Exploring the Platform for Expressing SystemVerilog Assertions in Model Based System Engineering

Muhammad Rashid, Muhammad Waseem Anwar, Farooque Azam and Muhammad Kashif

Abstract Despite the importance of Model Based System Engineering (MBSE) for early design verification, it is always challenging to represent the system properties / constraints at higher abstraction level due to complex behavioral and temporal aspects of embedded systems. To manage this, OCL (Object Constraint Language) and CCSL (Clock Constraint Specification Language) have been frequently used. On the other hand, SystemVerilog is a renowned hardware design and verification language that supports Assertion Based Verification (ABV). However, no real efforts have been made to employ SystemVerilog Assertions (SVA's) in MBSE. In this paper, we explore various possibilities to represent SVA's at higher abstraction level. Firstly, we evaluate the existing property specification approaches to represent SVA's at higher abstraction level. Consequently, we select OCL as an appropriate approach. Secondly, we investigate the syntax and semantics of OCL in the context of SVA's. The outcomes of research provide the sound platform to represent SVA's in OCL for model-based design verification of embedded systems.

Keywords MBSE · OCL · CCSL · SystemVerilog Assertions · Design verification

M. Rashid(✉)
College of Computer and Information Systems,
DCE, Umm Al-Qura University, Mecca, Saudi Arabia
e-mail: mfelahi@uqu.edu.sa

M.W. Anwar · M. Kashif
MODEVES Project, National Science,
Technology and Innovation Plan, Riyadh, Saudi Arabia
e-mail: waseemanwar@hotmail.com, muhammadkashif038@gmail.com

F. Azam
CEME, DCE, National University of Sciences and Technology, H-12, Islamabad, Pakistan
e-mail: farooq@ceme.nust.edu.pk

K.J. Kim and N. Joukov (eds.), *Information Science and Applications (ICISA) 2016*,
Lecture Notes in Electrical Engineering 376,
DOI: 10.1007/978-981-10-0557-2_53

1 Introduction

Model Based System Engineering (MBSE) is frequently practiced with static and/or dynamic verification techniques [1] for embedded systems design. MBSE dynamic verification process starts by identifying the embedded systems requirements [13], [48] so that the structural and behavioral aspects can be modelled. UML (Unified Modeling Language) [2], SYSML (Systems Modeling Language) [3] and MARTE (Modeling and Analysis of Real-Time and Embedded systems) [4] profiles are the leading approaches [5-7] to specify structural and behavioral aspects. Another important task in modelling is the abstract specification of system properties / constraints. OMG (Object Management Group) [9] standards like OCL (Object Constraint Language) [10] and CCSL (Clock Constraint Specification Language) [11] have been frequently used for property specification in the domain of embedded systems [12 -14].

On the other hand, SystemVerilog [16] is an emerging hardware design and verification language that supports Assertion Based Verification (ABV). Despite the proven ABV capabilities of SystemVerilog, it is rarely utilized (e.g. [17-18]) for early design (dynamic) verification in the context of MBSE for embedded systems. The primary reason is that the semantics / syntax of existing property specification approaches are entirely different as that of System Verilog Assertions (SVA's). Consequently, it is difficult to represent SVA's at higher abstraction level.

To target this problem, in this paper, we provide a basic platform for expressing SVA's at higher abstraction level by performing: (a) a comparative study of leading property / constraint specifications techniques and (b) evaluation of syntax and semantics of SVA's. It will allow not only to express SVA's at higher abstraction level but to develop correct transformation rules in order to derive SVA's from abstract property specification due to the matching semantics / syntax of property specification approach and SVA's.

The proposed research work starts with the search process in scientific databases (i.e. IEEE, Springer, Elsevier, and ACM) to identify the leading abstract property specification techniques in the domain of MBSE for embedded systems. As a result, MARTE stereotypes / annotations, CCSL, SYSML parametric diagram and OCL have been identified (Section 2.1). For comparative study of the leading techniques, we identify certain parameters such as complexity, tools support, applicability, extendibility etc. (Section 2.2). The results of comparative study have allowed to select OCL for further investigations. Once the OCL is selected, we evaluate/compare syntax and semantics of OCL with respect to SVA's (Section 3) to ensure the applicability of our proposal. It has been analyzed that SystemVerilog basic operators and control statements (Section 3.2) can be logically represented in OCL. However, SystemVerilog sampled value functions, sequence and repetition operators cannot be directly represented in OCL (Section 3.3). For such SystemVerilog constructs, we propose the possibility of OCL extension. Significant aspects of research have been discussed in Section 4. Finally, Section 5 concludes the paper.

