UML for the Impatient

Martin Gogolla
University of Bremen, FB 3, Computer Science Department,
Postfach 330440, D-28334 Bremen, Germany
gogolla@informatik.uni-bremen.de

Abstract

By examples we give a short introduction into the nine diagram forms provided by the Unified Modeling Language (UML). The running example we use is a small traffic light system which we first formally describe in an object specification language. Afterwards, central aspects of the specification and the specified system are visualized with UML constructs. As a conclusion, we discuss a classification of the various UML diagram forms.

1 Introduction

Developing software is supported by a large variety of object-oriented analysis and design approaches [CY90, WWW90, MO92, SM92, SGW94]. Recently, the Unified Modeling Language (UML) [BJR97c, BJR97b, BJR97a] was developed as the result of an effort in developing a single standardized language for object-oriented modeling. This language enhances and integrates concepts from three popular analysis and design methods, namely Booch [Boo91], OMT [RBP+91], and OOSE [JCJÖ92]. The aim of this paper is to give a quick overview on the UML features by explaining the different diagram forms using a running example. The view in this paper on the UML is a specification and conceptual modeling point of view in contrast to a more implementation and programming language influenced one.

The example is a small traffic light system which we formally describe in an object specification language. Central aspects of the specification and the specified system are visualized with UML constructs. The formal specification language we employ is TROLL light [GCH93, HCG94, GH95, RG97], but this paper is not on the technical content of this language. The TROLL light features we use are easy to understand. Other languages related to TROLL light are OBLOG [SSE87], Mondel [BBE+90], TROLL [JSHS91], CMSL [Wie91], and Albert [DDP93] (among many others).

The structure of the rest of this paper is as follows. In Sect. 2 we explain in detail the traffic light system we want to use. The central Sect. 3 shows examples of the nine UML diagram forms: class, object, use case, sequence, collaboration, statechart, activity, component, and deployment diagrams. The paper ends with concluding and summarizing remarks discussing a classification of the various UML diagram forms.

2 The Traffic Light System

The example system to be described is a simple street crossing as pictured in Fig. 1 where two streets (one in North-South direction, the other in West-East direction) meet and therefore the traffic has to be coordinated. The traffic lights are called North, South, West, and East. We assume the North and South traffic light always show identical signals as we do for the West and East ones. The central safety requirement is that two adjacent traffic lights, for example the West and the North light, never display identical signals. For example, it is forbidden that West displays Green and North displays Green as well.

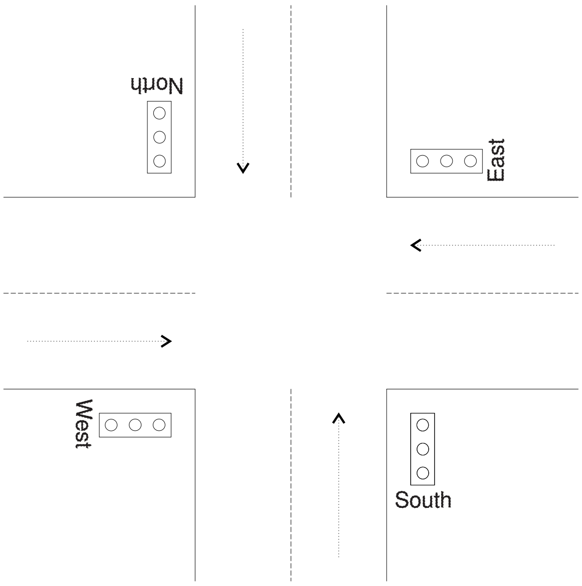


Figure 1: Traffic Lights on the Crossing

We further suppose that the traffic lights operate in an Italian style, i.e., the change of signals is done in the order Red-Green-GreenYellow with three phases. This is different from the order, for example, in Germany which is Red-RedYellow-Green-Yellow with four phases.

The TROLL light specification of the traffic light system comprises four parts: specifications for the data types `Int` and `Bool` and specifications for the object types `TrafficLight` and `Control`. Data types are assumed to be specified with a data type specification language. However, frequently used data types like the integer and boolean values are predefined and have not to be given explicitly. It remains to comment on the two remaining types, the central object types `TrafficLight` and `Control`: the specification of the object type `TrafficLight` as shown in Fig. 2 is

```
TEMPLATE TrafficLight                                            -- L01
DATA TYPES Int, Bool;                                            -- L02
ATTRIBUTES                                                       -- L03
  Phase : int;                                                   -- L04
  DERIVED Red, Yellow, Green : bool;                             -- L05
EVENTS                                                           -- L06
  BIRTH createLight(initPhase:int);                              -- L07
  switchLight;                                                   -- L08
CONSTRAINTS                                                      -- L09
  1<=Phase AND Phase<=3;                                         -- L10
VALUATION                                                        -- L11
  [createLight(initPhase)] Phase=initPhase;                      -- L12
  [switchLight] Phase=(Phase MOD 3)+1;                           -- L13
DERIVATION                                                       -- L14
  Red    = (Phase=1);                                            -- L15
  Yellow = (Phase=3);                                            -- L16
  Green  = (Phase=2 OR Phase=3);                                 -- L17
BEHAVIOR                                                         -- L18
  PROCESS TrafficLight = ( createLight -> LightLife );           -- L19
  PROCESS LightLife = ( switchLight -> LightLife );              -- L20
END TEMPLATE                                                     -- L21
```

Figure 2: Traffic Light Specification in TROLL light

designed to generate the four traffic light instances; these will be managed by exactly one object of the object type `Control` as depicted in Fig. 3; the controlling object will contain the traffic lights as its parts. The distinction between data and object types is motivated by the observation that instances of data types represent stateless values whereas instances of object types (objects) can change their state.

We now shortly explain the different syntactical clauses of a TROLL light specification following the templates (as object types or classes are called in the language) given in Figs. 2 and 3. For a deeper discussion we refer to [GCH93, HCG94, GH95, RG97].

`TEMPLATE` section: This section gives a name to the template (object type, class) being defined. Template names start with capital letters.

`DATA TYPES` section: Here, the imported data types are mentioned and their signature is made known. Thus, for example, importing `Int` means that the sort `int` together with operations like `1: -> int` and `+: int int -> int` are made known to `TrafficLight`. Sorts start with lower case letter. The difference between the data type `Int` and the data sort `int` is that `Int` comprises the sort `int` and all operations like `1` and `+` which work on the sort `int`.

`ATTRIBUTES` section: Attributes define observable properties of objects by specifying an attribute name and an attribute sort. Attributes can be data- or object-valued. They can be classified as `DERIVED`. Derived attributes are not stored

```
TEMPLATE Control                                                  -- C01
DATA TYPES Int;                                                   -- C02
TEMPLATES TrafficLight;                                           -- C03
SUBOBJECTS                                                        -- C04
  West, East, North, South : trafficLight;                        -- C05
ATTRIBUTES                                                        -- C06
  Phase : int;                                                    -- C07
EVENTS                                                            -- C08
  BIRTH createControl;                                            -- C09
  createLights;                                                   -- C10
  switchControl;                                                  -- C11
VALUATION                                                         -- C12
  [createControl] Phase=4;                                        -- C13
  [switchControl] Phase=(Phase MOD 4)+1;                          -- C14
CONSTRAINTS                                                       -- C15
  1<=Phase AND Phase<=4;                                          -- C16
  West.Phase=East.Phase;                                          -- C17
  North.Phase=South.Phase;                                        -- C18
  West.Phase<>North.Phase;                                        -- C19
  NOT(West.Green AND North.Green);                                -- C20
  NOT(West.Red AND North.Red);                                    -- C21
INTERACTION                                                       -- C22
  createLights >>                                                 -- C23
    West.createLight(3), East.createLight(3),                     -- C24
    North.createLight(1), South.createLight(1);                   -- C25
  {Phase=4 OR Phase=2} switchControl >>                           -- C26
    West.switchLight, North.switchLight;                          -- C27
  {Phase=1} switchControl >> North.switchLight;                   -- C28
  {Phase=3} switchControl >> West.switchLight;                    -- C29
  West.switchLight >> East.switchLight;                           -- C30
  North.switchLight >> South.switchLight;                         -- C31
BEHAVIOR                                                          -- C32
  PROCESS Control = ( createControl -> CreateLights );            -- C33
  PROCESS CreateLights = ( createLights -> ControlLife  );        -- C34
  PROCESS ControlLife = ( switchControl -> ControlLife );         -- C35
END TEMPLATE                                                      -- C36
```

Figure 3: Control Specification in TROLL light

explicitly, but their values are calculated from other information by means of a derivation rule.

