Design-by-Contract for Embedded Systems

Daniel Klünder, Jianmin Li and Stefan Kowalewski

Abstract—Design-by-contract is a software engineering technology from the object-oriented world that exploits runtime assertions to define precise verifiable interface specifications with so-called invariants and pre- and post-conditions. This paper describes our idea of extending design-by-contract to support assertions dealing with platform and environment features essential in embedded systems like battery power, localization, or memory and CPU consumption to support variation of these on different deployment targets. While others have focused on contracts as a means for component-based design, i. e. for design time issues or dynamic service discovery, we suggest a runtime version to enhance system reliability and aim at an actual implementation. This paper motivates the approach, compares it to related work and sketches a possible implementation as well as an example application for mobile phones.

1 Introduction

The ever increasing presence of information technology along with the pervasion of hardware and software into everyday life boosts the need for engineering methods and tools for the development of high quality embedded systems. This trend is further amplified by the rising complexity of such systems and their usage for safety critical tasks. As complexity is steadily increasing, variability techniques like product lines gain ground in the development of embedded systems.

One major point that distinguishes embedded systems from desktop or enterprise software is their tight interaction with the underlying hardware and especially the performance constraints that are imosed on them. It is common practice that over the lifetime of an embedded system the underlying hardware is changed because of cost considerations or supplier changes. The approach described in this paper treats these different deployment targets like variation points of the full hard- and software system.

When reusing software components reliability is even more important than for application-specific development. A pragmatic technique introduced to tackle reliability problems in case of system modifications is design-by-contract (DBC) [1]. Nevertheless, software engineering approaches from the non-embedded world like this are not common practice in the embedded field.

We agree with [2] that this is due to the following reasons: embedded systems are engineering artifacts involving computation that is subject to physical constraints which arise through interactions with the physical world. This includes reactions to the physical environment (reaction constraints) and execution on a physical platform (execution constraints). Hence, embedded systems design means modeling of hardware, software, and environment all of which are subject to physical constraints. The separation of computation (software) from physicality (platform and environment) does not work. On the software side difficulties arise in taming concurrency and incorporating physical constraints.

While others are working on the inclusion of reaction constraints into DBC [3], this paper focuses on the execution constraints caused by the platform and the environment of the system like e. g. power supply, memory or CPU shortage, or localization. Thereby a contract between software and underlying hardware is established that is ensures the validity of its rules throughout hardware variations.

There is some related work on the usage of DBC in embedded systems: Li et al. [4] focus on contracts used in combination with a

• *D. Klünder, J. Li and S. Kowalewski are with the Embedded Software Laboratory at the Computer Science Department, RWTH Aachen Technical University, Germany*
http://www-i11.informatik.rwth-aachen.de

component model similar to CORBA (common object request broker architecture). The contract is defined in XML accompanying each component and is used for dynamic service discovery at runtime, though no application example is given. Urting et al. [5] focus on timing issues and the representation of components and their contracts in different views. In contrast to this related work we plan to use contracts included into the programming language and their utilization not for service discovery but for expression of execution constraints related to platform and environment. Furthermore, we plan to examine its usefulness and possible improvements in a case study using a mobile phone application.

The rest of this paper is structured as follows: the next section gives an introduction into DBC and the inclusion of platform and environment features. Section 3 details the implementation idea, sketches an application example, and concludes the paper.

2 DESIGN-BY-CONTRACT

It has long been recognized that runtime assertion checking can be a beneficial software engineering technology [6]. The first object-oriented programming language to advocate the use of assertions is Eiffel [7], [8]. Meyer, the founder of Eiffel, also introduced the term DBC, the usage of assertions in object-oriented programs [1]. It is used as a permanent defensive mechanism for fault detection and was mainly influenced by the work of Floyd [9], Hoare [10], and Dijkstra [11]. Assertions in other languages like Alphard [12], Euclid [13], and Anna [14] also influenced DBC in Eiffel. Further influences were the specification approach advocated by Liskov and Guttag [15], and the Z notation [16].

Eiffel offers contracts by including assertions in routines as preconditions and postconditions, in loops and classes as invariants and contract handling in case of inheritance. A contract entails benefits and obligations for both contract parties. In case of a routine call by a class' client this means if the caller guarantees the precondition of a routine then the callee promises to deliver a final state in which the postcondition is satisfied. Thus, the precondition binds the client while the postconditions binds the class.

While preconditions and postconditions describe properties of individual routines, invariants prescribe global properties of class instances which must be preserved by all routines, such that they are satisfied by all instances of the class at all stable times. This means that an invariant must be true at instance creation and before and after each remote routine call but can be violated during the call.

DBC is used during design-time for clear interface specification and debugging. During runtime contracts can be switched off or combined with exception handling to treat unexpected runtime situations. Apart from Eiffel there are contracts for many other languages, e. g. for .Net [17] or Java [18].