2 Investigation of Existing Property Specification Approaches

This section investigates various property specification techniques, through a Systematic Literature Review (SLR), to represent embedded systems properties / constraints at higher abstraction level. The details of the SLR can be found at [8]. In subsequent sections, we first summarize the leading techniques in Section 2.1 and then perform comparative analysis in Section 2.2.

2.1 Summary of Leading Property Specification Approaches

It has been concluded from the SLR that MARTE stereotypes / annotations, CCSL, SYSML parametric diagram and OCL are leading property specification techniques:

MARTE Stereotypes / Annotations. It has been analyzed that MARTE stereotypes / annotations have been frequently used to specify embedded systems properties / constraints at higher abstraction level. Basically, UML and SYSML diagrams have been customized through MARTE stereotypes / annotations to capture embedded systems properties / constraints at higher abstraction level. For example, Andrade et al. [19], Moreira et al. [20] and Xiaopu et al [21] customize activity diagram, sequence diagram and state machine diagram respectively, through MARTE annotations, to capture properties / constraints at higher abstraction level.

Clock Constraint Specification Language (CCSL). The CCSL [11] has been first introduced in MARTE profile to express the time constraints. The Synchronous languages [22] and their polychromous extensions [23] provide the basis for CSSL. It has been analyzed that CCSL has been frequently used for embedded systems property specification. For example, [6], [24] and [25] utilize CCSL to represent system properties / constraints at higher abstraction level.

SYSML Parametric Diagram. We analyzed that SYSML parametric diagram has also been used to specify properties / constraints at higher abstraction level. For example, Berrani et al. [26] capture system properties / constrains through SYSML parametric diagram. Modelica [27] code is generated to perform dynamic verification. Similarly, Rahman et al. [28] utilize parametric diagram for abstract property specification.

Object Constraint Language (OCL). As OCL is an OMG standard, it has been frequently put into practice for the specification of embedded systems constraints / properties. For example, Iqbal et al. [29] propose Real Time Embedded Systems (RTES) scheme to model constraints, structure and behavior of the environment in UML / MARTE and OCL. Erwan et al. [30] present novel model verification approach by specifying safety critical systems properties / constraints in OCL.

Temporal Extensions of OCL. Although OCL has enhanced features to specify constraints at higher abstraction level, it has certain limitations while dealing with the temporal properties of embedded systems. Consequently, temporal aspects of embedded systems cannot be expressed in OCL unless it is extended. It has been analyzed that OCL extensions can be classified into three major types i.e. Template based, OCL meta-level based and Pattern based. Template based OCL extensions utilize template clauses to represent temporal dimensions [32], [33]. In Meta-level based OCL extensions, new classes and functions / operators have been added in the existing meta-level of OCL language while preserving the core OCL semantics as defined in the meta-level [35]. In Pattern based OCL extensions, property specification patterns have been employed to extend OCL for temporal dimensions [37-39].

Other Property Specification Techniques. In this section, we summarize the various property specification approaches that have not been practiced frequently. For example, we found an interesting research work by Daniel et al. [5] in which TEPE language has been proposed to graphically express temporal properties through parametric diagrams of SysML. We have also found different property specification techniques like Probabilistic Computation Tree Logic (PCTL) [42], Linear Temporal Logic LTL [43] and Property Specification Language (PSL) [44] which have been used by Samir et al. [45], Siveroni et al [46] and Guglielmo et al. [47] respectively. However, we analyzed that such property specification approaches should not be considered at higher abstraction level because properties / constraints are not represented in pure abstract design rather properties are specified in between higher and lower abstraction level.

