Temporale Erweiterungen der OCL – Überblick und Aussichten –

Stephan Flake, Wolfgang Müller

C-LAB, Universität Paderborn, Fürstenallee 11, 33102 Paderborn
email: {flake, wolfgang}@c-lab.de

Zusammenfassung

Die Object Constraint Language (OCL) wurde entwickelt, um Modelleinschränkungen beim objektorientierten Softwareentwurf mit der UML [14] ausdrücken zu können. Sie wird hauptsächlich benutzt, um Invarianten für Objekte sowie Vor- und Nachbedingungen von Operationen zu spezifizieren.

Zurzeit bieten OCL und Echtzeiterweiterungen der UML nur bedingt geeignete Mittel, um temporale zeitbehaftete Modelleigenschaften zu beschreiben. Insbesondere kann man mit OCL keine Einschränkungen über das dynamische Verhalten eines UML-Modells formulieren, die die Reihenfolge von Objektzuständen und Zustandsübergängen betreffen. Um ein korrektes Systemverhalten zu garantieren, ist es jedoch insbesondere bei zeitkritischen Anwendungen notwendig, solche zustandsbasierten zeitbehafteten Einschränkungen in einer formalen Art und Weise ausdrücken zu können.

Es sind daher verschiedene Vorschläge veröffentlicht worden, in denen die OCL erweitert worden ist, um Modellierern die Möglichkeit zu geben, temporale Einschränkungen zu formulieren. Dieser Artikel gibt einen Überblick über die zurzeit bekannten Vorschläge und zeigt Ansätze für weitere Entwicklungen in diesem Bereich auf.

1 Object Constraint Language

Die Object Constraint Language (OCL) ist seit der UML Version 1.3 von 1997 offizieller Bestandteil der UML (vgl. [14] , Kapitel 6). Es handelt sich um eine textbasierte Sprache, die Modellierern ermöglicht, Einschränkungen über einem gegebenen UML-Modell zu formulieren. OCL wird zur vornehmlich zur Spezifikation von Invarianten im Kontext von benutzerdefinierten Klassen, zur Formulierung von Vor- und Nachbedingungen für Operationen und in Bedingungen von Zustandsübergängen eingesetzt. Insbesondere wird OCL auch zur Definition des UML-Metamodells eingesetzt. Da OCL als deklarative Sprache und nicht als eine Programmiersprache zu verstehen ist, hat die Auswertung von OCL-Ausdrücken keine Seiteneffekte auf das zugehörige UML-Modell. OCL-Invarianten sowie Vor-

und Nachbedingungen werden üblicherweise als Kommentare in ihren zugehörigen UML-Klassendiagrammen spezifiziert.

Jeder OCL-Ausdruck hat einen Typ. Neben benutzerdefinierten Typen (basierend auf den Klassen eines gegebenen UML-Modells) und einigen vordefinierten Basistypen (z.B. Integer, Real oder Boolean) gibt es in OCL spezielle Typen für Mengen, Multimengen und Sequenzen von Objekten (Set, Bag und Sequence). Über diesen "Ansammlungstypen" sind diverse nützliche Operationen vordefiniert, um auf Einzelobjekte zugreifen und sie manipulieren zu können.

Zur weiteren Erklärung der Notation und Konzepte betrachten wir ein UML-Modell mit den Klassen `Machine` und `Buffer` und eine Assoziation `buffers` zwischen diesen Klassen. Die folgende einfache Invariante stellt sicher, dass jede Instanz der Klasse `Machine` mindestens einen zugeordneten Puffer hat:

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

Der Klassenname nach dem Schlüsselwort `context` gibt die Klasse an, für die der nachfolgende Ausdruck gelten soll. Das Schlüsselwort `self` steht für jedes einzelne vorhandene Objekt der zuvor spezifizierten Klasse. Auf Attribute, Operationen und Assoziationen wird mit Punktnotation zugegriffen, z.B. liefert `self.buffers` eine (möglicherweise leere) Menge von Objekten der Klasse `Buffer`. Der Typ des Ausdrucks `self.buffers` ist statisch festgelegt; er ist `Set(Buffer)`. Bei Operationen über Ansammlungen von Objekten wird eine Pfeilnotation eingesetzt. Dem Pfeil muss eine typkonforme vordefinierte Operation folgen, um einen gültigen OCL-Ausdruck zu formen. Zum Beispiel liefert die Operation `notEmpty()` den Wert `true`, wenn die betreffende voranstehende Menge nicht leer ist.