`EVENTS` **section:** Events are state changing entities. They are described by their name and parameters. A special event in an object's life is the `BIRTH` event bringing the object into life.

`CONSTRAINTS` **section:** Constraints are employed to restrict the possible object states to the ones satisfying the given formulas. They can be considered as invariants. The set of constraints given has to necessarily to be minimal in the sense that one constraint cannot be deduced from another one. For example, Constraints `C20` and `C21` are consequences from line `C19` and the derivation rules for the attributes `Green` and `Red`.

`VALUATION` **section:** The valuation rules specify the effect the events have on attributes. They can be considered as assignments.

`DERIVATION` **section:** The derivation rules state how the value of a derived attribute is calculated from other information already present.

`BEHAVIOR` **section:** The `BEHAVIOR` section specifies the allowed life cycles (event sequences) of the objects in question by means of process definitions. An object's life starts with a process having the same name as the current template.

`TEMPLATES` **section:** Here the signature (consisting of object sorts again, starting with lower case letters, attributes, and events) of other templates, which are used in the current template, are made known.

`SUBOBJECTS` **section:** The `SUBOBJECTS` section defines the exclusive components of an object. Subobjects cannot be shared, but they can be referenced by object-valued attributes.

`INTERACTION` **section:** `INTERACTION` specifies the object's communication patterns by means of event calling rules. An interaction rule like `West.switchLight >> East.switchLight` states that whenever the event `switchLight` in the object `West` occurs, then simultaneously (in the same transition) the event `switchLight` in the object `East` also occurs. When any of the two is not allowed to occur (for example due to integrity constraint violation), the complete transition cannot take place. Comma separated lists of events on the right hand side of an interaction rules like `E >> E1, E2` are short for `E >> E1` and `E >> E2`. Preconditions express that the interaction only takes place when the precondition is satisfied.

3 UML Diagrams for the Specification

UML defines in the UML Notation Guide nine different diagram types:

- Static structure diagrams divided into (1) Class and (2) Object diagrams,
- (3) Use case diagrams,
- (4) Sequence diagrams,
- (5) Collaboration diagrams,
- (6) Statechart diagrams,

- (7) Activity diagrams, and
- Implementation diagrams divided into (8) Component and (9) Deployment diagrams.

We now shortly introduce these different diagram types by pointing out special aspects of the above object specification and the described system. With the exception of discussing first use case diagrams, we follow the order from above which is taken from the UML Notation Guide.

3.1 Use Case Diagrams

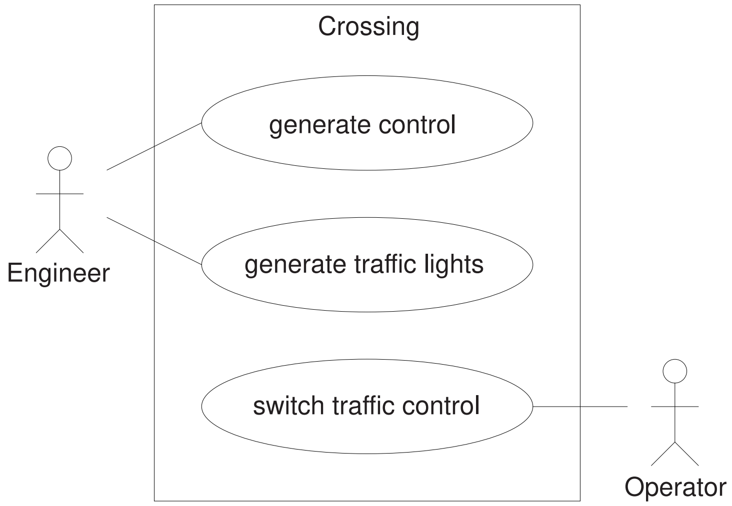


Figure 4: Use Case Diagram for Traffic Light System

A use case diagram serves to display the relationships between the actors and the use cases in a system. Use case diagrams are designed to give a rough, informal overview of the possible classes of users and the services and functionalities the system provides to them.

Actors can be visualized as “stick man” icons with the name of the actor below the icon. In Fig. 4 two different actors are shown: an engineer actor and an operator actor. Use cases are pictured as ellipses containing the name of the use case. In Fig. 4 we have three use cases: two connected with the engineer actor and one connected with the operator actor. The use cases “generate control” and “generate traffic lights” for the engineer express that the system initialization and installation of the traffic lights is done by an engineering expert. The operation of the system is symbolized by the use case “switch traffic control” done by the system operator (in our example, this system operator is an idealized and not a real actor; it could be another software unit working in the larger context of synchronizing different traffic lights). The system boundary is pictured as a rectangle enclosing the use cases and separating the actors from them. The rectangle also gives a name to the use case.

3.2 Class Diagrams

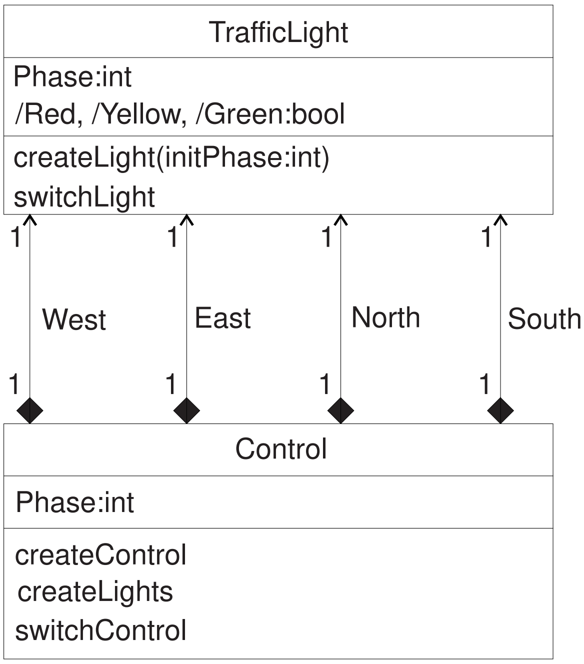


Figure 5: Class Diagram for Object Types

As the name indicates, static structure diagrams serve to represent static aspects of the system, i.e. these diagram serve to describe the structure of single system states. These diagrams are divided into class and object diagrams. The first type allows to describe states in a general form characterizing a set of allowed states, whereas the second type shows one concrete state.

Figures 5 and 6 show class diagrams. In Figs. 5 and 6 we have shown the structure, i.e. the signatures, of the data types and object types for the traffic system. Because TROLL light strictly distinguishes between data values and objects, we give two separate class diagrams. Alternatively, we could have given a single diagram for both kinds of types.

In Fig. 5 the object types (templates) `TrafficLight` and `Control` together with their attributes and operations are given. The three compartments in a class box specify the class name, the attributes of the class, and the operations of the class. The attributes are given by the attribute name and the attribute type, and the operations by their name and the type of their arguments. The slash `/` before an attribute name denotes a derived attribute. Relationships between classes are shown as connections between them. The four relationship `West`, `East`, `North`, and `South` denote the components of a `Control` object. The filled diamond indicates an unsharable component, the arrow expresses that the relationship is navigable only in the direction from `Control` objects to `TrafficLight` objects, and the numbers indicate cardinalities. Thus the diagram in Fig. 5 reflects the following sections of the TROLL light templates `TrafficLight` and `Control`.

