Temporal OCL Extensions for Specification of Real-Time Constraints

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

Abstract

The Unified Modeling Language (UML) receives increasing attention by designers of real-time systems. Several approaches already extend the UML notation for modeling real-time applications. In this context, it is essential to be able to specify time-bounded temporal constraints as a prerequisite to validate a model for correct system behavior.

But currently, UML and corresponding extensions only provide limited means to express temporal constraints over the dynamic behavior of objects. Furthermore, the Object Constraint Language (OCL) which was developed to express restrictions over (parts of) UML models currently lacks of means to specify temporal constraints.

We think that an appropriate notation to specify temporal constraints about the dynamic behavior of objects should preferably base upon already existing UML concepts. We therefore propose an extension of OCL that complies to existing concepts and is suitable to express temporal state-oriented, time-bounded constraints.

1. Introduction

Domain-specific approaches frequently make use of UML Profiles and extension mechanisms like *stereotypes* to introduce new model elements for their particular needs [19, Sections 2.6 and 3.16]. One of the most popular approaches to apply UML in the domain of real-time systems is the UML-RT profile based on the ROOM methodology [27]. Nevertheless, several other real-time extensions of UML exist that also enable designers to adequately model the *structure* or *architecture* of real-time systems.

These approaches have partially influenced the UML 2.0 infrastructure and superstructure proposals that have recently been adopted by the OMG [20, 21]. In the UML 2.0 superstructure proposal, three new kinds of diagrams (component diagrams, composite structure diagrams, and timing diagrams) and several new model elements (e.g., protocol state machines for ports and interfaces) have been introduced.

Besides modeling of a system, other important aspects of system development are analysis, validation, test, and – especially in safety-critical application domains – formal verification. In this context, an additional *specification* of desired properties that the system under consideration has to satisfy is employed. Properties that need closer attention primarily concern dynamic behavior, i.e., they are *temporal constraints*, sometimes even with specific time-bounds. A given model has then to be investigated, i.e., analyzed, validated, tested, and/or verified, w.r.t. these (time-bounded) temporal constraints.

In this article, we review those UML model elements that are used to express temporal constraints with a focus on time-bounded constraints. We first consider existing UML documents, i.e., the UML 1.5 and UML 2.0 specifications, the UML Profile for Schedulability, Performance, and Time, and the latest OCL 2.0 proposal (Section 2). In Section 3, we then outline extensions of UML/OCL that introduce means for temporal constraints. Finally, Section 4 concludes with our expectations on the future of UML/OCL extensions for specification of temporal constraints.

2. Time-Related Concepts in UML

The current **UML 1.5** standard provides two ways to specify behavioral time-related properties, i.e.,

- timing expressions in Sequence Diagrams that indicate times of sending and receiving messages to formulate constraints like `{reply().receiveTime-request().sendTime < 1 sec}` [19, Section 3.60] and
- elapsed-time events in Statecharts to trigger state transitions after a specific time has passed without a state change, e.g., `after(5 sec)` [19, Section 3.77].

For advanced modeling of real-time behavior, extensions on behavioral UML diagrams can be applied. For example, time bounds for state durations can be modeled using a stereotype on the metaclass `State`, semantically based on Harel's definition of Statecharts [12].

But generally, the UML standard does not have a notion of time. Therefore, different extensions of UML have been developed by means of UML profiles, the most popular being RT-UML [7] and UML-RT based on the ROOM methodology [27].

More recently, the **UML Profile for Schedulability, Performance, and Time** has been adopted by the OMG in September 2002 [18]. Though that profile provides a common framework of time-related concepts (e.g., clocks, timers, timed actions, and timed events), it also has only limited means concerning the specification of temporal object behavior.

In the **UML 2.0 superstructure proposal** [21], Sequence Diagrams are now equipped with improved modeling elements for time bounds. Arcs that represent messages that are sent between objects can now be annotated by expressions that refer to

- duration observations (e.g., "`Code d = duration`"),
- duration constraints (e.g., "{`d..3*d`}"),
- time observations (e.g., "`t=now`"), and
- time constraints (e.g., "{`t..t+3`}").

