Constraint Diagrams: Visualising Assertions in Object-Oriented Models

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

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

Subject areas: Analysis and design methods, language design, formal methods, software engineering practices.

Kind of paper: Research.

1 Introduction

There is a strand of object-oriented (OO) modelling, in particular Syntropy (Cook and Daniels, 1994) and Catalysis (D'Souza and Wills, 1995, 1997), where precision is held to be one of the central tenets of building object-oriented models. In this context, being precise means:

- being precise about the meaning of the visual notations (type models, statecharts etc.) employed in the model descriptions, in terms of a common model;
- supplementing these notations with precise mathematical expression of constraints (e.g. pre/post-conditions and invariants) that it is not easy or possible to express using the visual notation.

The latter is advocated as a way of achieving a level of detail necessary for a comprehensive behavioural 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. That this is so is evident from the limited success of formal methods in practical software development. Amongst other things, Parnas (1996) attributes this to the demanding mathematical skills current formal methods seem to expect of the software engineer, and to the lack of intuitive notation to make this maths more palatable:

> "Mathematical methods offered to the working software engineer are not very practical [...]. Most, but not all, are theoretically sound but very 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."

From our experience of teaching (potential) software engineers OO modelling techniques, Parnas seems to be quite accurate in his observations. Engineers have little difficulty in using, for example, instance diagrams to understand and explain what is happening, and to identify various cases of behaviour to be considered; but formalising these into mathematical notation is often hard for them to do. However, the process of formalisation can be extremely valuable in that it helps to uncover gaps and misunderstandings, as well as providing a general characterisation of behaviour that is simply not possible to achieve through instance diagrams alone.

What is required is a notation which is as intuitive as instance diagrams and as expressive and precise as mathematical assertions. This paper proposes a candidate notation.

In essence, all the OO modelling notations may be viewed as imposing constraints either on the set of allowable system states, examples of which can be illustrated using instance diagrams or ***snapshots***, or on the allowable execution paths through those states, examples of which can be illustrated through ***filmstrips*** - sequences of snapshots (one per frame) annotated with the actions performed between each frame. 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. We propose a visual notation, ***constraint diagrams***, which, in many cases, replaces the need to write assertions mathematically, and we argue, provides a far more intuitive picture of the constraints being imposed. The notation has similarities with informal diagrams used by mathematicians for illustrating relations and borrows much from Venn diagrams. It may be viewed as a generalisation of snapshot notation.

The paper is structured around the construction of an object-oriented (specification) model of a library system. The specification is presented through a number of views each one focussing on a different aspect of the model. Each view comprises a type diagram, mathematical assertions describing addional constraints (invariants, pre/post specifications of actions), and visualisations of those assertions as one or more constraint diagrams. Sometimes the maths is omitted, if previous examples already illustrate the relationship between the constraint diagrams and the maths.

§2, p.3 is a short problem description for the library system. This is introduced first as the rest of the paper uses this example for illustration. §3, p.3 gives the type diagrams for the library, illustrating the semantics in terms of snapshots, writing invariants both mathematically and using constraint diagrams. In this way, the main components of the notation are introduced. §3.3, p.10 also shows how constraint diagrams can be used to define constraints on states, as an alternative to state types on type diagrams and some aspects of statecharts. §4, p.12 gives some action specifications, illustrates their semantics using filmstrips, writing the pre and post-conditions mathematically and then visually. The interesting extensions here are how changes in state can be expressed in the notation, that is how the old state may be depicted visually in the constraint diagrams representing the pre and post-conditions; and also how the creation of new objects may be depicted. §5, p.23 and §6, p.23 are a summary and an indication of further work, respectively.

Apart from constraint diagrams, the notation used throughout is essentially Catalysis, which extends OMT/UML notation with mathematical expression of constraints. The OMT (Rumbaugh et al., 1991) style of notation e.g. for type diagrams, is used in this paper, though this could easily be replaced with UML (UML, 1997) or similar notations; it makes no difference to the essential concepts.

2 Problem Description for a Library System

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 (books, CD's etc.) 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.

3 Invariants

Invariants are constraints which restrict the set of allowable snapshots that the system being modelled can enter. In OO modelling invariants accompany type diagrams. A type diagram defines the kinds or ***types*** of object that may appear in the system being modelled and the links or ***associations*** that may exist between those objects. It also includes notation for constraining the multiplicity of associations: how many objects at the target of the association may be connected to the object at the source.

When modelling a system, it is often a good idea to draw different type diagrams focussing on different aspects of functionality. This is the approach taken here. There are three main aspects to the library system as described in §2, p.3.

Users, publications and copies. These are the "real world" objects the system needs to keep track of.

Loans. The part of the system for tracking loans.

Reservations. The part of the system for tracking reservations.

Each is dealt with in turn below, following the format: type diagram, illustration with snapshots, and invariants, expressed both mathematically and using constraint diagrams.

Where it is clear what is happening, the snapshots and mathematical expression of invariants may be omitted.

3.1 Basic Notation: Users, publications and copies

This section introduces the basic notation, at the same time introducing the essential "real world" objects appearing in a model of a library system.

A library needs to know about users, publications and copies of publications. Thus Figure 1 on page 4 includes types for all these objects and shows the possible associations between them and library system objects and between each other.

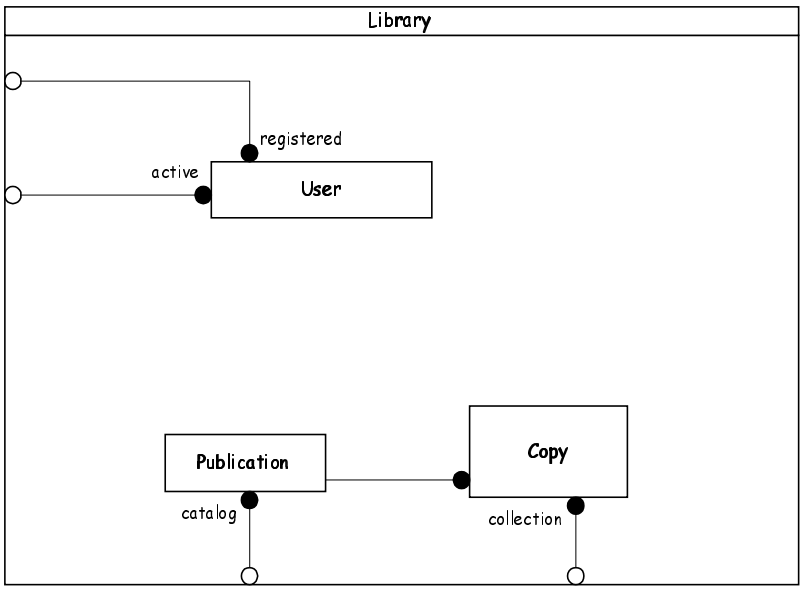


Figure 1: Basic view of library

The types User, Publication and Copy have been enclosed inside the type Library, for two methodological reasons, and one semantic:

- To make it clear that these types are being defined in the context of a library, so no thought has been put into how they might be used in a different context.
- Because it is much easier to draw these diagrams if the "system" type encloses the others (nearly all other types will have a link with the system type).
- We suppose that all the objects that a type within the enclosure refers to are those that become associated with the system object, either through internal creation, or by being passed as a parameter. A consequence of this is that the associations to Library are all optional, or single. One can navigate from a copy either through the association publication.~catalog, or through the association ~collection. Whatever route is taken, the same library object must be reached (hence the associations can not be multiple). Note that e.g. a copy may be passed into the system as a parameter, in which case it need not be connected to the library object by a permanent association (hence the associations can be optional).