2.2 Comparative Analysis

We develop five significant evaluation parameters for comparative analysis of leading property specification techniques as follows:

1. **Applicability:** It is defined as the ability of approach to express the properties / constraints of embedded systems in the UML-based models. “YES” means it can be used in UML-based models. “NO” means it cannot be used in UML-based models. This parameter ensures that the property specification approach should be compatible with the leading modeling language.

2. **Representation Complexity:** It is defined as the complexity of constructs, semantics and syntax of property specification approach. This parameter gives us a clear idea of the complexity involved to express properties / constraints at higher abstraction level. It is evaluated as low, high and very high accordingly where “low” means it is simple to express properties, “high” means it is difficult to express properties and “very high” means it’s very difficult to express properties / constraints.

3. **Transformation Complexity:** It is defined as the complexity involved to transform the abstract properties / constraints into target dynamic verification platform. This parameter is mandatory to evaluate because some properties specification approaches have very low representation complexity but it is very challenging to attain desired dynamic verification platform due to their very high transformation complexity. Consequently, it is evaluated as low, high and very high accordingly.

4. **Extension:** It is defined as the possibility to extend the particular property specification approach in order to meet some specific objectives. For example, OCL doesn't support temporal aspects of embedded systems but researchers intensively extend it to support temporal aspects. Extension can be evaluated as No, Partial and Full. "No" means that the approach does not support any extensions. "Partial" means the approach can be extended to meet particular objectives with some limitations. "Full" means the approach can be customized to support broad extensions.

5. **Tool Support:** It is defined as the range of tools that support a particular approach. It is evaluated as low, medium and high. Various factors have been considered while evaluating tools support like support of editor to express constraints / properties, syntax checking, validation and possible customization of tool to accommodate the extension of the approach.

The above mentioned evaluation parameters facilitate us to select the appropriate property specification approach in order to represent SVA's at higher abstraction level. The evaluation results, summarized in Table 1, reveal that all the leading techniques provide support to represent embedded systems properties / constraints in the UML-based models. It has also been analyzed that all the evaluation parameters (except Applicability) are directly related with Tools support parameter. For example, representation complexity is increased if property specification editor is not available. Similarly, transformation complexity is also increased in the absence of appropriate transformation tools.

Table 1 Comparative study of leading property specification approaches

Parameter	**MARTE Stereotypes / annotations**	**CCSL**	**SYSML Parametric Diagram**	**OCL**
Applicability	YES	YES	YES	YES
Representation Complexity	High	Very High	Very High	High
Transformation Complexity	Very High	Very High	Very High	High
Extension	Partial	Partial	Partial	Full
Tools support	Medium	Medium	Medium	High

We analyzed that MARTE stereotypes / annotation has high representation complexity because it is required to logically combine MARTE stereotypes / annotations with semantics of different UML diagrams. Consequently, it is difficult to develop transformation rules due to mix behavioral and verification aspects that leads to very high transformation complexity. MARTE stereotypes can be partially extended to achieve a particular property specification objective. There are sufficient modeling tools available for MARTE profile like papyrus [50]. Furthermore, different existing transformation tools (e.g. QVTO [51]) can be used to develop customized tools.

It is investigated that CCSL has very high representational complexity due to its difficult syntax and semantics. Furthermore, it is required to understand CCSL semantics before developing the transformation rules that makes the transformation complexity very high. However, it is possible to partially extend CCSL. For example, Ling et al. [52] propose CCSL extension (ccLTS) for embedded systems design verification. There is sufficient tools support available for CCSL. For example, TimeSquare [53] tool is explicitly developed for CCSL specifications that directly simulates the design without model transformation.

It is very difficult to model embedded systems constraints in SYSML parametric diagram due to its complex semantics. Consequently, the transformation complexity is also very high. However, there are tools available to directly simulate the SYSML parametric diagram for dynamic verification e.g. Rhapsody Parametric Constraint Evaluator (PCE) [54]. It can be argued that both SYSML parametric diagram and CCSL support direct simulation that results in low transformation complexity because transformation is not required at all. It is also possible to partially extend SYSML parametric diagram [5].

