To appear in Proceedings of OOPSLA97, ACM Press.

Constraint Diagrams: Visualizing Invariants in Object-Oriented Models

Stuart Kent
Division of Computing,
University of Brighton, Lewes Rd., Brighton, UK.
http://www.it.brighton.ac.uk/staff/Stuart.Kent
Stuart.Kent@brighton.ac.uk
fax: ++44 1273 642405, tel: ++44 1273 642494

Abstract. A new visual notation is proposed for precisely expressing constraints on object-oriented models, as an alternative to mathematical logic notation used in methods such as Syntropy and Catalysis. The notation is potentially intuitive, expressive, integrates well with existing visual notations, and has a clear and unambiguous semantics. It is reminiscent of informal diagrams used by mathematicians for illustrating relations, and borrows much from Venn diagrams. It may be viewed as a generalization of instance diagrams.

Key words: Analysis and design methods, language design, formal methods, software engineering practices.

1 Introduction

This paper proposes a new diagrammatic notation for precisely expressing invariant constraints on object-oriented (OO) models.

In essence, all OO modelling notations may be viewed as documenting constraints either on the set of allowable system states, i.e., the instance diagrams which one is allowed to draw, or on the allowable execution paths through those states.

Current graphical notations are inadequate in the constraints they are able to impose, so need to be supplemented by mathematical assertions describing the more intricate constraints, as is done in methods such as Syntropy (Cook and Daniels, 1994), Catalysis (D'Souza and Wills, 1995, 1997) and BON (Walden and Nerson, 1995). So far this has been the only way of achieving a level of detail necessary for a comprehensive behavioral description, at a level of abstraction that avoids irrelevant implementation or design detail. Unfortunately it is also unintuitive and off-putting to many working software engineers. Parnas (1996) characterizes the problem as follows:

> "Mathematical methods offered to the working software engineer are not very practical [...]. Most, but not all, are theoretically sound but much more difficult to use than the mathematics that has been developed for use in other areas of engineering. [...] We need a lot more work on notation. The notation that is purveyed by most formal methods researchers is cumbersome and hard to read. Even the best notation I know (mine of course) is inadequate."

The diagrammatic notation proposed here, called ***constraint diagrams***, replaces the need to write many assertions mathematically and is potentially more intuitive to, hence more likely to be used by, the practising software engineer. The notation has similarities with informal diagrams used by mathematicians for illustrating properties of functions and relations (e.g., Gerstein 1987) and borrows much from Venn diagrams. It may be viewed as a way of describing sets of instance diagrams, and is a natural development of instance (object) diagrams in UML (UML 1997).

Whilst we have some clear ideas of how the notation might be used to improve the lot of the software modeler/designer, it is inevitable that the focus of this first paper on the subject is on describing the notation. This is best done by relating it to something that is more familiar. Therefore a careful characterization of an object-oriented model is given in Section 2, using diagrammatic notation from UML, supplemented with invariants expressed mathematically where UML diagrams are unable to express the desired constraints. This section also serves to demonstrate the claim made above that there are some constraints that can not be expressed using diagrammatic notations currently proposed for object-oriented modelling.

Section 3 introduces the notation by using it to express the invariants identified in section 2. This demonstrates that it is indeed more expressive than current diagrammatic notations.

Section 4 is a discussion focussing on:

- use of the notation to visualize action contracts (pre/post specifications),
- relationships with other diagrammatic notations,
- limitations, possible extensions, semantic issues and usability of the notation,
- impact on automated tool support for modelling.

The latter gives a sketch of how the notation could be used as a basis for automated tool support to the construction of precise specifications of constraints from instance diagrams, and vice-versa. It is proposed that similar techniques could be used for semantic checking of models.

A summary of the notation is given in an appendix.

2 Library System in UML and Catalysis

This section sets the context for the constraint diagram notation introduced in this paper, by specifying a small case study - a library system - using the diagrammatic notation of UML supplemented with mathematical notation from Catalysis to express those constraints that can not be expressed using diagrams from UML (or, indeed, other OO modelling notations). UML has been chosen as it incorporates and unifies many of the notations of its predecessors, so may be viewed as representative of them, and is fast becoming the de facto standard in OO modelling.

The approach taken is to express as much as possible diagrammatically, resorting to mathematical assertions only where strictly necessary. This serves three purposes: it proves the point that there are some constraints that can not be expressed diagrammatically using

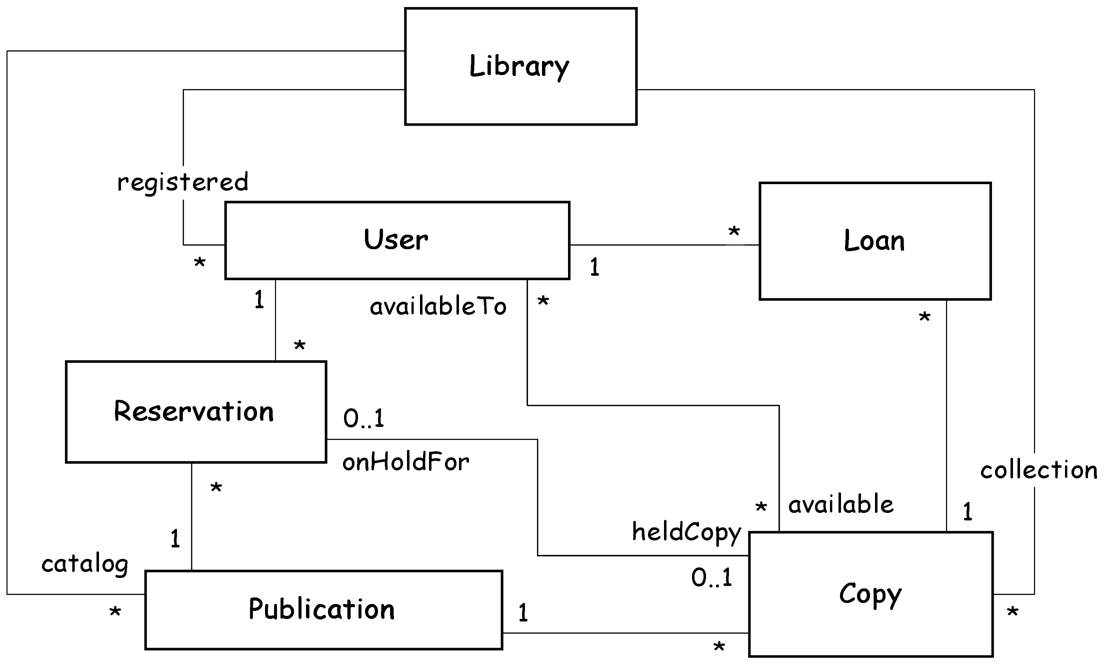


Figure 1: Type diagram for library

existing notations; it provides the examples (those constraints) for the section following which introduces the constraint diagram notation; it illustrates the pattern of specification required (mixing of type and state diagrams) to express as much as possible using diagrams from existing notations.

2.1 Informal Requirements

The general requirements are to produce a computerized system to support the management of loans in a university library. A library maintains a catalog of publications which are available for lending to users. There may be many copies of the same publication. Publications and copies may be added to and removed from the library. Copies available for lending may be borrowed by active users registered with the library. When a publication (or more specifically a particular copy) has been borrowed it is on loan, and is not available for lending to other users. However, it still belongs to the library and so is still part of its collection. Users are able to reserve publications, when none of the copies are available for loan. A user may not place more than one reservation for the same publication. When a copy is returned after it has been out on loan, it may be put back on the shelf or, alternatively, held for a user who has reserved the publication of which it is a copy. This may be done immediately on return, or delayed, and done as part of a batch of returned copies.

2.2 Type Diagram

The main type diagram for the library is given in Figure 1. This is UML notation. Publication, User and Copy are types of object we expect to find in a library. A publication is a record of all the details of a book: title, authors, ISBN, etc. A publication may have many copies (or none), which correspond to the physical books. A copy only has one set of publication details. A copy may be availableTo loan to many users, and a user may have many copies which are available for loan to her. All the publications known to the library are in the catalog, all the copies form the library's collection, and all the users known to the library must be registered.

Loan and Reservation characterize objects used to record loans and reservations, respectively. A loan records which copy is on loan to which user, and a reservation records which publication has been reserved by which user. A copy may be put onHoldFor a reservation, which means that it is waiting to be collected by the user who made the reservation. After the copy has been collected the reservation will have been fulfilled.

