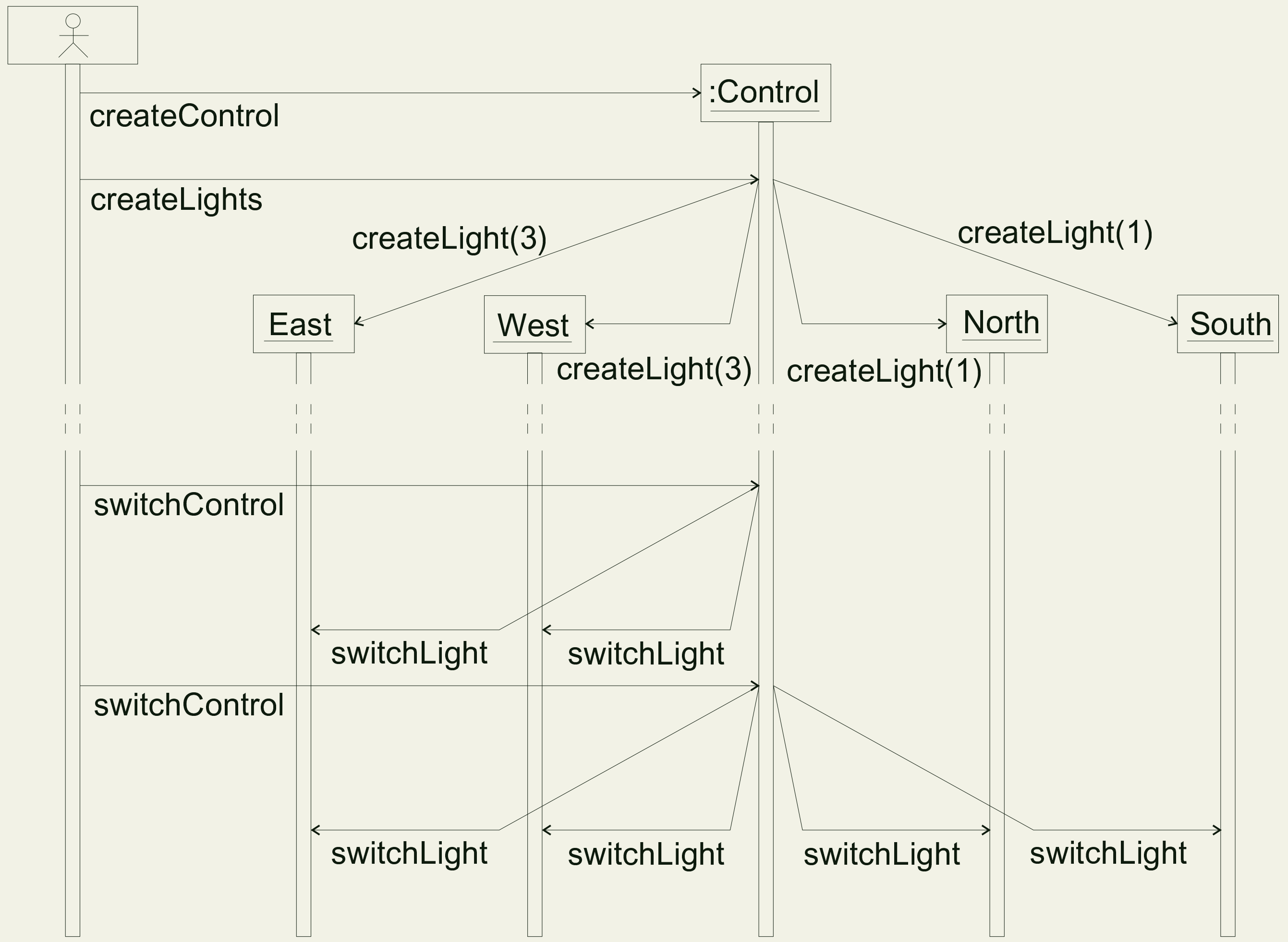
Sequence Diagram Part 1



Sequence Diagram Part 2



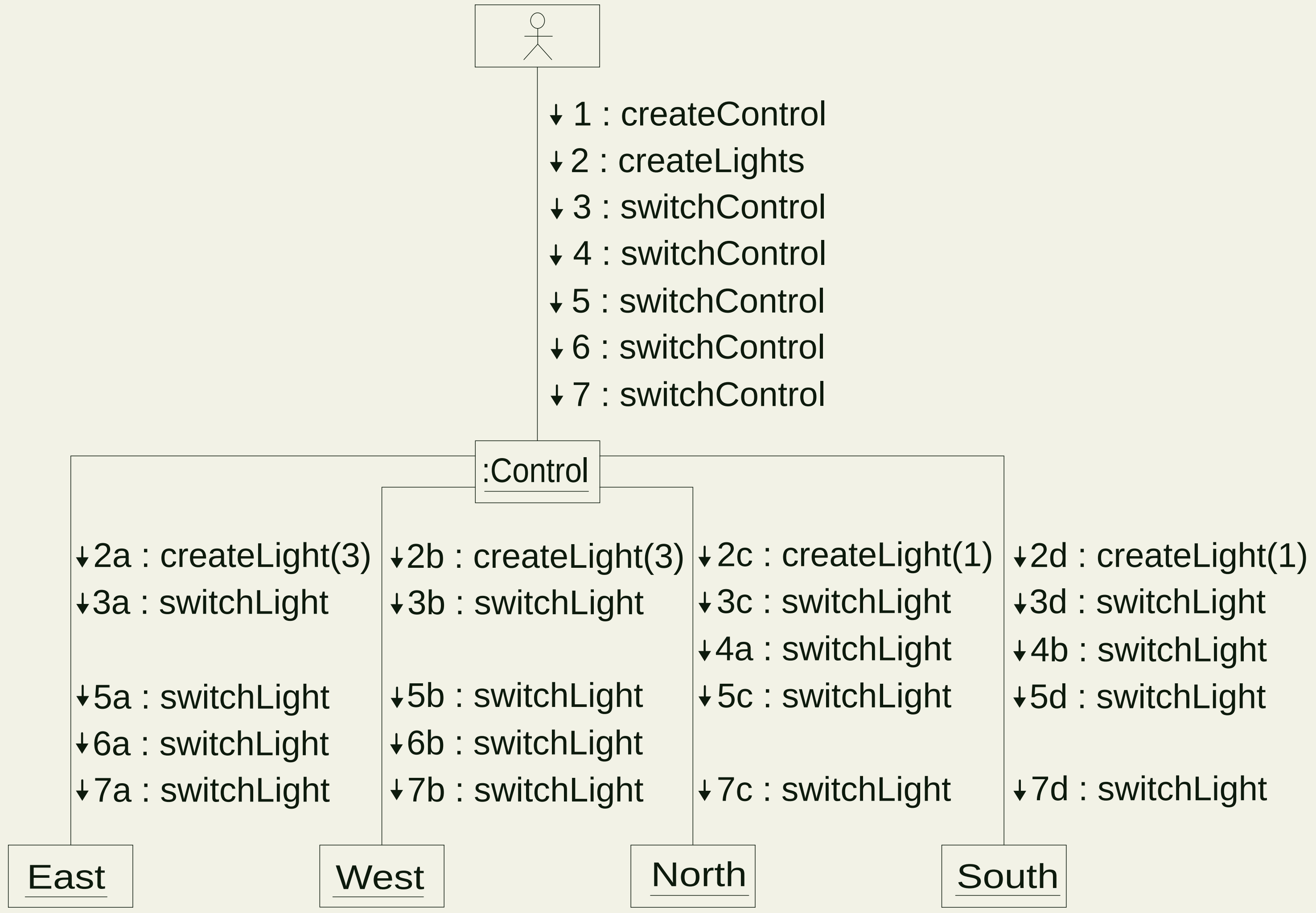
Sequence Diagram Part 3



Explanation 7 (Sequence diagrams)

- show a pattern of interaction among objects
- show an interaction arranged in time sequence
- show objects participating in interaction by their lifelines and messages that they exchange arranged in time sequence
- do not show associations among objects
- can exist in a generic form (describe all possbile sequences) and in an instance form (describe one actual sequence constistent with the generic form); without loops or branches, the two forms are isomorphic