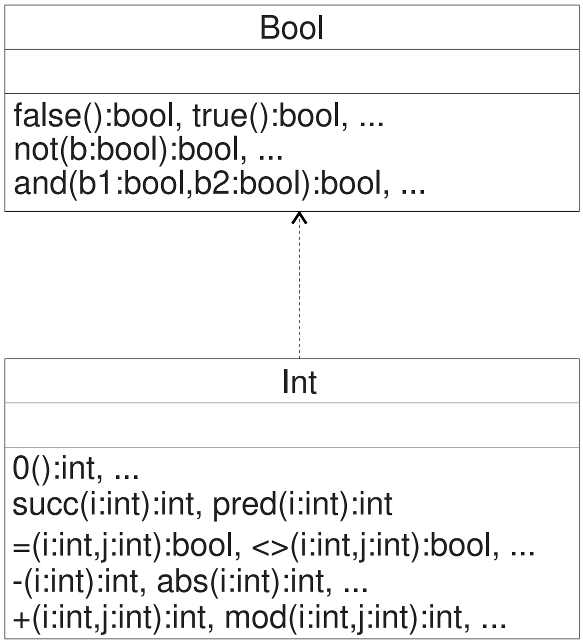


Figure 6: Class Diagram for Data Types

```
ATTRIBUTES                                                  -- L03
  Phase : int;                                              -- L04
  DERIVED Red, Yellow, Green : bool;                        -- L05
EVENTS                                                      -- L06
  BIRTH createLight(initPhase:int);                         -- L07
  switchLight;                                              -- L08

SUBOBJECTS                                                  -- C04
  West, East, North, South : trafficLight;                  -- C05
ATTRIBUTES                                                  -- C06
  Phase : int;                                              -- C07
EVENTS                                                      -- C08
  BIRTH createControl;                                      -- C09
  createLights;                                             -- C10
  switchControl;                                            -- C11
```

Alternatively to the filled diamond notation, unsharable aggregations (compositions) can also be depicted by graphical nesting of class rectangles. In Fig. 7 this is done for the four traffic lights as components of the control object. Cardinalities for the components are given in the upper right corner of the component class rectangle.

In Fig. 6 the data types `Bool` and `Int` together with their operations are given. As indicated by the empty attribute compartment, both types do not possess attributes but only operations. As indicated by the dashed arrow the class `Int` depends on the class `Bool`: it uses some operations of `Bool` to implement some of its own operations.

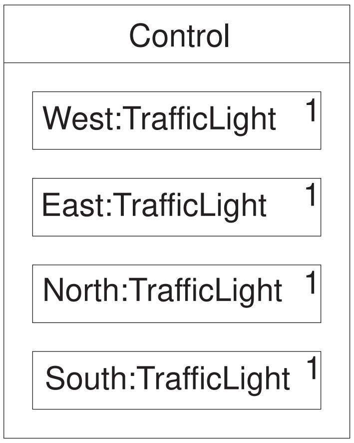


Figure 7: Class Diagram with Graphical Nesting

3.3 Object Diagrams

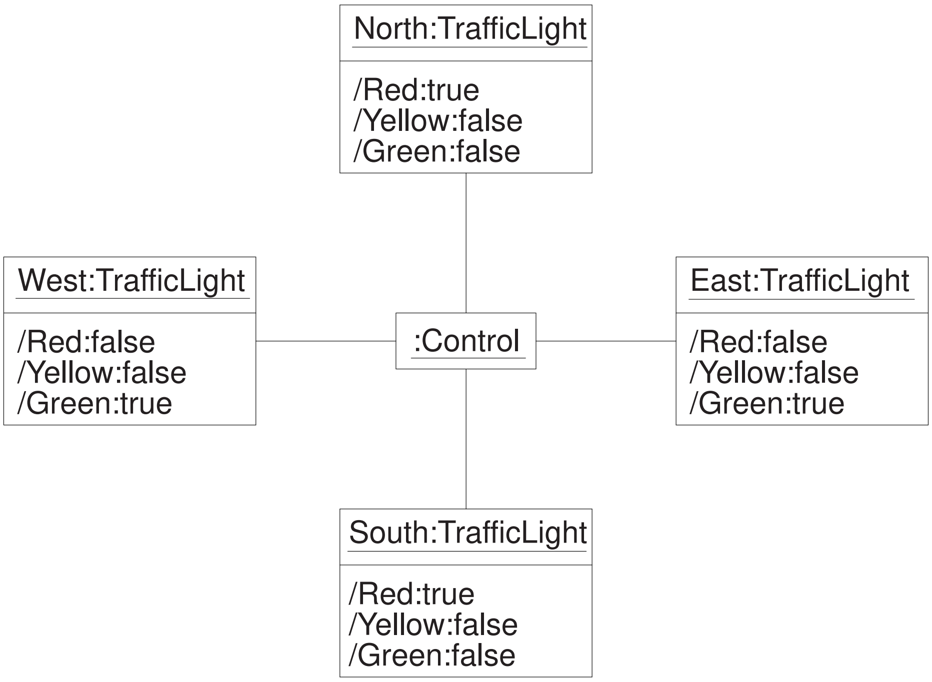


Figure 8: Object Diagram

Let us now consider the second form of static structure diagrams. The diagram in Fig. 8 shows a system state as an object diagram. The state displayed allows cars in West-East direction to pass their traffic light while the ones in North-South direction have to wait. Object instances are labeled by an optional name and the type of the object. In order to better distinguish between types and instances, instance labels are underlined. The diagram can be considered as the formal counterpart of Fig. 1. We have decided to show only some of the important attributes, namely the colors of the traffic lights. Alternatively, we could have given all attributes and their corresponding values.

Static structure diagrams show in the first place signatures and some constraints restricting the possible systems states and denoted in a graphical way, for example, the cardinality restrictions. With respect to TROLL light, the `Attributes`, `Subobjects`, and `Events` sections of a template are given.