Some associations in the diagram have been given rolenames at one or both ends. Following Catalysis (D'Souza and Wills, 1997) we adopt a convention for constructing default rolenames where they have been omitted.

- If the association is unlabeled the name of the type at either end is used, with the first letter in lowercase and pluralized if this makes sense: for example we can refer to the loans of a user or the user associated with a loan.
- If the association is labelled at one end only, then the name used at the reverse end will be that label prefixed with ~: for example ~collection, ~catalog, ~registered will always take us back to the library object(s) from an object of the corresponding types.

2.3 States

The type diagram on its own can not express all the constraints that we would wish to express for the library system. In particular, multiplicity restrictions do not allow relationships between associations to be expressed. For example, it should be a constraint that the publication of a copy onHoldFor a reservation is the one that has been reserved. Figure 2 shows an instance diagram which satisfies the type model, but which does not satisfy this constraint. It satisfies the type model because it satisfies all the multiplicity restrictions. It does not satisfy the constraint because the held copy is not associated with the same publication as the reservation it is held for.

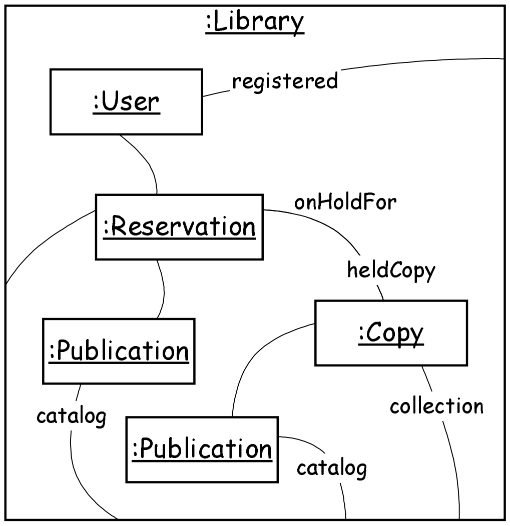
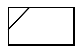


Figure 2: Instance diagram not satisfying constraint, but satisfying type diagram

However, we have not yet made any use of states. Often further constraints can be imposed by introducing states for particular types of object, and then adding further multiplicity conditions on associations, which will depend on the state chosen. In this section we show that for this system (and by extrapolation probably for most systems) this is still not enough. In particular the constraint above can not be expressed in this way.

Figure 3 shows four diagrams defining the states of particular types. The notation used is similar to that used in OMT (Rumbaugh et al. 1991), Syntropy and Catalysis, transformed to UML, which strangely says nothing about defining states outside state diagrams. The filled in subtype arrow indicates partitioning: the type Copy is partitioned between those objects in the state Available and those in the state Unavailable. Subtyping is used to bring states into type diagrams, based on the semantic intuition that every Available object is a Copy object, but not vice-versa.

In an attempt to unify notation, we have chosen to use a box with rounded corners to represent a state, i.e. the same shape of box that is used in state diagrams. The aforementioned methods generally use  .

In the true spirit of subtyping states can be further constrained: they can have associations that objects not in that state do not have and may further constraint multiplicities on associations obtained from the supertype/superstate.

The diagrams in Figure 4 place additional constraints on states for this example. The left hand diagram indicates that when the copy is in the OnHold state it is onHoldFor exactly one reservation in the ToBeCollected state, but is not onHoldFor any reservation when in a different state. Similarly, a reservation in the ToBeCollected state has exactly one heldCopy, but no held copies when in a different state. Clearly this diagram does place some constraint on the nature of the copy held, that it is in the state OnHold, but this is not enough; in particular it does not state that the publication associated with the held copy is the same as that reserved.

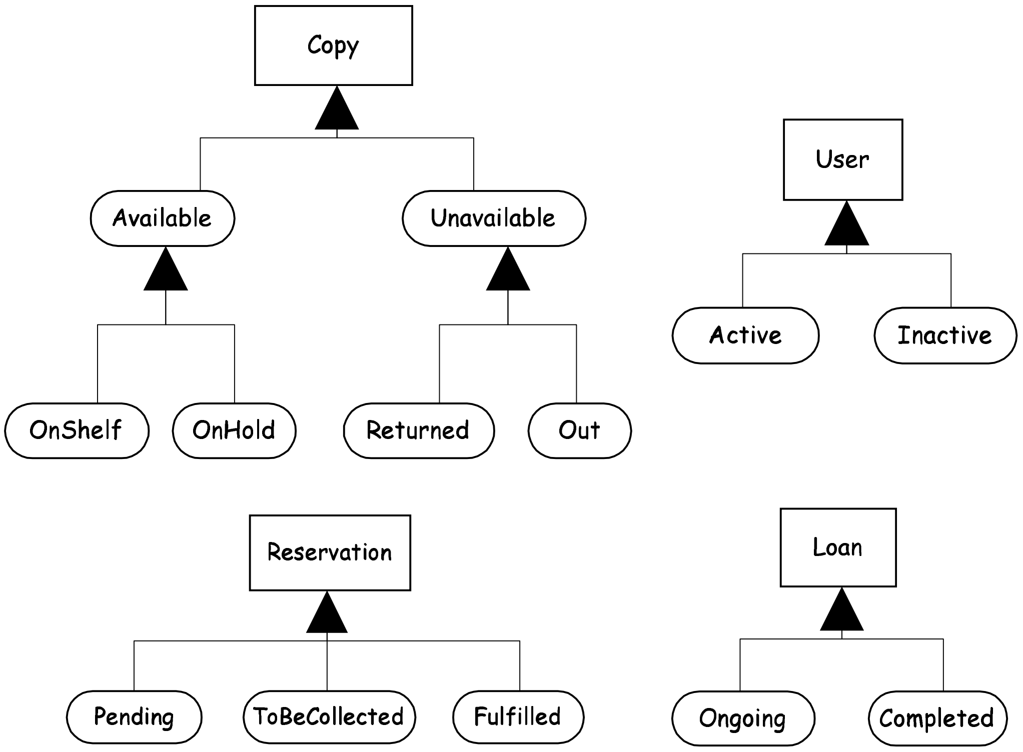


Figure 3: Type diagrams showing states

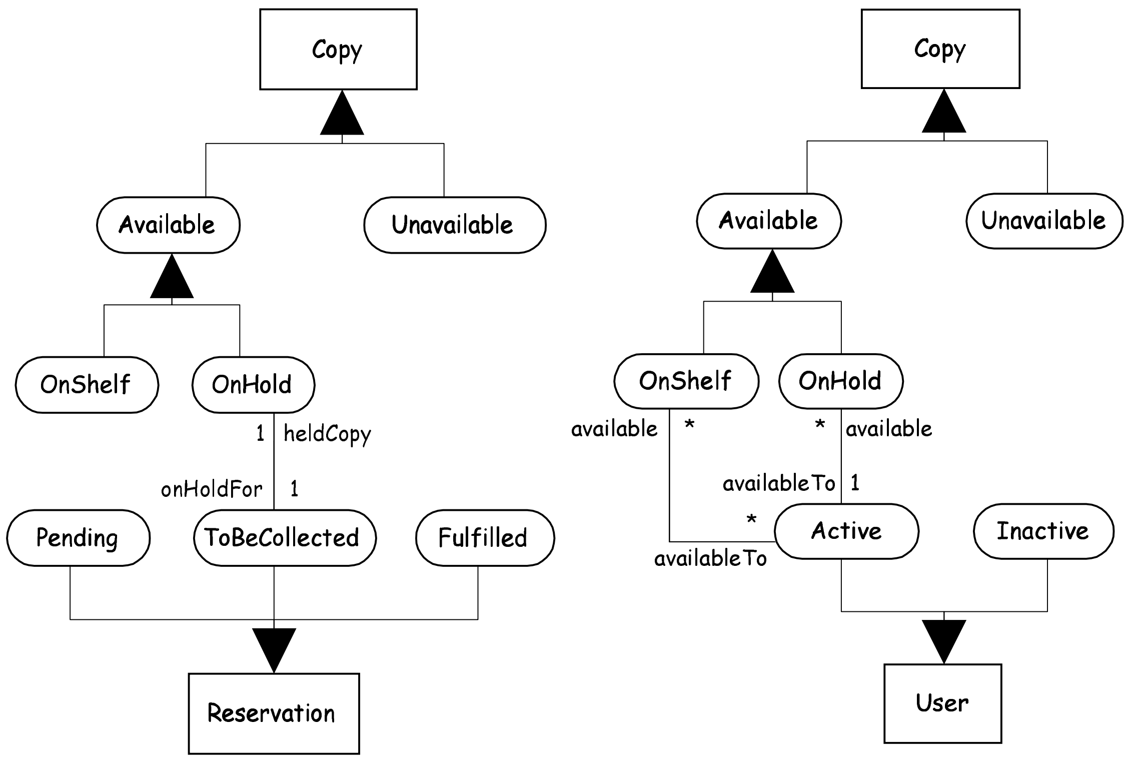


Figure 4: Multiplicities on associations between states

Similarly the right hand diagram places constraints on the number and state of users to which a copy is available, depending on the state of the copy. However, again it is not sufficient; specifically it does not state that the single user to whom a copy OnHold is available for loan is the one who made the reservation, or that a copy OnShelf is available to all active users.