Da OCL eine Modellierungssprache ist, werden keine Annahmen über die Auswertung von OCL-Ausdrücken zur Laufzeit gemacht. Modellierer müssen daher selbst entscheiden, wie OCL-Ausdrücke in ausführbaren Code zu übersetzen sind und wann Einschränkungen zur Laufzeit überprüft werden. Darüber hinaus kann man mit OCL nicht spezifizieren, welche Aktionen ausgeführt werden sollen, wenn eine Invariante zu einem bestimmten Zeitpunkt nicht erfüllt ist.

2 Ansätze zur Spezifikation temporaler Eigenschaften

In diesem Abschnitt stellen wir die uns zurzeit bekannten Vorschläge vor, die entweder OCL für die Spezifikation von temporalen Eigenschaften erweitern oder einen anderen Weg finden, diese im Kontext der UML auszudrücken. Auf die in der UML bereits vorhandenen Mittel, wie z.B. zeitbehaftete Ausdrücke in Sequenzdiagrammen oder Ablaufzeiten in Zustandsdiagrammen (*elapsed-time events*), gehen wir hier nicht weiter ein, sondern verweisen auf die Abschnitte 3.60 und 3.77 der aktuellen UML-Spezifikation, Version 1.4 [14].

Temporale Erweiterungen ohne explizite Zeitangaben. Ramakrishnan et al. [16] erweitern OCL durch unäre und binäre zukunftsorientierte temporale Operatoren wie `always` und `never` zur Spezifikation von Sicherheits- und Lebendigkeitseigenschaften. Ein sehr ähnlicher Ansatz im Bereich der Geschäftsprozessmodellierung, bei dem auch vergangenheitsorientierte temporale Operatoren betrachtet werden, ist von Conrad und Turowski veröffentlicht worden [4, 5]. Es ist jedoch anzumerken, dass diese Ansätze erlauben, auf beliebige benutzerdefinierte Operationen zuzugreifen, obwohl es eigentlich in OCL nur erlaubt ist, Abfrageoperationen (sog. *Query*-Operationen) zu benutzen[1].

Kleppe und Warmer stellen die sogenannte *action clause* vor [13]. In diesem neuen Spezifikationsparagraphen können Modellierer geforderte (synchrone oder asynchrone) Operationsaufrufe oder Entsendungen von Ereignissen spezifizieren. Auf ähnliche Weise werden in einem Vorschlag für die kommende OCL2.0-Version Ausdrücke über Nachrichten definiert [20].

Bradfield et al. [3] erweitern OCL mit nützlichen Templates, die kausale temporale Beziehungen herstellen. Im Wesentlichen bestehen die Templates aus neu eingeführten Spezifikationsparagraphen mit einer Ursache und einer nachfolgenden Auswirkung. Letztere wird mit einem temporalen Operator wie *eventually*, *immediately* oder *infinitely* eingeleitet. Die Templates sind durch eine Abbildung in eine erweiterte Form des μ-Kalküls (*observational μ-calculus*) formal definiert.

Distefano et al. definieren *Object-Based Temporal Logic* (BOTL) zur Spezifikation von statischen und dynamischen Modelleigenschaften [8]. BOTL ist eigentlich keine Erweiterung von OCL; vielmehr wird ein Teil der OCL in eine objektorientierte Version der Computation Tree Logic (CTL) übersetzt. BOTL hat daher eine Syntax, die der klassischen CTL sehr ähnlich ist.

Temporale zeitbehaftete Erweiterungen. Roubtsova et al. stellen ein UML-Profile mit stereotypisierten Klassen für kontinuierliche Zeit vor [17, 18]. Darüber hinaus werden durch parametrisierte Stereotypen Templates zur Spezifikation von Fristen, Zählern und Zustandsfolgen definiert. Jede Instanz dieser Templates hat eine strukturell äquivalente Formel in TCTL (*Timed Computation Tree Logic*), einer Temporallogik mit einem kontinuierlichen Echtzeitbegriff. Roubtsova et al. erweitern absichtlich nicht die OCL. Sie begründen dies dadurch, dass OCL kein Sprachkonzept für Ausführungspfade hat [18]:

> *... OCL has no path notion. Any extension of OCL to present properties of computation paths breaks the idea of the language and makes it eclectic.*

Sendall und Strohmeier führen zeitbehaftete Bedingungen auf Zustandsübergängen in einer eingeschränkten Form von UML-Zustandsdiagrammen für soge-

