VCL, a Visual Language for Modelling Software Systems Formally

Nuno Amálio and Pierre Kelsen

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

Abstract. This paper overviews design of VCL, a new visual language for abstract specification of software systems at level of requirements. VCL is designed to be visual, formal and modular, and aims at expressing precisely structural and behavioural properties of software systems. Novelty of VCL design lies in its emphasis on modularity.

Keywords: formal modelling, visual languages, modularity.

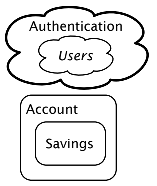
1 Introduction

Visual languages are widely used to describe software systems. Mainstream visual languages, like UML, however, have several shortcomings:

– They were mostly designed to be semi-formal (with a formal syntax, but no formal semantics). Although, there have been successful formalisations of semantics (e.g subsets of UML, see [1]), they are mostly used semi-formally. This brings numerous problems: it is difficult to be precise, unambiguous and consistent, and resulting models are not mechanically analysable.
– They cannot express a large number of properties diagrammatically; hence, UML is accompanied by the textual Object Constraint Language (OCL).
– They lack effective mechanisms to support coarser-grained modularity and separation of system-specific concerns.

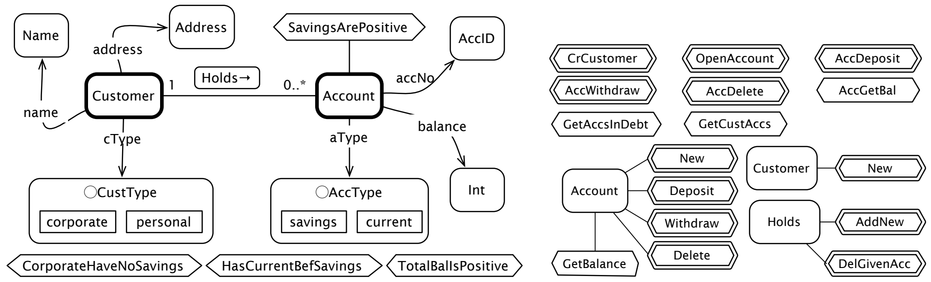
To address these problems, this paper proposes the visual contract language (VCL) [2,3], designed to be formal and modular, and to target abstract specification at level of requirements. The paper presents an overview of VCL's design.

2 Visual Primitives



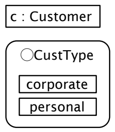
A VCL model is organised around *packages*, whose symbol is the *cloud*. Packages encapsulate structure and behaviour, and are built from existing ones using *extension*. They are VCL's coarse grained modularity construct. A package extends those packages that it encloses (e.g. *Authentication* to the left).

VCL *blobs* are labelled rounded contours denoting a *set*. They resemble Euler circles because topological notion of *enclosure* denotes subset relation (e.g. to the left, *Savings* is subset of *Account*).

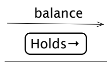


(a) Structural Diagram (b) Behavioural Diagram

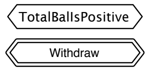
Fig. 1. Sample VCL structural and package diagrams (taken from [2])



Objects, represented as rectangles, denote an element of some set. Their label includes their name and may include set to which they belong (e.g. c to the left). Blobs may also enclose objects, and be defined in terms of things they enclose by preceding blob's label with symbol ○. To the left, *CustType* is defined by enumerating its elements.



Property edges are represented as labelled directed arrows; they denote some property possessed by all elements of a set, like *attributes* in the object-oriented (OO) paradigm (e.g. *balance* to the left). *Relational edges*, represented as directed lines where direction is indicated by arrow symbol above line, denote some relation between blobs (*associations* in OO) – e.g. $Holds$ to the left.

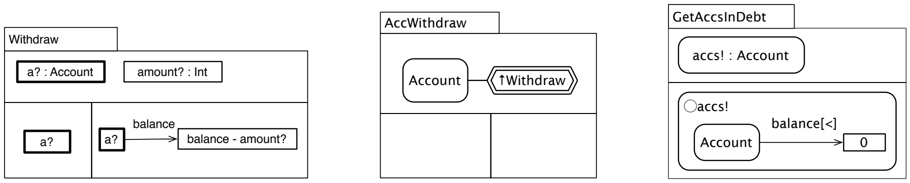


Represented as labelled hexagons, *constraints* denote some state constraint or observe operation (e.g $TotalBalIsPositive$ to the left). They refer to a particular state of some structure or ensemble. *Contracts*, represented as labelled double-lined hexagons, denote operations that change state; hence, their representation as double-lined hexagons as opposed to single-lined constraints (e.g. $Withdraw$ to the left).

3 VCL Diagrams

VCL diagrams are constructed using the visual primitives presented above. VCL *package diagrams* define packages. VCL *structural diagrams* (SDs) define structures of a package; the ensemble of structures makes the package's state space. Structures and their ensembles are subject to constraints (invariants), which are identified in SDs and defined in *constraint diagrams*. A sample SD is given in Fig. 1(a); see [2,3] for details.

Behavioural diagrams (BDs) identify all operations of a package. Operations are either local (operate upon individual structure), or global (scope of overall package, operate upon ensemble of structures). Update operations are represented in BDs as contracts, observe operations as constraints; these are defined in constraint and *contract* diagrams. A sample BD is given in Fig. 1(b); sample constraint and contract diagrams are given in Fig. 2; see [2] for details.



(a) Contract Diagram of *Account.Withdraw*

(b) Contract Diagram of global *AccWithdraw*

(c) Constraint Diagram of global *GetAccsInDebt*

Fig. 2. Sample VCL contract and constraint diagrams (taken from [2])

4 Semantics

VCL's design overviewed here is accompanied by a formal semantics. VCL takes a *generative* (or *translational*) approach to semantics. Currently, VCL diagrams are mapped into ZOO [4,1], a OO semantic domain expressed in formal language Z. We intend to support other formal languages in the future.

5 Using VCL

VCL has been applied to several case studies. Sample VCL diagrams of Figs. 1 and 2 are part of VCL model of *secure simple Bank* case study documented in [2]. In [5], VCL is used to model a large-scale case study.

6 Conclusions and Future Work

This paper overviews design of VCL, a visual language for formal abstract specification of software systems. A prominent feature of VCL is its support for modularity: constraints, contracts and packages are all modular constructs. Currently, we are completing formal definition of VCL, and developing VCL's tool[1].

References

1. Amálio, N.: Generative frameworks for rigorous model-driven development. Ph.D. thesis, Dept. Computer Science, Univ. of York (2007)
2. Amálio, N., Kelsen, P., Ma, Q.: The visual contract language: abstract modelling of software systems visually, formally and modularly. Tech. Report TR-LASSY-10-03, Univ. of Luxembourg (2010), `http://bit.ly/9c5YwQ`
3. Amálio, N., Kelsen, P., Ma, Q.: Specifying structural properties and their constraints formally, visually and modularly using VCL. In: Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Ukor, R. (eds.) EMMSAD 2010. LNBIP, vol. 50, pp. 261–273. Springer, Heidelberg (2010)
4. Amálio, N., Polack, F., Stepney, S.: An object-oriented structuring for Z based on views. In: Treharne, H., King, S., Henson, M.C., Schneider, S. (eds.) ZB 2005. LNCS, vol. 3455, pp. 262–278. Springer, Heidelberg (2005)
5. Amálio, N., Kelsen, P., Ma, Q., Glodt, C.: Using VCL as an aspect-oriented approach to requirements modelling. Transactions on Aspect Oriented Software Development 7 (to appear 2010)

[1] *Visual Contract Builder*, `http://vcl.gforge.uni.lu`