Collaboration Diagram



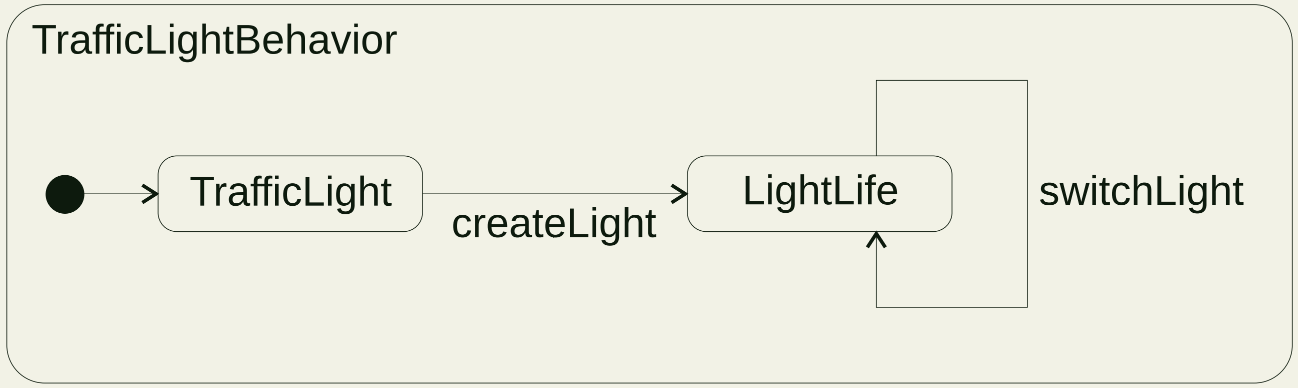
Explanation 8 (Collaboration diagrams)

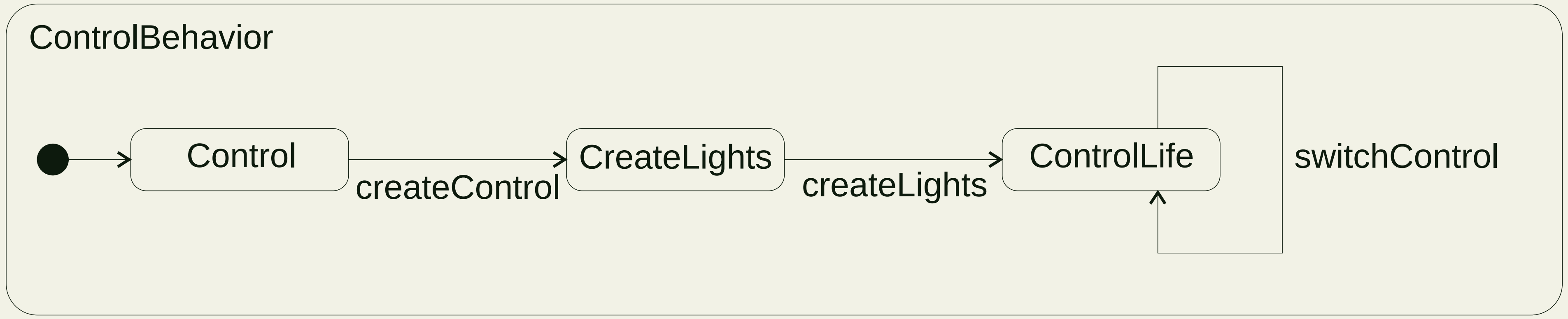
- show a pattern of interaction among objects
- show an interaction organized around the objects in the interaction and their links to each other
- show relationships among object roles
- do not show time as a separate dimension, so sequence of messages and concurrent threads must be determined using sequence numbers
- represent a collaboration, which is a set of objects related in a particular context, and an interaction, which is a set of messages exchanged among objects within a collaboration to effect a desired operation or result