2.1 Design-by-contract Utilizing Platform and Environment Features

When environment and platform features are used in DBC those two become contractors of the affected class, i. e. the contract entails benefits and obligations for them. The precondition of a routine call must then not only be guaranteed by a calling client but also by the environment, e. g. a certain quality of radio reception, localization, or available battery power. On the other hand the postcondition of the called routine might ensure benefits for the environment like limited usage of resources.

Beugnard et al. [19] categorize contracts in four categories:

- syntactic contracts
- behavioral contracts
- synchronization contracts
- quality of service contracts

Contracts including environment and platform features belong into the second and fourth category. Collet et al. [20] suggest a contract should provide:

- a specification formalism
- a rule of conformance
- a runtime monitoring technique

Thus, the environment and platform features supported in contracts have to be quantifiable and measurable at runtime.

3 Application Example

Modern Jass [21] implements DBC for Java and supports pre- and postconditions for methods, invariants for classes, invariants and variants for loops as well as check statements and inheritance of assertions. On top of this tool we plan to utilize a mobile phone's platform API for statements about localization of the device, battery power left, quality of radio reception, and memory and cpu usage.

Fig. 1 shows a small example of an alarm clock on a mobile phone, that can only be set and activated if the device has enough power left. The shown diagram is a static diagram of the Business Object Notation (BON) used in the Eiffel world as an alternative to UML class diagrams. As shown for the routine `set_alarm_time` pre- and postconditions are indicated by a question mark and an exclamation mark respectively. If the routine `turn_alarm_on` is called because of user interaction and the runtime assertion on battery power fails, an exception would be thrown in the class responsible for power management, in this case `PowerManager`. It is the obligation of this class to handle the exception by e. g. informing the user and offering to switch off unneeded devices like GPS.

The main advantage of putting the assertions into a contract is a clear interface to the routine with precise obligations for caller, callee and environment. A programmer might as well test the parameters of the assertions upon entering the routine but there is no definite guideline how wrong parameters should be handled or how the method caller or the environment could be informed about the failure. Furthermore, the application of DBC prevents doubled checking of parameters in caller and callee, and enables the evaluation of assertions at runtime including a statement on who to blame for the failure. This advantage is even more important if third party libraries are used for development.

References

[1] B. Meyer, "Applying "design by contract"," *Computer*, vol. 25, no. 10, pp. 40–51, 1992. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=619797&type=external&coll=GUIDE&dl=GUIDE&CFID=25681883&CFTOKEN=58143031

[2] T. A. Henzinger and J. Sifakis, "The embedded systems design challenge," in *Proceedings of the 14th International Symposium on Formal Methods*, ser. Lecture Notes in Computer Science, vol. 4085. Springer, 2006, pp. 1–15.

[3] P. Nienaltowski and V. Arslan, "SCOOPLI: a library for concurrent object-oriented programming on .NET," *Journal of .NET Technologies*.

[4] S. Li, X. Li, and J. Wu, "Components and contracts for embedded software," in *ECBS '05: Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 19–24. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=1053859&type=external&coll=GUIDE&dl=GUIDE&CFID=25680250&CFTOKEN=79034257

[5] D. Urting, S. v. Baelen, T. Holvoet, and Y. Berbers, "Embedded software development: Components and contracts," in *IASTED international conference: parallel and distributed computing and systems (IASTED-PDCS)*, T. F. Gonzalez, Ed., 2001, pp. 685–690.

[6] L. A. Clarke and D. S. Rosenblum, "A historical perspective on runtime assertion checking in software development," *SIGSOFT Softw. Eng. Notes*, vol. 31, no. 3, pp. 25–37, 2006. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=1127900&type=pdf&coll=GUIDE&dl=GUIDE&CFID=25680250&CFTOKEN=79034257

[7] B. Meyer, *Eiffel: The Language*. Prentice Hall, 1992.

[8] ——, "Eiffel: a language and environment for software engineering," *J. Syst. Softw.*, vol. 8, no. 3, pp. 199–246, 1988.

[9] R. Floyd, "Assigning meanings to programs," *Mathematical Aspects of Computer Science*, vol. 19, no. 19-32, p. 1, 1967.

[10] C. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[11] E. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," *Communications of the ACM*, vol. 18, no. 8, pp. 453–457, 1975.

[12] W. Wulf, R. London, and M. Shaw, "An introduction to the construction and verification of alphard programs," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 253–265, 1976.

[13] G. Popek, J. Horning, B. Lampson, J. Mitchell, and R. London, "Notes on the design of euclid," *Proceedings of an ACM conference on Language design for reliable software table of contents*, pp. 11–18, 1977.

[14] D. Luckham and F. Von Henke, "An overview of anna, a specification language for ada," *Software, IEEE*, vol. 2, no. 2, pp. 9–22, 1985.

[15] B. Liskov and J. Guttag, *Abstraction and specification in program development*. The MIT Press, 1986.

[16] J. Spivey, *The Z notation: a reference manual*. Prentice Hall, 1992.