Figure 5 is attempting to capture the idea that a copy may have only one onGoing loan at any point in time (it can only be out on loan to one user). This it does by introducing the current association. This nearly achieves the desired effect, but again is not quite enough. It is necessary to say how the loan identified through this link is related to the association indicated between Copy and Loan, which is taken from the type model. Is it included in this association, in which case it must always be explicitly excluded when the loan history of a copy (i.e. all completed loans for that copy) is retrieved, or is it always excluded?[1]

2.4 Invariants

The previous section identified the following constraints that can not be expressed using existing diagrammatic notations:

1. The publication associated with the held copy is the same as that reserved.
2. The user to whom a copy OnHold is available for loan is the one who made the reservation.

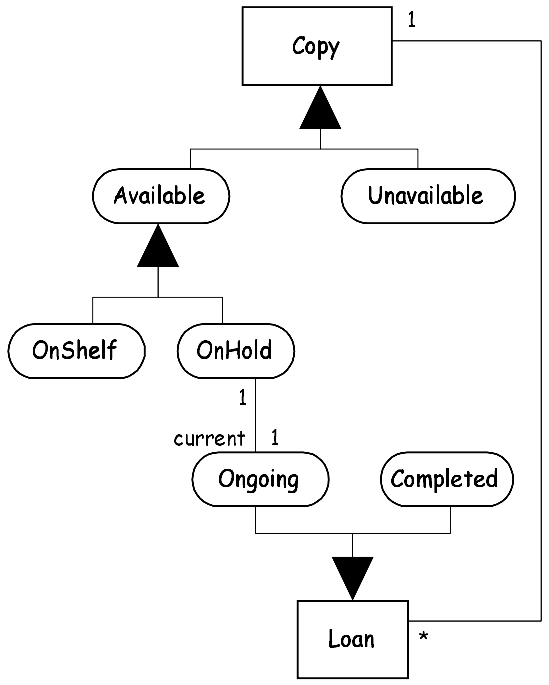


Figure 5: Associations between states of Copy and Loan

1. To maintain a consistency in design decisions we would probably wish to say it is included: no distinction has been made between ongoing and completed loans for a user, so why should we be forced by a lack of expressiveness in the notation to make a similar distinction here.

3. A copy OnShelf is available to all active users.

4. A copy may be associated with only one ongoing loan.

In addition there are some additional constraints which have not been touched upon:

5. The collection is equivalent to all the copies of all publications in the catalog.

6. The reservations known to the library are exactly all the reservations made by users or all the reservations there are for the publications in the catalog.

7. The loans known to the library are exactly those associated with copies in the collection and these are exactly those associated with registered users.

These seven constraints can not be expressed even if one is prepared to fully exploit the inclusion of states on type diagrams. More naive designs which e.g. use associations to model states will come up with many more constraints relating associations which can not be expressed using current diagrammatic notations. It seems that naive designs are likely to be par for the course, considering, for example, that the latest UML documentation (UML 1997) makes no mention of including states on type diagrams.

Methods such as Catalysis and Syntropy solve this problem by the use of mathematical assertions to write ***invariants***. The invariants stated are written in mathematical notation using Catalysis syntax, as follows:

1. The publication associated with a copy OnHold is the same as that which has been reserved.

$$\forall c{:}Copy,\ (c.OnHold \wedge c \in collection) \\ \Rightarrow c.onHoldFor.publication = c.publication$$

$c \in collection$ has been included as really this constraint only applies to copies in the collection; it can not be guaranteed e.g. for copies passed into the library via a parameter which have yet to be placed in the collection. Of course multiplicity constraints in type diagrams don't make this distinction.

2. The user to whom a copy OnHold is available for loan is the one who made the reservation.

$$\forall c{:}Copy,\ (c.OnHold \wedge c \in collection) \\ \Rightarrow c.availableTo = c.onHoldFor.user$$

3. A copy OnShelf is available to all active users.

$$\forall c{:}Copy,\ (c.OnShelf \wedge c \in collection) \\ \Rightarrow c.availableTo = registered[Active]$$

where $registered[Active]$ is the set of registered users in the active state.

4. A copy may be associated with only one ongoing loan.

$$\forall c{:}Copy,\ ((c.Out \wedge c \in collection) \\ \Rightarrow |c.loans[Ongoing]| = 1) \\ \wedge ((\neg c.Out \wedge c \in collection) \\ \Rightarrow |c.loans[Ongoing]| = 0)$$

This will work whether the current association suggested in Figure 5 on page 4 is included or not. If this association is included, then the above constraint is not necessary; instead the constraint $\forall c{:}Copy,\ c \in collection \Rightarrow c.current \in c.loans$ is required.

5. The collection is equivalent to all the copies of all publications in the catalog.

$$collection = catalog.copies$$

The meaning of navigation in this instance originated in Syntropy (Cook and Daniels 1994, p57) and is used in Catalysis. catalog is a set. catalog.copies is then the union of the sets arrived at by navigating the copies link from each publication in catalog. That is catalog.copies is equivalent to writing

$$\{x{:}Copy \mid \exists y{:}Publication,\ y \in catalog \wedge y.copy \\ = x\}$$

The Syntropy/Catalysis interpretation of navigation is fully exploited in the constraint diagram notation introduced here.

6. The reservations made by users are exactly all the reservations there are for the publications in the catalog.

$$registered.reservations \\ = catalog.reservations$$

7. The loans associated with copies in the collection and these are exactly those associated with registered users.

$$collection.loans = registered.loans$$

3 Constraint diagrams: visualizing invariants

This section introduces the constraint diagram notation, by giving a constraint diagram for the invariants written mathematically in §2.4, p.4. Thus it demonstrates how constraint diagrams can express invariants that can not be expressed diagrammatically in current notations. A summary of the notation can be found in the Appendix.

Perhaps the easiest invariants to show diagrammatically are 5, 6 and 7. The corresponding constraint diagram is given in Figure 6. The basic notation can be summarized as follows:

- Associations are depicted as relations between *sets* of objects.
- Venn diagrams can be used to express relationships between associations. In this case the relationships are simply equivalences, i.e. a number of arrows target on the same set. In general, relationships such as intersection and containment can be depicted. Examples of such relationships appear throughout this section.
- There are four ways of depicting a set depending on the number of elements in the set:

  ● has 1 element,

  ○ has 0..1 elements,

  ⬭ has 0 or more elements,

  ⬭ $\boxed{n}$ has n elements.

- Types are depicted as (universal) sets.
- Navigation always begins at an object or set with no incoming arrows. In this diagram the only such item is the self object. Enclosing the types within the object has no semantic relevance. It is an aid to drawing the diagram, and gives a sense of where navigation begins.

