Building VCL Models and Automatically Generating Z Specifications from them

Nuno Amálio, Christian Glodt, and Pierre Kelsen

University of Luxembourg, 6, r. Coudenhove-Kalergi, L-1359 Luxembourg
{nuno.amalio,christian.glodt,pierre.kelsen}@uni.lu

Abstract. VCL is a visual and formal language for abstract specification of software systems. Its novelty lies in its capacity to describe predicates visually. This paper presents work-in-progress on a tool for VCL; the tool version presented here supports the VCL notations of structural and assertion diagrams (a subset of the whole VCL suite), enabling the generation of Z specifications from them.

Key words: formal methods, visual languages, Z, model-driven development

1 Introduction

Diagrams are widely used in modern day software engineering. There are, however, several issues with existing visual languages (VLs) such as UML. Diagrams are effective provided they make use of certain properties of visual descriptions that benefit cognition [1,2]. One problem is that most VLs, like UML, have not been designed to be *cognitive-effective* and to make use of such properties [2]. Another problem is that mainstream VLs have not been designed with a formal semantics [3]; they are mostly used without formal semantics, which precludes formal model analysis. Furthermore, most VLs cannot describe all properties visually; UML, for instance, uses the textual OCL to describe invariants and operations.

This paper presents our first results on a tool to support the visual contract language (VCL) [4,5]. VCL is formal, designed with usability in mind and to express predicates visually. It aims at making formal methods usage more practical; in particular: (a) to be usable by a variety of engineers, not necessarily formal methods experts; (b) to enable engineers to focus on software design; and (c) to use formal methods, but hiding them from the lay user. This enables the formation of teams with a good combination of skills; some individuals may be experts in the domain, others in the formal method.

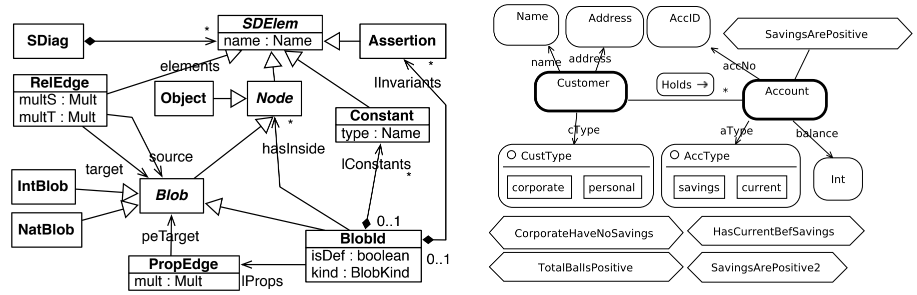
2 VCL Tool

The *visual contract builder* (VCB)[1], VCL's tool presented here, is an Eclipse plug-in built using the GMF framework[2]. VCB's version presented here supports the construction of VCL structural and assertion (or constraint) diagrams (a subset of the whole VCL suite, see [4]).

VCB provides editors to construct VCL diagrams. These editors are built from metamodels that describe the abstract syntax of the VCL notations. VCL's

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

[2] http://www.eclipse.org/modeling/gmf/



(a) Metamodel of VCL structural diagrams (b) A VCL Structural Diagram

Fig. 1. Metamodel of VCL structural diagrams and sample instance

metamodels are described in terms of object-oriented (OO) class *metamodels* specified in Alloy, which were refined to build diagram editors using GMF. All Alloy metamodels can be found in `http://vcl.gforge.uni.lu/metamodels`.