Sequence diagrams versus Collaboration diagrams

- Sequence diagrams and collaboration diagrams express similar information but show it in different ways
- Sequence diagrams show explicit sequence of messages and are better for real-time specifications and for complex scenarios
- Collaboration diagrams show relationships among objects and are better for understanding all of effects on a given object and for procedural design

Statechart Diagram for BEHAVIOR Sections

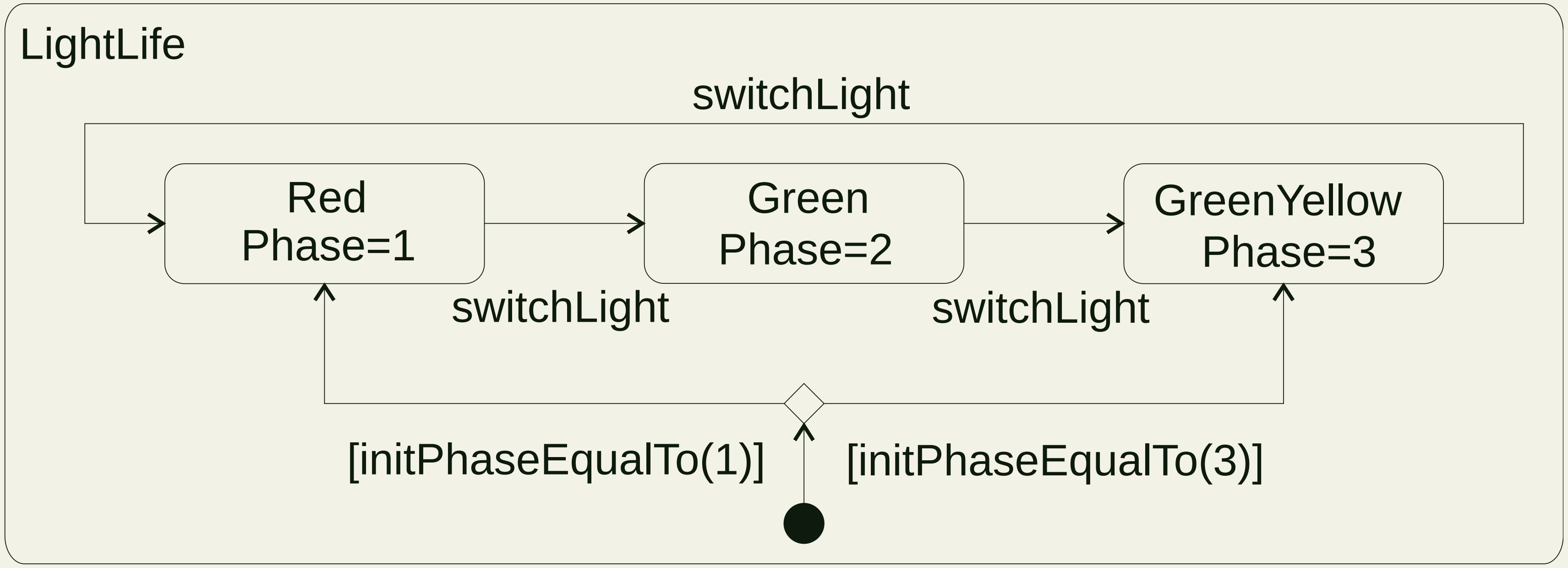




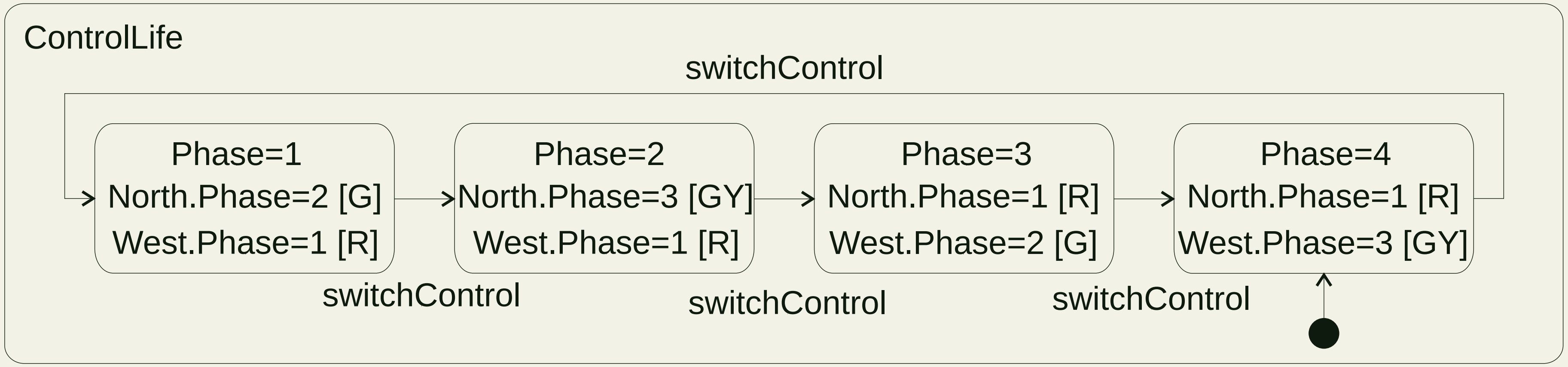
Explanation 9 (Statechart diagrams)