We analyze that the syntax and semantics of OCL are very close to most of the target dynamic verification platform (e.g. VHDL, SystemC etc.). However, it is still required to study OCL standard library before expressing the properties / constraints. There are various tools available to support transformation of OCL constraints (e.g. MOFScript [55]). Moreover, tools are available to directly evaluate OCL constraints for dynamic verification (e.g. OCL constraint solver [40]). Furthermore, it is possible to customize standard OCL library through Xtext [49] to support OCL extensions. The tool support for OCL is high. OCL fully supports extensions to support particular embedded systems property specification objective. Based on the above analysis, we select OCL to represent SVA's at higher abstraction level.

3 Investigating OCL Semantics/Constructs for SystemVerilog

3.1 Significant SystemVerilog Constructs

SystemVerilog is a Hardware Description Language (HDL) that also provides enhanced verification features. In the following, we identified the important constructs of SystemVerilog, which are mandatory to design simple as well as complex SVA's.

1. **Basic operators and control statements:** The SystemVerilog provides set of basic operators and control statements which are partially similar as that of other programming languages. These operators and control statements are used to develop hardware design and assertions. We investigate almost all basic operators of SystemVerilog like arithmetic (e.g. multiply, division etc.), relational (e.g. greater than, less than, etc.), logical (e.g. and, or etc.) and equality operators. Table 2 summarizes the SystemVerilog basic operators with equivalent OCL operators. Similarly, we have also identified four SystemVerilog control

statements (i.e. 1) *If else* 2) *Case* 3) *While* 4) *For Loop*) for further analysis (Section 3.2).

2. **SystemVerilog sampled value functions:** SystemVerilog provides a set of sampled value functions to evaluate current values with respect to past values. These functions are frequently used for hardware design and verification (assertions). We identify five important sampled value functions (i.e. *$sampled, $fell, $rose, $past*, and *$stable*) for further investigation (Section 3.3).

3. **SystemVerilog sequence, repetition and implication operators:** SystemVerilog sequence and repetition operators are usually used to manage past and future temporal dimensions. Furthermore, SystemVerilog provide two types of implications (i.e. *Overlapped Implication (*|->*)* and *Non-Overlapped Implication (*|=>*))* to design / check the occurrence of particular sequence of conditions. We consider most common SystemVerilog sequence and repetition operators for further investigation i.e. Consecutive repetition *([*])*, Goto repetition *([->), ##, [*], [=]* and *[->]*. We also consider both types of SystemVerilog implication for further investigation.

3.2 Similarities Between SystemVerilog and OCL Constructs

Basic Operators. It has been analyzed that most of the basic operators of both SystemVerilog and OCL have similar semantics. Results are summarized in Table 2. Furthermore, SystemVerilog overlapping implication can be managed through OCL implies operator.

Table 2 Similarities between basic operators of SystemVerilog and OCL

Sr. #	Type of Operator	SystemVerilog Operators	Equivalent OCL Operators	Operator Description
1	**Arithmetic**	*	*	Multiply
		/	/	Division
		+	+	Add, Unary plus
		-	-	Subtract, Unary minus
2	**Comparison**	>	>	Greater than
		<	<	Less than
		>=	>=	Greater than or equal
		<=	<=	Less than or equal
3	**Boolean**	&&	and	Logical And
		!	not	Logical Not
		||	or	Logical OR
		|->	implies	Implies
		^	xor	Exclusive OR
4	**Equality**	==	=	Equality
		!=	<>	inequality

Control Statements. It has been analyzed that OCL does not directly support SystemVerilog control statements like *foreach* etc. However, we examine that the OCL loop operations on collection types can be used to logically represent particular SystemVerilog control statements. For example, OCL *forAll* operation on collection types returns true if expression is true for all elements in the collection. Consequently, SystemVerilog *foreach* statement can be logically expressed in OCL through *forAll* operation on collection types. Similarly, other OCL loop operations on collection types (e.g. *Any*, *Exists* etc.) can also be used to represent different SystemVerilog control statements. The details of OCL loop operations on collection types and description of OCL iterator variables can be found in OCL specifications version 2.4 [56]. We also analyze that SystemVerilog *case / if else* statements can be directly expressed in OCL *IfExp*.