Figure 1(a) gives a UML class diagram that partially describes the metamodel of VCL structural diagrams (SDs). An instance of this metamodel is given in Fig. 1(b). The rounded contours in Fig. 1(b) are blobs. Blob `Int` is an instance of metaclass `IntBlob`; all other blobs are instances of `BlobId`. `Customer` and `Account`, drawn with a bold line, are domain blobs (`kind` meta-attribute has value `domain`); all others are value blobs (`kind` meta-attribute has value `value`). `CustType` and `AccType`, represented with symbol ○, are definition blobs (`isDef` meta-attribute has value `true`); the objects inside these blobs (meta-association `hasInside`) are instances of metaclass `Object`. Property edges (instances of metaclass `PropEgde`), such as `name` and `address`, define state properties of blobs. `Holds` is a relation edge (metaclass `RelEdge`) connected to `Customer` (meta-association `source`) and `Account` (meta-association `target`) with multiplicity `one` (meta-attribute `multS`) to `many` (meta-attribute `multT`). Elements of Fig. 1(b) depicted as elongated hexagons are instances of metaclass `Assertion`; `SavingsArePositive` is a local assertion of blob `Account` (meta-association `lInvariants`); all others are global assertions.

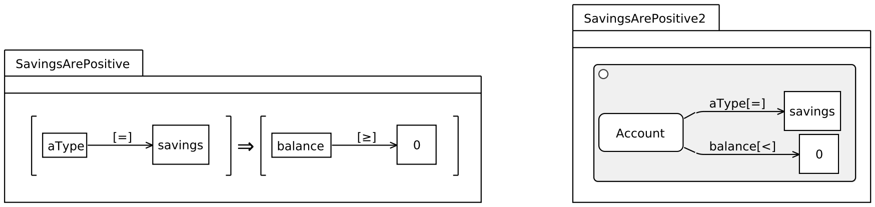
VCB uses type-checking to check well-formedness. This helps to find subtle errors in an efficient way. VCB type-checks diagrams based on VCL's type system, and generates Z from type-correct diagrams only. Currently, VCB maps diagrams to Z specifications expressed in the ZOO style of object-orientation [3,6]. The generated Z is type-checked using Community Z Tools (CZT)[3]; in most cases, this is a mere sanity check: all type errors in diagrams are captured using VCL type-checking.

3 Illustration

Figures 1(b) and 2 present VCL structural and assertion diagrams of the *Simple Bank* case study [6,4] drawn using VCB (see [4] for details). The Z generated by VCB for this case study's VCL model, along with a demo of VCB, can be found at `http://vcl.gforge.uni.lu/SBDemo`.

Assertion diagram (AD) of Fig. 2(a) describes the local invariant `SavingsArePositive`, identified in the SD of Fig. 1(b); in VCB, double-clicking the invariant on the SD takes the user to the AD. This AD says that savings accounts

[3] `http://czt.sourceforge.net/`



(a) Assertion Diagram of invariant `SavingsArePositive` of blob `Account`

(b) Assertion Diagram of global invariant `SavingsArePositive2`

Fig. 2. VCL diagrams of *Simple Bank* [6,4] constructed using the VCL tool

must have positive balances using a logical implication formula. The Z predicate that is generated for this AD is: $aType\ =\ savings \Rightarrow\ balance\ \geq\ 0$.

The same constraint is expressed in the AD of Fig. 2(b), using a set formula. It defines the set of `Account`s that are savings and have negative balances using a definition blob, which is shaded to say that the set must be empty. The Z predicate resulting from the diagram is:

$$\{o : sAccount \mid (stAccount\ o).aType = savings \land (stAccount\ o).balance\ <\ 0\} = \varnothing$$

The blob definition results in a Z set comprehension. Here, *sAccount* is set of *Account* objects; *stAccount* is a function mapping account objects to their state.

4 Discussion

We believe that we achieve usability gains by hiding the formal method from the lay user[4]. Four people developed the large VCL model of [5], only one is a Z expert. The VCL approach may be relevant for the critical systems industry that uses formal methods, such as Z [7]. Users writing specifications do not necessarily need to be trained in a formal method, they could use a VL, such as VCL, that abstracts away from the underlying formal method(s); based on [5], VCL appears to be easier to learn than Z.