Links are directed for the following reason. Consider the top diagram in Figure 7. The association available indicates, for any user, which set of copies is available for loan to that user. If the arrow was omitted then we would not know in which direction to read the diagram. Reading the link in the other direction would mean that any set of copies are always available only to a single user which is the same for all copies in that set. This is clearly not

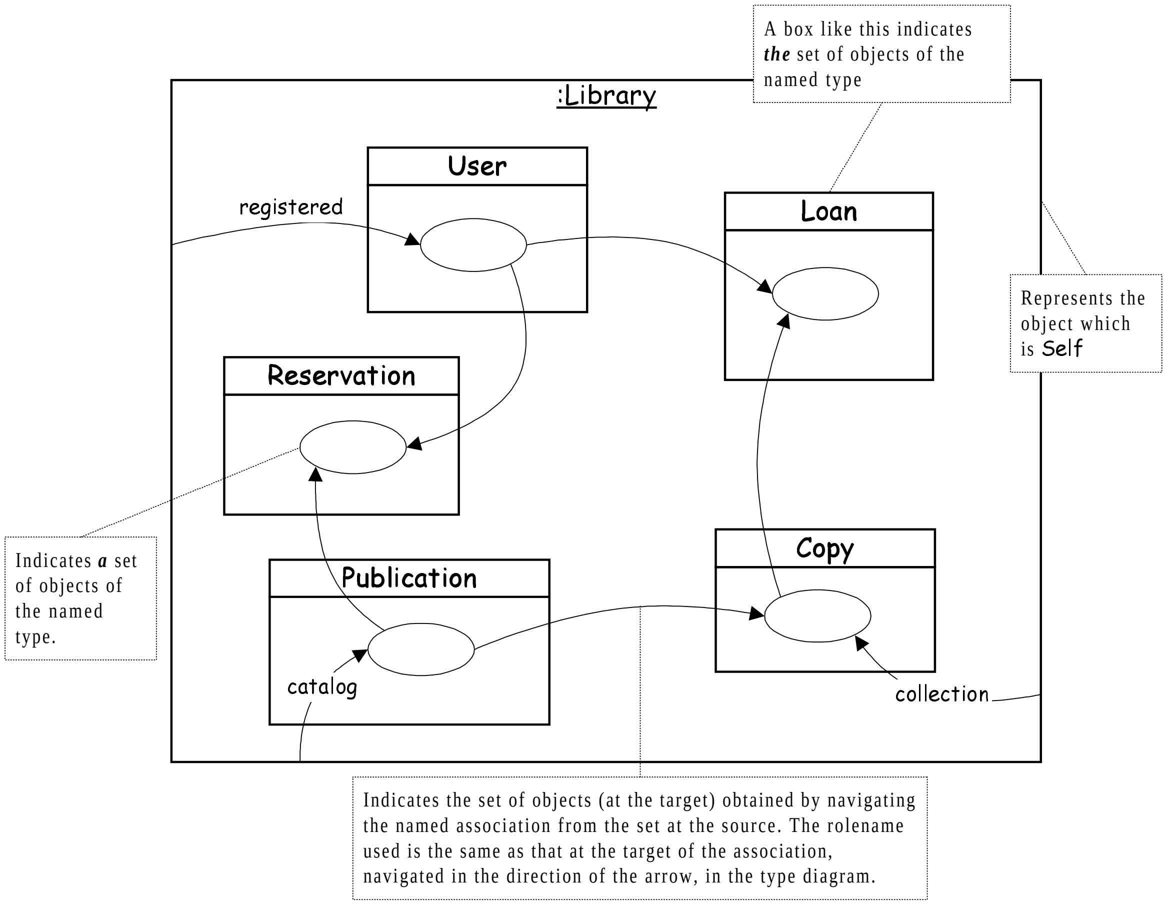

A box like this indicates *the* set of objects of the named type
:Library
User
registered
Loan
Represents the object which is Self
Reservation
Indicates *a* set of objects of the named type.
Publication
Copy
catalog
collection
Indicates the set of objects (at the target) obtained by navigating the named association from the set at the source. The rolename used is the same as that at the target of the association, navigated in the direction of the arrow, in the type diagram.

Figure 6: Constraint diagram for invariants 5-7

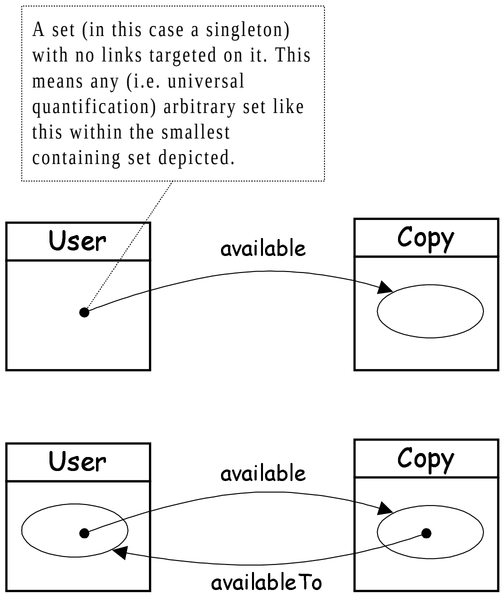

A set (in this case a singleton) with no links targeted on it. This means any (i.e. universal quantification) arbitrary set like this within the smallest containing set depicted.
User
available
Copy
User
available
Copy
availableTo

Figure 7: The availableTo link

the case. Instead we could draw the bottom diagram, which says that for any user there is a set of copies available to that user (possibly empty), and for any copy in that set, that user is *one of* the users to which the copy is available for loan.

The constraint diagram, Figure 8, for invariant

1. The publication associated with a copy OnHold is the same as that which has been reserved.

$\forall c{:}Copy,\ (c.OnHold \wedge c \in collection)$
$\Rightarrow c.onHoldFor.publication = c.publication$

introduces a new piece of notation: a shorthand for representing those objects in a particular set (in this case collection), which are in a particular state (in this case OnHold). It is probably worth highlighting how a constraint can be placed on an arbitrary object chosen from a particular set: by showing a set (if a singleton then this corresponds to an object) with no arrows targeted on it. In this case, c represents any object in the state OnHold and in the collection. The label c is not strictly necessary; it has been included so to clarify the mapping from the diagram to the mathematical form of the invariant.

Equivalently one could draw Figure 9, where, as in §2.3, p.2, a box with rounded corners is used to represent the set of objects in the named state.

The short hand notation is especially useful when objects in two or more different states need to be referred to.

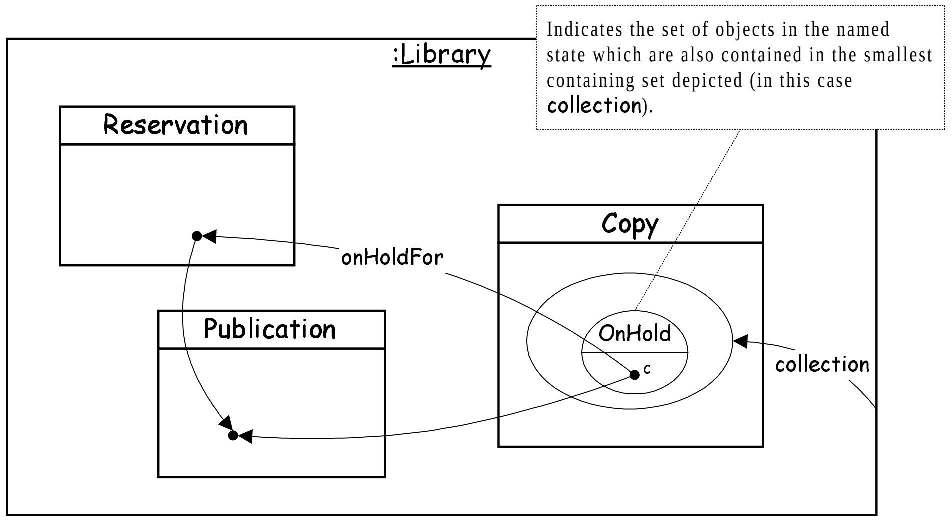

:Library
Indicates the set of objects in the named state which are also contained in the smallest containing set depicted (in this case collection).
Reservation
Publication
Copy
onHoldFor
OnHold
c
collection