3.3 Dissimilar Semantics of SystemVerilog and OCL

1. **SystemVerilog sampled value functions:** These SystemVerilog functions are based on past temporal dimensions and cannot be expressed in OCL. For example, SystemVerilog *$past* function returns any past value of expression as per given past number of clocks. However, in OCL, we can only access the last preceding value through *@pre* operator with certain restrictions (e.g. it is mandatory to use *@pre* operator in post condition). Consequently, SystemVerilog sampled value functions cannot be expressed in OCL without its appropriate past temporal extension.
2. **SystemVerilog non-overlapping implication (|=>):** In this type of implication, the expression on right hand side is evaluated after one clock tick on the occurrence of left hand side expression. On the other hand, OCL doesn't support any kind of future temporal dimensions. Consequently, non-overlapping implication cannot be expressed in OCL without its appropriate future temporal extension.
3. **SystemVerilog sequence and repetition operators:** These operators are commonly used with past / future temporal dimensions. For example, SystemVerilog expression a ##7 b means that b should be true after seven clock ticks of a. However, OCL doesn't support future temporal dimensions. Consequently, it is mandatory to extend OCL for past and future temporal dimensions in order to handle SystemVerilog sequence and repetition operators.

4 Discussion and Limitations

The objective of this research is to logically represent SVA's in OCL so that the target SystemVerilog code can be generated with minimum transformation efforts. Of course SystemVerilog is a complete design / verification language and it is not possible to represent all SystemVerilog semantics in OCL. For example, in SystemVerilog, both overlapping and non-overlapping implications have the concept of vacuous success. However, it is not necessary to include such

SystemVerilog semantics in OCL because the ultimate dynamic design verification is performed in SystemVerilog. Therefore, the main purpose is to logically represent SVA's in OCL to ensure the simplicity and correctness of model transformation for the target SystemVerilog code.

Although OCL provides support to basic SystemVerilog constructs and control statement, it is not sufficient to represent SVA's due to missing past and future temporal dimensions. Therefore, it is mandatory to extend OCL for inevitable temporal aspects of embedded systems. We already highlighted different research works (2.1) where OCL is extended to deal with the complex temporal aspects. Consequently, it can be concluded that the development of such OCL extension is practicable. Furthermore, it is easy to tailor OCL extension in the context of SVA's as basic semantics / syntax of SystemVerilog operators and control statements have already been supported by OCL (Section 3.2). We are currently working on OCL extension for SystemVerilog and represent it in our next article.

5 Conclusions and Future work

In this paper, we explored various property specification techniques and provided a basic platform to represent SystemVerilog Assertions (SVA's) at higher abstraction level in the context of MBSE for embedded systems. Analysis of existing techniques identified the four leading approaches: (i.e. MARTE stereotypes / annotations, CCSL, SYSML parametric diagram and OCL). A comprehensive comparative study of the leading techniques allowed to select OCL for SVA's. Subsequently, we investigated semantics / syntax of OCL constructs with respect to SystemVerilog constructs for representation of SVA's. It has been concluded that OCL supports the representation of SystemVerilog constructs like basic operators and control statements. It has also been analyzed that some SystemVerilog constructs like sampled value functions and sequence / repetition operators cannot be represented in OCL unless it is extended for past and future temporal dimensions. This research provided the basis/platform for researchers and practitioners of the domain to introduce SystemVerilog Assertions in MBSE for embedded systems. We are currently working on OCL extension for SystemVerilog and intend to present the details in our next article.

Acknowledgments This project is funded by NSTIP (National Science Technology, Innovative Plan), Saudi Arabia under the Technology Area "Information Technology Strategic Priorities" and Track "Software Engineering and Innovated Systems". We acknowledge the support of KACST (King Abdulaziz City for Science and Technology) and STU (Science and Technology Unit) Makkah (grant no. 13-INF761-10).