Timing diagrams are one of the new kinds of diagrams in UML 2.0 [21, Section 14.4]. Timing diagrams model changes of object states over time along a linear time axis. Basically, modelers can specify conditions that imply object state changes as part of object lifelines. The behavior of objects as well as interactions among objects can thus be restricted. Though some examples are provided and a guideline for the basic graphical notation is given in the UML 2.0 superstructure proposal, the semantics description of timing diagram is still incomplete, e.g., the semantics of tick mark values and timing rulers is unclear.

The **Object Constraint Language (OCL)** is an integral part of UML [19, Chapter 6]. It is a declarative 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 conditions on state transitions.

So far, OCL does not have means to formulate temporal constraints. But the latest OCL 2.0 proposal [14] introduces a language concept that enables modelers to reason about messages that must have been sent (so called *OCL messages*). In a broad sense, OCL messages establish a temporal notion to OCL, as the set of messages sent *during operation execution* is regarded. Apart from OCL messages, there is no other concept in OCL to specify temporal constraints.

Nevertheless, different approaches have taken OCL as a basis and developed temporal extensions to enable modelers to specify temporal constraints. An overview is given in the next section.

3. Temporal Extensions of OCL

In recent years, a number of OCL extensions have independently been proposed to enable modelers to specify temporal constraints. We first review work on temporal OCL extensions by other authors, then outline our approach, and finally provide a comparison of these approaches w.r.t. syntax, semantics, and support of explicit timing specification.

3.1. Related Work

Ramakrishnan et al. [23]extend OCL by additional rules with unary and binary temporal operators, e.g., `always` and `never` to specify safety and liveness properties. A very similar approach in the area of business modeling that also considers past temporal operators is published by Conrad and Turowski [5]. However, the resulting syntax of these works does not combine well with standard OCL, as temporal expressions appear to be similar to temporal logics formulae.

Kleppe and Warmer [16] introduced a so-called action clause to OCL. Basically, action clauses enable modelers to specify required (synchronous or asynchronous) executions of operations or dispatching of events. This work has influenced the previously mentioned message concept in the OCL 2.0 proposal.

Distefano et al. [6] define BOTL (Object-Based Temporal Logic) in order to facilitate the specification of static and dynamic properties. BOTL is not directly an extension of OCL; it rather maps a subset of OCL into object-oriented Computation Tree Logic (CTL). Syntactically, BOTL looks very similar to temporal logics formulae in common CTL.

Bradfield et al. [2] extend OCL by useful causality-based templates for dynamic constraints. Basically, a template consists of clauses, the cause and the consequence. The cause clause starts with the keyword *after*, followed by a boolean expression, while the consequence is one of *eventually, immediately, infinitely* etc., followed by an OCL expression. The templates are formally defined by a mapping into observational μ-calculus, a two-level temporal logic, using OCL as the lower level logic.

Ziemann and Gogolla [29] present an OCL extension, in which future-oriented temporal development of attribute values and existence of objects and links can be restricted. Similar to other approaches, temporal operators like `always`, `next`, and `sometime` are introduced. For defining a formal semantics, they build upon the set-theoretic OCL semantics developed by M. Richters [24] and define *traces*, i.e., sequences of system states. Such a trace employs a high-level notion of the development of a running system with only that information which is necessary to evaluate OCL expressions.

Note that none of the approaches mentioned so far considers real-time constraints. Besides the rudimentary and

Table 1. Temporal OCL Extensions and Real-Time Specification

Approach	Syntax	Formal Semantics	Real-Time
Ramakrishnan et al. [23]	OCL + temporal operators	–	no
Conrad/Turowski [5]	OCL + temporal operators	–	no
Kleppe/Warmer [16]	OCL + action clause	–	no
Distefano et al. [6]	CTL + OCL subset	BOTL	no
Bradfi eld et al. [2]	OCL + template clauses	Observational μ-calculus	no
Ziemann/Gogolla [29]	OCL + temporal operators	Trace semantics	no
Roubtsova et al. [25, 26]	Stereotyped classes	TCTL	yes
Sendall/Strohmeier [28]	OCL consistent	–	yes
Cengarle/Knapp [3]	OCL + temporal operators	Trace semantics	yes
Flake/Mueller [9, 11]	OCL consistent	Clocked CTL	yes

mostly informal UML modeling elements described in Section 2, we know of the following approaches.

The work presented by Roubtsova et al. [25, 26] defines a UML profile with stereotyped classes for dense time as well as parameterized specification templates for deadlines, counters, and state sequences. Each of these templates has a structural-equivalent dense-time temporal logics formula in TCTL (Timed Computation Tree Logic).