- show sequences of states that an object or an interaction goes through during its life in response to received stimuli, together with its responses and actions
- show graphs of states and transitions that describe response of an object of a given class to receipt of outside stimuli
- are attached to a class or a method
- show states represented by state symbols and transitions represented by arrows connecting state symbols
- may contain subdiagrams by physical containment

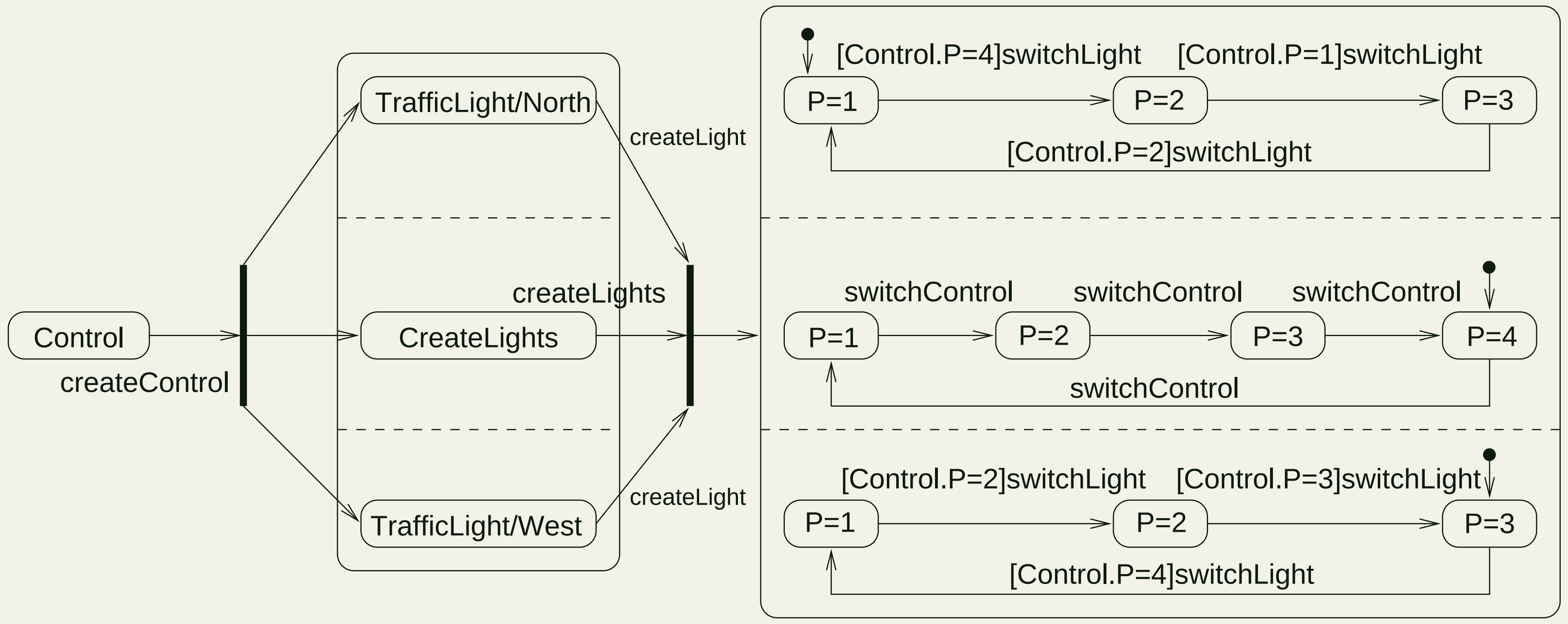
Statechart Diagram for LightLife



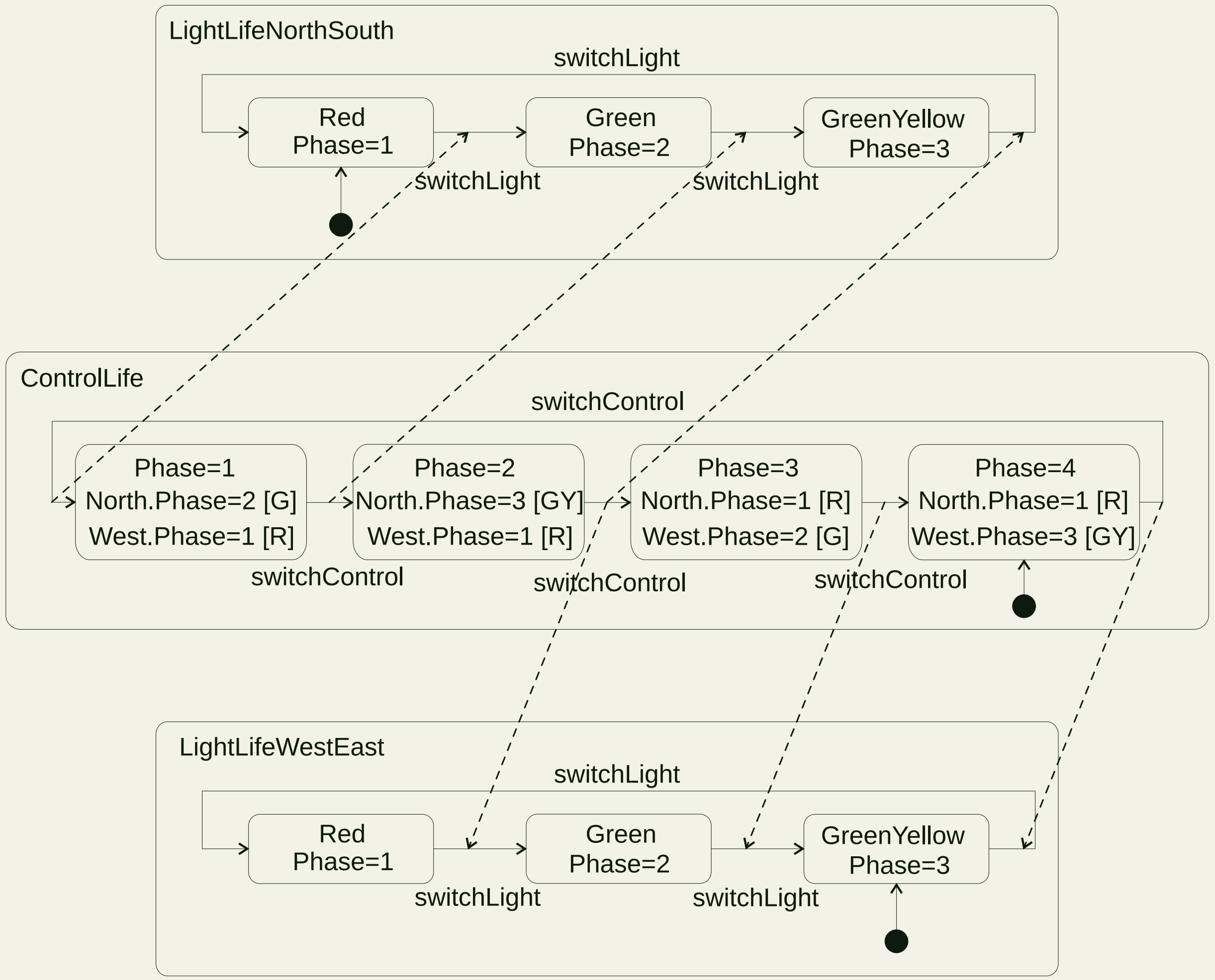
Statechart Diagram for ControlLife



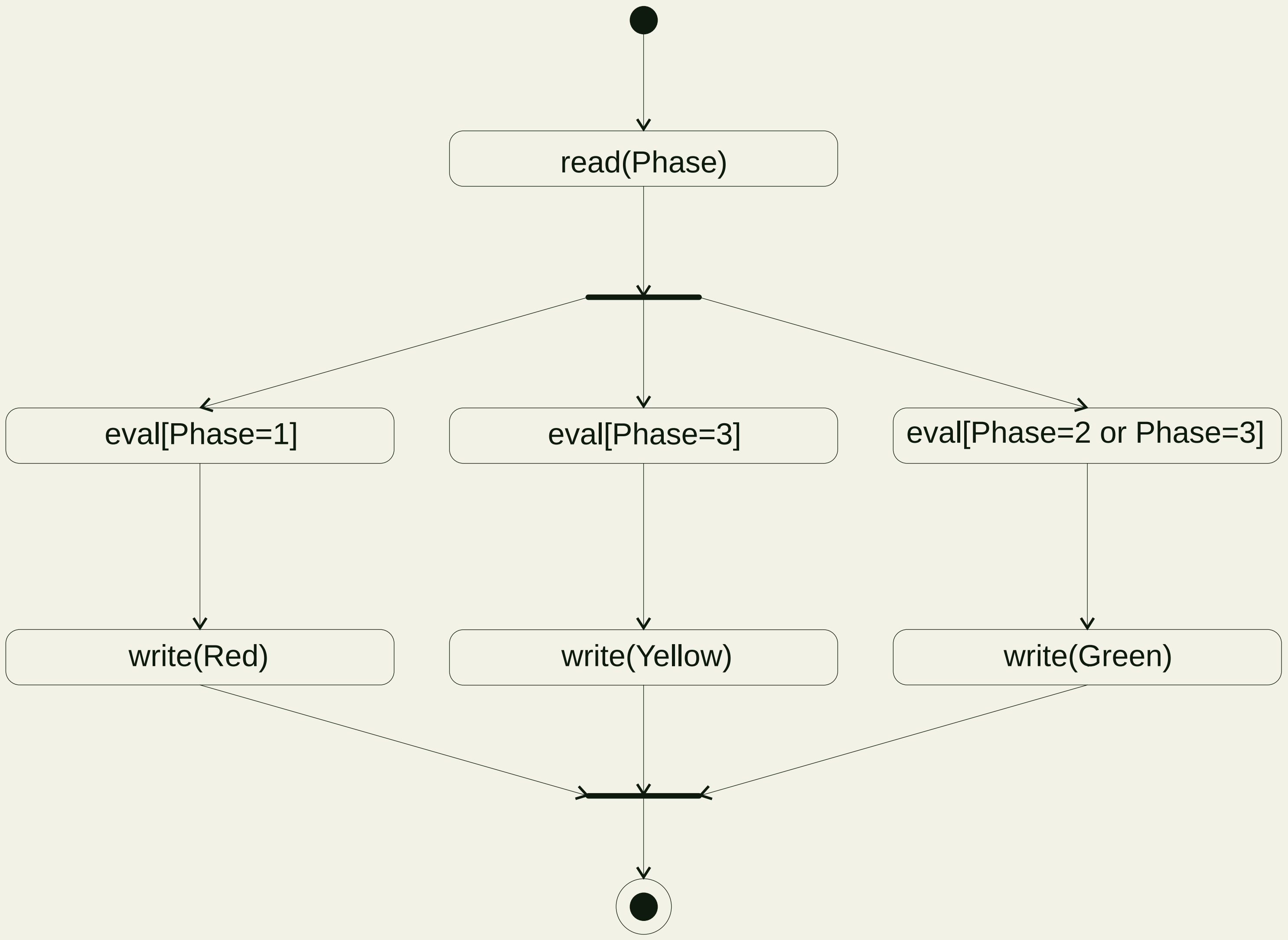
Statechart Diagram with Concurrent Substates



