Semantics through Pictures

Stuart Kent, Ali Hamie, John Howse, Franco Civello, Richard Mitchell[1]
Division of Computing,
University of Brighton, Lewes Rd., Brighton, UK.
http://www.biro.brighton.ac.uk/biro/index.html, biro@brighton.ac.uk
fax: ++44 1273 642405, tel: ++44 1273 642494

Abstract. A diagrammatic approach to the semantics of OO modelling notations is proposed. This is based on an innovative and expressive notation dubbed "constraint diagrams", which can be used to precisely characterise a range of sophisticated, static constraints on OO models. Other notations, such as those found in UML, can be viewed as projections of constraint diagrams. Work on using constraint diagrams at the core of a 3D modelling notation is also briefly described as a means of similarly providing the semantics of diagrams imposing constraints on dynamic behaviour.

1 Introduction

This paper outlines a pictorial approach to constructing a precise semantics for object-oriented modelling notations. There are at least four reasons why one might want to build a precise semantics:

1. To clarify meaning leading to refinements of the notation.
2. To clarify meaning for developers using the notation.
3. To clarify meaning for tool developers, thereby increasing the likelihood of inter-operability between tools at a semantic level (e.g. code generated from different tools for the same model has the same behaviour).
4. To support semantic checking of models, automated if possible. This includes checking that implementations meet their specifications, checking internal consistency of components, and checking for inconsistencies and conflicts between components.

(1) just requires the semantics to be written down in a precise form. (2) and (3) require it to be written down in a form which developers and tool developers can easily understand. In addition, it would be desirable for (3) to provide a semantics in a form which directly assists the construction of tools, e.g. the automation of (4).

We propose that the semantics is given in terms of an expressive and innovative diagrammatic modelling notation, dubbed constraint diagrams, which can be used to precisely characterise a range of sophisticated, static constraints on OO models and which is particularly targeted on (2) and (3) (with some impact on (1) and (4)). A 3D notation based on constraint diagrams may be similarly used to characterise the semantics of dynamic behaviour. In essence, our notation is rich enough to characterise a model that would otherwise require many different kinds of diagram. The latter can then be viewed as projections of this model.

1. This research was partially funded by the UK EPSRC under grant number GR/K67304

Section 2 briefly surveys some of the notations used for describing OO models found in UML (UML) and Catalysis (d'Souza and Wills, 1997). Section 3 introduces constraint diagrams and contract boxes. Section 4 outlines how these could be used to give a semantics, and discusses how the approach could be extended to other diagrams in UML.

2 Generic Descriptors: Perspectives on a Model

In essence an OO model *is* the set of states it is allowed to enter, where a state can be visualised as an object diagram (*snapshot*), together with the set of allowed paths through those states. These sets are in general infinite, or at best very large, so impossible to enumerate. Therefore modellers need notations that are able to define very large sets in only a few diagrams. UML calls these notations *generic descriptors*. Essentially generic descriptors provide ways of writing rules or constraints which determine whether any particular snapshot or filmstrip is allowed in a model or not. Here we consider type and state diagrams (from UML) combined with invariants and action specifications (from Catalysis).

2.1 Type Diagrams

Type diagrams define most of the language that can be used in snapshots and constrains cardinalities of links between objects. The type diagram for the specification of a library system is given in Figure 1.