[1] Query-Operationen sind Operationen, die bei ihrer Ausführung keine Seiteneffekte auf dem zugehörigen UML-Modell verursachen, d.h., keine Änderungen des aktuellen Gesamtzustands des Modells nach sich ziehen.

nannte *System Interface Protocols* (SIPs) ein [19]. Ein SIP definiert eine geforderte zeitliche Abfolge von Operationen. Bei diesem Ansatz sind für jeden Zustandsübergang implizit fünf Attribute definiert, die u.a. den Zeitpunkt des letzten Transitionsendes, die Transitionsdauer und die Häufigkeit der Transition in einer bestimmten Zeitspanne angeben. In erweiterten Zustandsübergangsbedingungen darf man auf diese Attribute zugreifen, um z.B. Verletzungen von Zeitschranken zu spezifizieren und entsprechende auszuführende Aktionen anzugeben.

Zustandsorientierte zeitbehaftete OCL-Erweiterung. Wir schlagen eine zustandsorientierte zeitbehaftete Erweiterung der OCL vor, die auf einem Metamodell von Baar und Hähnle basiert [1]. Als geringfügige Änderungen dieses Metamodells fügen wir zwei neue vordefinierte Ansammlungstypen ein. Der Typ `OclConfiguration` repräsentiert die möglichen Konfigurationen von Zustandsdiagrammen. Eine Konfiguration ist eine Menge von parallelen einfachen Zuständen, die zusammen den Gesamtzustand eines Zustandsdiagramms vollständig beschreiben. Sequenzen solcher Konfigurationen werden durch den zweiten neuen Typ `OclPath` repräsentiert. Instanzen von `OclPath` werden demnach als mögliche Ausführungspfade interpretiert. Die meisten in OCL vordefinierten Operationen über Mengen können direkt für `OclConfiguration` übernommen werden. Entsprechendes gilt für Sequenzen und `OclPath`.

Mit diesen Erweiterungen ist man in der Lage, geforderte Ausführungspfade in OCL-konformer Syntax zu spezifizieren. Zum Beispiel stellt die folgende Invariante sicher, dass jedes Objekt der Klasse `Machine` immer wieder innerhalb von 100 Zeiteinheiten (und auf allen möglichen zukünftigen Ausführungspfaden) nacheinander die Zustände `Loading` und `Working` erreicht.

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

Die Operation `@post()` wurde von uns neu definiert und kann auf alle benutzerdefinierten Klassen, für die ein Zustandsdiagramm vorliegt, angewandt werden. `@post()` liefert die Menge möglicher zukünftiger Ausführungspfade ausgehend vom aktuellen Zustand und kann durch Angabe von Parametern auf ein bestimmtes Zeitintervall beschränkt werden.

In UML gibt es keinen vorgegebenen Zeitbegriff, sodass üblicherweise Uhren explizit durch benutzerdefinierte Klassen modelliert werden (vgl. auch das *UML Profile for Scheduling, Performance, and Time* [15]). Eine formale Semantik unserer Erweiterung ist jedoch durch eine Abbildung von zeitannotierten Zustandsdiagrammen in erweiterte Kripke-Strukturen und von unserer temporalen OCL-Erweiterung in eine zeitbehaftete Variante der klassischen CTL, Clocked CTL (CCTL), gegeben. Details zu der Semantik und eine Anwendung unserer Erweiterung sind in [11] und [12] zu finden.

3 Vergleich der vorgestellten Ansätze

Nicht nur in den zitierten Publikationen ist erkannt worden, dass in der UML zurzeit ausreichende Spezifikationsmöglichkeiten für temporale und zeitbehaftete Einschränkungen fehlen. Für eine Erweiterung in dieser Hinsicht bieten sich aufgrund der starken graphischen Ausprägung der UML zunächst neue Diagrammarten bzw. Erweiterungen bestehender UML-Diagramme an, mit denen z.B. geforderte oder nicht erlaubte Abfolgen von Operationen spezifiziert werden können (vgl. hierzu beispielsweise *Life Sequence Charts* [6]). In diesem Artikel konzentrieren wir uns jedoch vornehmlich auf textuelle Ansätze zur Spezifikation temporaler Modelleigenschaften, denn sie bieten u.a. die Möglichkeit, in vergleichsweise kompakter Form auch umfangreiche, tief verschachtelte Bedingungen ausdrücken zu können. Dies ist insbesondere dann ein Vorteil, wenn zusätzlich auch noch Zeitannotationen in Betracht gezogen werden müssen. Wir überprüfen im Folgenden die im Abschnitt 2 betrachteten Ansätze im Hinblick auf folgende Eigenschaften:

1. **Syntax:** Die temporale Erweiterung soll konform mit den bestehenden Konzepten der UML sein. Nur so kann wegen der bereits sehr umfangreichen Modellierungssprache UML mit einer Akzeptanz bei den Benutzern gerechnet werden.