Sendall and Strohmeier [28] introduce timing constraints on state transitions in the context of a restricted form of UML protocol state machines called SIP (System Interface Protocol). A SIP defines the temporal ordering between operations. Five time-based attributes on state transitions are proposed, e.g., (absolute) completion time, duration time or frequency of state transitions. Using these attributes, one can then relate actions to timing constraint failures in an extended form of transition condition (or, in UML terms: transition guard).

Cengarle and Knapp [3] present OCL/RT, a temporal extension of OCL with modal operators `always` and `sometime` over event occurrences. These can be used for specifying deadlines and timeouts of operations and reactions on received signals. On the metamodel level, events are equipped with time stamps by introducing a metaclass `Time` with attribute `now` to refer to the time unit at which an event occurs. In turn, each instance can access the set of current associated events at each point of time, i.e., at each *system state*.

3.2. State-oriented Temporal OCL Extension

With current OCL, it is already possible to check the activated state of an object using the operation `oclInState(statename:OclState)`. For more extensive reasoning about states, our temporal OCL extension (called RT-OCL) has a notion of state sequences based upon a formal definition of *active state configurations*.[1] The syntax of our OCL extension is consistent with common OCL syntax and builds upon the metamodel of the OCL 2.0 proposal [9]. Basically, we provide means to specify temporal constraints over Statechart states, such as safety and liveness properties. Beyond that, we have shown in [10] that our OCL extension has the expressive power to specify all of those properties that are regarded as being relevant in practice (based upon the property specification pattern system by Dwyer et al. [8]). A formal semantics is given by a trace semantics similar to the approach by Ziemann and Gogolla in [29]. But additionally, we map state-based temporal OCL expressions to time-annotated temporal tree logic formulae (i.e., Clocked CTL) for further application in model checking tools.[2]

3.3. Comparison

We think that a successful approach to formulate temporal (time-bounded) constraints in UML and/or OCL should have the following characteristics:

- **Syntax**: Temporal constraints should be notated conforming to existing concepts of UML. This is important w.r.t. the already quite extensive notation of UML to be adopted and accepted by UML users.

- **Semantics**: Temporal constraints should be provided with a formal semantics. Only by a formal semantics a unique notational meaning is guaranteed and formal verification techniques can be employed.

[1]Note that the current notion of active state confi gurations is only informally defi ned and has some defi ciencies, e.g., it does not consider fi nal states in active state confi gurations.

[2]Note that a mapping of the referred UML model into appropriate model checking input has additionally to be defi ned.

- **Real-Time**: The notation for temporal constraints should support specification of time bounds and timing intervals. This is especially important for modeling of safety-critical systems.

Table 1 lists once more the mentioned approaches and compares them w.r.t. the desired characteristics.

Most of the approaches with a formal semantics have formal verification by model checking in mind. Formal verification by theorem proving using OCL is investigated in the KeY project. That approach aims to facilitate the use of formal verification for software specifications [1]. Here, OCL is applied without modifications to specify constraints on design patterns. As standard OCL currently has no formal semantics, this approach translates OCL specifications to dynamic logic (DL), an extension of Hoare logic. DL is used as input for formal verification by theorem proving.

4. The Future of OCL

In recent years, many complaints about the concrete OCL syntax could be observed, e.g., [22, Section 5]. For UML 2.0, the metamodel approach of the OCL 2.0 proposal might enable tool developers to overcome this problem in the future. Basically, tools can employ their own constraint language in UML 2.0; they only have to provide a mapping to the OCL metamodel. Thus, a tool does not have to stick to the concrete OCL syntax provided in the OCL 2.0 proposal. For example, there is already work available on a graphical OCL variant [15].

However, the semantics of OCL still has some deficiencies. In the OCL 2.0 proposal, two semantic descriptions are provided. On the one hand, a metamodel-based semantics is given that associates the abstract syntax (i.e., the metamodel) with values on the actual M1 level. On the other hand, a formal semantics by means of a naive set-theoretic approach (i.e., object models) is provided that is based on work by Mark Richters [24]. Unfortunately, the two semantics are currently neither consistent nor complete, as (a) the formal semantics does not consider the newly introduced concept of OCL messages and (b) both semantics lack an integration of Statecharts and a semantic definition of state-related operations.