This semantics is similar to that suggested for Catalysis; the main difference is that they do not seem to account for the possibility that the "system" object may only know about other objects through temporary links, such as when they are passed through as parameters to system actions.

Table 1 on page 5 gives some snapshots indicating whether or not they are consistent with the type diagram and/or consistent with the "real world" situation we are trying to model.

Snapshots 1 and 2 represent undesirable situations, which are rejected by the type diagram as required. 1 shows a copy attached to a publication from a different library: this is specifically denied by enclosing the type Copy within the type Library (the optional multiplicities on the catalog and collection associations are not enough by themselves). 2 shows a case where a copy is associated with two publications, which is denied by the single multiplicity on the association from Copy to Publication.

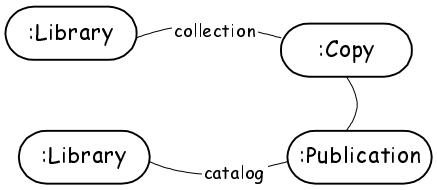
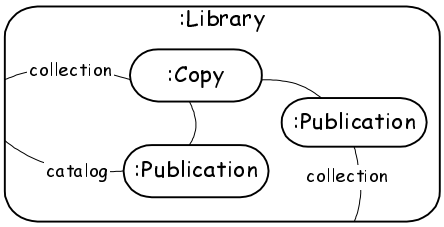
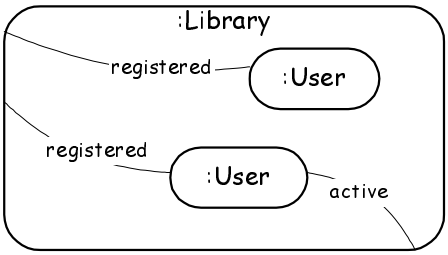
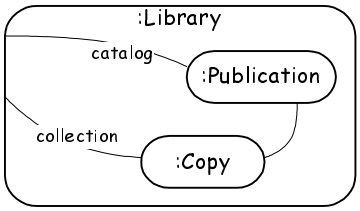
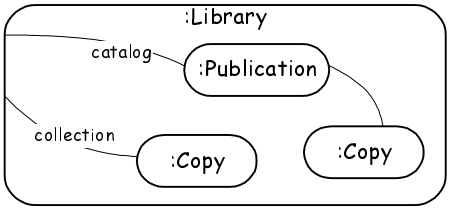
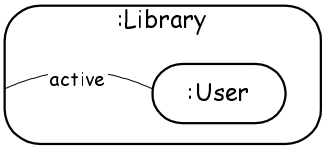
snapshot	consistent with type model	consistent with "real world"
1. 	✘	✘
2. 	✘	✘
3. 	✓	✓
4. 	✓	✓
5. 	✓	✘
6. 	✓	✘

Table 1: Snapshots of users, publications and copies

Snapshots 3 and 4 represent desirable situations which are accepted by the type diagram. 5 and 6, on the other hand, show *undesirable* situations accepted by the type diagram; additional constraints - ***invariants*** - are required to ensure their rejection. The invariant corresponding to snapshot 5 is

$$catalog.copies = collection$$

Invariants apply to **self**. Thus this says that the set of objects obtained by first traversing the association **catalog** from **self**, followed by **copies**,[1] is equal to that obtained by traversing the association **collection**.[2]

Similarly, the invariant corresponding to snapshot 6 is:

$$active \subseteq registered$$

The active users must also be registered. A constraint diagram visualising these invariants is given in Figure 2 on page 6. Notes of explanation are included on the diagram. In particular note:

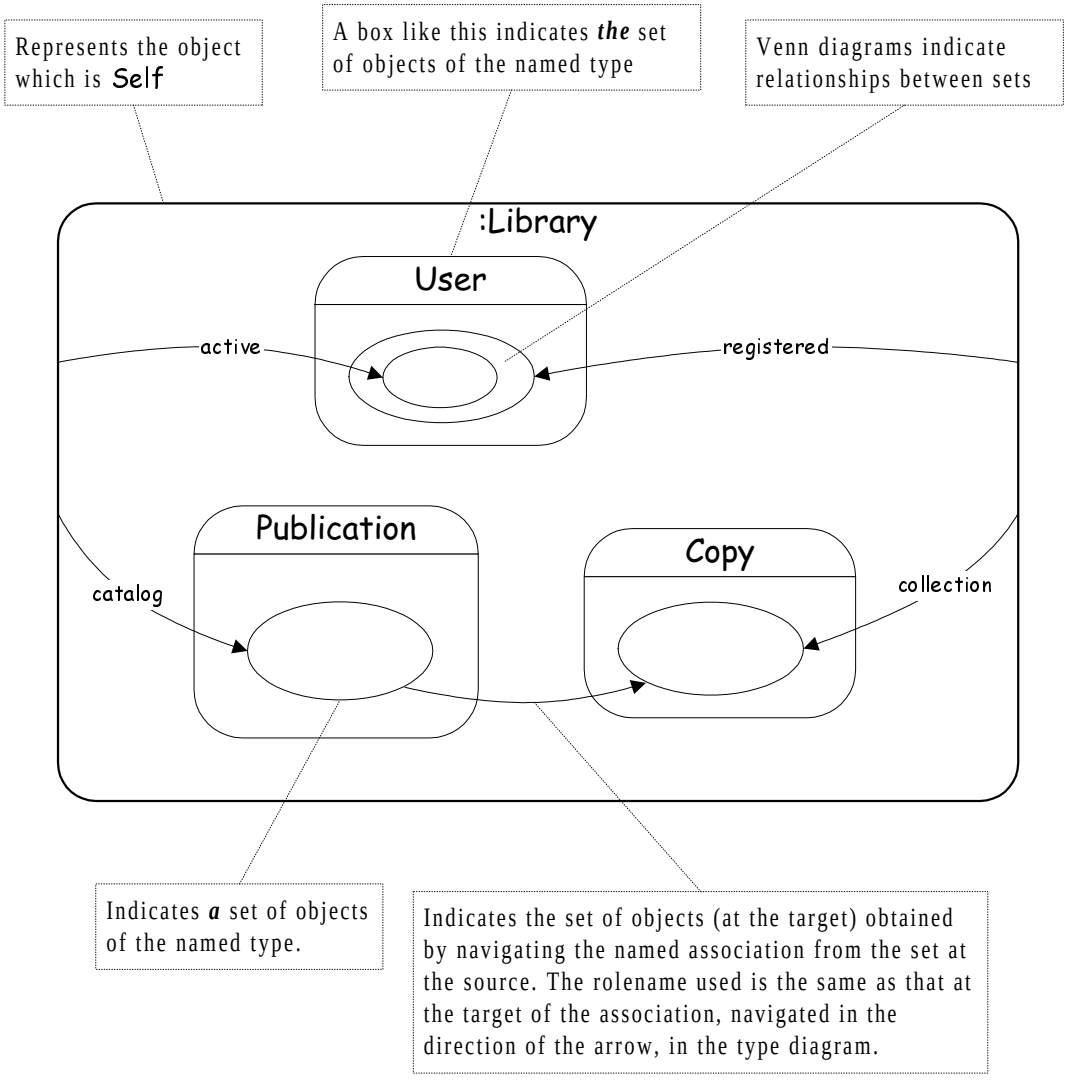


Figure 2: Constraint diagram for basic view

1. In line with common practise, if the rolename of an association is omitted then the name of the type to which the association is directed is used as the rolename, in its plural form if the association is multiple.
2. This is the interpretation of navigation expressions used by Catalysis, and as will be seen this is a (the) key concept underpinning the visual notation. In other attempts to integrate formal assertions with OO modelling notation (e.g. Syntropy) this expression would have to be rewritten $\{x{:}Copy \mid \exists y{:}Publication, y \in catalog \wedge y.copy = x\} \supseteq collection$.

- associations are depicted as relations between sets of objects (after all that's what they are!)
- the use of Venn diagrams to express relationships between associations
- the depiction of types as sets
- navigation always begins at the self object

Links are directed for the following reason. Consider Figure 3 on page 7. The association availa-

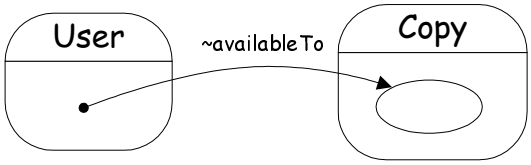


Figure 3: The availableTo link (i)

bleTo 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 the case. Instead we could draw Figure 4 on page 7, which says that for any user there is a set of copies available to that user (possibly empty),

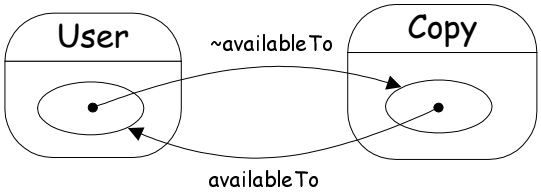


Figure 4: The availableTo link (ii)

and for any copy in that set, that user is *one of* the users for which the copy is available.

3.2 A More Substantial Example: Loans

The loans view illustrates a more substantial example of a constraint diagram, in particular how it tends to lead to more comprehensive coverage of invariants than an alternative method of generating snapshots and deriving invariants from them.

Figure 5 on page 8 is the part of the type diagram focussing on loans.

As with the basic view of the library system, we generate snapshots to establish what additional invariants need to be expressed, looking, in particular, for snapshots which are consistent with the type model but not with the real world. Two such snapshots are given in Table 2 on page 8. Snapshot 1 represents an undesirable situation, as the current loan associated with the copy is actually part of the history of loans. The intention is that the sets of current and historical loans are disjoint. Snapshot 2 is undesirable on two counts: the current loan depicted is associated with a user which is not active, and it is also not a current loan of any copy.

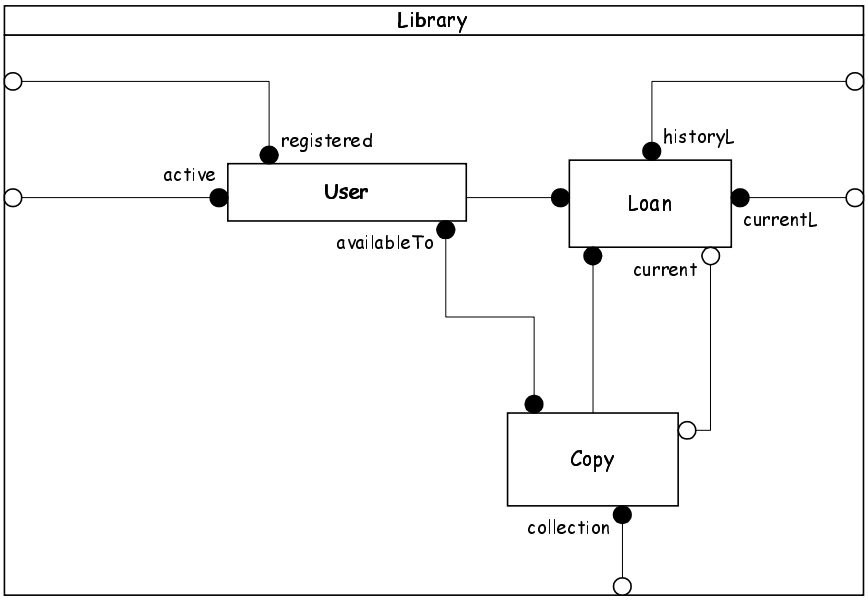


Figure 5: Loans

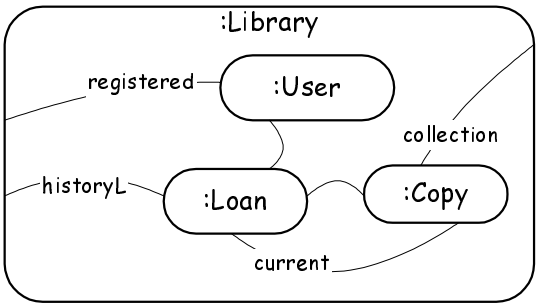
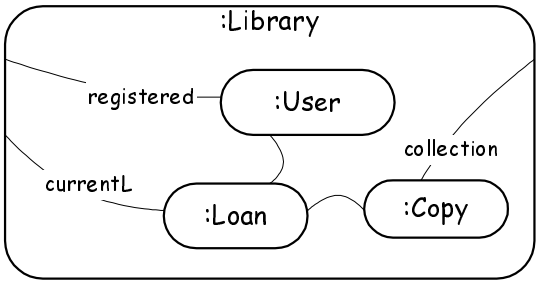
snapshot	consistent with type model	consistent with "real world"
1. 	✓	✘
2. 	✓	✘

Table 2: Snapshots for loans

Reading directly off from these snapshots we might come up with the invariants below.