2. **Semantik:** Die temporale Erweiterung soll eine formale Semantik haben. Nur so ist gewährleistet, dass die Ausdrücke eindeutig und mit formalen Verifikationstechniken überprüfbar sind.

3. **Echtzeit:** Die temporale Erweiterung soll die Angabe von Zeitschranken und Zeitintervallen erlauben. Die Angabe zeitbehafteter Einschränkungen ist insbesondere bei der Modellierung echtzeitkritischer Systeme essentiell.

Tabelle 1 listet die Beiträge aus Abschnitt 2 noch einmal auf und betrachtet die Ansätze hinsichtlich der oben genannten drei Eigenschaften.

Der Ansatz von Distefano et al. (Nummer 5 in Tabelle 1) lässt sich vom syntaktischen Standpunkt zwar nicht direkt in die UML einordnen und betrachtet darüber hinaus auch keine Zeiteigenschaften, obwohl das zugrunde liegende formale Modell sehr interessant ist und z.B. auch die Arbeit von Bradfield et al. beeinflusst hat.

Bezüglich der Syntax führen die auf OCL basierenden Ansätze 1, 2, 3 und 4 entweder neue temporale Operatoren ein, die sich nicht direkt in die bestehenden Konzepte der OCL integrieren, oder fügen sogar komplett neue Spezifikationsparagraphen zu OCL hinzu. Jene Arbeiten weisen damit wesentliche Spracherweiterungen der OCL auf und erfüllen damit nur bedingt die erste gewünschte Eigenschaft. Darüber hinaus werden in keinem der vier Ansätze zeitbehaftete Einschränkungen betrachtet. Die Ansätze 1, 2, 3 und 7 haben keine formale Semantik und erfüllen damit nicht die gewünschte zweite Eigenschaft.

Tabelle 1: Ansätze zur Spezifikation temporaler Eigenschaften

Nr.	Ansatz	Syntax	Formale Semantik	Echtzeit
1	Ramakrishnan et al. [16]	OCL + temporale Operatoren	–	nein
2	Conrad/Turowski [4, 5]	OCL + temporale Operatoren	–	nein
3	Kleppe/Warmer [13]	OCL + *action clause*	–	nein
4	Bradfield et al. [3]	OCL + temporale Templates	Observational μ-Calculus	nein
5	Distefano et al. [8]	ähnlich CTL	BOTL	nein
6	Roubtsova et al. [18, 17]	parametrisierte Stereotypen	TCTL	ja
7	Sendall/Strohmeier [19]	OCL-konsistent	–	ja
8	Flake/Müller [11, 12]	OCL-konsistent	CCTL	ja

Der formale zeitbehaftete Ansatz 6 von Roubtsova et al. nutzt eine UML-konforme Erweiterung über parametrisierte stereotypisierte Klassen und erfüllt zwar alle gewünschten drei Eigenschaften, konzeptuell erscheint es jedoch nicht erstrebenswert, eine jede temporale Eigenschaft als separate Klasse zu modellieren.

Unser eigener Vorschlag 8 erfüllt ebenfalls die drei gewünschten Eigenschaften. Dabei benutzen wir einen Pfadbegriff, der sich im Gegensatz zu dem in Abschnitt 2 aufgeführten Zitat von Roubtsova et al. sehr gut in die bestehenden Konzepte von OCL integriert. Unser Ansatz sollte jedoch noch um weitere Spezifikationsmöglichkeiten ergänzt werden, z.B. hinsichtlich vergangenheitsbezogener Pfade und in Bezug auf Zeitbeschränkungen einzelner Zustands*übergänge*.