And even if these problems were fixed during the finalization process of OCL 2.0, there remains the more general problem of the level of abstraction used to define the semantics of OCL. In this context, H. Hussmann argues in [13] that neither naive set theory nor a metamodel-based approach is fully adequate for building a conceptual bridge between the programming artifacts produced from UML/OCL and the formal semantics currently defined for OCL. Basically, this is due to the direct mapping of attributes to values of a particular semantic domain. Further research is therefore necessary to overcome this issue.

The evolution, application, and extension of OCL is comparable to the evolution of UML in general: Modelers, both in academia and industry, identify deficiencies when they apply the language in their specific domain and consequently extend the language for their needs. Examples can be found in the area of business processes, databases, and real-time systems (cf. [4] for an overview of recent research efforts on OCL).

We expect that several extensions of OCL will be developed in different application domains, e.g., at this year's UML conference, an OCL extension concerning low-coupling preserving contracts is presented [17]. Similarly, there will most likely be further OCL extensions in the domain of modeling real-time systems. Semantical issues will play an important role in this context, and interdependencies with other UML extensions such as the UML Profile for Schedulability, Performance, and Time have to be studied more extensively. It is imaginable that OCL extensions make use of that profile and extend it by appropriate means, resulting in a notation that enables UML users to specify time-bounded temporal constraints.

Acknowledgements

This work receives funding by the Deutsche Forschungsgemeinschaft in the course of the Special Research Initiative 614 “Self-optimizing Concepts and Structures in Mechanical Engineering”.

References

[1] W. Ahrendt, T. Baar, B. Beckert, M. Giese, R. Hähnle, W. Menzel, and P. H. Schmitt. The KeY Approach: Integrating Object Oriented Design and Formal Verifi cation. In M. Ojeda-Aciego et al., editors, *8th European Workshop on Logics in AI (JELIA), Malaga, Spain*, volume 1919 of *LNCS*, pages 21–36. Springer, October 2000.

[2] J. Bradfi eld, J. Kuester Filipe, and P. Stevens. Enriching OCL Using Observational mu-Calculus. In R.-D. Kutsche and H. Weber, editors, *Fundamental Approaches to Software Engineering (FASE 2002), Grenoble, France*, volume 2306 of *LNCS*, pages 203–217. Springer, April 2002.

[3] M. Cengarle and A. Knapp. Towards OCL/RT. In L.-H. Eriksson and P. Lindsay, editors, *Formal Methods – Getting IT Right, International Symposium of Formal Methods Europe, Copenhagen, Denmark*, volume 2391 of *LNCS*, pages 389–408. Springer, July 2002.

[4] T. Clark and J. Warmer, editors. *Object Modeling with the OCL*, volume 2263 of *LNCS*. Springer, Heidelberg, Germany, February 2002.

[5] S. Conrad and K. Turowski. Temporal OCL: Meeting Specifi cations Demands for Business Components. In K. Siau and T. Halpin, editors, *Unifi ed Modeling Language: Systems*

Analysis, Design, and Development Issues, pages 151–165. IDEA Group Publishing, 2001.

[6] D. Distefano, J.-P. Katoen, and A. Rensink. On a Temporal Logic for Object-Based Systems. In S. F. Smith and C. L. Talcott, editors, *Formal Methods for Open Object-Based Distributed Systems IV (FMOODS'2000), Stanford, CA, USA*, pages 285–304. Kluwer Academic Publishers, September 2000.

[7] B. P. Douglass. *Doing Hard Time: Developing Real Time Systems with UML, Objects, Frameworks, and Patterns*. Addison-Wesley, 2000.

[8] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. A System of Specifi cation Patterns, September 1998. http://www.cis.ksu.edu/santos/spec-patterns.

[9] S. Flake and W. Mueller. A UML Profi le for Real-Time Constraints with the OCL. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *UML 2002 – The Unifi ed Modeling Language. Model Engineering, Languages, Concepts, and Tools, Dresden, Germany, September/October 2002*, volume 2460 of *LNCS*, pages 179–195. Springer, 2002.

[10] S. Flake and W. Mueller. Expressing Property Specifi cation Patterns with OCL. In *The 2003 International Conference on Software Engineering Research and Practice (SERP'03), Las Vegas, Nevada, USA, June 2003*, pages 595–601. CSREA Press, 2003.