3.4 Sequence Diagrams

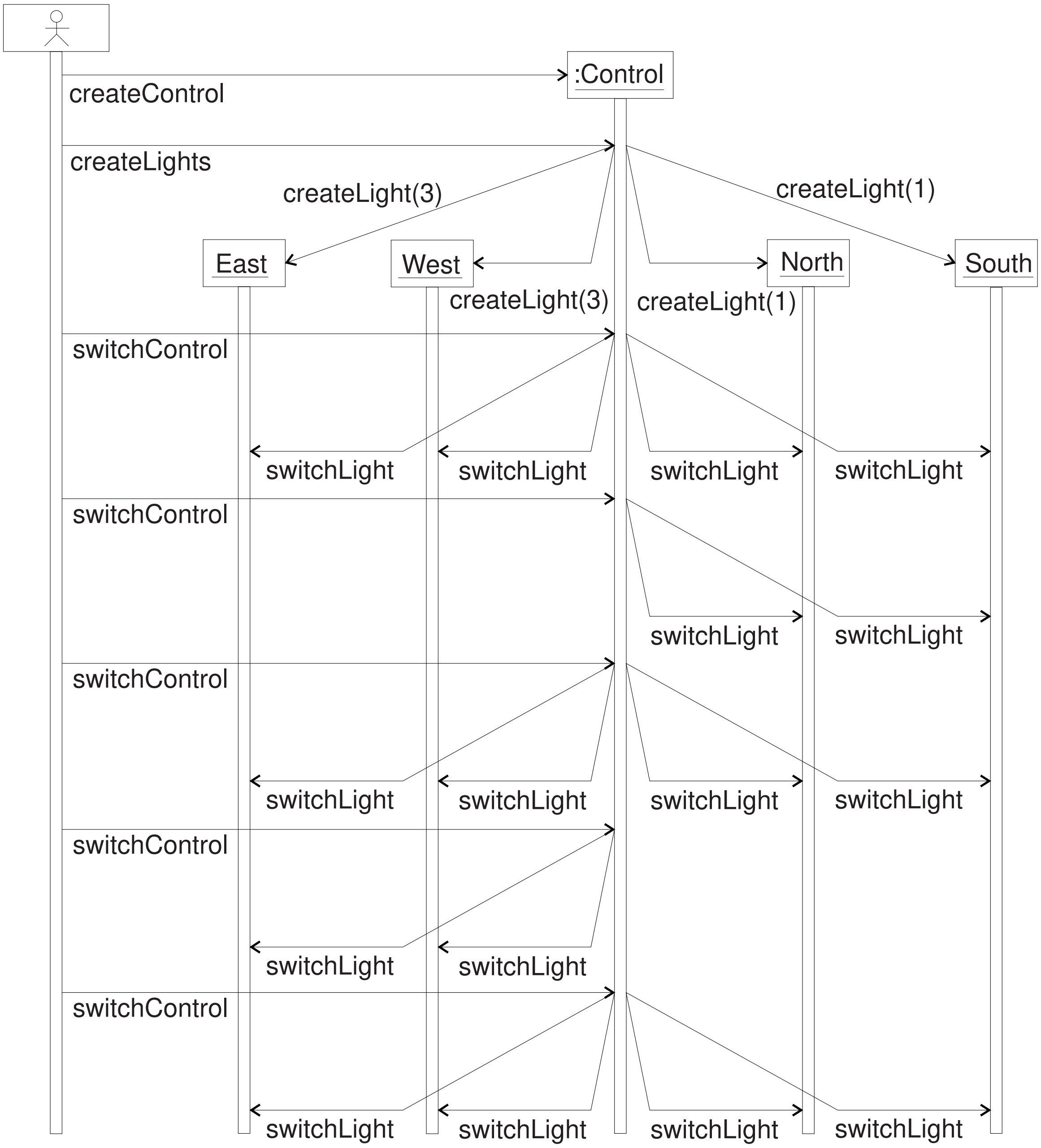


Figure 9: Sequence Diagram

A sequence diagram pictures interaction among objects. It shows the participating objects together with a lifeline symbolizing life cycles or parts of them. The messages the objects exchange are ordered on the lifeline with respect to their occurrence in time.

In Fig. 9 a snapshot of a typical life cycle of the described traffic light system is shown. The objects involved are a user object, one instance of the `Control` object, and the four traffic lights. Object lifelines are pictured as long, narrow rectangles below wider object rectangles including the object names. Events (or messages) together with their arguments are notated close to arrows going from the sender to the receiver. Birth events creating objects touch the wider object rectangles thus symbolizing that objects enter the scene. Usually, the triggering and the triggered event are shown on the same level on the lifeline. But because of the complex interaction mechanism in the example this would be confusing. Therefore we have taken the freedom to show concurrent events a bit below their triggering events (it is unclear how to visualize in sequence diagrams that one event simultaneously triggers e.g. three other events). For example, the second from above `switchControl` event triggers the `switchLight` event in the `North` and `South` traffic lights. These arrows are the graphical counterpart of lines `C28` and `C31` in the TROLL light specification. These three events are supposed to occur simultaneously. This sequence diagram symbolizes the `Interaction` axioms of the TROLL light specification. The triggered events are especially determined by the preconditions of these axioms.

```
INTERACTION                                                 -- C22
  createLights >>                                           -- C23
    West.createLight(3), East.createLight(3),               -- C24
    North.createLight(1), South.createLight(1);             -- C25
  {Phase=4 OR Phase=2} switchControl >>                     -- C26
    West.switchLight, North.switchLight;                    -- C27
  {Phase=1} switchControl >> North.switchLight;             -- C28
  {Phase=3} switchControl >> West.switchLight;              -- C29
  West.switchLight >> East.switchLight;                     -- C30
  North.switchLight >> South.switchLight;                   -- C31
```

3.5 Collaboration Diagrams

In principle, a collaboration diagram shows the same information as a sequence diagram but it emphasizes the objects and their links not the behavior in time. Collaboration diagrams extend object diagrams. Recall object diagrams visualize class instances (objects) and their concrete relationships (links). Interaction is displayed in collaboration diagrams on the links by stating the event sent (the message sent). In addition, arrows identify sender and receiver objects and a numbering system captures the sequences the events take (the event sequences were captured by the lifelines in sequence diagrams).

The collaboration diagram in Fig. 10 shows the same events as the sequence diagram

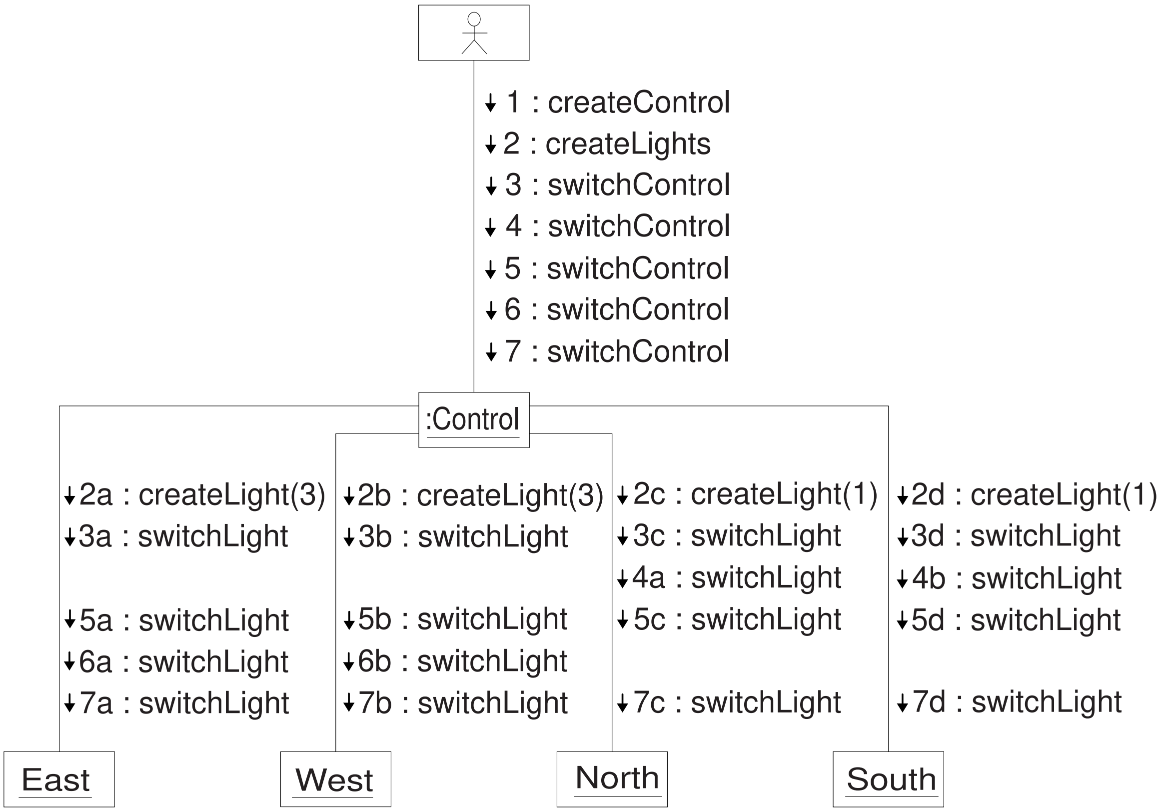


Figure 10: Collaboration Diagram

in Fig. 9. Thus the diagram deals with the same objects: the user object, the `Control` object, and the four traffic lights. Sets of events belonging together are grouped by the similar event numbers on the links. For example, the second from above `switchControl` event in Fig. 9 together with the called events `switchLight` in `North` and `switchLight` in `South` is represented in the collaboration diagram by (1) the label `4` on the link between the user object and `:Control` object, (2) the label `4a` on the link between the `:Control` object and the `North` traffic light, and (3) the label `4b` on the link between the `:Control` object and the `South` traffic light. The number `4` indicates that this is the forth event in the considered sequence, and `4a` and `4b` indicate that these events are "children" of the `4` event, and the letters show that they occur concurrently (in contrast to numerical labels which stand for sequential occurrences).

3.6 Statechart Diagrams

A statechart diagram reflects the state sequences occurring in an object or in an interaction during its life. State changes occur in response to received stimuli. The state diagram can also show responses of the object.

