Thesis proposal for MAP-I 2011/2012 doctoral program

André de Matos Pedro

April 11, 2012

Abstract

Design by contract (DbC) is a technique being developed in the last decades to ensure software correctness through the specification, during design, of the required conditions and guaranteed behavior of software components. It is a technique that is applied with success, not only, but also, in safety-critical systems. There are several modeling languages for DbC that support the specification of the functional properties of a system. Nevertheless, the specification and verification of contracts for non-functional properties, such as timing constraints, do not have yet the same maturity level, particularly for the case of dynamic analysis approaches, which intend to enforce that the system behaves correctly, on a certain degree, on non-expected run-time scenarios. This work will address this problem, and intends to propose a specification-refined language (e.g. based in real-time logics, like RTL, MTL and TECTL) together with the underlying enforcement mechanisms to specify and verify at run-time non-functional requirements of hard real-time systems.

1 Thesis Proposal

Dynamic contracts for verification and enforcement of real-time systems properties

Supervisors

- TBD

1.1 Introduction

The last two decades have witnessed an immense increase in research activities in the area of static analysis [26], where innumerous theories and methods have been developed to verify both sequential and concurrent programs [1]. However, these techniques proved to be hard, expensive and non intuitive for the common programmer. The exploration of other techniques, such as dynamic analysis, will be necessary in order to decrease the burden of program verification, alternatively (or as a complement) verifying correctness during the execution [21, 12].

In the context of formal verification, dynamic analysis also emerged as a means to enforce that the system behaves well on non expected run-time situations, and techniques such as monitoring were actively developed in order to support such run-time verification process. Run-time verification is thus a join between formal verification and run-time execution of a system. It is complementary to the problem of model checking, i.e., we have to be able to create a model automatically that verifies a set of logical properties (a contract), and the created contract (the model) can then monitor the implementation of a system during run-time.

These monitoring techniques are considered [21, 12] in many cases an alternative to theorem proving [30] and model checking [2, 25], and a complement in safety-critical systems as a means of increasing their reliability.

This work is part of the VipCore project (PTDC/EIA-CCO/111799/2009) that intends, as one of the goals, to specify approaches for the correct design of component-based real-time systems. In this context, design by contract and run-time verification are important approaches to support the composition and isolation of independently developed components. Particularly, for mixed-criticality systems, it will be possible to verify, during execution, that lower-criticality components do not interfere with higher-criticality ones.

1.2 Objectives

Although formal methods are increasingly recognized by software engineers as being essential in the software development for safety-critical systems, a transparent manner to combine the formal methods with the standard process of software development proves to be a challenge for mathematicians and engineers. Design by contract (DbC) is not an exception, and establishes a compromise between the complexity and expressiveness of the contract specification language and the formal verification analysis (i.e., completeness, soundness, and decidability).

The main approach of this thesis will be to identify and address safety and time constraints that can be contractualized and automatically verified in run-time. In this context, the proposal aims to

- propose a specification language for dynamic contracts that enforces non-functional requirements such as time constraints and system resources distribution for real-time systems,
- explore a run-time strategy using software monitoring to verify and enforce these requirements,
- validate the proposed approach through a proof of concept implementation.

1.3 Brief review of the State of the Art

Real-time systems are those systems where the correctness of operation depends not only on the logical result of computation, but also on the instant at which the results are produced [34]. Several formal techniques have been devised [13, 36] to verify their correctness, considering temporal constraints. Nevertheless, and although there are a set of tools and languages which can be used to guarantee the functional requirements of real-time systems (such as Frama-C [27], SPARK [3], or Eiffel [22]), work still needs to be done for the non-functional requirements.

Design by Contract approaches. The aforementioned languages are examples of both static and dynamic contracts. The former implies a static analysis, i.e., the correctness of the program is assured before its execution, whilst the latter is checked during execution and being only able to ensure the correctness of past and current states of the system.