Statechart Diagrams with Messages Sent



Activity Diagram for Derivation Rules



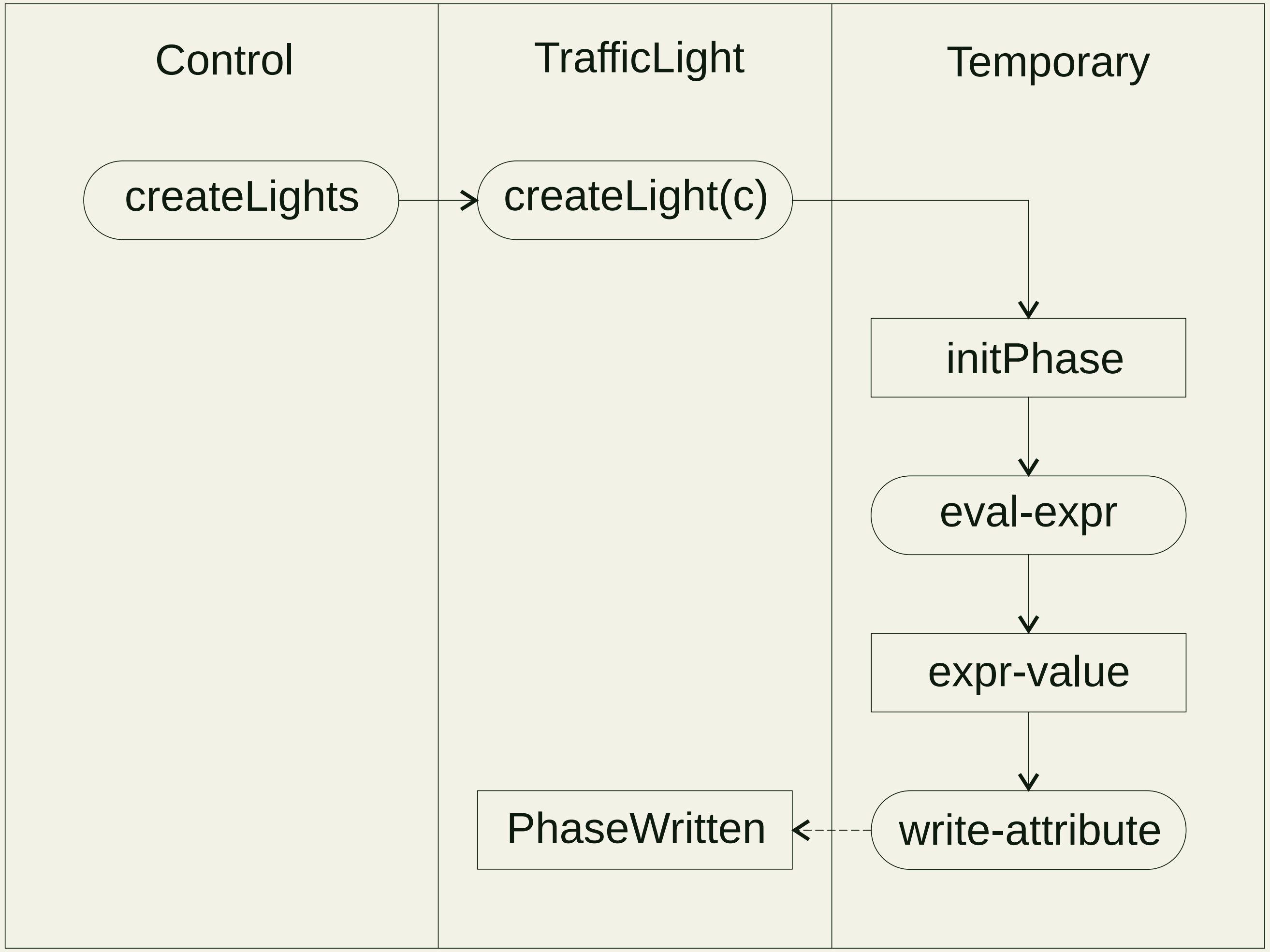
Explanation 10 (Activity diagrams)

- are variations of state machines in which states are activities representing performance of operations and the transitions are triggered by completion of operations
- represent a state machine of a procedure itself; the procedure is the implementation of an operation on owning class
- are special cases of state diagrams in which all (or at least most) of states are action states and in which all (or at least most) of transitions are triggered by completion of actions in the source states
- are attached to a class or to implementation of an operation or a use case
- focus on flows driven by internal processing (as opposed to external events)

