<<UML>>2000 Workshop Dynamic Behaviour in UML Models,
Technical Report, Technical University of Munich, G. Reggio,
A. Knapp, B. Rumpe, B. Selic, R. Wieringa (Eds.), 2000.

Analysing Atomic Dynamic UML Notions by Surfing through the UML Metamodel

Martin Gogolla, Oliver Radfelder, Ralf Kollmann, Mark Richters
University of Bremen, Computer Science Department

Abstract. This paper analyses atomic notions in UML which are fundamental for the understanding of dynamic aspects. The notions considered are: Action, Event, Exception, Message, Method, Signal, Stimulus, Operation, and Reception. We surf through the UML metamodel by combining the different metamodel class diagrams, where these notions are defined, into a single class diagram. Thereby we point out the intent, similarities and differences between these notions. Thus before doing a formalization, we try to make the concepts a bit clearer than they appear in the UML Semantics [OMG99].

1 Motivation

The class diagram in Fig. 1 combines aspects of six different UML metamodel class diagrams and concentrates on the following notions: Action, Event, Exception, Message, Method, Signal, Stimulus, Operation, and Reception. In the UML Semantics these notions are distributed over Figs. 2-5 (Operation, Method), 2-14 (Signal, Exception, Reception), 2-15 (Action), 2-16 (Stimulus), 2-17 (Message), and 2-22 (Event). These notions constitute atomic building blocks in the UML behavior diagrams, in particular statechart, activity, sequence, and collaboration diagrams. The chosen notions are closely related because they all refer to atomic entities which are involved in fundamental system state changes. Other more involved entities use these notions to build more complex system changes, for example, a transition may use an event and an action. In order to explain the class diagram, we now shortly repeat the explanations given in the UML Semantics for the metaclasses under consideration.

Action: An *action* is a specification of an executable statement that forms an abstraction of a computational procedure that results in a change in the state of the model, and can be realized by sending a *message* to an object or modifying a link or a value of an attribute.

Event: An *event* is a specification of a type of observable occurrence. The occurrence that generates an *event* instance is assumed to take place at an instant in time with no duration.

Exception: An *exception* is a *signal* raised by behavioral features typically in case of execution faults.

Message: A *message* defines a particular communication between instances that is specified in an interaction.

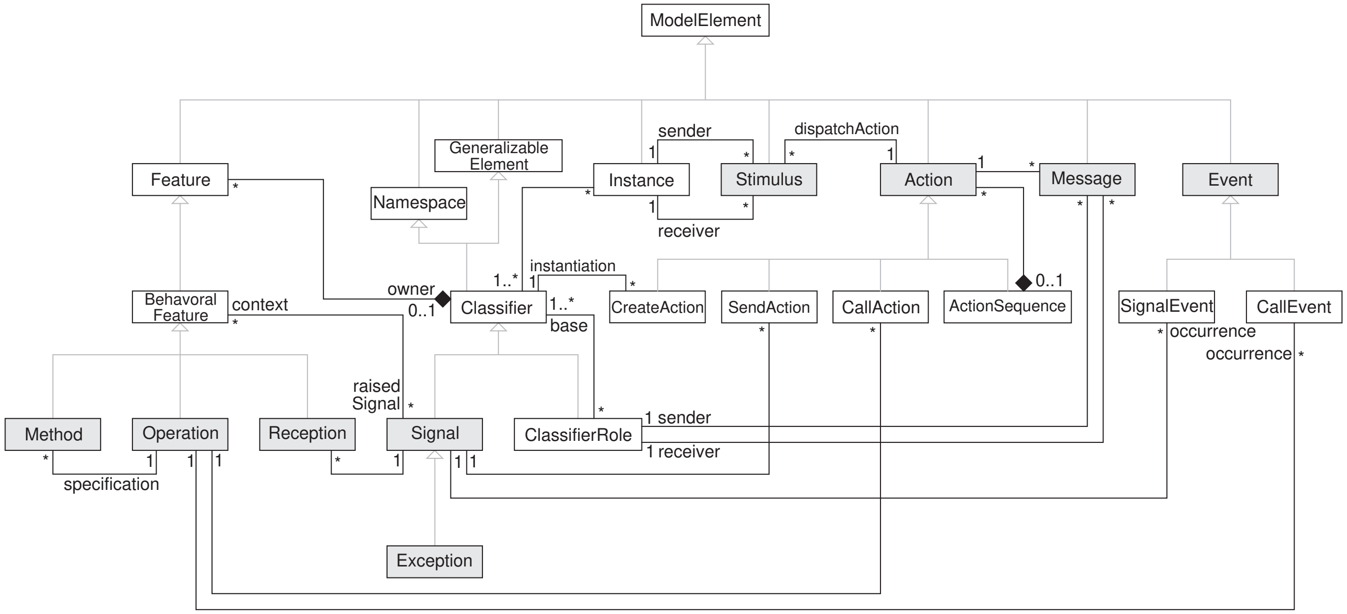

Fig. 1. UML Metamodel for Atomic Dynamic Notions

Method: A *method* is the implementation of an *operation*. It specifies the algorithm or procedure that effects the results of an operation.

