Real-Time Constraints with the OCL

Stephan Flake
C-LAB, Paderborn University, Fürstenallee 11, 33102 Paderborn, Germany

Abstract

In recent years, the Unified Modeling Language (UML) has received increasing attention from designers of real-time systems. Several approaches apply and enrich the UML notation for modeling of real-time applications. In addition to that, UML modelers can make use of the Object Constraint Language (OCL) to restrict their models by additional constraints.

Currently, OCL and real-time extensions of the UML only provide limited means to express temporal constraints. In particular, OCL lacks sufficient means to specify constraints over the dynamic behavior of a UML model, i.e., the evolution of states and state transitions as well as time-bounded constraints. However, it is essential to be able to specify such constraints for real-time systems to guarantee correct system behavior. We argue for a consistent extension of OCL that enables modelers to express state-related time-bounded constraints.

1. Real-Time UML

Domain-specific approaches frequently make use of the UML *stereotype* extension mechanism to introduce new model elements for their particular needs (see [9], Section 3.18). One of the most popular approaches to apply UML to the domain of real-time systems is the UML-RT profile based on the ROOM methodology [11]. Nevertheless, several other real-time extensions of UML exist that also enable designers to adequately model the structural issues of real-time systems.

Concerning real-time behavior, the UML standard notation currently provides two ways to specify timing properties, (a) for messages in Sequence Diagrams by timing expressions and (b) for state transitions in Statecharts by elapsed-time events (see [9], Sections 3.60 and 3.77). For advanced modeling of real-time behavior, extensions on behavioral UML diagrams can be applied. For instance, timing bounds for state durations could be modeled using a stereotype on the metaclass `State`, semantically based on Harel's definition of Statecharts [7].

2. Object Constraint Language

The Object Constraint Language (OCL) is part of the UML since version 1.3. It is an expression language that enables modelers to formulate constraints in the context of a given UML model. OCL is used to specify invariants attached to classes, pre- and postconditions of operations, and guards on state transitions (see [9], Chapter 6). It is a declarative language, not a programming language, i.e., evaluation of OCL expressions does not have side effects on the corresponding UML model. To integrate constraints into the visual modeling approach of UML, invariants, pre- and postconditions are modeled as comments and attached to the respective model elements in class diagrams.

Each OCL expression has a type. Besides user-defined model types (e.g., classes or interfaces) and some pre-defined basic types (e.g., Integer, Real, or Boolean), OCL has the notion of object collection types, i.e., `Set`, `Sequence`, and `Bag`. OCL defines several useful operations to access and select objects from such object collections.

To give an example, assume that we have a model with classes `Machine` and `Buffer` and an association `buffers` between these classes. The following invariant ensures that each instance of class `Machine` has at least one buffer:

```
context Machine
inv: self.buffers->notEmpty()
```

The class name that follows the `context` keyword specifies the class for which the following expression should hold. The keyword `self` refers to each object of the context class. Attributes, operations, and associations can be accessed by dot notation, e.g., `self.buffers` results in a (possibly empty) set of instances of `Buffer`. The arrow notation indicates that a collection of objects is manipulated by one of the pre-defined OCL collection operations. For example, operation `notEmpty()` returns true, if the accessed set is not empty.

As OCL is a modeling language, it does not take runtime issues into account. Modelers must decide how to translate OCL expressions into executable code and when constraints are to be checked during runtime. Moreover, it is not possible with OCL to specify what actions are to be taken if an invariant does not hold at a specific point in time.

3. Temporal Extension of OCL

Works on temporal extensions for OCL basically introduce temporal logic operators (e.g., eventually, always, or never) that enable modelers to specify required occurrences of actions, events, and states [3, 4]. Unfortunately, the resulting syntax of these extensions does not combine well with current OCL concepts, as temporal expressions are very similar to rather cryptic temporal logic formulae. Moreover, none of the approaches explicitly considers real-time.

In contrast to these approaches, we propose an extension to OCL with only minor modifications on the language metalevel, so that the use of current OCL is not affected. Our extension is based on a metamodel presented in [1], however, adaptation to other metamodels is possible, e.g. to [10]. The formal semantics of our extension is given by a direct correspondence to time-annotated temporal tree logic formulae.