SPARK (a restricted subset of Ada, augmented with annotations) supports static contracts that describe the correct functional behavior of a sys-

tem. Nevertheless, in order to provide analysis decidability, a strict number of options is provided [18]. On the contrary, Eiffel is a language that supports dynamic contracts (it has been nevertheless recently extended to consider static contracts with aid of Boogie [35, 5]). Its authors have initially decided to support only dynamic contracts to avoid complex proofs and theories which could be undecidable for verifying pre-, postconditions, and invariants [23]. Thus, they created a code generator (AutoTest [24]) to produce the assertions (i.e., monitor clauses or constraints) to verify it in a run-time execution. Consequently, the guarantees are incomplete, i.e., it is only ensured that systems behave correctly on the execution paths that were already executed and checked, and not for all system executions. This approach is also recently followed by Ada 2012 [4], which supports directly in the language the specification of aspects (used for specifying pre-, postconditions and invariants).

Specification languages (logics) for real-time systems. A set of logics has already been put forward in order to specify properties of a system, which are able to formulate a formal specification according to given requirements. LTL (linear-time temporal logic) and CTL (computation tree logic) are logics used for model checking of non timed systems (i.e., verifies if a model is according to a given logical formula). Another interesting logic for properties specification is the *past temporal* LTL (PTLTL) [19]. However, real-time systems require the use of approaches that are powerful enough to deal with time constraints, therefore real-time variants of these logics are available, such as: RTL (real time logic) [17], TPTL (timed propositional temporal logic) [8], MTL (metric temporal logic) [8], and TECTL (timed event CTL) [37]. Other approaches not based on these logics could be applicable, but not specifically for real-time systems, such as: ERE (extended regular expressions) [31, 15] and CFG (context free grammars) (e.g., JML [20], ACSL [7] , E-ACSL [32], Praspel [11]).

These approaches can be used together with algorithms that generate models (according to a given formal specification), thus producing finite state automata that can be used to monitor the systems at run-time [29, 6].

Monitoring. A monitor process is a supervisor that observes the system behavior and checks if the system is consistent with a given specification [14]. Monitoring can be applied to non functional aspects of a system, such as time constraints, performance, and resource usage during execution.

Monitoring a system can be performed on-line (i.e., during execution) or off-line (i.e., during run-time a program trace is collected but the actual

monitoring process is performed after execution). The off-line analysis has the advantage of reducing the constraints the monitoring process imposes in the execution of the application. However, it is not able to enforce correct execution and isolate misbehaving components during execution. Concerning intrusiveness, the system can be monitored by in-line or out-line approaches. For instance, using a process that is independent of the system under observation (e.g. by using dedicated separate hardware), it is possible to analyze properties without interference in the system; therefore the monitoring is out-line [28]. Otherwise, the interference of the monitoring process always occurs, and in this case a monitor may be considered in-line (as is the case of MOP [10] and MaC [33] both run-time checkers for Java, and RMOR [16], a more recent work for the C language).

1.4 Work plan

The research plan for the thesis will include the following tasks:

Review of the state of the art. Detailed literature review concerning the DbC paradigm (such as Frama-C, SPARK, Eiffel and Ada 2012) and monitoring approaches (such as MOP, MaC or RMOR). Theoretical background complement in the area of model checking, and run-time verification. Careful study of specification languages suitable for program verification, such as logics, ERE and CFG (e.g., JML, E-ACSL), and model-based decision methods for real-time logics.

Design of a specification contract language. The specification language will be based in real-time logics, analyzed in the previous task. It should be expressive enough to represent the real-time requirements and the system resources control. The model for the run-time verification methodology for real-time systems will consider the Ada 2012 programming language where the contracts may be expressed using the developed specification language.

Development of an algorithm for automatic monitor generation from a given contract. Generating monitors from an automatic process is needed in order to check during execution the inherent properties. Methods such as [29, 6] will be used as basis to produce models from another logic specification. This development will be the base of run-time verification for real-time contracts.

Development of an on-line monitoring process Real-time systems require bounded and known computation times. Therefore, any on-line monitoring process must be combined with a scheduler for bounded and known interference on the system under analysis. There are two alternatives to combine monitoring processes with the system under analysis. Using a scheduler where interference is predicted and controllable or separating this problem into two distinct blocks by hardware (monitoring and system) [38]. The proposed work will address the former, where separate, dedicated, hardware is not required.