1. $\forall l$:Loan, $l \in$ historyL $\Rightarrow l.\sim$current $=$ ni
2. $\forall l$:Loan, $l \in$ currentL $\Rightarrow$
   $(l.\text{user} \in \text{active} \wedge l.\sim\text{current} \neq \text{nil})$

This is only a small selection of the snapshots, with corresponding invariants, that would need to be generated, so that, even for this small system, it begins to get difficult to ascertain when all cases have been considered.

On the other hand, drawing a constraint diagram tends to provide the kind of overarching view that leads to comprehensive coverage more quickly. One considers each association in turn, drawing the appropriate set at the target of the association on the diagram. As a new association is included the notation forces you to consider its relationship with those already there.

The constraint diagram for the loans view is given in Figure 6 on page 9. An explanation of new

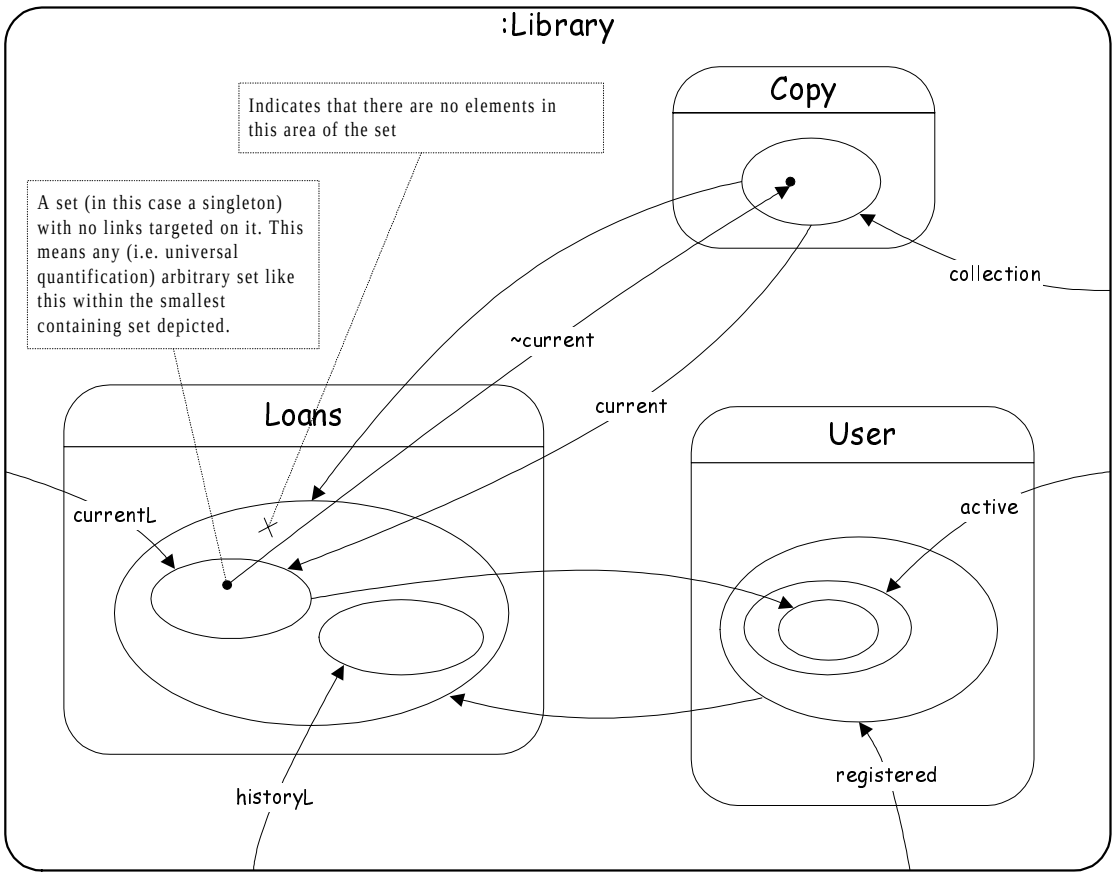


Figure 6: Constraint diagram for loans

notation is annotated on the diagram.

The invariants derived from this diagram are listed in Table 3 on page 10 are obtained. Comments

currentL ∩ historyL = ∅	
currentL.users ⊆ active	
active ⊆ registered	
registered.loans = currentL ∪ historyL	currentL and historyL partition the set at the target of registered.loans, as they do not intersect and the area outside of them is indicated as having no elements.
collection.loans = currentL ∪ historyL	As above.
collection.current = currentL	
∀l:Loan, l ∈ currentL ⇒ l.~current ≠ nil	Notice the use of universal quantification. In addition one could say that l.current ∈ collection , but this is derivable from invariant above

Table 3: Invariants for loans (from constraint diagram)

are written next to some invariants to highlight the most interesting cases.

3.3 Use with Statecharts: The Reservations View

The reservations view illustrates how constraint diagrams connect with and can be used in conjunction with statecharts as an alternative to using state types on type diagrams, as suggested in e.g. Syntropy, Catalysis and UML. The part of the type model concerned with reservations is given in Figure 7 on page 10. The invariants on reservations have a lot to do with the state of copies, in partic-

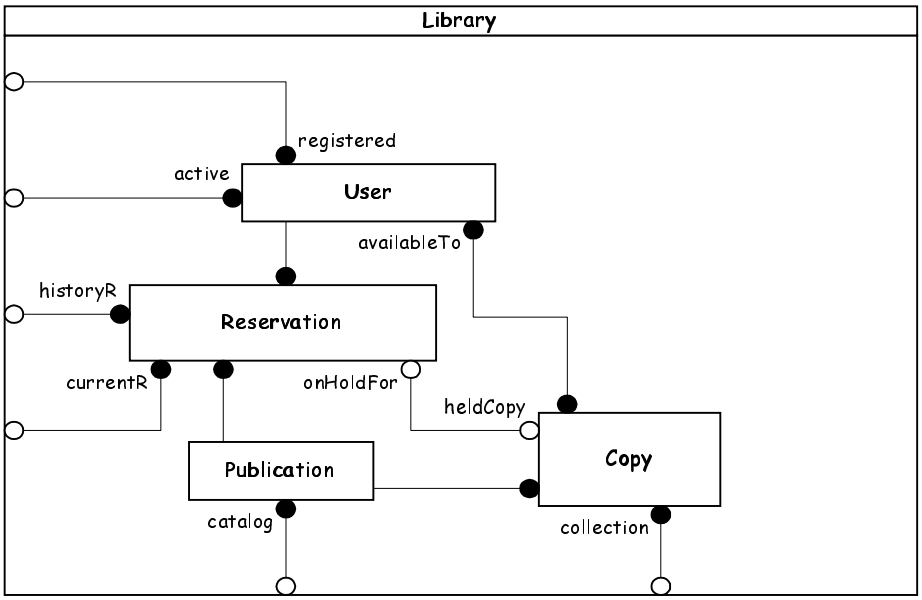


Figure 7: Reservations

ular whether they are on hold or not. It is therefore worth first exploring the states of Copy, and draw-

ing up some invariants to relate them to the associations on the type model. Similarly it is worth exploring the states of Reservation. Doing this also illustrates how constraint diagrams relate to statecharts.