4 Weitergehende temporale OCL-Erweiterungen

Die vorgestellten Arbeiten unterscheiden sich zum einen im Hinblick auf die jeweils eingeführten temporalen Konzepte. So führen einige Ansätze nützliche Templates ein, während andere ganze temporale Logiken umfassen, wenn diese meist auch nur implizit und nicht formal definiert sind. Zum anderen kann man eine Unterscheidung bezüglich der Zeitausrichtung treffen; einige Arbeiten umfassen sowohl zukünftige als auch vergangene Ereignisse und Ausührungen von Aktionen. Ein weiterer Aspekt betrifft die betrachteten Modellelemente; einige Ansätze – darunter auch unser eigener – beschränken sich auf die Spezifikation von temporalen Einschränkungen in Zustandsdiagrammen. Es gilt daher zu untersuchen, ob die einzelnen Erweiterungen zusammengeführt und z.B. auch auf andere Diagrammarten angewendet werden können.

Dies sollte auf Basis eines OCL-Metamodells geschehen, das es jedoch noch nicht in einer standardisierten Form gibt. Zurzeit werden Metamodell-Vorschläge für die OCL Version 2.0 von der Object Management Group (OMG) überprüft. Dabei hat der Vorschlag von Warmer et al. [20] sehr gute Chancen akzeptiert zu werden, weil dort neben einem umfassenden Metamodell auch eine Semantik für wesentliche Teile der OCL definiert wird. Weiterhin sind in diesen Vorschlag auch die bereits erwähnten *action clauses* eingearbeitet. Die Semantik hierfür ist jedoch noch nicht klar definiert.

Arbeitsaufgaben. Im Einzelnen kann unsere temporale Erweiterung der OCL unter folgenden Aspekte näher untersucht werden:

1. Integration der temporalen Erweiterung in das (noch zu standardisierende) OCL-Metamodell. Eine Alternative zu einer direkten Erweiterung im Metamodell kann ein UML-Profile sein, das das OCL-Metamodell mit den UML-Erweiterungsmechanismen wie Stereotypen, Tagged Values und Constraints entsprechend ergänzt. Allerdings ist nicht vollständig definiert, wie bei der Definition eines solchen Profiles formal vorzugehen ist.

2. Einbeziehung vergangenheitsbezogener, zeitbehafteter temporaler Operatoren bzw. Operationen. Manchmal ist es für Modellierer einfacher, eine Modelleigenschaft in Bezug auf vergangene Zustände (bzw. Aktionen oder Ereignisse) zu spezifizieren.

   Damit einher geht die Angabe einer Semantik. Für die neuen temporalen Operatoren bzw. Operationen in OCL muss eine Abbildung in einen mathematisch fundierten Formalismus angegeben werden.

3. Anwendung von temporalen OCL-Erweiterungen auf weitere verhaltensorientierte Diagramme, insbesondere Aktivitätendiagramme, in denen das objektübergreifende parallele Modellverhalten im Vordergrund steht.

4. Temporale Templates an sich sind keine neue Errungenschaft. Dwyer et al. haben beispielsweise einen Katalog mit Spezifikationsmustern erstellt, der auf einer Vielzahl von praxisrelevanten Spezifikationen basiert [9, 10]. Zu jedem Muster sind dort temporallogische Formeln angegeben. Es bleibt zu untersuchen, inwieweit es ausreicht, in OCL eine begrenzte Auswahl an temporalen Spezifikationsmustern anzubieten, oder ob ein allgemeinerer Ansatz nötig ist.

5. Dadurch, dass die UML kein inhärentes Zeitmodell besitzt, ist ein Bezug von zeitbehafteten Eigenschaften zum Rest der UML nicht direkt gegeben. Es gibt jedoch seit kurzer Zeit ein offizielles *UML Profile for Scheduling, Performance, and Time* [15], das auch Echtzeiterweiterungen aufgreift. Es sollte untersucht werden, ob die temporale OCL-Erweiterung dieses Profile erweitern könnte.

5 Fazit

Wir werden auch weiterhin auf der Erweiterung von OCL im Hinblick auf zeitbehaftete temporale Eigenschaften forschen. Dadurch, dass wir im Projekt GRASP[2] im Bereich der Modellierung von Produktionsautomatisierungssystemen arbeiten und in diesem Zusammenhang die Modellierungssprache MFERT [7] anwenden, wollen wir uns in der Zukunft vornehmlich der oben genannten Aufgabe 2 widmen, um Modellierern weitere geeignete Spezifikationsmittel zur Hand zu geben. Die formale Integration von MFERT und der OCL-Erweiterung könnte dabei durch die Beschreibung eines weiteren UML-Profiles für MFERT unterstützt werden.