Validation of the developed run-time verification process. The validation of the proposed approach will be performed. It will incorporate the specification language, the generation of monitoring models from dynamic

contracts and the properly solution for an on-line and in-line/out-line monitor scheduler.

Lastly, the work will end with the thesis writing.

2 Thesis Planning

2.1 Review of the state of the art

Design by contract (DbC) Review of the recent work on DbC, more specifically concerning the verification of dynamic contracts. Thus, the exploration of tools such as Frama-C E-ACSL and Eiffel AutoTest, and languages such as Ada 2012 will be the basis for this topic.

Logics for real-time systems verification. Review of the recent work on specification languages and logics for run-time verification of programs (i.e., logics like MTL and TECTL).

Run-time verification - Monitoring by software. Review of the recent work on run-time verification, including automatic monitor generation given a set of logic formulas that specifies the system requirements. Scheduling for monitor process will also be reviewed here.

2.2 Evaluation and research plan.

Evaluation of the literature review undertaken, detailed specification of the research objectives and plan of the PhD dissertation, eventually with a revision of the tasks presented in the preliminary thesis proposal.

3 Free option

3.1 Topic

Concurrent and Real-Time Software

3.2 Plan

Software is increasingly the most complex component of any real-time system, and its correct behavior is of paramount importance, both in terms of functional and non-functional behavior and correctness. Real-time software is also inherently concurrent; real-time applications interact and interface with the external world (people, cars, robots, conveyor belts, planes, etc.) in situations that are concurrent in nature. Dealing with the nature of the system in the semantics of the program makes for a more readable, maintainable and reliable application. Nevertheless, building concurrent and real-time software brings a number of new problems and challenges. It is important to understand the main characteristics of this type of software, in order to guarantee its correct specification and behavior.

This project aims to study the development of concurrent real-time software, via the analysis and implementation of a case study [9]. Upon completion of the project, the student will be able to understand and apply methods, and tools to implement software solutions for real-time embedded systems problems.

Bibliography

[1] K.R. Apt, F.S. de Boer, and E.R. Olderog. *Verification of Sequential and Concurrent Programs*. Texts in Computer Science. Springer, 2009.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[3] John Barnes. *High Integrity Software: The SPARK Approach to Safety and Security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[4] John Barnes. Rationale for ada 2012: Contracts and aspects. Technical report, Caversham, UK, 2012.

[5] Michael Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and K. Rustan M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In *FMCO*, pages 364–387, 2005.

[6] David A. Basin, Felix Klaedtke, Samuel Müller, and Birgit Pfitzmann. Runtime monitoring of metric first-order temporal properties. In *FSTTCS*, pages 49–60, 2008.

[7] Patrick Baudin, Jean C. Filliâtre, Thierry Hubert, Claude Marché, Benjamin Monate, Yannick Moy, and Virgile Prevosto. *ACSL: ANSI C Specification Language*, preliminary design v1.2 edition, May 2008.

[8] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of tptl and mtl. *Inf. Comput.*, 208(2):97–116, 2010.

[9] A. Burns and A. M. Lister. A framework for building dependable systems. *Comput. J.*, 34(2):173–181, April 1991.

[10] Feng Chen and Grigore Roşu. Mop: an efficient and generic runtime verification framework. In *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, OOPSLA '07, pages 569–588, New York, NY, USA, 2007. ACM.

[11] Ivan Enderlin, Frédéric Dadeau, Alain Giorgetti, and Abdallah Ben Othman. Praspel: a specification language for contract-based testing in php. In *Proceedings of the 23rd IFIP WG 6.1 international conference on Testing software and systems*, ICTSS'11, pages 64–79, Berlin, Heidelberg, 2011. Springer-Verlag.

[12] Yliès Falcone. You should better enforce than verify. In *Proceedings of the First international conference on Runtime verification*, RV'10, pages 89–105, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] M.C.W. Geilen. *Formal techniques for verification of complex real-time systems*. Eindhoven, 2002.

[14] Alwyn Goodloe and Lee Pike. Monitoring distributed real-time systems: A survey and future directions. Technical Report NASA/CR-2010-216724, NASA Langley Research Center, July 2010.