Figure 11 directly shows the behavior patterns of the TROLL light specifications represented as state machines. The allowed sequence of events for `TrafficLight`

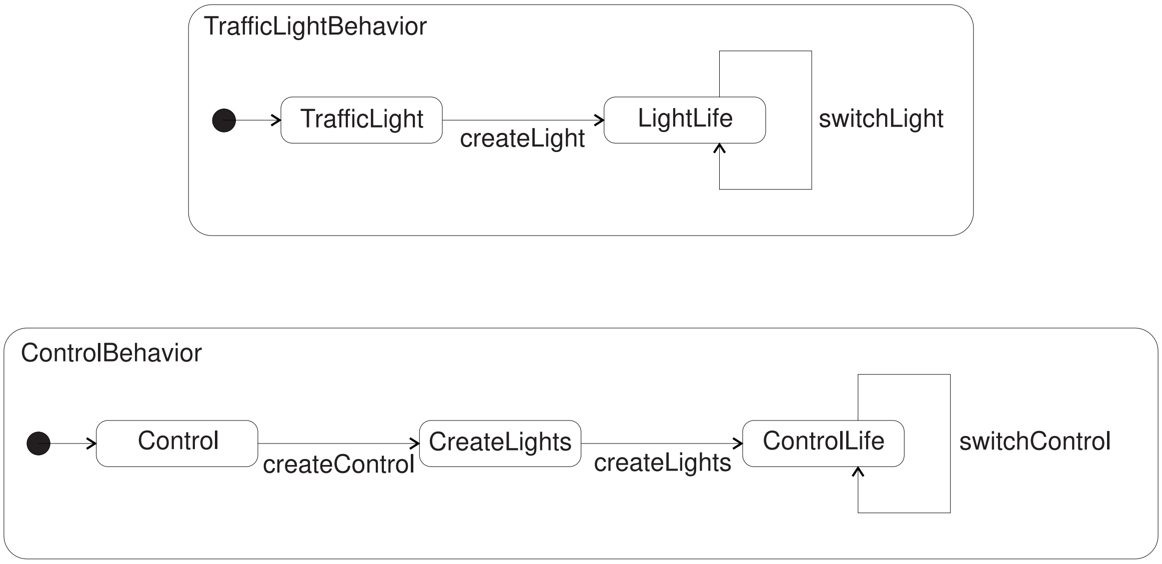


Figure 11: Statechart diagram for `BEHAVIOR` sections

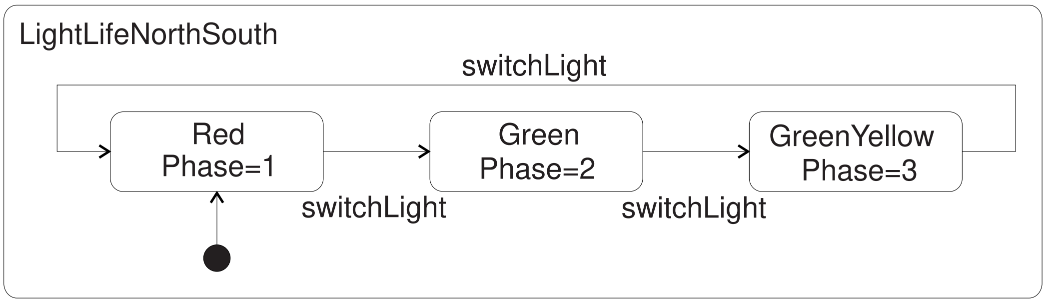
and `Control` are visualized. For example, in the template `Control` a life cycle starts with one `createControl` event followed by one `createLights` event. Afterwards an arbitrary number of `switchControl` events are allowed.

```
BEHAVIOR                                                  -- L18
  PROCESS TrafficLight = ( createLight -> LightLife );    -- L19
  PROCESS LightLife = ( switchLight -> LightLife );       -- L20

BEHAVIOR                                                  -- C32
  PROCESS Control = ( createControl -> CreateLights );    -- C33
  PROCESS CreateLights = ( createLights -> ControlLife  ); -- C34
  PROCESS ControlLife = ( switchControl -> ControlLife ); -- C35
```

The statechart diagrams in Figs. 12, 13 and 14 also characterize allowed state sequences. However, they do not correspond directly to pieces of code as the state machines in Fig. 11. Figures 12 and 13 can be considered as a refinement of state `LightLife` of Fig. 11, and Fig. 14 makes state `ControlLife` of Fig. 11 more concrete.

The refinement of state `LightLife` given in Fig. 13 has states `Red/Phase=1`, `Green/Phase=2`, and `GreenYellow/Phase=3`. The chosen state names reflect the colors the traffic light shows and indicate the value of the attribute `Phase`. This value is modified by the `VALUATION` axioms for the event `switchLight` in the TROLL light specification. After entering the initial state denoted by the filled circle (in UML terminology a pseudo-state), the next state is determined by the decision for which object the state diagram is used for. For the `North` and `South` traffic lights the “proper” initial state is `Red/Phase=1`, for the `West` and `East` traffic lights it is `GreenYellow/Phase=3`. Figure 13 can be considered as an abstraction of the two statechart in Fig. 12 where separate diagrams for the `North` and `South` traffic light



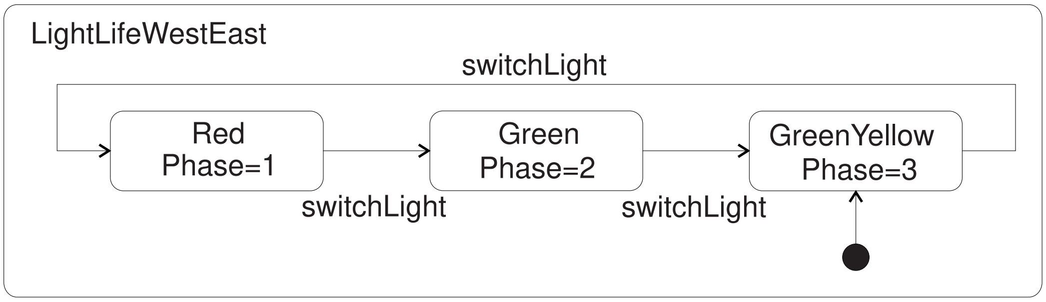


Figure 12: Statechart diagrams for `LightLife`

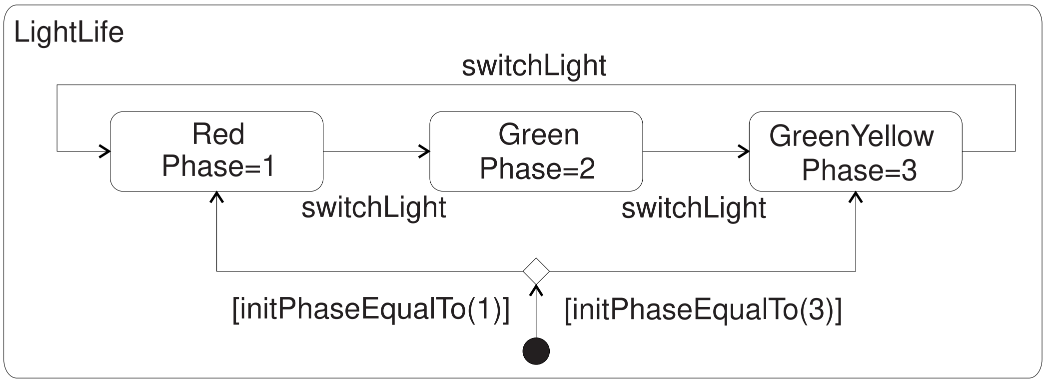


Figure 13: Alternative Statechart diagram for `LightLife`

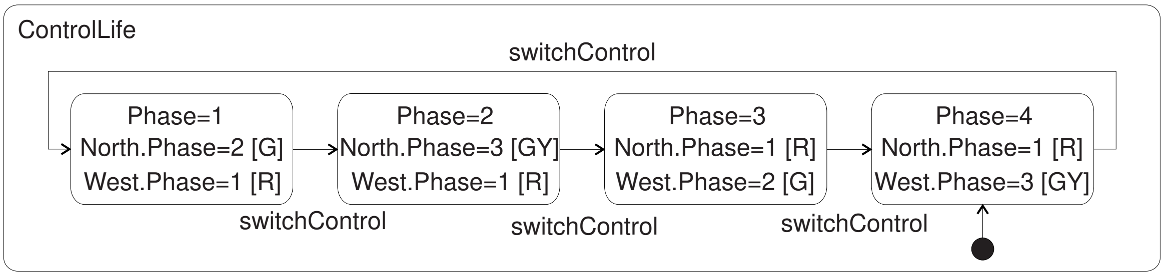


Figure 14: Statechart diagram for `ControlLife`

and the `West` and `East` traffic lights are pictured. Analogously to the refinement of state `LightLife`, the statechart diagram in Fig. 14 shows the internal structure of the `Control` object in dependency from the `switchControl` event.

```
VALUATION                                              -- L11
  [createLight(initPhase)] Phase=initPhase;            -- L12
  [switchLight] Phase=(Phase MOD 3)+1;                 -- L13

VALUATION                                              -- C12
  [createControl] Phase=4;                             -- C13
  [switchControl] Phase=(Phase MOD 4)+1;               -- C14
```