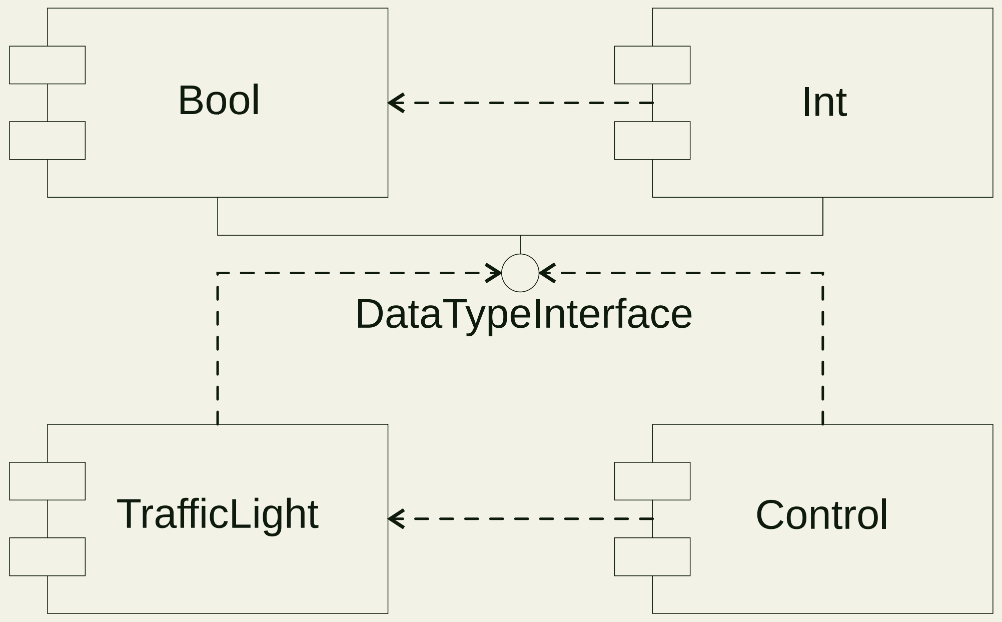
Activity diagrams versus Statechart diagrams

- Use activity diagrams in situations where all or most of the events represent completion of internally-generated actions (that is, procedural flow of control)
- Use ordinary state diagrams in situations where asynchronous events occur