[15] J. Goyvaerts and S. Levithan. *Regular expressions cookbook*. O'Reilly Series. Oreilly, 2009.

[16] Klaus Havelund. Runtime verification of c programs. In *Proceedings of the 20th IFIP TC 6/WG 6.1 international conference on Testing of Software and Communicating Systems: 8th International Workshop*, TestCom '08 / FATES '08, pages 7–22, Berlin, Heidelberg, 2008. Springer-Verlag.

[17] Farnam Jahanian and Aloysius K. Mok. Modechart: A specification language for real-time systems. *IEEE Trans. Softw. Eng.*, 20(12):933–947, December 1994.

[18] William Landi. Undecidability of static analysis. *ACM Lett. Program. Lang. Syst.*, 1(4):323–337, December 1992.

[19] François Laroussinie, Nicolas Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, LICS '02, pages 383–392, Washington, DC, USA, 2002. IEEE Computer Society.

[20] Gary T Leavens, Albert L Baker, and Clyde Ruby. Jml: A notation for detailed design. *Behavioral Specifications of Businesses and Systems*, pages 175–188, 1999.

[21] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[22] Bertrand Meyer. Eiffel: Analysis, Design and Programming Language. Technical Report ECMA-367, Ecma International, June 2006.

[23] Bertrand Meyer. Verified software: Theories, tools, experiments. chapter Eiffel as a Framework for Verification, pages 301–307. Springer-Verlag, Berlin, Heidelberg, 2008.

[24] Bertrand Meyer, Arno Fiva, Ilinca Ciupa, Andreas Leitner, Yi Wei, and Emmanuel Stapf. Programs that test themselves. *Computer*, 42(9):46–55, September 2009.

[25] Nicolas Navet and Stephan Merz. *Modeling and Verification of Real-time Systems*. John Wiley & Sons Inc., Hoboken, 2008.

[26] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[27] Dillon Pariente and Emmanuel Ledinot. Formal Verification of Industrial C Code using Frama-C: a Case Study. *In Proceedings of First International Conference on Formal Verification of Object-Oriented Software (FoVeOOS'10)*, June 2010.

[28] Rodolfo Pellizzoni, Patrick Meredith, Marco Caccamo, and Grigore Rosu. Hardware runtime monitoring for dependable cots-based real-time embedded systems. In *Proceedings of the 29th IEEE Real-Time System Symposium (RTSS'08)*, pages 481–491, 2008.

[29] Grigore Rosu, Feng Chen, and Thomas Ball. Synthesizing monitors for safety properties: This time with calls and returns. In *RV*, pages 51–68, 2008.

[30] Johann Schumann. *Automated theorem proving in software engineering*. Springer, 2001.

[31] Koushik Sen. Generating optimal monitors for extended regular expressions. In *In Proc. of the 3rd Workshop on Runtime Verification (RV'03), volume 89 of ENTCS*, pages 162–181. Elsevier Science, 2003.

[32] Julien Signoles. *E-ACSL: Executable ANSI/ISO C Specification Language*, preliminary design v1.5-4 edition, 2012.

[33] Oleg Sokolsky, Usa Sammapun, Insup Lee, and Jesung Kim. Run-time checking of dynamic properties. *Electron. Notes Theor. Comput. Sci.*, 144(4):91–108, May 2006.

[34] John A. Stankovic. Real-time computing system: The next generation. Technical report, Amherst, MA, USA, 1988.

[35] Julian Tschannen, Carlo A. Furia, Martin Nordio, and Bertrand Meyer. Usable verification of object-oriented programs by combining static and dynamic techniques. In *Proceedings of the 9th international conference on Software engineering and formal methods*, SEFM'11, pages 382–398, Berlin, Heidelberg, 2011. Springer-Verlag.

[36] Farn Wang. Formal verification of timed systems: a survey and perspective. *Proceedings of the IEEE*, 92(8):1283–1305, August 2004.

[37] Farn Wang. Model-checking distributed real-time systems with states, events, and multiple fairness assumptions. In *AMAST*, pages 553–568, 2004.

[38] Haitao Zhu, M.B. Dwyer, and S. Goddard. Predictable runtime monitoring. In *Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on*, pages 173 –183, july 2009.