Visual Behavioural Modelling with Contracts

Nuno Amálio Pierre Kelsen
University of Luxembourg, 6, r. Coudenhove-Kalergi,
L-1359 Luxembourg
nuno.amalio@uni.lu pierre.kelsen@uni.lu

This paper presents the Visual Contract Language (VCL). VCL is a new visual language for abstract software specification at level of requirements. It is designed to be visual, formal and modular, and aims at expressing precisely structural and behavioural properties of software systems. VCL takes an approach to behavioral modelling based on design by contract that emphasises modularity.

1 Introduction

Visual languages (VLs) are widely used to model software systems. VLs like UML are known as *semi-formal* methods; they have a formal syntax but no formal semantics. They have several shortcomings:

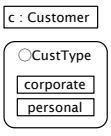
- They were mostly designed to be semi-formal (with a formal syntax, but no formal semantics). Although, there have been successful formalisations of semantics (e.g subsets of UML, see [1]), they are mostly used semi-formally. This brings numerous problems: it is difficult to be precise, unambiguous 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).

This paper presents the visual contract language (VCL) [2, 3, 4, 5, 6], which tries to address these problems. VCL targets abstract specification at level of requirements (or high-level designs), and takes an approach to behavioural modelling based on *design by contract* [7].

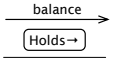
2 An Overview of VCL

2.1 Visual Primitives

VCL *blobs* are labelled rounded contours denoting a *set*; they are akin to classes of the object-oriented (OO) paradigm. Blobs resemble Euler circles because topological notion of *enclosure* denotes subset relation (e.g. to the left, `Savings` is subset of `Account`).



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

Submitted to:
FLACOS 2010

© N. Amálio & P. Kelsen
This work is licensed under the
Creative Commons Attribution License.

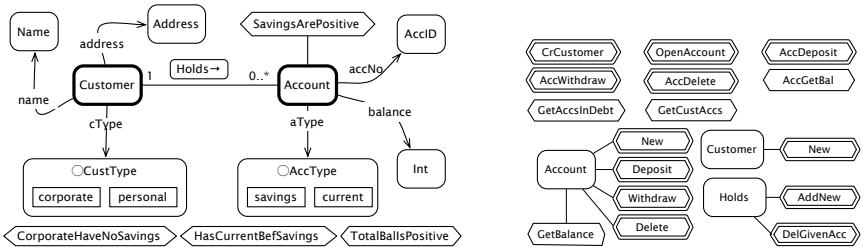
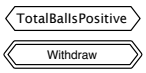


Figure 1: Structural (left) and behavioural (right) diagrams of simple Bank.



Represented as single-lined labelled hexagons, *constraints* denote some state constraint or observe (or query) 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, they are double-lined (e.g. `Withdraw` to the left).

2.2 Semantics

VCL's design is accompanied by a formal semantics. VCL takes a *generative* (or *translational*) approach to semantics. Currently, VCL diagrams are mapped into ZOO [8, 1], a OO semantic domain expressed in the formal language Z. We intend to support other formal languages in the future.

3 Defining structures and overall behaviour

VCL is illustrated with the *simple bank* case study, which is used to illustrate the ZOO semantic domain [1, 8] and VCL's structural [2] and behavioural parts [3]. Full VCL model of case study together with generated Z are given in [6][1].

VCL diagrams are constructed using the visual primitives presented above. *Structural diagrams* (SDs) define structures; an ensemble of structures makes a state space. Structures and ensembles are subject to constraints (invariants), which are identified in SDs and defined in *constraint diagrams*. SD of simple bank is defined in Fig. 1 (left); it defines main problem domain concepts (blobs `Customer` and `Account`), their relationships (relational edge `Holds`) and invariants (e.g. `TotalBalIsPositive`).

Behavioural diagrams (BDs) identify operations of some ensemble. Operations can either be local (operate upon individual structure) or global (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. BD of simple Bank (Fig. 1, right) identifies eight system operations, represented as global units, one operation of blob `Customer`, five operations of blob `Account`, and two operations of relational edge `Holds`. Next section shows how to describe in VCL some of these behaviours.

4 Defining Behaviour using contracts

4.1 Contract diagrams of local operations

Local operations have a localised scope and effect. They factorise the local effects of the overall (global) package behaviour.

Figure 2 presents VCL contracts for local operations `New`, `Delete` and `Withdraw` of blob `Account`. Remaining local contracts are given in [6]. Semantics of contract diagrams is given in [3]. Local contracts

[1] [6] extends *simple bank* case study with security concerns; simple Bank is localised in VCL package *Bank* in [6].

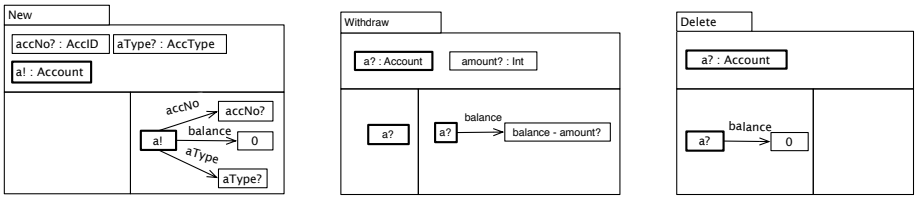


Figure 2: VCL contracts diagrams describing local operations of simple Bank.

