Science of Computer Programming 98 (2015) 52–79

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A tool for visual and formal modelling of software designs

Nuno Amálio [a,*], Christian Glodt [b]

[a] Escuela Politécnica Superior, Engeniería Informática, Universidad Autónoma de Madrid, 28049 Madrid, Spain
[b] Faculty of Science, Technology and Communication, University of Luxembourg, L-1359 Luxembourg

HIGHLIGHTS

- The paper presents the Visual Contract Builder (VCB) tool supporting the Visual Contract Language (VCL).
- VCL is a graphical language for describing software designs formally.
- VCL and VCB have been applied to several case studies.
- The paper evaluates VCB based on a survey carried out in the context of a controlled experiment.
- The paper includes several reflections on strengths and weaknesses of VCB and lessons learnt.

ARTICLE INFO

Article history:
Received 20 November 2012
Received in revised form 5 April 2014
Accepted 7 May 2014
Available online 21 May 2014

Keywords:
Visual modelling languages
MDE
Formal methods
Tool-support
Empirical evaluation

ABSTRACT

Diagrams are ubiquitous in software engineering and widely used for software modelling. The visual contract language (VCL) enables an approach to software design modelling that is entirely graphical and has a mathematical basis. VCL's main novelties lie in its capacity to describe predicates visually and in its graphical front-end to formal modelling. VCL is brought to life in the visual contract builder (VCB) tool presented in this paper. VCB provides diagram editors for the whole VCL suite, it type-checks diagrams and generates Z formal specifications from them; the Z specification enables formal verification and validation using Z theorem provers. The paper evaluates VCB based on the results of a survey carried out in the context of a controlled experiment. The work presented here is a contribution to the area of visual design modelling: the paper presents a state of the art tool supporting the novel VCL language and concrete empirical results on the usefulness and effectiveness of tool and language in particular, suggesting benefits of visual modelling in general.

1. Introduction

The prevalence of visual notations in software engineering (SE) can be explained by certain properties of diagrams that benefit cognition [1,2]. In the broad field of SE and in the specific field of software modelling, the "visual" is the most widespread representation [2]. This SE trend is entirely consistent with the ubiquity and importance of visual representations in traditional engineering [3,4].

Despite this prominence, mainstream visual languages for software design modelling that are driven by SE practice, such as the industry standard UML [5,6] and its predecessors, OMT [7], Booch [8] and others [9], have certain drawbacks:

* Corresponding author.
E-mail addresses: nuno.rodrigues@uam.es (N. Amálio), christian.glodt@uni.lu (C. Glodt).

http://dx.doi.org/10.1016/j.scico.2014.05.002
0167-6423/

- UML and similar languages have not been designed to be cognitively effective [2]. UML, for instance, is known to breach many principles of visual language design [2].
- UML and similar languages have several semantic issues. The semantics of UML, its accompanying constraint language OCL, and similar languages that preceded UML, have been, over the years, criticised for vagueness, ambiguity and imprecision [10,9,11–19]. This is attributed to the definitions of semantics in these languages, which are known to lack precision and formality [10,9,17,19]. There have been many formalisations of UML to date [20–27], but they only cover subsets of UML; no formalisation or formal framework covering the whole UML has been produced to date. This suggests that UML-like languages are appropriate for sketching, but not satisfactory for precise and rigorous modelling, due to their many semantic interpretations and lack of firm mathematical foundations. A consequence of this, is that consistency defects are a serious problem [28–30]. Furthermore, the task of model analysis is severely impaired as these languages do not provide means for analysing a model mechanically and exhaustively, using theorem-proving or model-checking, because their semantics is not formally-defined.
- UML and similar languages cannot describe all properties graphically. UML, for instance, uses the textual OCL to describe contracts and invariants.
- There is some dissatisfaction with the diagrams provided by UML-like languages for describing behaviour [15–17,31]. Descriptions of behaviour provided by UML collaboration and sequence diagrams are partial as they describe scenarios through interactions between instances; UML state and activity diagrams provide total descriptions, but their emphasis on explicitly defined states and transitions between these states may be appropriate for reactive systems, but they fall short of meeting the requirements of more general information systems.

There is another group of SE modelling languages that puts a strong emphasis on semantics. The so-called *formal methods* (FMs) [32,33] are rigorous, precise and have sound semantic definitions; their mathematical-based semantics enables mechanical and exhaustive verification and validation. FMs, however, are not without shortcomings:

- FMs are seen as difficult to use as they require a considerable amount of expertise; their entry costs are seen as a barrier for their adoption in industry [33,34].
- Despite a considerable number of success stories, FMs have not seen a widespread adoption in terms of industrial practice [33]. Although there has been substantial progress in recent years, the onus of formality does not appear to justify their general uptake [35,34]; the effort and expertise that they require appears to be justified only in domains where cost of software error is very high, such as the safety-critical niche [36,34,33].
- From a market perspective, current FMs do not appear to offer a business advantage in today's competitive software industry where *time to market* is a main competing factor [37,38].

The Visual Contract Language (VCL) [39–41], a general-purpose *object-oriented* (OO) modelling language for the abstract graphical description of software designs with a formal semantics, tries to address the above-mentioned drawbacks of existing SE modelling languages. On the one hand, VCL aims at improving mainstream UML-like languages: (a) it introduces rigour, precision and means for mechanical and exhaustive model analysis, (b) it extends the realm of what can be described visually, and (c) it improves existing graphical representations for the description of behaviour. On the other hand, VCL aims at improving the *practicality* of FMs by using visual descriptions as a front-end for sound mathematical approaches; this is done to (a) move the onus of formality from the user (the engineer) to the designer of the modelling language, (b) tap into knowledge and education of SE practitioners on OO modelling, (c) mitigate the need for expertise in formal methods and (d) enable engineers with diverse backgrounds to participate in a common modelling effort.

VCL tries, therefore, to contribute to both model-driven engineering (MDE) [42] using visual languages and FMs. It addresses the MDE side of the equation by proposing an approach to modelling that is entirely visual providing novel diagram types that do graphically what UML does textually with OCL. The FMs side of the equation is addressed by enabling the generation of formal specifications from VCL models, which can be subject to exhaustive verification and validation.

Like UML, VCL follows an OO style of description, expressing a software system as a collection of data and associated behaviour. It supports, however, a more abstract approach to modelling that is closer to *set theory* and is inspired in formal modelling languages based on set theory, such as Z [43,44], B [45] and Alloy [46]; this is done without breaching the compatibility with UML. VCL's distinguishing feature lies in its capacity to express predicates visually, which enables a graphical approach to behavioural modelling based on *design by contract* [47]. In addition, VCL tackles the modelling of large and complex systems with an approach to *separate concerns* based on coarse-grained modularity mechanisms to enable the definition of modules to represent requirements and design concerns. VCL builds up on previous work that combines UML and Z [26,27,48] to introduce rigour in UML-based development and is compatible with UML-based development processes.

This paper presents the tool that brings VCL to life: the *visual contract builder* (VCB).[1] Together, VCL and VCB, establish a symbiotic relationship that is important to address the needs of software modelling: better modelling techniques and tools to support them [49]. A prototype version of VCB, which only supported two diagram types of VCL (structural and assertion

[1] http://vcl.gforge.uni.lu.