A statechart for Copy is given in Figure 8 on page 11. If the transitions are ignored, then the diagram

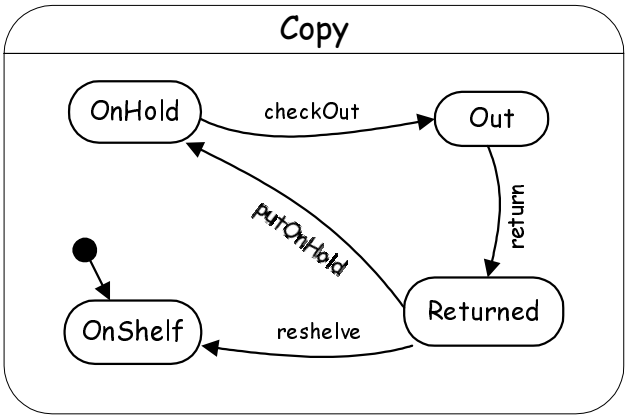


Figure 8: Statechart for Copy

looks just like part of a constraint diagram. This is in fact the case. The states represent sets of objects, namely those objects in that state. They are, of course, disjoint. It is natural, therefore, to draw a constraint diagram relating associations with states. This is done for the states of Copy and Reservation in Figure 9 on page 12. As discussed in §3.1, p.3, the type Copy represents all the Copy objects known in any way to self, for any particular snapshot, whether by permanent or temporary association. Similarly for Reservation and User. The states represent the subsets of those sets of objects which are in a particular state. The diagram informs us that if an object is in any of the states OnShelf, Returned, OnHold or Out, then it is a copy in the library's collection. In other words, a copy object should not be passed in as a parameter in a state OnShelf, for example.[1] The set of objects in the Returned state is exactly that reached through the association returned.[2] Similarly All the waiting and pending reservations are in currentR, and the fulfilled ones in historyR. The associations of objects in particular states with objects of other types are also intuitively depicted on the type diagram. Thus we see that the set of copies on hold map into the set of waiting reservations and vice-versa, and copies/reservations in other states do not have such an association. The use of constraint diagrams in this way provides an alternative and, we think, more intuitive approach than introducing state subtypes on a type diagram, as is done in e.g. Syntropy, Catalysis and UML.

The constraint diagram for reservations can now be given. Figure 10 on page 13 is similar to the diagram for loans, with the addition of the notation to identify the sets of objects from a type that are in a particular state (as introduced in Figure 9). This diagram includes additional information that was not really relevant to the diagram constraining states; for example that navigating via catalog.reservations gets you to the union of historyR and currentR. Also included is the constraint that a copy on hold, must be on hold for a current reservation, and that the only user to whom it is available

1. If this is not desired, then there is notation, introduced in the latter part of §4.4, p.17, which allows the intersection of objects in a particular state with objects from another set (e.g. collection) to be easily represented.
2. Of course this association is redundant. However, having it does make the visual specification of the clearReturns transaction in §4.4, p.17 less cluttered.

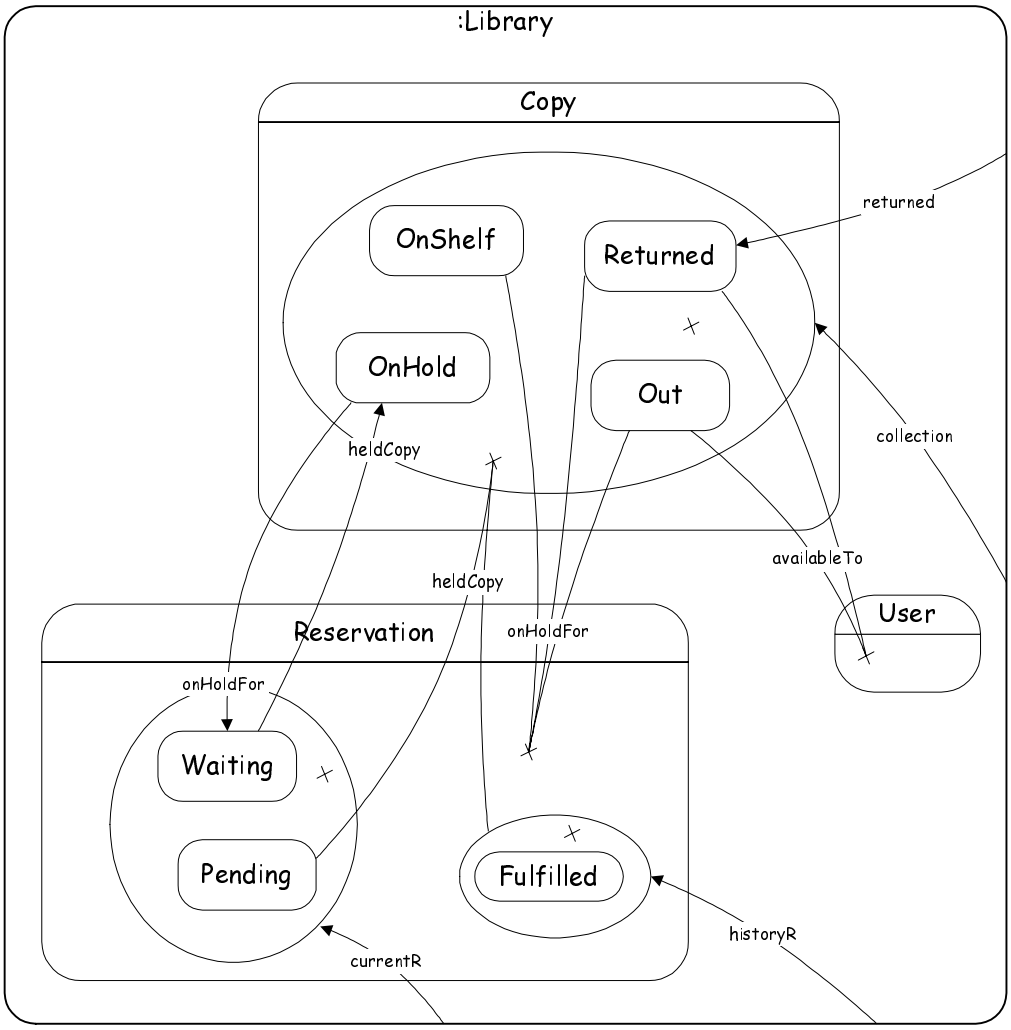


Figure 9: Constraints on states of Copy and Reservation

for loan is the user associated with that reservation, and that copies on the shelf are available to all active users. Textual versions of these constraints are given, as they illustrate how sets of objects in a particular state translate into the mathematical assertion language.