VCL has been designed to exploit many target formal languages, not just Z. We intend to map VCL to other languages in the future to produce partial or total specifications from VCL diagrams; this enables use of the verification capabilities of the target language.

5 Related Work

The AutoZ tool [8] generates Z specifications, expressed in the ZOO style, from UML class (similar to VCL SDs) and state diagrams, following the *UML + Z* approach of [3]. Predicates of invariants are expressed textually in Z. VCL describes predicates visually and generates Z from them. AutoZ uses Z type-cheking to check well-formedness of diagrams, which complicates error-reporting. VCB type-checks diagrams directly based on VCL's type-system.

Constraint diagrams [9] is a VL that describes predicates visually. Its semantic basis is similar to VCL (set theory). To our knowledge, there are no visual diagram editors to support this VL. VL of [9] emphasises reasoning with diagrams; VCL generates Z to enable formal reasoning at the Z level.

[4] We intend to assess this claim empirically in the future.

Visual OCL (VOCL) [10], a VL based on OCL, expresses predicates visually and has a semantics based on graph transformations. There is a tool with a visual editor (`http://tfs.cs.tu-berlin.de/vocl/`) for VOCL. This tool, however, does not support UML class diagrams (similar to VCL SDs). Unlike VCB, VOCL tool only supports basic types and only generates simple OCL expressions. VCB supports both ADs and SDs, checks consistency of ADs (predicates) against SDs (structures), and generates complete Z Specifications.

6 Conclusions and Future Work

This paper presents work-in-progress on VCL's tool: the *Visual Contract Builder* (VCB)[5]. VCB version presented here supports the construction of VCL structural and assertion (or constraint) diagrams (a subset of the whole VCL suite), enabling the generation of Z specifications from them. It is the first step towards tool support for VCL. The most relevant contribution of the work presented here is a tool supporting a software engineering visual language that has a formal basis, expresses predicates visually, and enables the generation of formal specifications that can be processed independently on their own. To our knowledge, no other tool supports a language that describes predicates visually, and generates complete formal specifications. This is also a contribution to VCL's development: (a) it demonstrates VCL's formal semantics, and (b) it supports the novel notation of assertion diagrams (ADs).

Future work will complete the tool support for ADs (modular mechanisms of VCL ADs [4] are currently not supported), the type system for structural and assertion diagrams, and will extend the tool to support VCL's descriptions of behaviour using the VCL notations of behaviour and contract diagrams (see [4]).

References

1. Larkin, J. H., Simon, H. A. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11:65–99 (1987)
2. Moody, D. L. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE TSE*, 6(35):756–779 (2009)
3. Amálio, N. *Generative frameworks for rigorous model-driven development*. Ph.D. thesis, Dept. Computer Science, Univ. of York (2007)
4. Amálio, N., Kelsen, P. Modular design by contract visually and formally using VCL. In *VL/HCC 2010* (2010)
5. Amálio, N., Kelsen, P., Ma, Q., Glodt, C. Using VCL as an aspect-oriented approach to requirements modelling. *TAOSD*, VII:151–199 (2010)
6. Amálio, N., Polack, F., Stepney, S. An object-oriented structuring for Z based on views. In *ZB 2005*, vol. 3455 of *LNCS*, pp. 262–278. Springer (2005)
7. Hall, A. Correctness by construction: Integrating formality into a commercial development process. In *FME 2002*, vol. 2391 of *LNCS*, pp. 139–157. Springer (2002)
8. Williams, J., Polack, F. Automated formalisation for verification of diagrammatic models. *ENTCS*, 263:211–226 (2010)
9. Fish, A., Flowe, J., Howse, J. The semantics of augmented constraint diagrams. *Journal of Visual Languages and Computing*, 16:541–573 (2005)
10. Bottoni, P., Koch, M., Parisi-Presicce, F., Taentzer, G. A visualisation of OCL using collaborations. In *UML 2001*, vol. 2185, pp. 257–271 (2001)

[5] `http://vcl.gforge.uni.lu/`