References

1. Rashid, M., Anwar, M.W., Khan, A.M.: Towards the Tools Selection in Model Based System Engineering for Embedded Systems - A Systematic Literature Review. JSS **106**, 150–163 (2015)
2. Booch, G., Rumbaugh, J., Jacobson, I.: UML, User Guide. Addison-Wesley (1999)
3. OMG, SysML (2008). http://www.omg.org/spec/SysML/1.3/ (retrieved March 6, 2012)
4. OMG, UML Profile for MARTE, v1.0, November 2009, formal/2009-11-02
5. Knorreck, D., Apvrille, L.: TEPE: A SysML Language for Time-Constrained Property Modeling and Formal Verification. ACM SIGSOFT **36**(1), 1–8 (2011)
6. Ge, N., Pantel, M., Cregut, X.: Formal specification and verification of task time constraints for real-time systems. LNCS, vol. 7610, pp. 143–157 (2012)
7. Baresi, L., Blohm, G., Kolovos, D.S., Matragkas, N., Motta, A., Paige, R.F., Radjenovic, A., Rossi, M.: Formal verification and validation of embedded systems: the UML-based MADES approach. SSM Springer (2013)
8. MODEVES project, details of SLR. http://www.modeves.com/slr2.html
9. Object management Group: http://www.omg.org/ (accessed September 2015)
10. Object Constraint Language: http://www.omg.org/spec/OCL/ (accessed September 2015)
11. Clock Constraint Specification Language (CCSL): http://www.omgmarte.org/node/66 (accessed September 2015)
12. Rashid, M., Anwar, M.W., Khan, A.M.: Identification of trends for model based development of embedded systems. In: 12th IEEE International Symposium on Programming and Systems (ISPS), April 2015, Algiers, pp. 1–8 (2015)
13. Rashid, M., Pottier, B.: A multi-objective framework for characterization of software specifications. In: Embedded and Real Time System Development: A Software Engineering Perspective: Concepts, Methods and Principles. Springer, pp. 185–209 (2014)
14. Rashid, M.: An efficient cycle accurate performance estimation model for hardware software co-design. In: Embedded and Real Time System Development: A Software Engineering Perspective: Concepts, Methods and Principles. Springer, pp. 213–234 (2014)
15. Rashid, M., Ferrandi, F., Bertels, K.: HARTES design flow for heterogeneous platforms. In: Proceedings of the 10th International Symposium on Quality of Electronic Design (ISQED 2009), CA, USA, March 2009, pp. 330–338 (2009)
16. IEEE SystemVerilog: http://standards.ieee.org/getieee/1800/download/1800-2012.pdf
17. Jin, N., Shen, C., Ni, T.: Engineering of An Assertion-based PSLSimple-Verilog Dynamic Verifier by Alternating Automata. ENTCS **207**, 153–169 (2008)
18. Li, L., Coyle, F.P., Thornton, M.A.: UML to SystemVerilog synthesis for embedded system models with support for assertion generation. In: ECSI Forum on Design Languages (2007)
19. Andrade, E., Maciel, P., Callou, G., Nogueira, B.: A methodology for mapping SysML activity diagram to time petri net for requirement validation of embedded real-time systems with energy constraints. In: ICDS 2009, pp. 266–271 (2009)
20. Moreira, T.G., Wehrmeister, M., Pereira, C.E., Pétin, J.F., Levrat, E.: Automatic code generation for embedded systems: from UML specifications to VHDL code. In: IEEE 8th INDIN 2010, pp. 1085–1090 (2010)