Im Bereich der Aufgabe 3 kooperieren wir mit dem Projekt SafeRail[2], indem wir gemeinsam die dort identifizierten domänenspezifischen Safety-Patterns [2] untersuchen und auf unsere OCL-Erweiterung abbilden.

Literatur

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

[2] F. Bitsch. Safety Patterns - The Key to Formal Specification of Safety Requirements. In *20th International Conference SAFECOMP 2001 - Computer Safety Reliability and Security*, Budapest, Hungary, September 2001. Springer-Verlag, Belin.

[3] J. Bradfield, J. Küster Filipe, and P. Stevens. Enriching OCL using observational mu-calculus. In *Proceedings of FASE 2002*, Lecture Notes in Computer Science, Grenoble, France, April 2002. Springer-Verlag, Heidelberg. (to appear).

[4] S. Conrad and K. Turowski. Vereinheitlichung der Spezifikation von Fachkomponenten auf der Basis eines Notationsstandards. In J. Ebert and U. Frank, editors, *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik (Beiträge des Workshops Modellierung 2000)*, pages 179–194, St. Goar, April 2000. Koblenzer Schriften zur Informatik, Band 15, Fölbach-Verlag, Koblenz. (in German).

[5] S. Conrad and K. Turowski. Temporal OCL: Meeting Specifications Demands for Business Components. In K. Siau and T. Halpin, editors, *Unified Modeling Language: Systems Analysis, Design, and Development Issues*, pages 151–165. IDEA Publishing Group, 2001.

[6] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[7] W. Dangelmaier. Dynamic Modelling of Production Processes as Basis for Distributed Object-oriented Production Planning and Control. In M. D. Cin, U. Herzog, G. Bolch, and A. Kaylan, editors, *Proc. of 7th European Simulation Symposium ESS'95*, pages 615–620, Erlangen-Nürnberg, Germany, 1995.

[8] 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, *Proc. of FMOODS'2000 – Formal Methods for Open Object-Based Distributed Systems IV*, Stanford, CA, USA, September 2000. Kluwer Academic Publishers.

[2] Projekt im DFG-Schwerpunktprogramm 1064: Integration von Techniken der Softwarespezifikation für ingenieurwissenschaftliche Anwendungen (http://tfs.cs.tu-berlin.de/SPP/index.html)

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

[10] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specifications for Finite-State Verification. In *21st International Conference on Software Engineering, Los Angeles, California*, May 1999.

[11] S. Flake and W. Mueller. An OCL Extension for Real-Time Constraints. In *Object Modeling with the OCL*, volume 2263 of *Lecture Notes in Computer Science*, pages 150–171. Springer-Verlag, Heidelberg, Germany, February 2002.

[12] S. Flake and W. Mueller. Specification of Real-Time Properties for UML Models. In *Proc. of the Hawaii Internatinal Conference on System Sciences (HICSS-35)*, Hawaii, USA, January 2002. IEEE Computer Society Press.

[13] A. Kleppe and J. Warmer. Extending OCL to include Actions. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000*, volume 1939 of *Lecture Notes in Computer Science*, pages 440–450. Springer-Verlag, Heidelberg, 2000.

[14] OMG – Object Management Group. Unified Modeling Language 1.4 Specification. OMG Document formal/2001-09-67, September 2001. URL: http://www.omg.org/technology/documents/formal/uml.htm.

[15] OMG – Object Management Group. UML Profile for Scheduling, Performance, and Time Specification. OMG Document ptc/02-03-02, March 2002. URL: http://www.omg.org/technology-/documents/modeling_spec_catalog.htm.

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

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

[18] E. Roubtsova, J. van Katwijk, W. Toetenel, and R. de Rooij. Real-Time Systems: Specification of Properties in UML. In *ASCI 2001 Conference*, pages 188–195, Het Heijderbos, Heijen, The Netherlands, May 2001.

[19] S. Sendall and A. Strohmeier. Specifying Concurrent System Behavior and Timing Constraints Using OCL and UML. In M. Gogolla, editor, *UML 2001 - The Unified Modeling Language: Modeling Languages, Concepts and Tools, Fourth International Conference*, volume 2185 of *Lecture Notes in Computer Science*, pages 391–405, Toronto, Canada, October 2001. Springer-Verlag, Heidelberg.

[20] J. Warmer et al. Response to the UML2.0 OCL RfP (Revised Submission), Version 1.3. Technical report, Boldsoft, Rational, IONA, Adaptive Ltd. et al., March 2002. URL: http://-www.klasse.nl/ocl/ocl-specification-v1-3.zip.