Figure 8: Constraint diagram for invariant 1

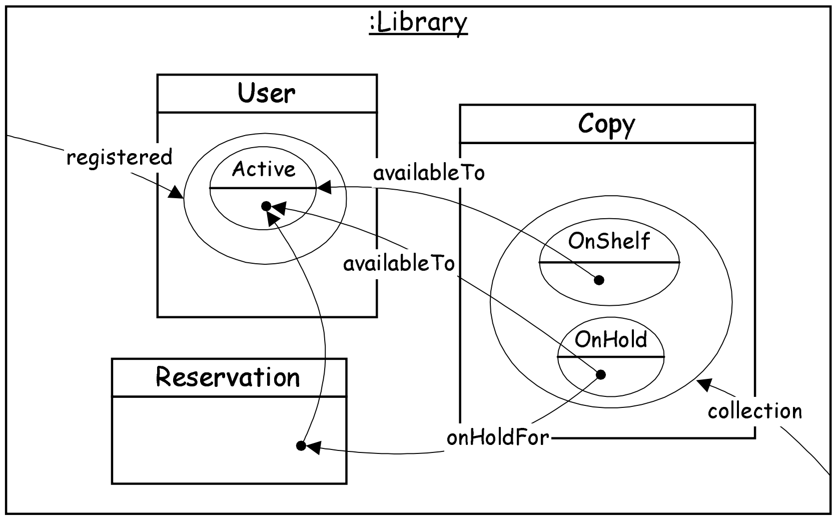


Figure 10: Constraint diagram for invariants 2, 3

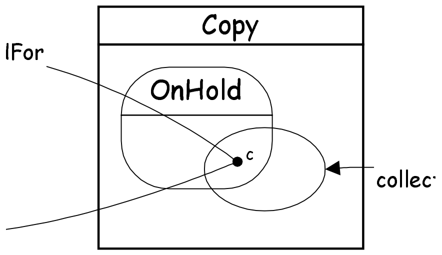


Figure 9: States on constraint diagrams

It is natural to combine invariants

2. The user to whom a copy OnHold is available for loan is the one who made the reservation.

$\forall c{:}Copy, (c.OnHold \land c \in collection)$
$\Rightarrow c.availableTo = c.onHoldFor.user$

3. A copy OnShelf is available to all active users.

$\forall c{:}Copy, (c.OnShelf \land c \in collection)$
$\Rightarrow c.availableTo = registered[Active]$

into the same diagram, Figure 10, as they both concern the same association. There is little that requires comment here.

Finally, Figure 11 is the diagram for the invariant

4. A copy may be associated with only one ongoing loan.

$\forall c{:}Copy, ((c.Out \land c \in collection)$
$\Rightarrow |c.loans[Ongoing]| = 1)$
$\land ((\neg c.Out \land c \in collection)$
$\Rightarrow |c.loans[Ongoing]| = 0)$

This diagram introduces one new piece of notation, which is explained on the diagram in one place that it is used. In the second place it is used, it indicates that, apart from the single object depicted, there are no other Ongoing loans in the set of loans for a copy that is Out, i.e. a copy which is Out is associated with exactly one Ongoing loan.

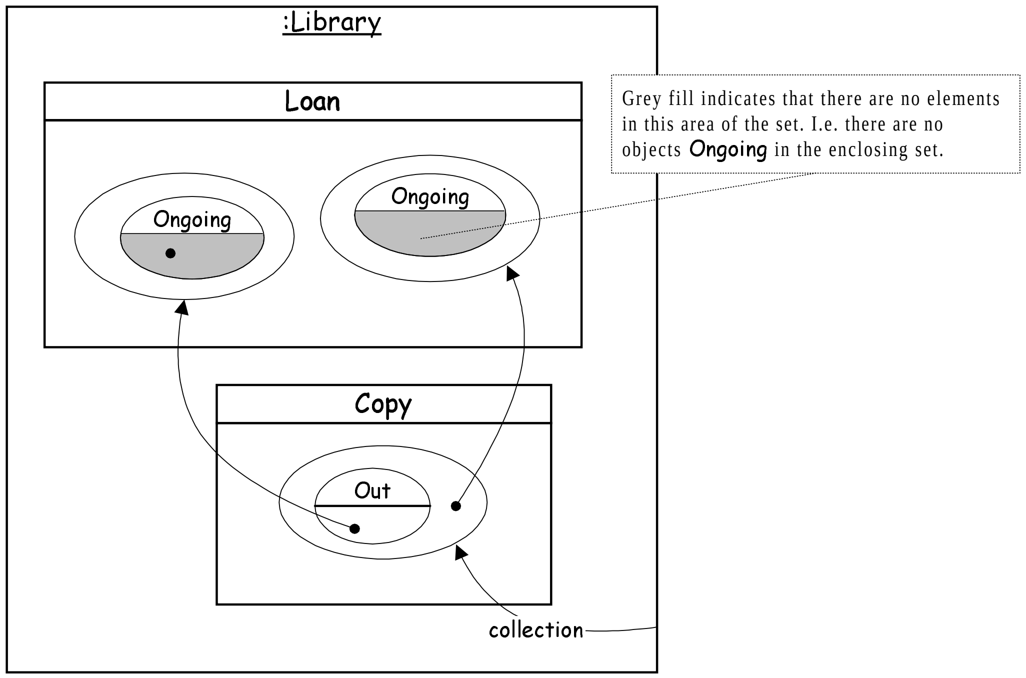

:Library
Loan
Ongoing
Ongoing
Copy
Out
collection
Grey fill indicates that there are no elements in this area of the set. I.e. there are no objects Ongoing in the enclosing set.

Figure 11: Constraint diagram for invariant 4

For drawing by hand or in a tool where shading the desired region may be difficult, a cross may be placed in the affected area as an alternative.

4 Discussion

This section discusses:

- use of the notation to visualize action contracts (pre/post specifications),
- relationships with other diagrammatic notations,
- limitations, possible extensions, semantic issues and usability of the notation,
- impact on automated tool support for modelling.

4.1 Visualizing action contracts

Some preliminary work (Kent 1997) has been done on using constraint diagrams to visualize behavioral specifications for actions, expressed in terms of pre and post conditions. A pre/post specification is sometimes called a contract. In UML and its precursors, state diagrams (based on Harel's statecharts – Harel, 1987) are used to specify dynamic behavior, and as is the case with invariants, they are limited in the constraints they are able to express. Methods such as Catalysis and Syntropy therefore supplement state diagrams with pre and post conditions which must be expressed using mathematical notation if precision is required.

Constraint diagrams can be used unchanged to express pre-conditions. However, a post-condition is predicated over two states, and this causes further complications. The idea proposed to solve this is to borrow from the Catalysis idea of a ***filmstrip*** (a sequence of instance diagrams), replacing instance with constraint diagrams. An example is given in Figure 12. This is the filmstrip characterizing the post-condition of the action borrow(u:User,c:Copy). Items appear on the first constraint diagram if they are subject to change by the action. Three new pieces of notation have been introduced, and these are explained on the diagram. It may be considered that the notation for null is redundant: couldn't the association just be omitted from the diagram? Omission doesn't work as constraint diagrams are, by their very nature, partial: if an association is omitted, it just means that the diagram imposes no constraints on it. The symbol chosen for null is used in many standard books on data structures (e.g., Thomas et al., 1988) to represent a null pointer. It also has the appearance of an arrow, suggesting the direction in which the association should be read.

For further clarification of the diagrams, the post-condition written in mathematical notation is given next.

borrow(u:User, c:Copy)
<u>pre</u>
...
<u>post</u>
c is out and no longer available for lending.

$c.availableTo = null \wedge c.Out$

The loan of c to u is recorded and marked as ongoing.

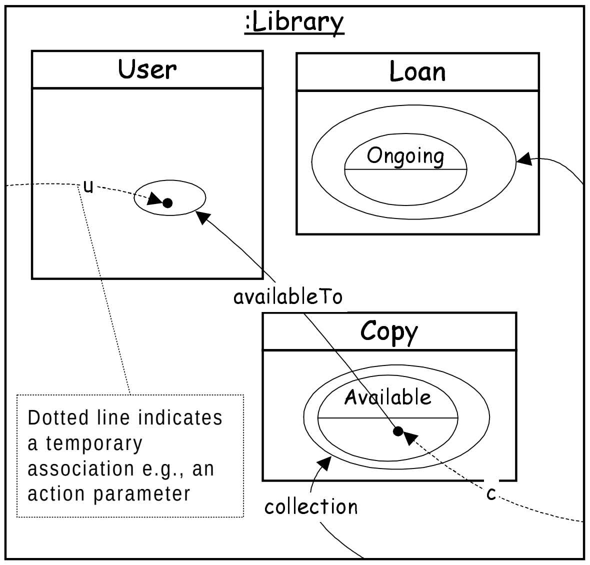
$\exists l : loan, l \in new \wedge l.Ongoing \wedge l.user = u \wedge l.copy = c$

Although presented as two separate diagrams here, a case tool could show the change dynamically e.g. by "running" the filmstrip with changes shown in different colors.