Operation: An *operation* is a service that can be requested from an object to effect behavior. An operation has a signature, which describes the actual parameters that are possible (including possible return values).

Reception: A *reception* is a declaration stating that a classifier is prepared to react to the receipt of a *signal*. The *reception* designates a *signal* and specifies the expected behavioral response. A *reception* is a summary of expected behavior. The details of handling a *signal* are specified by a state machine.

Signal: A *signal* is a specification of an asynchronous *stimulus* communicated between instances. The receiving instance handles the signal by a state machine. *Signal* is a generalizable element and is defined independently of the classes handling the signal. A *reception* is a declaration that a class handles a *signal*, but the actual handling is specified by a state machine.

Stimulus: A *stimulus* reifies a communication between two instances.

2 Discussion

We now show the intent of these metaclasses by giving two examples in Fig. 2 and 3. In the upper left part of Fig. 2 four classes and a signal are declared, whereas in the upper right part an excerpt from a statechart diagram for one of the classes is shown. The lower part displays how aspects of this are represented in the UML metamodel as an object diagram. Thus we have example instantiations for the metaclasses Operation, Signal, Reception, and Event. Note that some few associations unimportant in this context are not shown, and we do not neccessarily display the 'lowest' class of the objects nor underline object names.

In the upper part of Fig. 3 we have an excerpt from a collaboration diagram with one operation call. This situation is represented in the UML metamodel as indicated in the lower part of the figure. There we have instantiations for the metaclasses Operation, Method, Message, Action, and Stimulus. On the right side of the dashed line the Stimuli with sender and receiver instances are shown, i.e. instances of concepts on the left side.

After giving these examples let us now comment on the similarities and differences between the chosen metaclasses.

Overview on Diagrams and Purpose: The table in Fig. 4 points out the main diagrams for the chosen metaclasses and tries to shortly characterize the main purpose of the respective notion.

Action-Event Pattern: The destilled metaclass diagram in Fig. 1 reveals a pattern for Action and Event. Signal production is done by SendAction and signal consumption by SignalEvent. Operation call production is done by CallAction and operation call comsumption by CallEvent. From the Action classes there are many-to-one associations to Signal resp. Operation, and from Signal resp. Operation there are one-to-many associations to the Event classes.