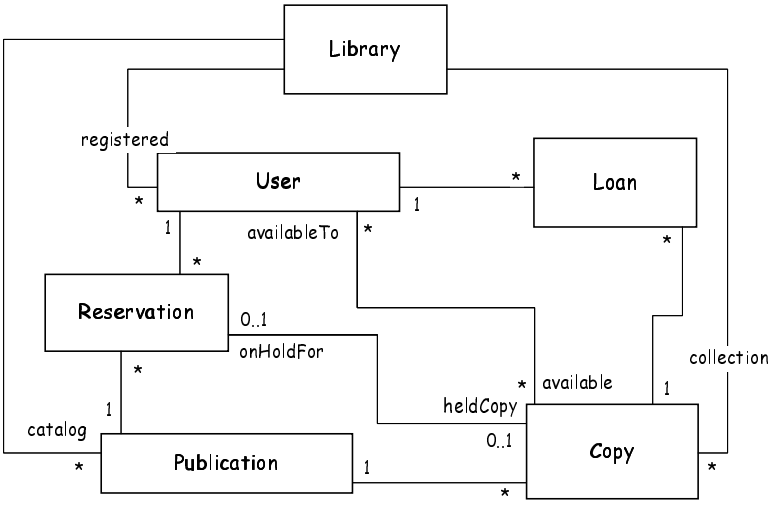

Figure 1: Type model for library

Only types and association rolenames appearing in the type model may appear in snapshots. Furthermore the number of links in a snapshot corresponding to a particular association may not exceed the cardinality constraints declared on the type model, for any objects of the types associated. For example, focusing on the (unlabelled) associations between Loan and User and Loan and Copy, a loan object may be linked with

only one user and one copy, though user and copy objects may be linked to many loan objects.

2.2 State Diagrams

A state diagram places constraints on both the static and dynamic models. The state diagram for the type Copy, in the context of the Library, is given in Figure 2.

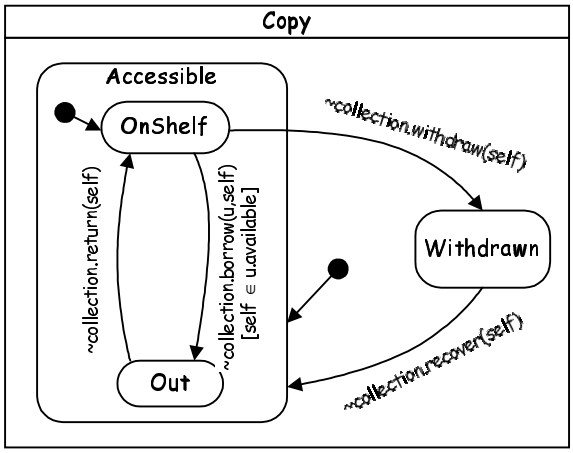

Figure 2: State Diagram for Copy

This is essentially UML notation, though we allow navigation expressions labelling the transitions. For example the diagram indicates that when the action borrow is performed on the object identified through ~collection with self as the copy argument, then, provided self is in the OnShelf state, the effect will be to move it into the Out state.

The constraints on the static model imposed by a state diagram are the introduction of new states and the relationships between them. In particular, states at the same nesting level are disjoint, so an object can only be in one state at a time.

The constraints on the dynamic model are on the transitions: for example, Figure 2 says that when a borrow action is performed on the library, the copy involved is self, and that copy is in the onShelf state, then the effect will be to put that copy in the Out state. This may be expressed, if desired, as a pre/post specification fragment for the action borrow (see Kent 1997).

2.3 Invariants

Diagrams currently in use in OO modelling, can not express all required static constraints. This is demonstrated in (Kent 1997). For example, in the library system we would like to say that a copy on hold is only available for lending to the user who reserved it, whereas a copy on the shelf is available to all users who are registered and active. Such constraints can not be expressed diagrammatically in UML, and some form of textual annotation is required. (Kent 1997) shows how this and other constraints may be written using the mathematical language of Catalysis.

2.4 Action Specifications

Similarly, diagrams in UML can not express all constraints on dynamic behaviour. For example, in the case of borrow, the state diagram of Figure 2 does not say that a new, loan object must be created recording the fact that the copy has been loaned out to the

user. Again such constraints have to be written using a textual notation. In Catalysis they are expressed precisely as pre/post conditions or *contracts* written in the mathematical language.