The diagram in Fig. 15 repeats three statecharts shown before. Additionally however, messages sent from one object to another object are indicated by dashed arrows. Thus, for example, the transition in `ControlLife` between states `Phase=2` and `Phase=3` induces transitions (A) in `LightLifeNorthSouth` between states `Phase=3` and `Phase=1` and (B) in `LightLifeWestEast` between states `Phase=1` and `Phase=2`. The messages sent correspond to the `INTERACTION` patterns of the TROLL light specification.

```
INTERACTION                                            -- C22
  ...
  {Phase=4 OR Phase=2} switchControl >>                -- C26
    West.switchLight, North.switchLight;               -- C27
  {Phase=1} switchControl >> North.switchLight;        -- C28
  {Phase=3} switchControl >> West.switchLight;         -- C29
  West.switchLight >> East.switchLight;                -- C30
  North.switchLight >> South.switchLight;              -- C31
```

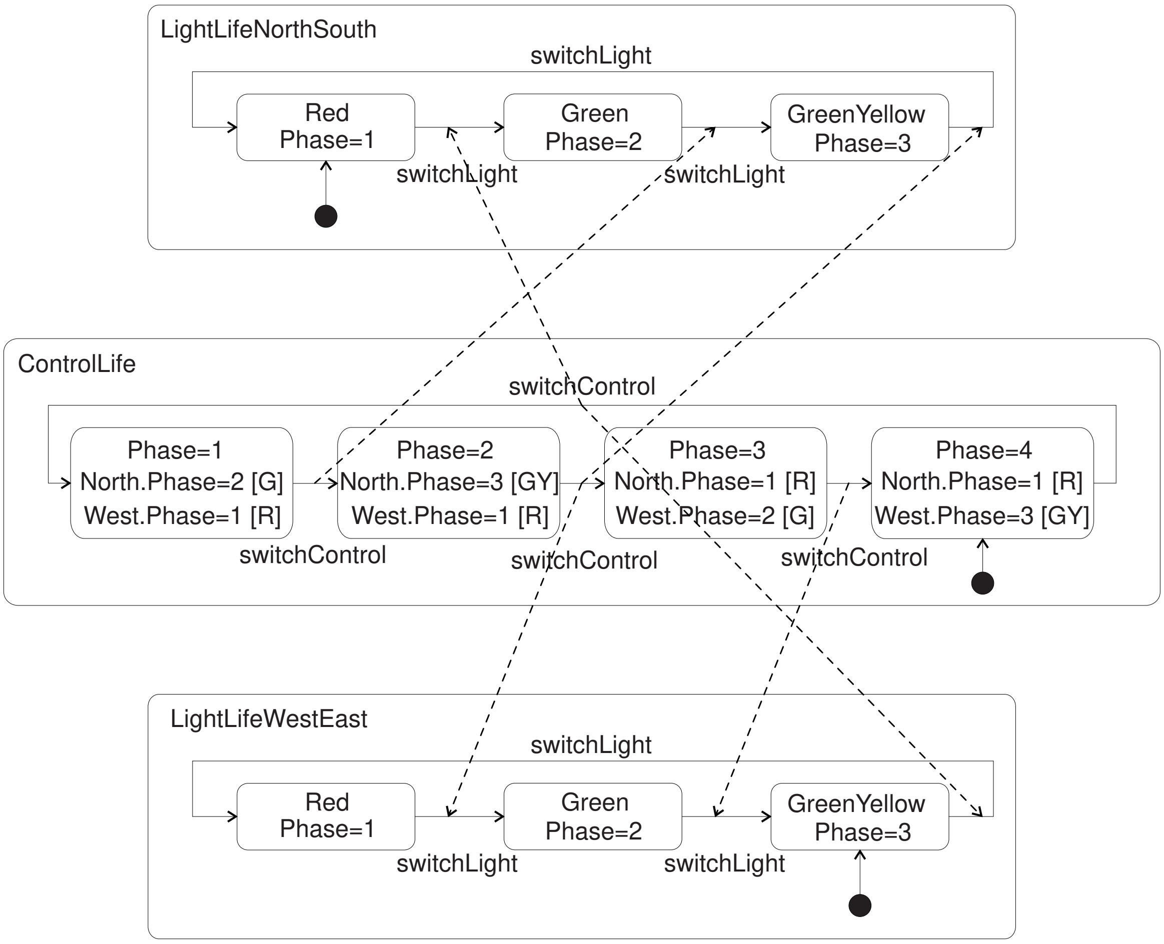


Figure 15: Statechart Diagrams with Messages Sent

3.7 Activity Diagrams

Typically, an activity diagram shows action states, i.e. states representing the execution of an atomic action. Most of the transitions are triggered by completion of actions in the source state. The purpose of this diagram form is to focus on flows driven by internal processes. Activity diagrams have a strong relationship to classical data flow diagrams.

Figure 16 shows an activity diagram for the first valuation axiom (`L12`) of `TrafficLight`. The action states are pictured as rounded rectangles. The actions taken represent activities needed for the evaluation of the axiom in order to modify the attributes `Phase`. We have grouped the diagram into so-called "swimlanes" to emphasize the participating objects. Thus in addition to the traffic light and the control object there is an object `Temporary` responsible for the evaluation of the expression on the right hand side of the valuation axiom. The actions consume or read objects along ingoing edges and produce or modify objects along outgoing edges.