21. Huang, X., Sun, Q., Li, J., Pan, M., Zhang, T.: An MDE-based approach to the verification of SysML state machine diagram. In: APSIS 2012, p. 9. ACM (2012)
22. Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., De Simone, R.: The synchronous languages 12 years later. Proc. of the IEEE **91**(1), 64–83
23. André, C., Mallet, F., Deantoni, J.: VHDL observers for clock constraint checking. In: ISIE, pp. 98–107
24. Peters, J., Wille, R., Drechsler, R.: Generating SystemC implementations for clock constraints specified in UML/MARTE CCSL. In: 19th ICECCS 2014, pp. 116–125 (2014)
25. Suryadevara, J.: Validating EAST-ADL timing constraints using UPPAAL. In: 39th Euromicro Conference SEAA 2013, pp. 268–275 (2013)
26. Berrani, S., Hammad, A., Mountassir, H.: Mapping SysML to modelica to validate wireless sensor networks non-functional requirements. In: 11th ISPS 2013, pp. 177–186 (2013)
27. Modelica Language: https://www.modelica.org/ (accessed October 2015)
28. Rahman, M.A.A., Nor, N.S.M., Mizukawa, M.: Evaluation for SysML-based design and analysis models using PCE. In: ICCSCE 2012, pp. 339–344 (2012)
29. Iqbal, M.Z., Arcuri, A., Briand, L.: Environment modeling and simulation for automated testing of soft real-time embedded software. SSM (2013)
30. Bousse, E., Mentré, D., Combemale, B., Baudry, B., Katsuragi, T.: Aligning SysML with the B method to provide V&V for systems engineering. In: WMDEVV, pp. 11–16. ACM (2012)
31. Universal Verification Methodology: http://accellera.org/downloads/standards/uvm
32. Bradfield, J., Filipe, J. K., Stevens, P.: Enriching OCL using observational mu-calculus. LNCS, vol. 2306, pp. 203–217 (2002)
33. Mullins, J., Oarga, R.: Model checking of extended OCL constraints on UML models in SOCLe. LNCS, vol. 4468, pp. 59–75 (2007)
34. Mentor Graphics: http://www.mentor.com/ (accessed October 2015)
35. Lavazza, Luigi, Morasca, Sandro, Morzenti, Angelo: A Dual Language Approach to the Development of Time-Critical Systems. ENTC **116**, 227–239 (2005)
36. Cengarle, M.V., Knapp, A.: Towards OCL/RT. LNCS, vol. 2391, pp. 390–409
37. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: 21st ICS, pp. 411–420 (1999)
38. Dou, W., Bianculli, D., Briand, L.: OCLR: a more expressive, pattern-based temporal extension of OCL. LNCS, vol. 8569, pp. 51–66 (2014)
39. Kanso, B., Taha, S.: Specification of temporal properties with OCL. Science of Computer programming **96**, Part 4, pp. 527–551 (2014)
40. Ali, S., Iqbal, M.Z., Arcuri, A., Briand, L.: A Search-based OCL constraint solver for model-based test data generation. In: The 11th International Conference on Quality Software (2011)
41. Bill, R., Gabmeyer, S., Kaufmann, P., Seidl, M.: Model checking of CTL-extended OCL specifications. LNCS, vol. 8706, pp. 221–240 (2014)
42. Baier, C., Katoen, J.P.: Principles of model checking. The MIT Press (2008)
43. Emerson, E.: Temporal and modal logic. In: Leeuwen, J.V. (ed.) HTCE. MIT Press (1990)
44. IEEE PSL Std.: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5446004
45. Ouchani, S., Mohamed, O.A., Debbabi, M.: A formal verification framework for SysML activity diagrams. ESA, 2713–2728 (2014)

46. Siveroni, I., Zisman, A., Spanoudakis, G.: Property specification and static verification of UML models. In: 3rd ICARS 2008, pp. 96–103 (2008)
47. Di Guglielmo, G., Di Guglielmo, L., Foltinek, A., Fujita, M., Fummi, F., Marconcini, C., Pravadelli, G.: On the integration of model-driven design and dynamic assertion-based verification for embedded software. JSS, 2013–2033 (2013)
48. Rashid, M., Picard, D., Pottier, B.: Application analysis for parallel processing. In: Proceedings of the 11th Euro Micro Conference on Digital System Design, Architectures, Methods and Tools (DSD 2008), Parma, Italy, September 2008, pp. 633–640 (2008)
49. Eciplse XText Editor: https://eclipse.org/Xtext/ (accessed October 2015)
50. Papyrus MDT: http://www.eclipse.org/modeling/mdt/papyrus/
51. OMG, M2M/Operational QTV: https://projects.eclipse.org/projects/modeling.mmt.qvt-oml
52. Yin, L., Liu, J., Ding, Z., Mallet, F., De Simone, R.: Schedulability analysis with CCSL specifications. In: 20th APSEC 2013, pp. 414–421 (2013)
53. TimeSquare: http://timesquare.inria.fr/ (accessed August 2015)
54. IBM: Rational PCE. http://www.304.ibm.com/support/docview.wss?uid=swg27018723
55. MofScript: http://www.eclipse.org/gmt/mofscript/ (accessed August 2015)
56. Object Constraint Language specification version 2.4. http://www.omg.org/spec/OCL/2.4/