Further work is required to reduce the complexity of these filmstrips for post-conditions more sophisticated than this.

4.2 Relationships with other diagrammatic notations

States in state and type diagrams. In constraint diagrams a state is viewed as the set of objects in that state. We have already argued for the use of the same shape of box (one with rounded corners) to represent a state wherever it appears. However, constraint


:Library
User
Loan
Ongoing
u
availableTo
Copy
Available
Dotted line indicates a temporary association e.g., an action parameter
collection
c

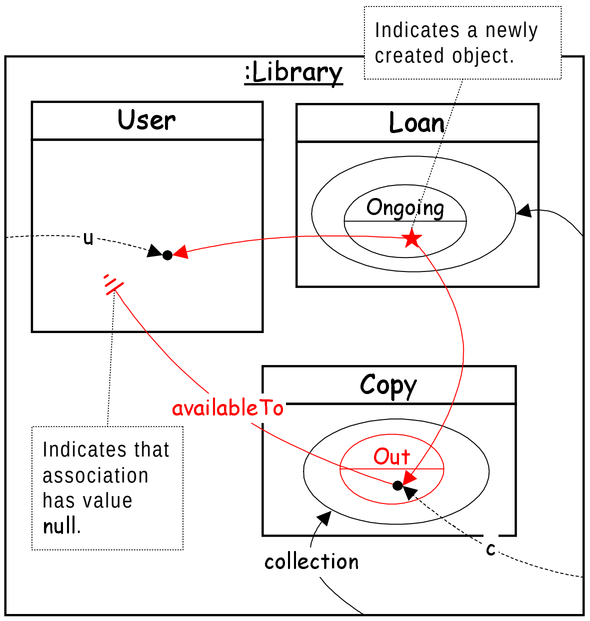

Indicates a newly created object.
:Library
User
Loan
Ongoing
u
availableTo
Copy
Out
Indicates that association has value null.
collection
c

Figure 12: Visual specification of borrow

diagrams also provide a natural way of expressing constraints on state partitioning and substating that are expressed in different ways on type diagrams and state diagrams.

Consider the states for the type Copy, as defined by Figure 3 on page 3. Figure 13 gives the same definition as a constraint diagram, remembering that a state on these diagrams represents the set of objects in that state. This is like a state diagram with the transitions omitted. The area around the states has also been grayed out to indicate that the type is partitioned by the two top-level states. This is required because, as stated in Section 4.1, constraint diagrams are partial; without explicitly indicating that the states partition, there is the possibility that there may be other states defined on a different diagram.

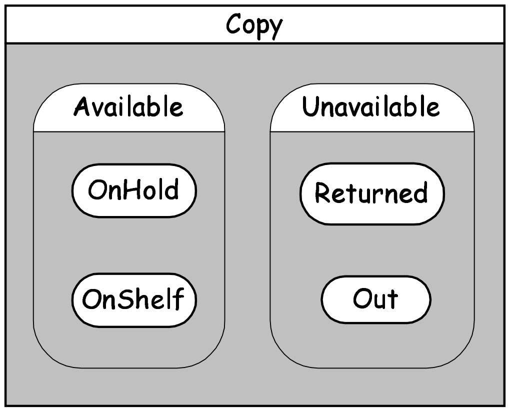


Figure 13: A grouping of states for Copy

A second constraint diagram (Figure 14), which is consistent with Figure 13, may be drawn. It defines a different grouping of nested

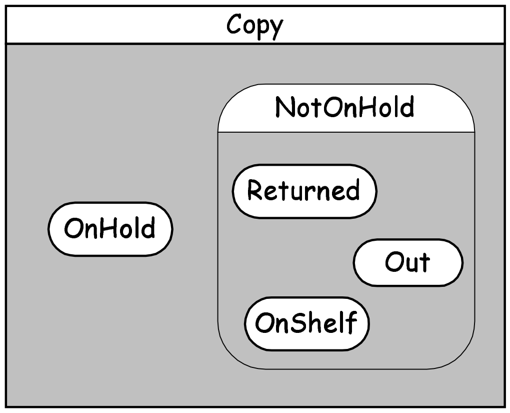


Figure 14: Another grouping of states for Copy

states for Copy. This can be expressed using states on one or more type diagrams, though this seems more cumbersome (Figure 15). It can not be expressed using a state diagram, as there may only be one state diagram per type, and a state diagram enforces a single grouping of nested states. The closest approximation is given by Figure 16, where transitions have been omitted as they are irrelevant to this discussion. Unfortunately, the usual interpretation of state diagrams treats, e.g., the two OnShelf states as different.

Subtyping in type diagrams. A similar notation could be used for showing relationships (partitions, disjointedness, etc.) between static subtypes. This accords with explanations of subtyping based on Venn diagrams, in e.g. Wirfs-Brock et al. (1990).

Object (instance) diagrams in UML. Object diagrams in UML already include a notation for showing sets of objects - placing a * in an object. However they do not allow sets to overlap, and they have no representation for navigation expressions (in our notation, arrows from sets targeted on other sets), which is fundamental to the expressiveness of constraint diagrams. Thus constraint diagrams could be regarded as the natural development of what has already been started in UML.

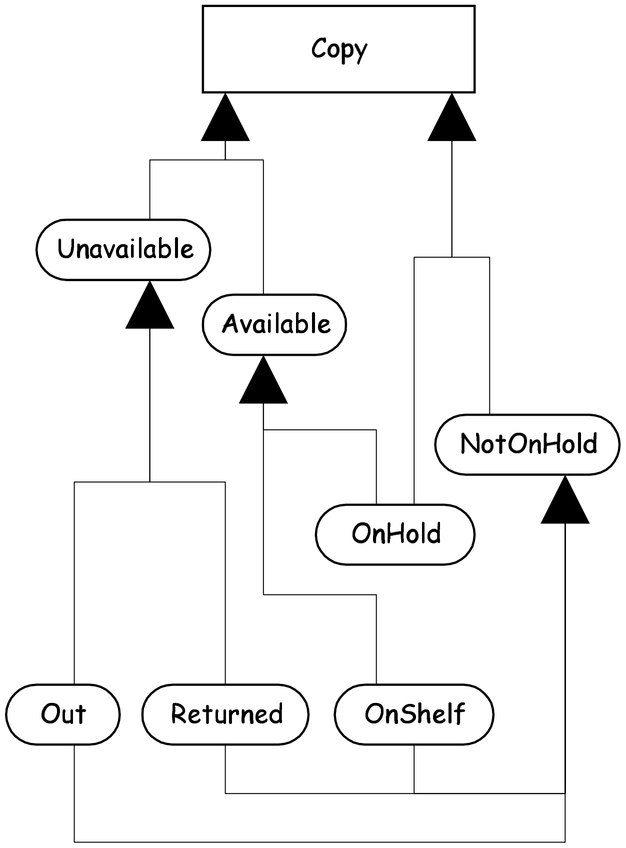


Figure 15: Grouping states on a type diagram

4.3 Limitations, Extensions, Semantics, Usability

Limitations and extensions. The assertions expressed using the notation in this paper have not involved all kinds of logical operators and connectives. Notable omissions are existential quantification and disjunction. Notation is given in the appendix for existential quantification over sets. With regard to disjunction, the two disjuncts could be represented as different constraint diagrams and composed disjunctively (see below).

Another limitation is the difficulty of showing constraints on attributes which hold values, e.g. of type Integer, rather than identify objects. It is expected that this could be cured by showing values like objects on instance diagrams, sets of values like sets of objects on constraint diagrams, and relationships between them as associations.

The notation also suffers from the same limitations as Venn diagrams - it is hard to show the intersection of more than three sets. We have a (rather cumbersome) notation that will get round this problem, but do not believe it will occur very often. There is inevitably other published research on this problem, but we have yet to find it.