For modeling object behavior, the UML makes use of Statechart diagrams that are attached to classes (see [9], Chapter 3). With OCL, it is already possible to check the current state of an object, using the pre-defined type `OclState` and the operation `oclInState()`. For more extensive reasoning about states, we introduce a new type `OclConfiguration` that deals with parallel substates and represents possible overall descriptions of Statecharts. Sequences of such configurations are specified by the new type `OclPath`.

This enables modelers to specify required sequences of states. For example, the following invariant requires that for each instance of `Machine`, at each point of time, within the next 100 time units, on all possible execution paths, the states `Loading` and `Working` must be subsequently entered:

```
context Machine
inv: self@post[1,100]->forAll(p:OclPath |
     p->includes(Sequence{Loading,Working}))
```

This notation is compliant with existing OCL syntax. The operation `@post` is newly introduced and can be used for all user-defined types. It extracts the set of possible future execution paths, optionally restricted by a timing interval, i.e., the result of that operation is a set of `OclPath` objects.

The UML does not define a notion of a (global) time, so that clocks are usually explicitly modeled by user-defined classes. The formal semantics of our work, however, is defined by a mapping of time-annotated Statecharts to extended Kripke structures and a mapping of our OCL extension to a time-bounded variant of Computational Tree Logic, called Clocked CTL. More details about the semantics and an application of our OCL extension in a formal verification environment can be found in [5] and [6].

4. The Future of OCL

As OCL is currently only loosely coupled to the UML on the metalevel of language definition, the standardization initiative OMG has initiated a request for proposals to define a consistent OCL metamodel. The submitted proposals can be retrieved from [8]. However, specific real-time aspects are not considered in these proposals.

The evolution, application, and extension of OCL is comparable to the evolution of the UML in general: Modelers, both in academia and industry, identify deficiencies when they apply the language in their specific domain and extend the language for their needs. Examples can be found in the area of business processes, databases, and real-time systems. An extensive overview on OCL's applications, extensions, and semantics is given in [2].

We are sure that in the near future more domain-specific extensions of OCL will be developed, including real-time systems as well.

References

[1] T. Baar and R. Hähnle. An Integrated Metamodel for OCL Types. In *Proc. of OOPSLA 2000, Workshop Refactoring the UML: In Search of the Core.*, Minneapolis, MN, USA, 2000.

[2] T. Clark and J. Warmer, editors. *Object Modeling with the OCL*, volume 2263 of *LNCS*. Springer-Verlag, 2002.

[3] S. Conrad and K. Turowski. Temporal OCL: Meeting Specifications Demands for Business Components. In *Unified Modeling Language: Systems Analysis, Design, and Development Issues*. IDEA Group Publishing, 2001.

[4] D. Distefano, J.-P. Katoen, and A. Rensink. On a Temporal Logic for Object-Based Systems. In *Proceedings of FMOODS'2000*, Stanford, CA, USA, 2000.

[5] S. Flake and W. Mueller. An OCL Extension for Real-Time Constraints. In Clark and Warmer [2].

[6] S. Flake and W. Mueller. Specification of Real-Time Properties for UML Models. In *Proc. of the Hawaii Internat. Conf. on System Sciences (HICSS-35)*, Hawaii, USA, 2002.

[7] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[8] Object Management Group. UML 2.0 OCL Request For Proposal. URL: http://www.omg.org/techprocess/meetings/schedule/UML_2.0_OCL_RFP.html.

[9] Object Management Group. Unified Modeling Language Specification, Version 1.4, September 2001. URL: http://www.omg.org/technology/documents/formal/uml.htm.

[10] M. Richters and M. Gogolla. A Metamodel for OCL. In *UML'99 - The Unified Modeling Language. Beyond the Standard*, volume 1723 of *LNCS*, Fort Collins, CO, USA, 1999.

[11] B. Selic and J. Rumbaugh. Using UML for Modeling Complex Real-Time Systems. White Paper. URL: http://www.rational.com/media/whitepapers/umlrt.pdf.