[17] K. Arnout and R. Simon, "The .net contract wizard: Adding design by contract to languages other than eiffel," in *TOOLS '01: Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented*

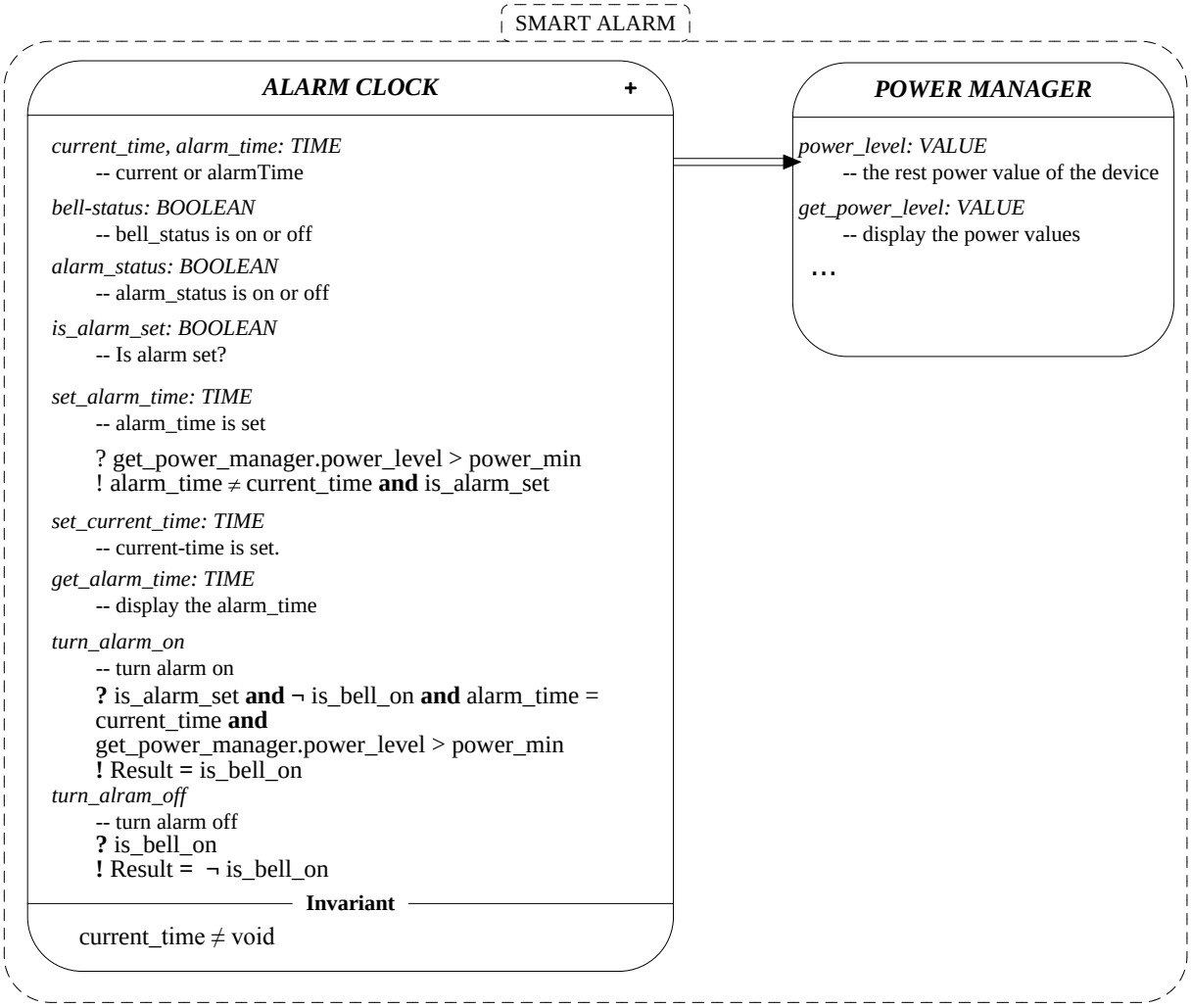

Fig. 1. Alarm clock example

Languages and Systems (TOOLS39). Washington, DC, USA: IEEE Computer Society, 2001, p. 14. [Online]. Available: http://portal.acm.org/ft_gateway.cfm?id=884716&type=external&coll=GUIDE&dl=GUIDE&CFID=9675813&CFTOKEN=60647003

[18] R. Kramer and C. Partners, "icontract-the java tm design by contract tm tool," *Technology of Object-Oriented Languages, 1998. TOOLS 26. Proceedings*, pp. 295–307, 1998.

[19] A. Beugnard, J. Jézéquel, N. Plouzeau, and D. Watkins, "Making Components Contract Aware," 1999.

[20] P. Collet, "Functional and Non-Functional Contracts Support for Component-Oriented Programming," *First OOPSLA Workshop on Language Mechanisms for Programming Software Components, OOPSLA 2001*.

[21] J. Rieken, "Design by contract for java - revised," Master's thesis, Universität Oldenburg, April 2007.