Semantics. Work has begun on describing the semantics of the notation in terms of logical theories in Larch (Guttag and Horning, 1993). This builds upon recent work in interpreting existing modeling notations (Bourdeau and Cheng, 1995; Hamie and Howse, 1997). The aim of this work is to check the consistency and expressiveness of the notation - basically to ensure that no stone has been left unturned. A particular area of interest is to look at diagram composition, both disjunction and conjunction. This may open up

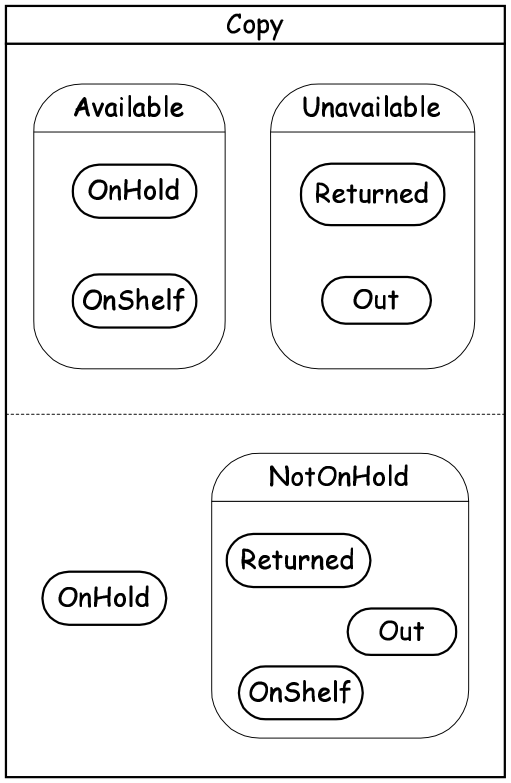


Figure 16: A possible state diagram for Copy (transitions omitted)

new avenues of investigation into the expression of frame conditions (see e.g. Borgida et al., 1995). As hinted above, the notation looks as if it could be used to give the semantics of existing diagrammatic notations. With its own formal semantics, we would then be in a position to provide formal, yet intuitive, semantic underpinnings to other OO modeling notations, such as those in UML. A more detailed proposal to this effect is given in (Kent et al., 1997).

Use of the notation. Further investigation is required into (a) whether the notation would be useful in practice and whether it is any more intuitive and easier to use than mathematical assertions; and (b) what are the most appropriate ways to use it, e.g. in conjunction with other notations such as state and type diagrams. It would also be interesting to compare its use with other approaches to making assertions easier to write and understand such as ADL (ADL, 1997).

4.4 Impact on automated tools

The paper has limited the use of constraint diagrams to expressing diagrammatically what otherwise has to be expressed mathematically in existing notations. It is hoped that practising engineers will find constraint diagrams more palatable than writing the math though we have no evidence to support this claim. However, this is not, we believe, the only or even main contribution of the notation. More importantly, it could open the way to providing automated support for the construction of sophisticated constraints in CASE tools, as well as semantic cross checking between different views of a model (e.g. between instance diagrams and type diagrams, type diagrams and state diagrams, etc.).

First notice the correspondence between constraint diagrams and instance diagrams; constraint diagrams are essentially characterizations of sets of instance diagrams. We can imagine a tool that could assist with constructing constraint diagrams from instance diagrams and vice versa.

Suppose, for example, that a tool was presented with the instance diagrams in Figure 17, where the first should be accepted by the model but the second not. The goal is to construct the constraint diagram which ensures that this is the case. Assuming that a type diagram has been drawn, a constraint diagram can be derived which represents the multiplicity constraints for associations mentioned in the instance diagrams, composed with the part of Figure 6 relevant to these associations. This is given in Figure 18.

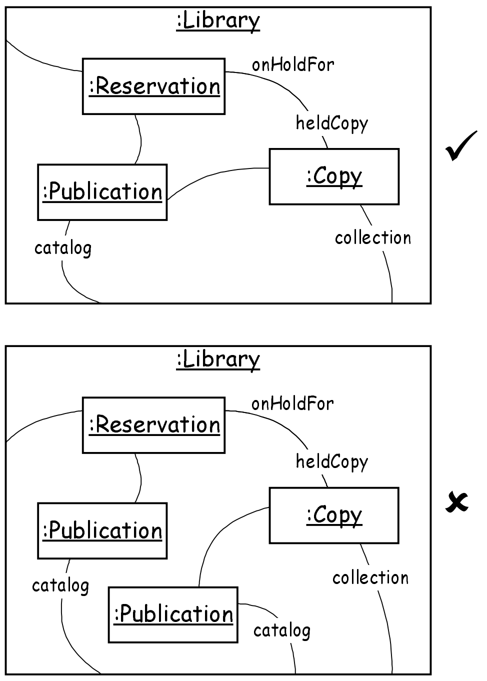


Figure 17: Instance diagrams: input to tool

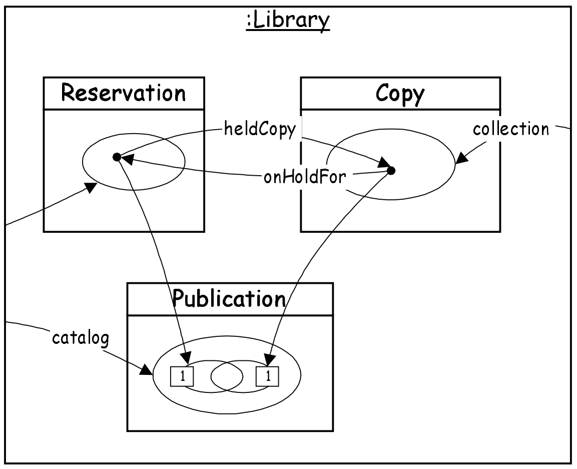


Figure 18: Constraint diagram: derived by tool from existing diagrams

Focussing on the Publication type, Figure 19 (a) admits two possibilities, (b) and (c). Pattern matching with the instance diagrams indicates that (b) should be accepted but (c) rejected. This gives rise to the final constraint diagram in Figure 20.

This is a very simple example and some details of the various steps have been omitted. However, it is hoped that the example is convincing enough to warrant that the approach is worthy of further investigation.

As this paper has illustrated, the mapping from constraint diagrams to mathematical notation is quite systematic. Once a constraint diagram had been constructed a tool could automatically generate an equivalent mathematical form, if desired.

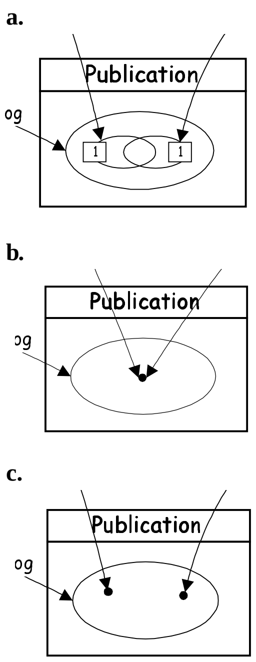


Figure 19: Fixing the key constraint

Constraint diagrams could also assist with cross-checking between different views of a model. Constraints imposed by existing diagrammatic notations may be expressed using constraint diagrams instead (for example, an indication of how multiplicity constraints may be characterized has just been given). This suggests that a systematic hence automated conversion could be performed. Cross-checking is then a matter of "overlaying" constraint diagrams and looking for conflicts. It may be possible to automate this process, in which case any conflict could be shown visually using an appropriate constraint diagram.

By a similar process instance diagrams could be checked against constraints by converting an instance diagram into a constraint diagram[1] and overlaying as above.

Finally, as constraint diagrams characterize sets of instance diagrams, they could be used for animating models where actions have under-determined specifications: that is, given a starting instance, there may be a set a of possible instances which could be reached and that satisfy the specification. This goal will be more achievable once the extensions of the notation for visualizing action specifications have been fully worked out.

Acknowledgments

I am grateful to the BIRO research team at Brighton, in particular Franco Civello, John Howse and Richard Mitchell for many useful comments and feedback. Thanks are also due to Alan Wills and Desmond D'Souza whose work on Catalysis has had considerable influence on my recent thinking. Finally I would like to thank the anonymous reviewers and the OOPSLA program committee, in particular Yossi Gil, for their positive and detailed comments and feedback. This research was partially funded by the UK EPSRC under grant GR/K67304.