3 Constraint Diagrams and Contract Boxes

Constraint diagrams (Kent 1997) are a diagrammatic notation for expressing static constraints on models. They build upon the effectiveness of snapshots in illustrating the import of constraints on a model. They may be viewed as a generalization of snapshot notation (i.e. UML object diagrams) – one constraint diagram represents a set of snapshots, which is more expressive than type diagrams. They make use of Venn diagram notation, with some extensions, to show relationships between the values of navigation expressions. They also show types and states as the sets of objects of that type or in that state, respectively.

Figure 3 is a constraint diagram for the invariant stated in Section 2.3 on page 3. Reading the diagram starting from the object x, part of that invariant is read off as follows: for all libraries x, and for all copies y in the collection of x that are on hold, y is available for lending to the (single) user associated with the reservation that y is on hold for.

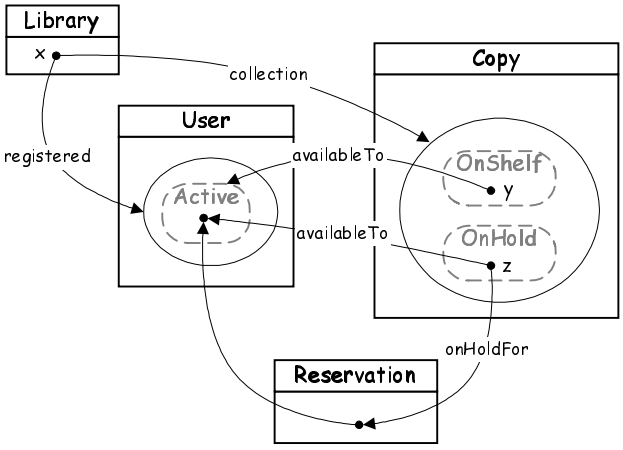

Figure 3: Constraint Diagram

The state diagram of Figure 2 with transitions removed is also a constraint diagram, showing the relationships of sets of objects in different states: onShelf and Out are disjoint, so an object can't be in both states at the same time; they are contained in Accessible, so if an object is onShelf or Out it must be accessible.

Contract boxes (Gil and Kent 1997) are a diagrammatic notation for showing dynamic constraints, that would otherwise be expressed textually using pre/post conditions (contracts). A contract box is shown in Figure 4. It is a pair of constraint diagrams linked by object lifelines. The constraint diagrams show constraints on the objects involved in or affected by the action associated with the box (in this case borrow). The top diagram is a general characterisation of the pre-state, and the bottom of the post-state. The lifelines are a visual aid to identifying how specific objects are affected by

the action. For example, in Figure 4 a lifeline makes it clear that the copy object is moved from being available to out.