Another part of TROLL light where activity diagrams can be used to represent

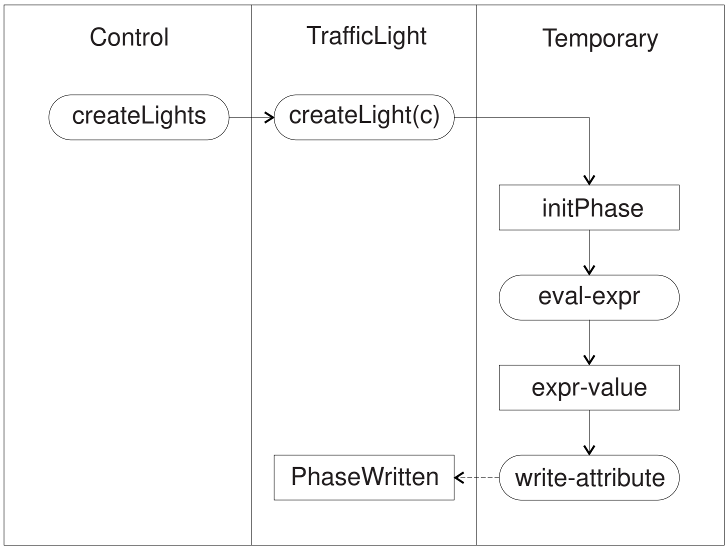


Figure 16: Activity Diagram for Valuation and Triggering of `Phase=initPhase`

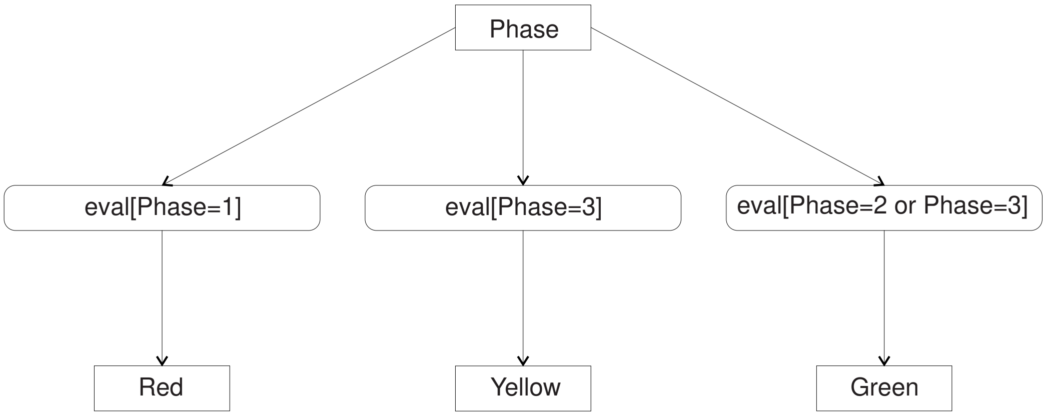


Figure 17: Activity Diagram for Derivation Rules

pieces of code are the `DERIVATION` axioms. In Fig. 17 we have pictured the respective axioms of the `TrafficLight` template. Roughly speaking, we have shown an action state for each derivation rule. The diagram captures the dependencies between the rules and attributes in the evaluation process.

```
DERIVATION                                                    -- L14
  Red    = (Phase=1);                                         -- L15
  Yellow = (Phase=3);                                         -- L16
  Green  = (Phase=2 OR Phase=3);                              -- L17
```

3.8 Component Diagrams

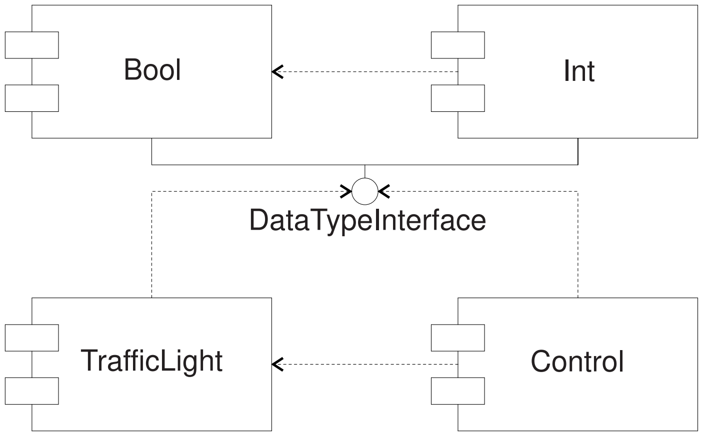


Figure 18: Component Diagram for Traffic Light System

Implementation diagrams are designed to represent implementation aspects. They are divided into component and deployment diagrams. Component diagrams point out the structure of the code itself. Deployment diagrams visualize the structure of the run-time system.

Figure 18 reflects the overall structure of the TROLL light specification. The dashed arrows indicated dependencies as given for example in the TROLL light `DATA TYPES` and `TEMPLATES` sections. Because we have a strict separation between data types and templates in TROLL light, the data type layer is represented by one UML interface (displayed by a small circle) called `DataTypeInterface`.

3.9 Deployment Diagrams

The second form of an implementation diagram is the deployment diagram. The deployment diagram in Fig. 19 shows a possible structure for a running traffic light control system. The contents of this diagram is not reflected by the TROLL light specification at all, because the language is intended to be a specification and not an implementation language. The diagram describes that the running system works on three resources (machines) of different nature. The participating objects are distributed over these machines.

4 Summary and Conclusion

As a summary we want to show a diagram and a table concentrating on central aspects of UML.

In Fig. 20 we have pictured the various relationships between UML diagram forms. We have used the UML class diagram notation (with additional adornments or stereo-

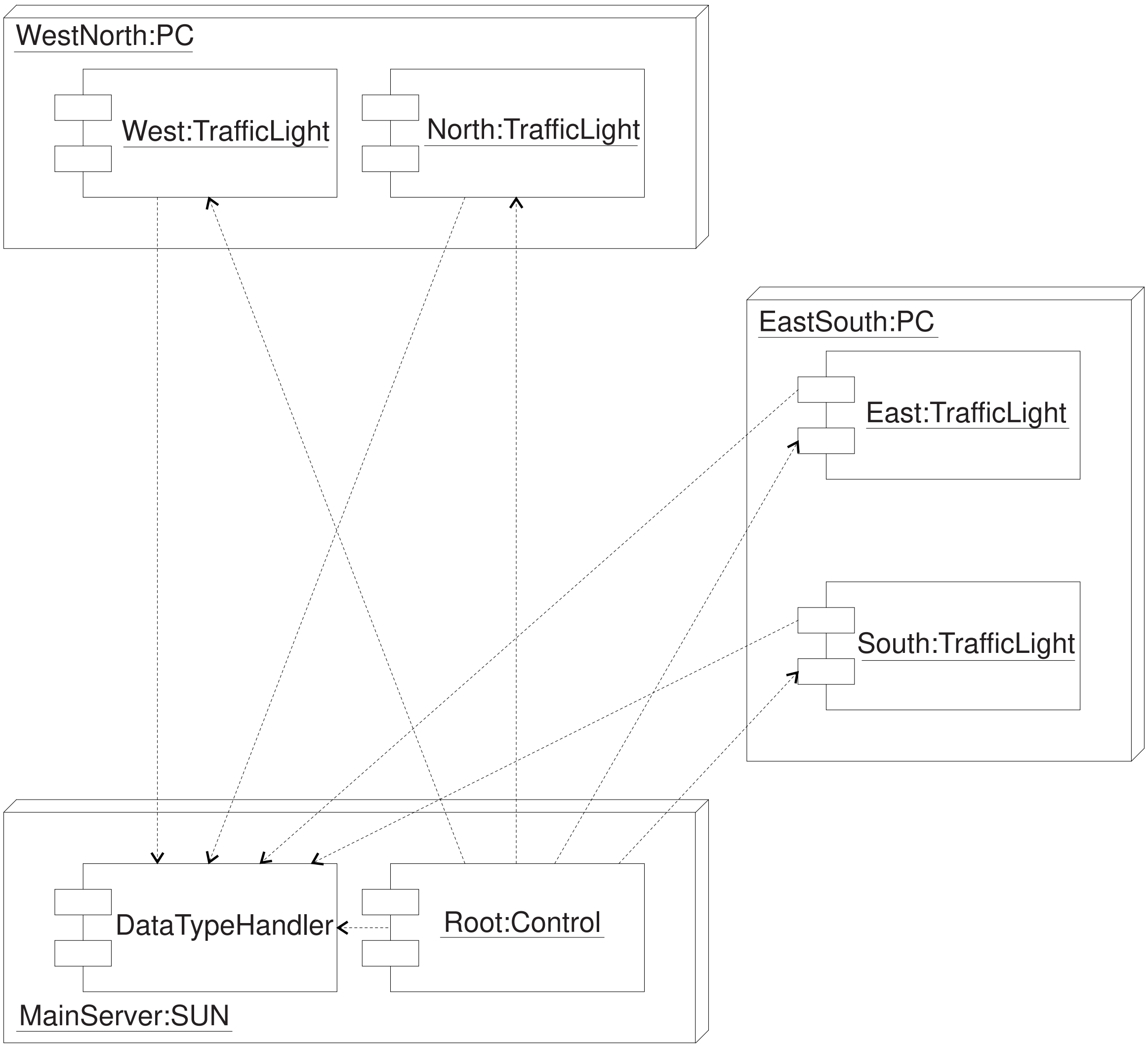


Figure 19: Deployment Diagram for Traffic Light System

types as they would be called in UML). We have mapped the diagram forms to different phases of a very simple, idealized software development process (at the bottom of the diagram), emphasizing the main initial usage of the respective diagram but not excluding use of the diagram in other steps of the process.