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

4 Action Specifications

Formal assertions are also used to write specifications for the behaviour of actions in terms of pre and post-conditions. This section shows how constraint diagrams can be used in this role.

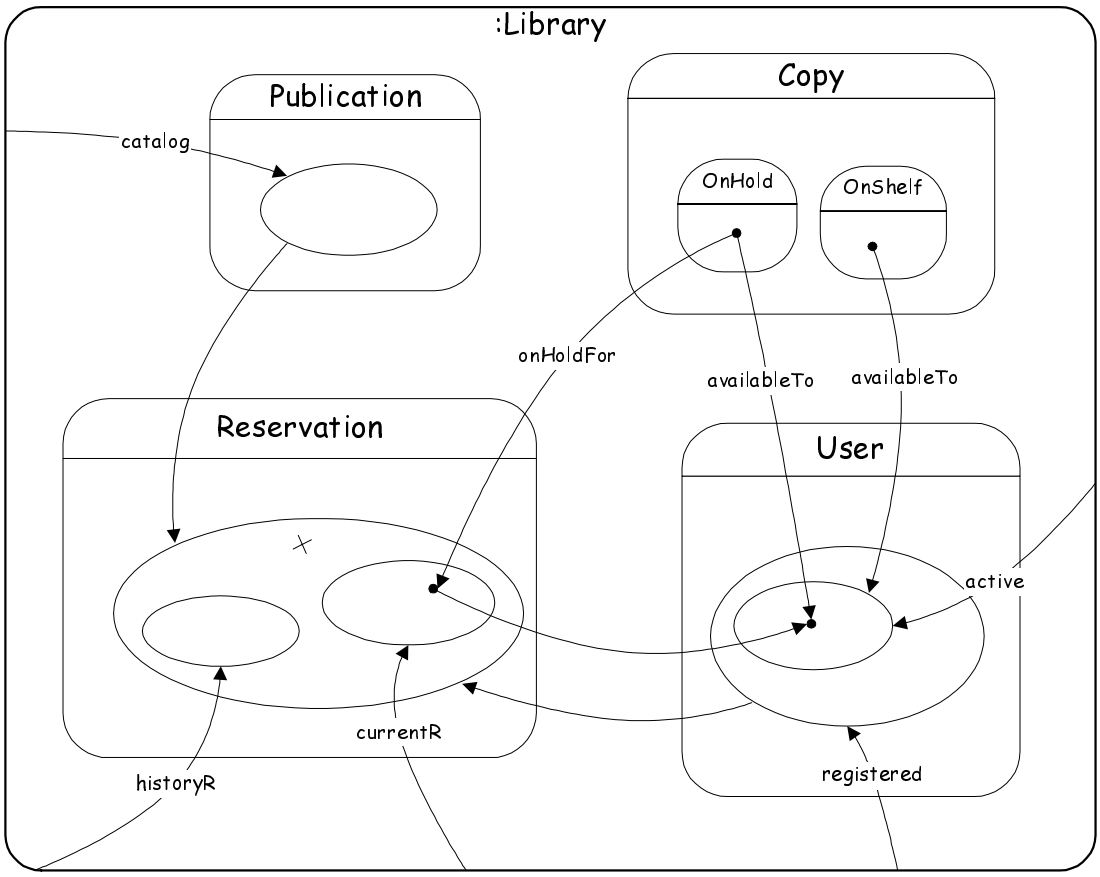


Figure 10: Constraint diagram for reservations

As before, the specifications are considered with respect to the different views of the library system model. Since this is a specification model, all the actions are assumed to take place on the system object (D'Souza and Wills, 1997). The actions considered are registerUser, as it illustrates the point made earlier that some objects may only be known to the system object through temporary associations, and checkout, return, reserve and clearReturns, as they have some of the most interesting behaviour and therefore give a good indication of the expressiveness and intuitiveness of the notation. Of course actions would also be required for adding and removing publications and copies to the library stock, removing users etc.

registerUser is specified in terms of the "basic" view of the library; reserve and clearReturns are specified in terms of the reservations view; and checkout and return are actions specified (largely) in terms of the loans view. We say "largely" as checkout does impact a little on reservations: this will provide an opportunity to show how the specification of an action may be factored into different components, each concerned with a different aspect of the system, which means drawing separate constraint diagrams, one per component, which can then be "composed".

4.1 Filmstrips: registerUser

This section introduces ***filmstrips***, first using snapshots, and then using constraint diagrams. We refer to the latter as ***generalised filmstrips***. The example used - registerUser - also illustrates why the objects known to the system object (in this case a library) can include objects not in permanent association with that object, as originally discussed in §3.1, p.3.

In working out the specification of an action it is usual to produce one or more filmstrips, each with two frames: the first a snapshot satisfying (or not) the pre-condition of the action; the second a snapshot satisfying (or not) the post-condition. A filmstrip for registerUser is given in Figure 11 on page 14, where the snapshots satisfy the pre and post-conditions, respectively. The filmstrip is placed