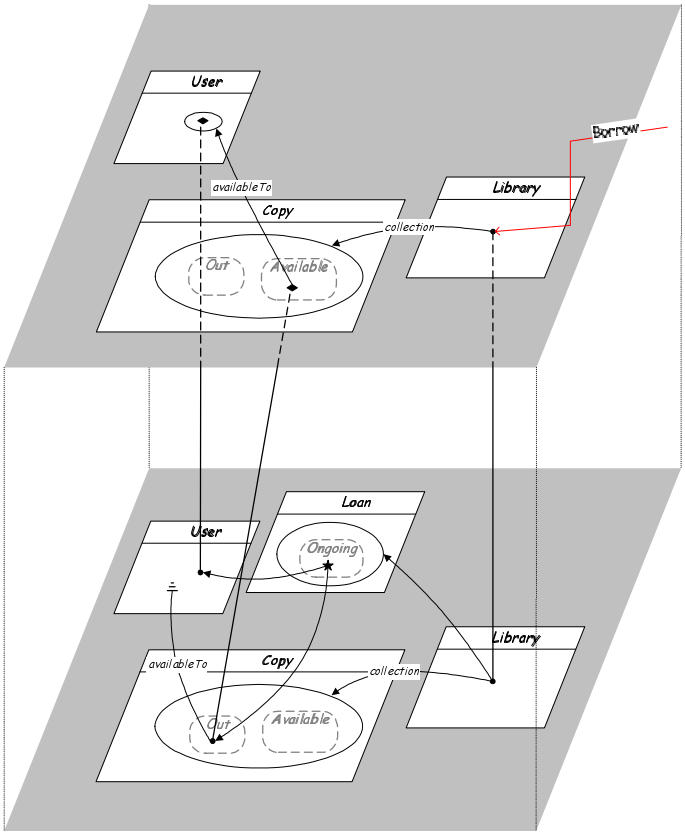

Figure 4: Contract Box

4 Pictorial Semantics

It appears that constraint diagrams and contract boxes can be used to express most, if not all, static and dynamic constraints that can be expressed with invariants and contracts. This includes constraints imposed by type diagrams – cardinality constraints are just a particular form of invariant; and constraints imposed by state diagrams, which contribute to the type model (dynamic types) and action contracts.

Thus using one notation – constraint diagrams and contract boxes (which are just pairs of constraint diagrams), it is possible to express a model with a rich set of constraints, that otherwise requires a range of different diagrams and textual annotations in other notations. This suggests that constraint diagrams and contract boxes could be used to give a semantics to the other notations. This could help to make the semantics easier to understand, and provide an alternative approach to tool support for semantic checking, through the direct comparison of diagrams (see Kent, 1997 and Gil and Kent, 1997, for more specific ideas). The semantics could be formalised by grounding the semantics of

constraint diagrams and contract boxes in a formal language such as Larch (Guttag & Horning 1993), adopting an approach similar to (Hamie & Howse, 1997).

In terms of the meta-model semantics currently proposed for UML, a model of (the abstract syntax of) our notations would have to be built, with well-formedness rules (as invariants or operations) to describe the mapping of all the UML diagrams (whose abstract syntax is also encoded as OO models) into this model. Then, when the state of the meta-model was instantiated with a particular system model, all parts concerned with UML diagrams could be stripped away and no information would be lost.

Contract boxes are just one part of a series of 3D notations currently under development (Gil & Kent 1997). Not only can these notations be used in their own right for modelling, it emerges that they can also be regarded as a visualisation of a single underlying semantic model, of which 2D notations such as sequence, collaboration and activity diagrams are just projections. An important result of this work is the identification of other projections (some 3D, some 2D) which are more precise and richer than current notations, but just as simple and intuitive. Indeed, it may be that this work results in a general improvement to existing 2D notations, which was one of the reasons stated in the Introduction for doing semantics work.

Much work remains to be done. The details need to be worked out for the mappings between our notation and standard 2D notations such as UML. The precise semantics of the former also need to be worked out, to ensure the integrity of the notation and to explore its expressiveness, as well as help with provision of tool support. Tools envisaged include assistance with: 3D visualisation; derivation of 2D and 3D projections; generation of the complete 3D model from projections; use of the complete model to perform integrity checks on the projections; and checking the integrity of the complete model itself.

References

Cook S. and Daniels J. (1994) *Designing Object Systems*, Prentice Hall Object-Oriented Series.

D'Souza D. and Wills A. (1997) *Component-Based Development Using Catalysis*, book submitted for publication, manuscript available at http://www.iconcomp.com.

Gil Y. and Kent S. (1997) Three Dimensional Models, submitted to ICSE98, available at http://www.biro.brighton.ac.uk/biro/index.html

Guttag J. and Horning J. (1993) *Larch: Languages and Tools for Formal Specifications*, Springer-Verlag.

Hamie A. and Howse J. (1997) *Interpreting Syntropy in Larch*, Technical Report ITCM97/C1, University of Brighton, available at http://www.biro.brighton.ac.uk/biro/index.html.

Kent S. (1997) *Constraint Diagrams: Visualising Assertions in Object-Oriented Models*, to appear in Procs. OOPSLA97, ACM Press.

UML (1997) *Unified Modelling Language v1.0*, Rational Software Corporation, available at http://www.rational.com.