Activity Diagram for Triggering and Valuation of `Phase=initPhase`



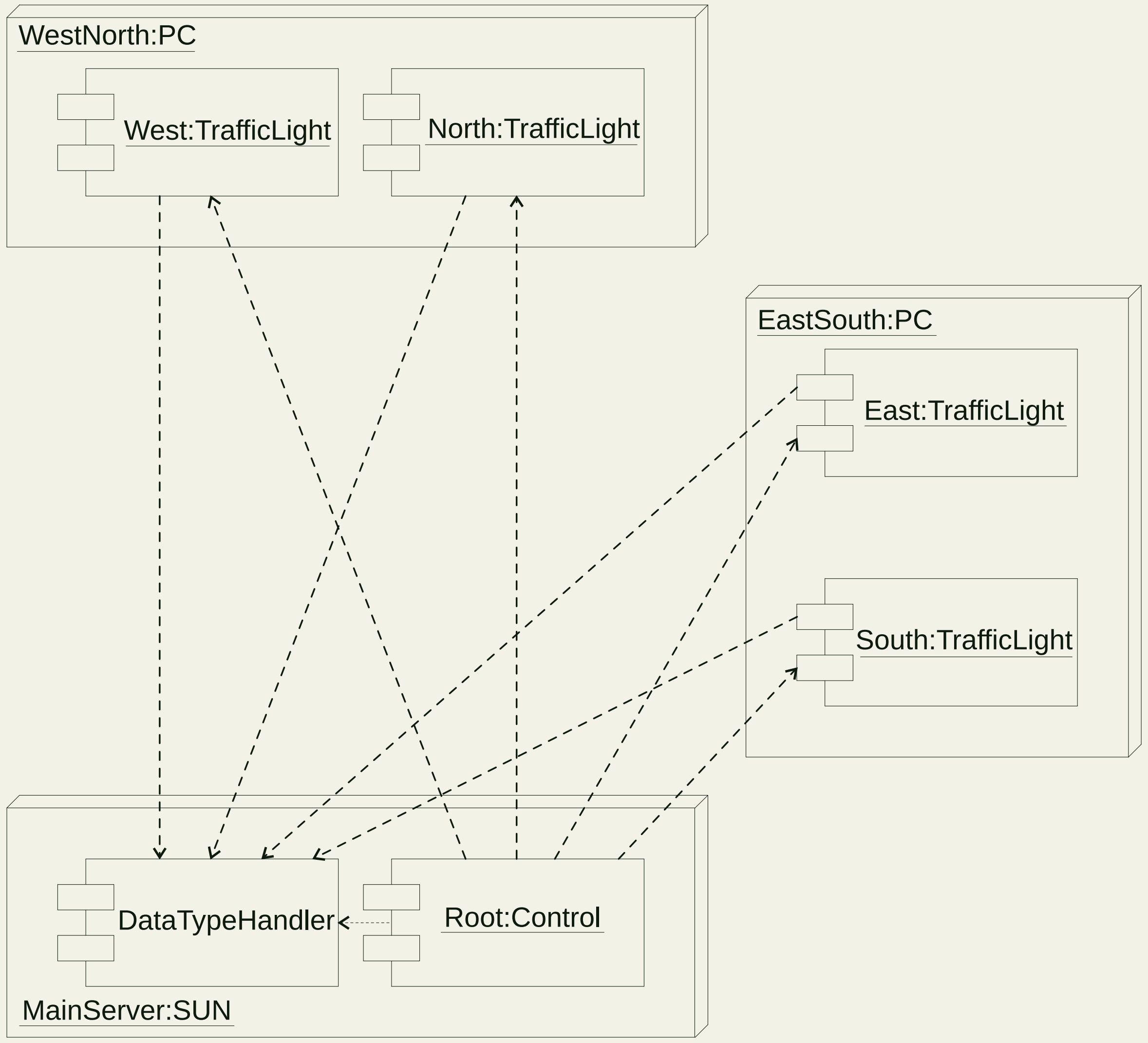
Component Diagram



Explanation 11 (Component Diagrams)

- show structure of the code itself
- show dependencies among software components, including source code components, binary code components, and executable components
- have only a type form, not an instance form
- are graphs of components connected by dependency relationships
- may also be used to show interfaces and calling dependencies among components

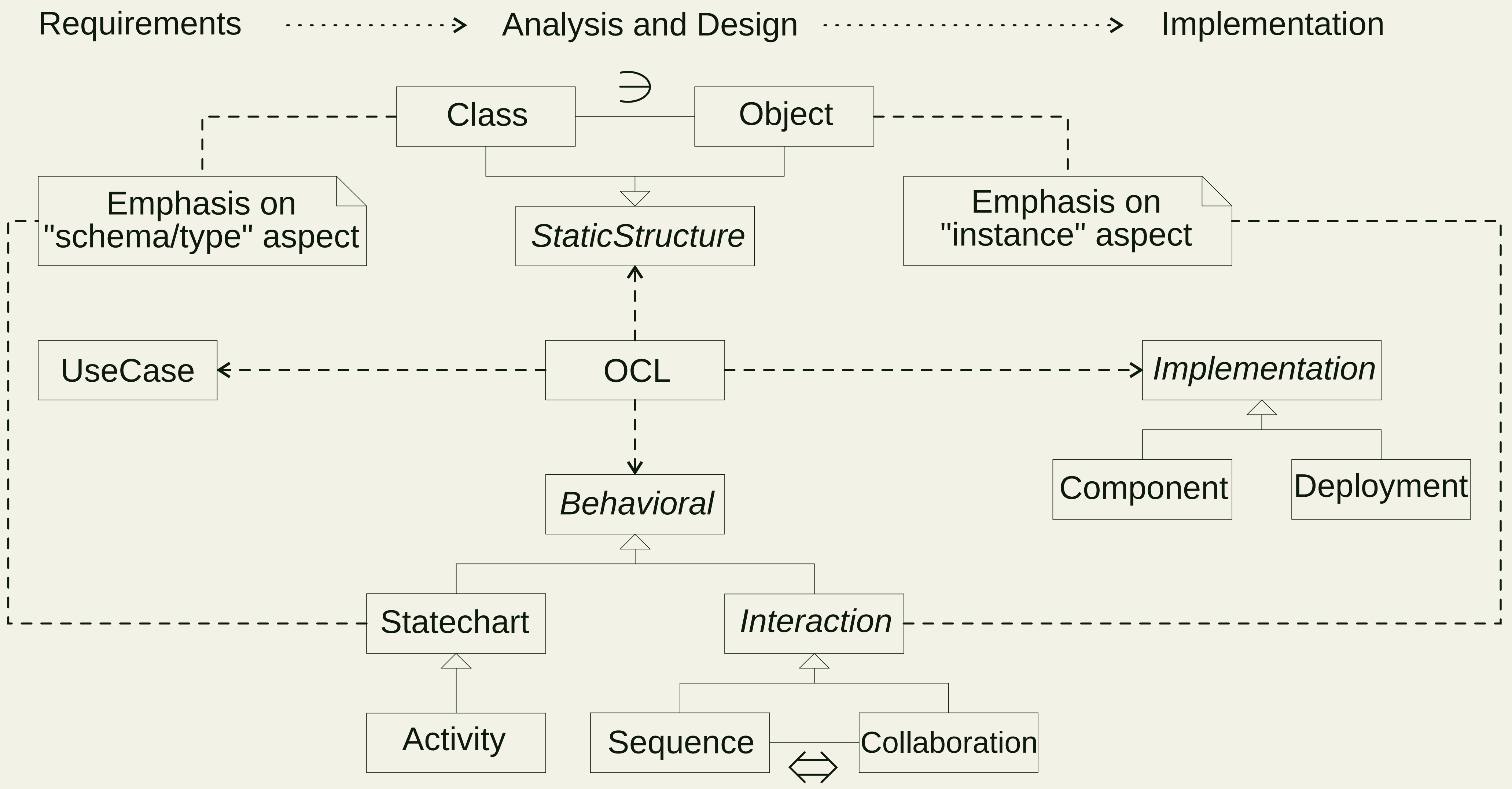
Deployment Diagram



Explanation 12 (Deployment Diagrams)

- show configuration of run-time processing elements and the software components, processes, and objects that live on them
- show structure of run-time system
- are graphs of nodes connected by communication associations

Relationship between UML Diagram Forms


Requirements
Analysis and Design
Implementation
∋
Class
Object
Emphasis on "schema/type" aspect
StaticStructure
Emphasis on "instance" aspect
UseCase
OCL
Implementation
Component
Deployment
Behavioral
Statechart
Interaction
Activity
Sequence
⇔
Collaboration