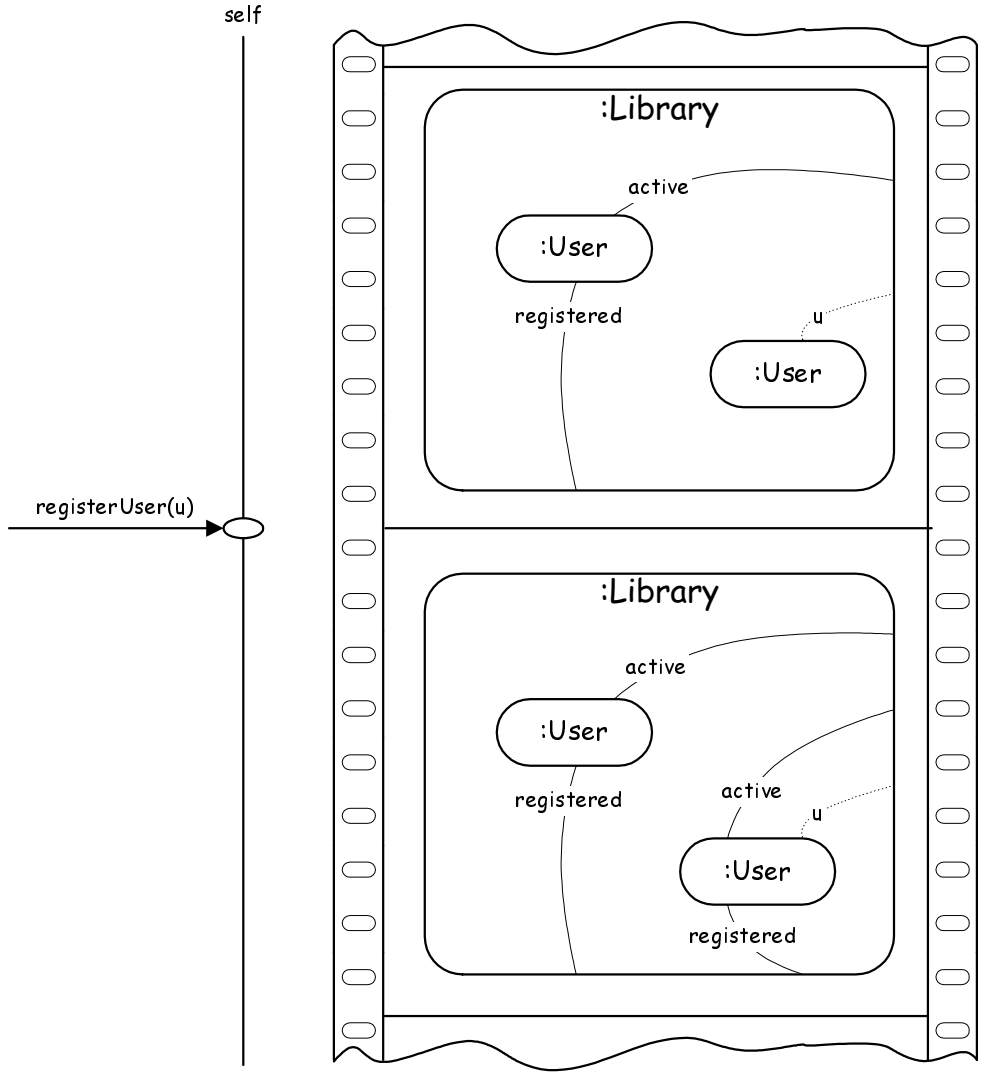


Figure 11: Filmstrip for registerUser

alongside an object interaction diagram (this is Catalysis notation), as a powerful technique of explaining how the state of a system changes as the actions on various objects are performed.

Following common practice, a dotted link indicates a temporary association with an object. Thus there are objects which the system needs to know about and which we need to refer to for specification purposes which are not permanently associated with the system. Here one such object is identified to the system through the temporary association u, and in the pre snapshot this object is not permanently associated with the system.

A formal specification of the action is:

register(u:User)
pre
u is not registered

$u \notin registered$

post

u is now registered and active

$u \in registered \wedge u \in active$

By replacing the snapshots with constraint diagrams, we produce a ***generalised filmstrip*** (Figure 12 on page 15) which expresses exactly what the formal specification expresses[1]. This, and the formal specification, are more general in that they cover all cases, not just the one depicted in Figure 11 on page 14 (for example, the cases when registered is an empty set or has more than a single element). The first constraint diagram both represents the pre-condition and also can be used to indicate

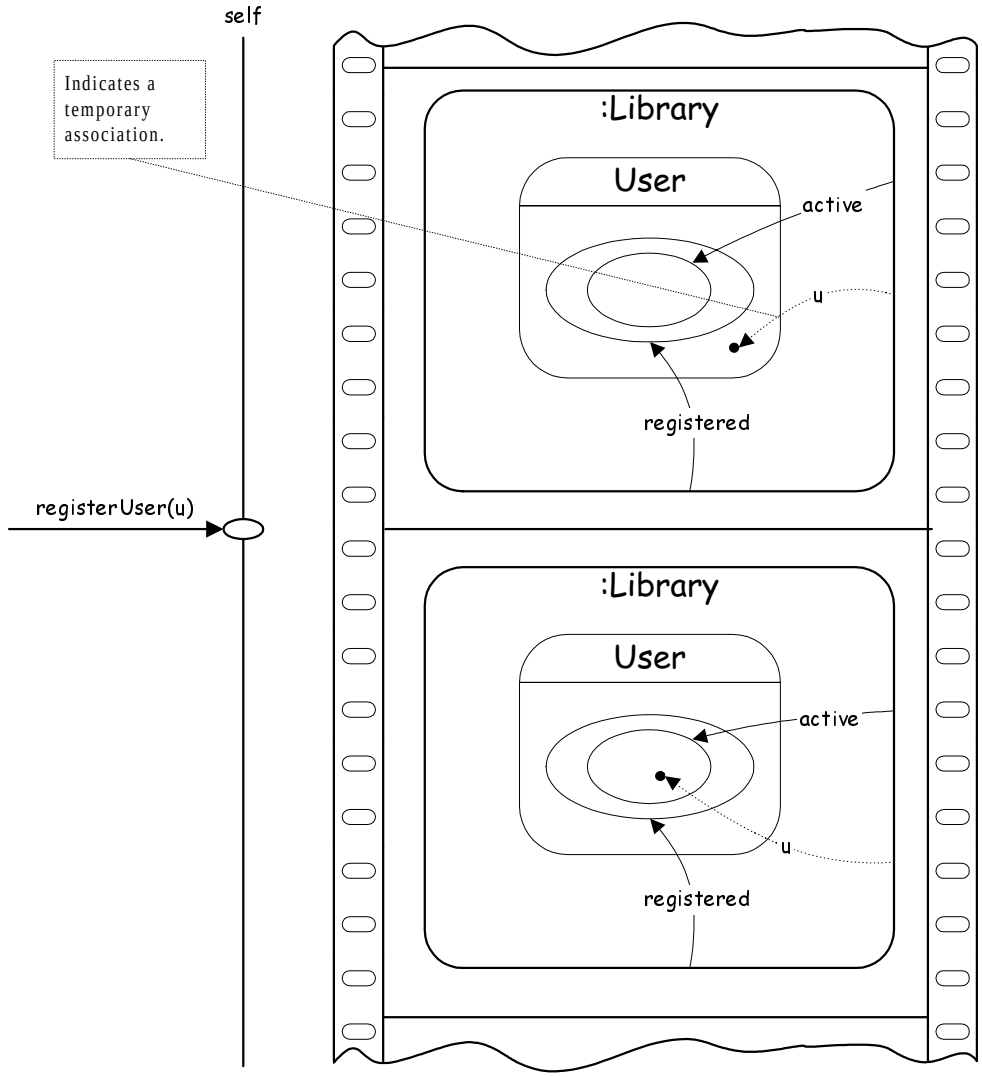


Figure 12: Generalised filmstrip for registerUser

1. It actually says a bit more, specifically that $active \subseteq registered$ in both the pre and post-conditions. In the pre-condition this is guaranteed by the invariant (see §3.1, p.3) so is redundant, and, assuming that nothing else changes except the placing of u in the sets active and registered (see following discussion), is guaranteed to be preserved by the action, so true in the post-condition.