1. A constraint diagram characterizes a set of instance diagrams, including singletons!

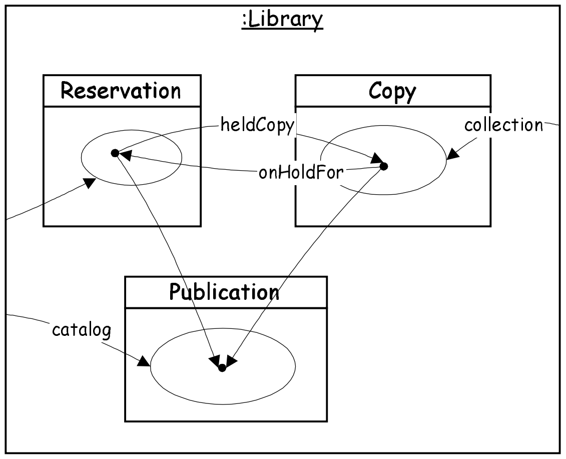


Figure 20: Constraint diagram: output from tool

References

ADL (1997) *Assertion Definition Language*, The Open Group (formerly X/Open), http://adl.xopen.org.

Borgida A., Mylopoulos J. and Reiter E. (1995) "On the Frame Problem in Procedure Specifications", *IEEE Transactions in Software Engineering*, Vol. 21, No. 10.

Bourdeau H. and Cheng B. (1995) *A Formal Semantics for Object Model Diagrams*, in IEEE Transactions on Software Engineering 21, 799-821.

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

D'Souza D. and Wills A. (1995) *Catalysis: Practical Rigor and Refinement*, technical report available at http://www.iconcomp.com.

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

Gerstein L. J. (1987) *Discrete Mathematics and Algebraic Structures*, Freeman New-York.

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.

Harel D. (1987) "Statecharts: a visual formalism for complex systems", *Science of Computer Programming* 8:231-274.

Kent S., Hamie A., Howse J., Civello F. and Mitchell R. (1997) "Semantics Through Pictures: towards a diagrammatic semantics for object-oriented modelling notations", to appear in *Procs. of ECOOP'97 Workshop on Precise Semantics for Object-Oriented Modeling Techniques*, Jyväskylä, Finland, June 10, 1997.

Kent S. (1997) *Constraint Diagrams: Visualizing Assertions in Object-Oriented Models*, Technical Report ITCM97/C2, University of Brighton.

Parnas D. (1996) "Mathematical methods: What we need and don't need", in *An Invitation to Formal Methods*, IEEE Computer.

Rumbaugh J., Blaha M., Premerali W., Eddy F. and Lorensen W. (1991) *Object-Oriented Modelling and Design*, Prentice Hall.

Thomas P., Robinson H. and Emms J. (1988) *Abstract Data Types: Their specification, representation, and use*, Oxford Applied Mathematics and Computing Series, Oxford University Press.

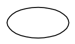
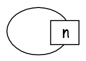
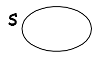
UML (1997) *Unified Modeling Language v1.0*, Rational Software Corporation, available at http://www.rational.com.

Walden K. and Nerson J-M. (1995) *Seamless Object-Oriented Architecture*, Prentice Hall, Object-Oriented Series.

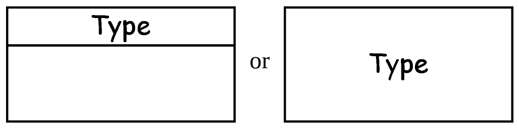
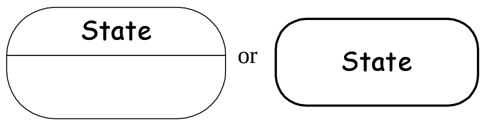
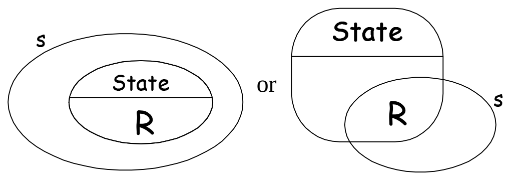
Wirfs-Brock R., Wilkerson B. and Wiener L. (1990) *Designing Object-Oriented Software*, Prentice Hall.

Appendix – Summary of Notation

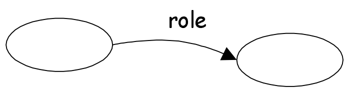
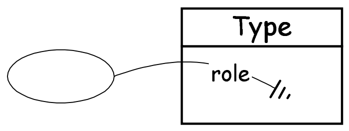
Normal Sets

	A set with one element.
	A set with 0..1 elements.
	A set with 0, 1 or more elements.
	A set with n elements; n may be any numerical expression.
	Optionally sets can be explicitly labeled. This can be useful for referring to them in accompanying explanations, or when mapping a constraint diagram to a math expression.
Venn Diagrams	Standard Venn diagram notation may be used to show relationships between sets.

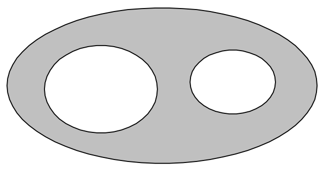
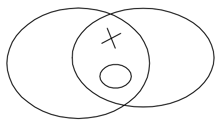
Types and States

	The set of objects of the named type Type.
	The set of objects in state State.
	The region labeled R is the set of objects in state State, which are also in the set labeled s.

Navigation

	The set at the target of the arrow is the union of the sets reached by navigating the association labeled role from each element in the set at the source.
	The value of the association to Type and labeled role, when navigated from the source set, is the empty set or null.

Areas with no elements

	Grey fill indicates that there are no elements in that area of the set. In this case this means that the two subsets partition the containing set.
	In those cases where grey fill is difficult to achieve (e.g., by hand or with some drawing tools), a simple cross in the area may be used as an alternative. In this case, the cross means that the set contained in the intersection *is* the intersection – particularly useful for sourcing/targeting arrows from/to an intersection.

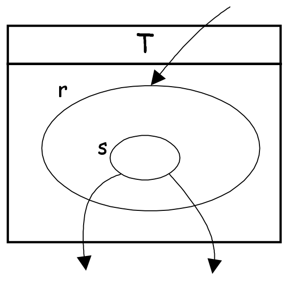
Typical (quantified) sets

A normal set with no arrows targeted on it is assumed to be a typical subset of the set in which it is contained. When converted to a math expression this translates to universal quantification: “for any subset of the containing set, ...”.

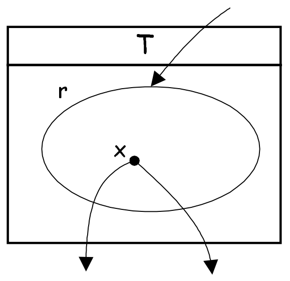
This notation is often used with singleton sets, with effect “for any object in the containing set, ...”.

Existential quantification is achieved by introducing a temporary, unlabeled association.

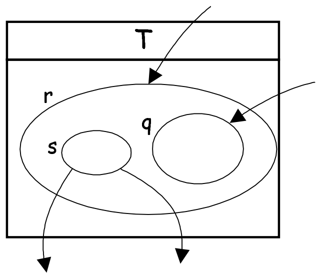
As usual specific labels for sets are optional.



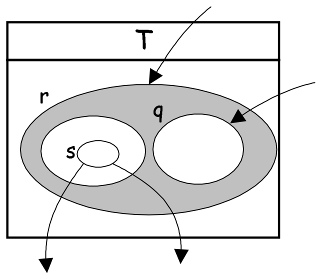
$$\forall s : Set(T), s \subseteq r \Rightarrow \ldots$$



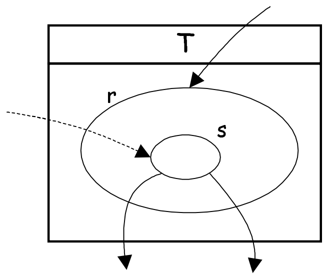
$$\forall x : T, x \in r \Rightarrow \ldots$$



$$\forall s : Set(T), s \subseteq r \Rightarrow \ldots$$



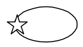
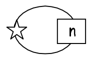
$$\forall s : Set(T), s \subseteq (r - q) \Rightarrow \ldots$$



$$\exists s : Set(T), s \subseteq r \wedge \ldots$$

New objects (filmstrips only)

The following symbols only appear in the second or subsequent frame of a filmstrip. Each represents a set of objects that did not exist in the previous frame. The symbol differs, depending on the cardinality of the set.

★	A set containing exactly one new object (i.e., a new object).
☆	A set containing 0..1 new objects.
	A set containing 0, 1 or more new objects.
	A set containing n new objects; n may be any numerical expression.