[11] S. Flake and W. Mueller. Formal Semantics of Static and Temporal State-Oriented OCL Constraints. *Journal on Software and System Modeling (SoSyM)*, 2(3), October 2003.

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

[13] H. Hussmann. Loose Semantics for UML/OCL. In A. E. H. Ehrig, B.J. Kraemer, editor, *6th World Conference on Integrated Design and Process Technology (IDPT 2002), Pasadena, CA, USA*, June 2002.

[14] A. Ivner, J. Högström, S. Johnston, D. Knox, and P. Rivett. Response to the UML2.0 OCL RfP, Version 1.6 (Submitters: Boldsoft, Rational, IONA, Adaptive Ltd., et al.). OMG Document ad/03-01-07, January 2003.

[15] C. Kiesner, G. Taentzer, and J. Winkelmann. VisualOCL: A Visual Notation of the Object Constraint Language. Technical Report 23, Computer Science Department of the Technical University of Berlin, Germany, 2002.

[16] A. Kleppe and J. Warmer. Extending OCL to include Actions. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000 – The Unifi ed Modeling Language. Advancing the Standard. 3rd International Conference, York, UK, October 2000*, volume 1939 of *LNCS*, pages 440–450. Springer, 2000.

[17] I. Nunes. An OCL Extension for Low-coupling Preserving Contracts. In G. Booch, P. Stevens, and J. Whittle, editors, *UML 2003 – The Unifi ed Modeling Language. Modeling Languages and Applications, San Francisco, CA, USA, October 2003*, LNCS. Springer, October 2003.

[18] Object Management Group. UML Profi le for Schedulability, Performance, and Time Specifi cation. OMG Document ptc/2003-03-02, April 2003.

[19] Object Management Group. Unifi ed Modeling Language 1.5 Specifi cation. OMG Document formal/2003-03-01, March 2003. http://www.omg.org/technology/documents/formal/-uml.htm.

[20] Object Management Group. Unifi ed Modeling Language: Infrastructure, Version 2.0. Adopted Specifi cation, OMG Document ad/2003-03-01, July 2003.

[21] Object Management Group. Unifi ed Modeling Language: Superstructure, Version 2.0. Final Adopted Specifi cation, OMG Document ptc/2003-08-02, August 2003.

[22] P. Padawitz. Swinging UML – How to Make Class Diagrams and State Machines Amenable to Constraint Solving and Proving. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000 – The Unifi ed Modeling Language. Advancing the Standard. 3rd International Conference, York, UK, October 2000*, volume 1939 of *LNCS*, pages 162–177. Springer, 2000.

[23] S. Ramakrishnan and J. McGregor. Extending OCL to Support Temporal Operators. In *21st International Conference on Software Engineering (ICSE99), Workshop on Testing Distributed Component-Based Systems, Los Angeles, CA, USA*, May 1999.

[24] M. Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen, Bremen, Germany, 2001.

[25] E. Roubtsova and W. Toetenel. Specifi cation of Real-Time Properties in UML. In *22nd IEEE Real-Time Systems Symposium RTSS, Work-In-Progress Section*, London, UK, December 2001.

[26] E. E. Roubtsova, J. van Katwijk, W. J. Toetenel, and R. C. M. de Rooij. Real-Time Systems: Specifi cation of Properties in UML. In *Proceedings of the 7th Annual Conference of the Advanced School for Computing and Imaging (ASCI 2001)*, pages 188–195, Het Heijderbos, Heijen, The Netherlands, May 2001.

[27] B. Selic and J. Rumbaugh. Using UML for Modeling Complex Real-Time Systems. White Paper, 1998. http://www.rational.com/media/whitepapers/umlrt.pdf.

[28] S. Sendall and A. Strohmeier. Specifying Concurrent System Behavior and Timing Constraints Using OCL and UML. In M.Gogolla and C.Kobryn, editors, *UML 2001 – The Unifi ed Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference, Toronto, Canada, October 2001*, volume 2185 of *LNCS*, pages 391–405. Springer, 2001.

[29] P. Ziemann and M. Gogolla. An Extension of OCL with Temporal Logic. In J. Jürjens, M. V. Cengarle, E. B. Fernandez, B. Rumpe, and R. Sandner, editors, *Critical Systems Development with UML*, pages 53–62. Technische Universität München, Institut für Informatik, 2002.