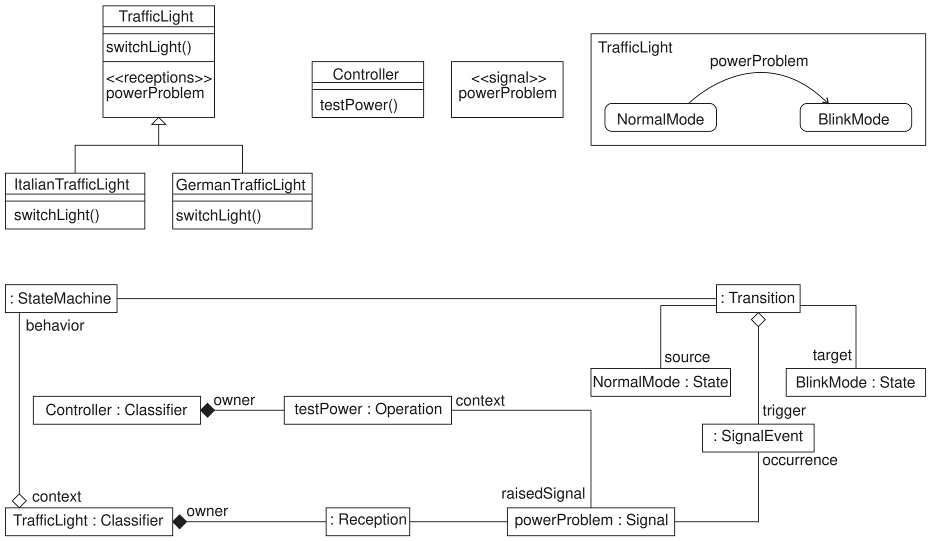

Fig. 2. Example Explaining Signal, Reception, Event

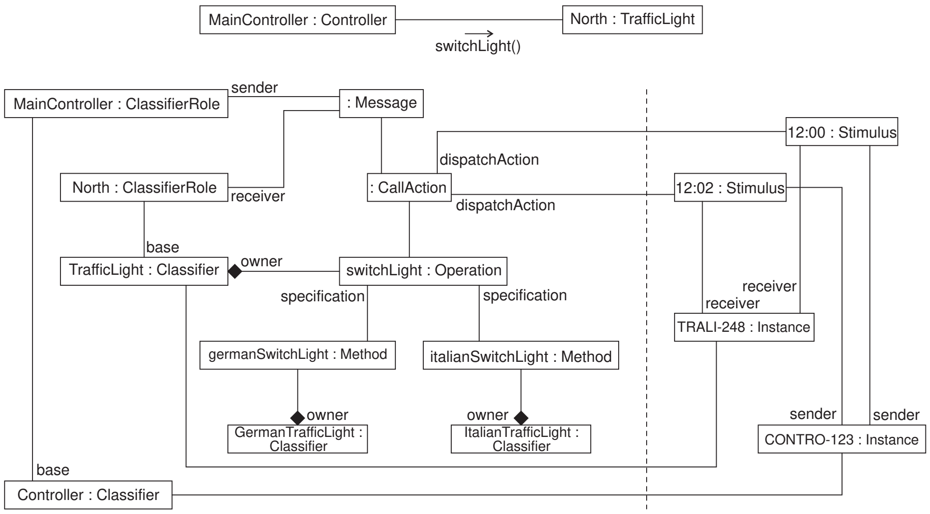

Fig. 3. Example Explaining Stimulus, Action, Message, Operation, Method

Metaclass	Main diagram	Main purpose
Action	Statech./Seq./Collab.	Abstraction of a computation
Event	Statech.	Specification of an observable occurrence
Exception	Class	Signal raised for execution faults
Message	Seq./Collab.	Specification of communication between instances
Method	Statech./Activ.	Implementation of an operation
Operation	Class	Declaration of an operation
Reception	Class	Declaration of signal receipt
Signal	Class	Asynchronous global communication
Stimulus	Seq./Collab.	Reification of a communication between instances

Fig. 4. Overview on Diagrams and Purpose of Chosen Metaclasses

Existential Dependencies from Operation, Signal, and Action:
(1) Method, CallAction, and CallEvent instances existentially depend on Operation, (2) Reception, SignalEvent, and CallEvent instances on Signal, (3) Stimulus and Message instances on Action. Thus Operation, Signal, and Action play a dominant role.

Class Diagram versus Behavior Diagram Concepts:
The five grey-shaded metaclasses on the left of Fig. 1 constitute the class diagram concepts whereas the four grey-shaded metaclasses on the right the behavior diagram concepts.

Message versus Stimulus: Stimulus and Message are closely related: They both are connected to one action and possess sender and receiver entities. Sender and receiver of Stimuli are however Instances whereas sender and receiver of Messages are ClassifierRoles which again are Classifiers. Thus Stimuli and Message both represent communication, but Stimuli on an instance level and Messages on a classifier level. As an observation we also state that the concept of Stimulus (which was called MessageInstance in UML 1.1) is not mentioned in the UML Reference Manual [RJB98].

Missing Rule for Reception, Signal, and SignalEvent:
The destilled metaclass diagram in Fig. 1 also reveals that a well-formedness rule for Reception, Signal, and SignalEvent is missing in the UML Semantics: The corresponding Signal of a SignalEvent appearing, for example, in a StateMachine owned by a Classifier must be declared as a Reception for that Classifier.

3 Conclusion

We have concentrated on nine notions important for aspects of dynamic behavior in UML and have pointed out relationships and connections between these notions. Our description destilles from the original UML description in the UML Semantics the relevant parts and can serve as a starting point for further in-depth discussion, for example, on the basis of a graph transformation approach [GPP98, Gog00].

References

[Gog00] M. Gogolla. Graph Transformations on the UML Metamodel. In J.D.P. Rolim, A.Z. Broder, A. Corradini, R. Gorrieri, R. Heckel, J. Hromkovic, U. Vaccaro, and J.B. Wells, editors, *Proc. ICALP Workshop Graph Transformations and Visual Modeling Techniques (GVMT'2000)*, pages 359–371. Carleton Scientific, Waterloo, Ontario, Canada, 2000.

[GPP98] M. Gogolla and F. Parisi-Presicce. State Diagrams in UML - A Formal Semantics using Graph Transformation. In M. Broy, D. Coleman, T. Maibaum, and B. Rumpe, editors, *Proc. ICSE'98 Workshop on Precise Semantics of Modeling Techniques (PSMT'98)*, pages 55–72. Technical University of Munich, Technical Report TUM-I9803, 1998.

[OMG99] OMG. *Unified Modeling Language Specification (Version 1.3)*. OMG, 1999.

[RJB98] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Guide*. Addison-Wesley, 1998.