At the top of the generalization hierarchies, we identify `UseCase`, `StaticStructure`, `Behavioral`, and `Implementation` diagrams. `StaticStructure` diagrams are specialized to `Class` and `Object` diagrams with a special association between them denoting that `Object` diagrams represent instances for `Class` diagrams (as indicated by the mirrored element sign). `Behavioral` diagrams specialize to `Sequence`, `Interaction`, and `Statechart` diagrams where `Statechart` diagrams are further specialized to `Activity` diagrams and the special association between `Sequence` and `Interaction` diagrams denotes that both diagram forms are capable of expressing equivalent content. Last, `Implementation` diagrams are specialized to `Component` and `Deployment` diagrams. The three classes which have their names shown in italics represent abstract

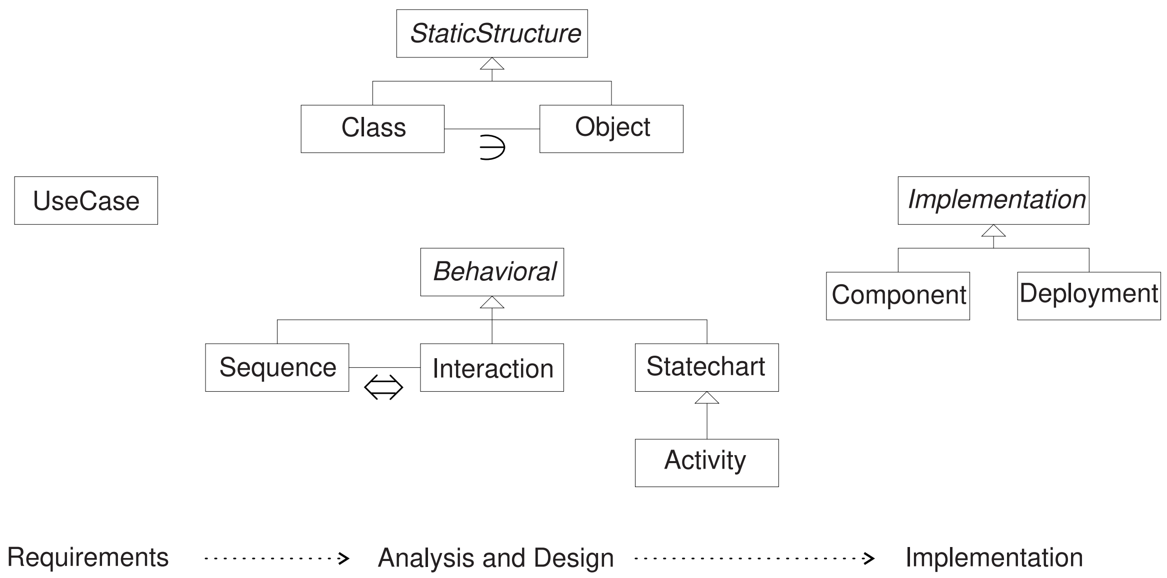


Figure 20: Relationships between UML Diagram Forms

	Cla	Obj	Use	Seq	Col	Sta	Act	Com	Dep
Instance		+	+	+	+	+	+	+	+
Generic	+		+	+	+	+	+	+	
Static	+	+							
Behavioral			+	+	+	+	+		
Passive State	+	+				+			
Active State						+	+		
Transition				+	+	+	+		
External			+						
Internal	+	+		+	+	+	+	+	+
Requirements			+						
Design	+	+		+	+	+	+		
Implementation								+	+
Time				+					
Space					+				
Compile	+	+		+	+	+	+	+	
Run									+

Figure 21: Relationship between Diagram Forms and Selected Criteria

classes which cannot be instanciated directly but only through one of their subclasses.

The matrix in Fig. 21 expresses whether a diagram applies to a fixed criterion (speaking in UML terms, the matrix is an object diagram showing links for a class diagram having two classes `Criterion` and `DiagramForm` and one association `AppliesTo`). The plus + indicates that the diagram is capable of making a statement about the given criterion. The criteria try to answer the following questions.

- Does the diagram describe things in instance form or generic form?
- Does the diagram emphasize static or behavioral aspects?
- Does the diagram apply to passive states, active states or transitions?
- Does the diagram support an external or an internal view on the system?
- In which development phase is the diagram mainly applied?
- Does the diagram emphasize time (sequences) or space (links)?
- Does the diagram apply to compile time or run time?

We have attempted to give introductory examples for and an overview on the various UML diagram forms. Of course there is much more to the UML story than we have shown. Probably, more questions about UML now arise than before. It remains to refer again to the original UML definition [BJR97c, BJR97b, BJR97a].

Acknowledgment

Thanks to Mark Richters for commenting on a draft version of this paper.

References

[BBE+90] G. v. Bochmann, M. Barbeau, M. Erradi, L. Lecomte, P. Mondain-Monval, and N. Williams. Mondel: An Object-Oriented Specification Language. Publication 748, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, 1990.

[BJR97a] Grady Booch, Ivar Jacobson, and James Rumbaugh, editors. *Object Constraint Language (Version 1.1)*. Rational Corporation, Santa Clara, 1997. `http://www.rational.com`.

[BJR97b] Grady Booch, Ivar Jacobson, and James Rumbaugh, editors. *UML Notation Guide (Version 1.1)*. Rational Corporation, Santa Clara, 1997. `http://www.rational.com`.

[BJR97c] Grady Booch, Ivar Jacobson, and James Rumbaugh, editors. *UML Semantics (Version 1.1).* Rational Corporation, Santa Clara, 1997. http://www.rational.com.

[Boo91] G. Booch. *Object-Oriented Design with Application.* Benjamin-Cummings, 1991.

[CY90] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Yourdon Press, Prentice Hall, 1990.

[DDP93] E. Dubois, P. Du Bois, and M. Petit. O-O Requirements Analysis: an Agent Perspective. In O.M. Nierstrasz, editor, *ECOOP'93 — Object-Oriented Programming*, pages 458–481. Springer, Berlin, LNCS 707, 1993.

[GCH93] M. Gogolla, S. Conrad, and R. Herzig. Sketching Concepts and Computational Model of TROLL light. In A. Miola, editor, *Proc. 3rd Int. Conf. Design and Implementation of Symbolic Computation Systems (DISCO)*, pages 17–32. Springer, Berlin, LNCS 722, 1993.

[GH95] Martin Gogolla and Rudolf Herzig. An Algebraic Semantics for the Object Specification Language TROLL light. In Egidio Astesiano, Gianna Reggio, and Andrzej Tarlecki, editors, *Proc. 10th Int. Workshop Abstract Data Types (WADT'94)*, pages 288–304. Springer, Berlin, LNCS 906, 1995.

[HCG94] R. Herzig, S. Conrad, and M. Gogolla. Compositional Description of Object Communities with TROLL light. In C. Chrisment, editor, *Proc. Basque Int. Workshop on Information Technology (BIWIT'94): Information Systems Design and Hypermedia*, pages 183–194. Cépaduès-Éditions, Toulouse, 1994.

[JCJÖ92] I. Jacobson, M. Christerson, P. Johnsson, and G. Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Prentice-Hall, 1992.

[JSHS91] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. Object-Oriented Specification of Information Systems: The TROLL Language. Informatik-Bericht 91-04, TU Braunschweig, 1991.

[MO92] J. Martin and J.J. Odell. *Object-Oriented Analysis and Design.* Prentice-Hall, Englewood Cliffs (NJ), 1992.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs (NJ), 1991.

[RG97] Mark Richters and Martin Gogolla. A Web-based Animator for Validating Object Specifications. In Bipin C. Desai and Barry Eaglestone, editors, *Proc. Int. Database Engineering and Applications Symposium (IDEAS'97)*, pages 211–219. IEEE, Los Alamitos, 1997.

[SGW94] Bran Selic, Garth Gullekson, and Paul T. Ward. *Real-Time Object-Oriented Modeling.* John Wiley Sons, 1994.

[SM92] S. Shlaer and S.J. Mellor. *Object Life Cycles: Modeling the World in States*. Prentice-Hall, 1992.

[SSE87] A. Sernadas, C. Sernadas, and H.-D. Ehrich. Object-Oriented Specification of Databases: An Algebraic Approach. In P.M. Stoecker and W. Kent, editors, *Proc. 13th Int. Conf. on Very Large Databases VLDB'87*, pages 107–116. VLDB Endowment Press, Saratoga (CA), 1987.

[Wie91] R.J. Wieringa. A Conceptual Model Specification Language (CMSL, Version 2). Technical Report IR-248, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1991.

[WWW90] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice-Hall, 1990.