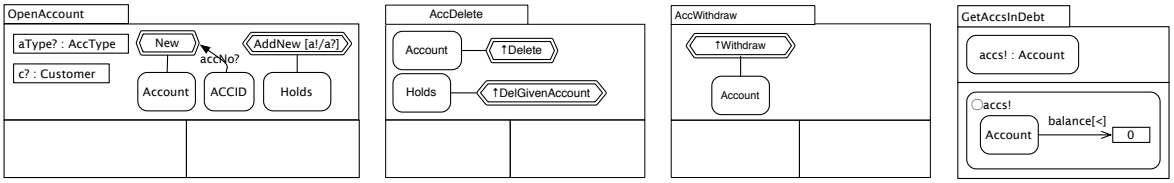


Figure 3: VCL contracts of global operations of Simple Bank.

that change state are expressed in terms of *action* units (represented with a bold line), identifying the object or link whose state is to change. Contracts of Fig. 2 are as follows:

- Contract *New* (Fig. 2, left) declares inputs new account's number (*accNo?*) and type (*aType?*), and output for created object (*a!*). Pre-condition is *true* as its compartment is empty. *Post-condition* sets properties of action object *a!*; *a!* is to be *created*: it is on the right predicate compartment, but not on the left.
- Contract *Delete* (Fig. 2, centre) declares account to delete as input (*a?*). Pre-condition says that action object *a?* must have a balance of 0. Post-condition compartment is empty. *a?* is to be *deleted*: it is on the left predicate compartment, but not on the right.
- Contract *Withdraw* (Fig. 2, right) declares two inputs: account (*a?*), and amount to withdraw (*amount?*). Pre-condition says that action object *a?* exists. Post-condition sets *balance* property of *a?* to value of expression *balance-amount?* (where *balance* refers to before-state value). *a?* is to be *updated*: it is both on the left and right predicate compartments.

4.2 Contract diagrams of global operations

Global operations define the behaviour of an ensemble of structures (such as a system). Their contracts are a composition of contracts of local operations plus some extra behaviour of their own. In VCL, this form of contract composition is done through *contract importing* (see [3] for details), enabling larger contracts to be built from smaller ones.

In VCL, a contract imports all contracts placed on its declarations compartment. Usually, importing is used to promote local contracts to a global scope. Semantically, contract importing is *conjunction*: of all pre- and post- conditions of imported contracts with pre- and post- condition of composite. Rules of contract importing are given in [3, 6].

Figure 3 describes in VCL some global operations of simple Bank; remaining operations are given in [6]. Diagrams of Fig. 3 are as follows:

- *OpenAccount* declares inputs *aType?* and *c?*, and partially imports[2] contracts *Account.New*

[2]There are two modes of importing (see [3] for details). *Partial importing* (the default) means that only the contract's predicate is imported; declarations are hidden. Total importing (symbol ↑) means that both declarations and predicates are imported.

(Fig. 2) and *Holds.AddNew* (creates a new tuple of relation *Holds*). *aType?* and *c?* are shared inputs; *accNo?* and *a!* (of *Account.New*) are used internally only. Arrow from *AccID* (set of all account identifiers) means that some object of this set (selected non-deterministically) is passed to *Account.New* through channel *accNo?*.

- *AccDelete* imports integrally both *Account.Delete* and *Holds.DelGivenAccount*. Action compartments are empty and so pre- and post-conditions are conjunction of imported contracts.
- *AccWithdraw* imports *Account.Withdraw* integrally. Action compartments are empty; pre- and post-conditions are those of imported contract.
- *GetAccsInDebt*, an observe contract, declares output *accs!* to hold set of accounts whose balance is less than 0[3].

5 Conclusions and Future Work

This paper presents VCL, a visual language for formal abstract specification of software systems. A prominent feature of VCL is its support for modularity: constraints and contracts are all modular constructs. We have recently applied VCL to model a large system [5]. Currently, we are completing formal definition of VCL, and developing VCL's tool[4].

We intend to leverage VCL's underlying formal semantics to enable formal analysis and verification of VCL models. This is to be done visually. Future work will look at incorporating the snapshot analysis technique for ZOO models of [1, 9] in VCL; this enables analysis of state spaces to validate invariants, and of behaviours to validate contracts.

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. Specifying structural properties and their constraints formally, visually and modularly using VCL. In *EMMSAD 2010*, vol. 50 of *LNBIP*. Springer (2010)

[3] Amálio, N., Kelsen, P. Modular design by contract visually and formally using VCL. In *VL/HCC 2010* (2010)

[4] Amálio, N., Kelsen, P. VCL, a visual language for modelling software systems formally. In *Diagrams 2010*, vol. 6170 of *LNAI*. Springer (2010)

[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*, 7:151–199 (2010)

[6] 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). Available at `http://bit.ly/9c5YwQ`.

[7] Meyer, B. Applying "design by contract". *Computer*, 25(10):40–51 (1992)

[8] 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)

[9] Amálio, N., Stepney, S., Polack, F. Formal proof from UML models. In *Proc. ICFEM 2004*, vol. 3308 of *LNCS*, pp. 418–433. Springer (2004)

[3] In contracts, property edges link object to some value; by default they denote equality, unless other relational operator is explicitly provided. In the contracts above, most property edges denote equality; here *balance* denotes $<$.

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