what associations are affected by the action. In this case the associations registered and active are depicted to indicate that they are likely to change when the action is performed in this case by gaining the object identified through u. Currently we make no claims about the representation of frame conditions i.e. stating what doesn't change, as this is a difficult area for formal specification in general (Borgida et al. 1995); for this reason frame conditions have deliberately been omitted from the textual representation of the action post-condition, for example by using set membership instead of set union. Nevertheless, intuitively one can see that anything not indicated as changed on the diagram may be assumed to be unaffected by this action. Working out the detailed semantics needs further work (see §6, p.23).

4.2 Object creation: checkout

The generalised filmstrip for checkout is given in Figure 13 on page 16. The new piece of notation introduced here is to indicate that a new object, previously unknown to the system, has been created. In this example, the object in question is a Loan, which is used in the post-condition to record that the copy (c) is on loan to the user (u). Nothing will be gained by further explanation, that can not be

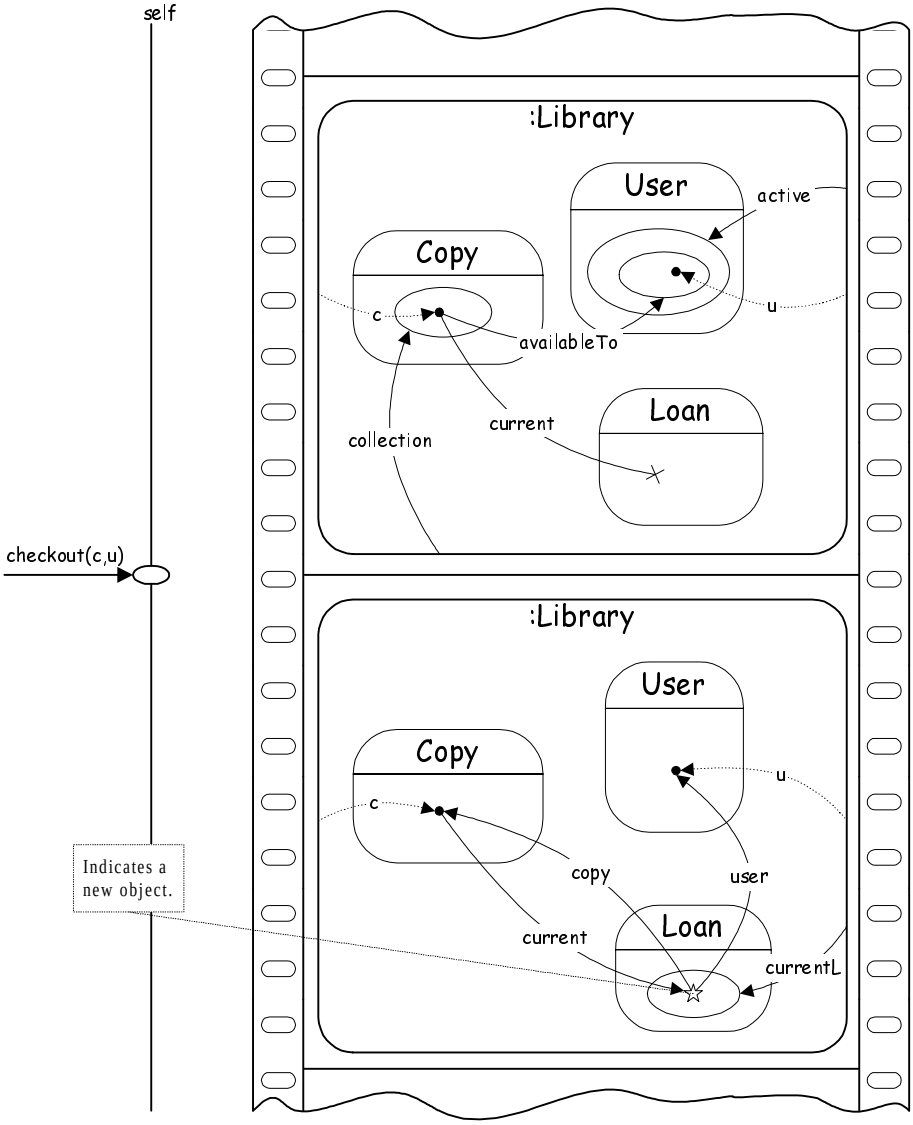


Figure 13: Generalised filmstrip for checkout

discovered by examining the textual specification read off from this diagram, except to note that in Catalysis new is the set of new objects created in moving from the pre state to the post state. The text is given below.

checkOut(u:User, c:Copy)
pre
u is an active user
$u \in active$
c is available for lending to the user u.[1]
$u \in c.availableTo$
post
c is no longer available for lending.
$c.current \neq nil$
The loan of c to u is recorded and marked as current.
$\exists l : loan, l \in new \wedge l \in currentL \wedge l.user = u \wedge l.copy = c \wedge copy.current = l$

4.3 old state: return

return illustrates a specification where direct reference needs to be made in the post-condition to the pre state. Textually, we write old x or $\bar{x}$, where x is an association, to identify the value of x in the pre state. Similar notation can be used in a constraint diagram, as is done in that for return in Figure 14 on page 18. Here, the loan that was current for c, the copy being returned, in the pre state, is placed into the loan history. c.current in the post state becomes unattached. The textual specification is again read off directly from the diagram.

return(c:Copy)
pre
c is on loan.
$c.current \neq nil$
post
c is marked as 'returned', waiting to be reshelved or put on hold.
$c.\sim returned \neq nil$
The loan of c to u is no longer current.
$c.current = nil \wedge c.\overline{current} \notin currentL \wedge c.\overline{current} \in historyL$

4.4 reserve

reserve introduces no new notation, but is included for the sake of completeness and because it helps to understand the next example. Its specification is visualised in Figure 15 on page 19. The post-condition is very similar to that of checkout: a new reservation object is created to record the fact that there is a current reservation for the user u of publication p. The pre-condition is more sophisticated than has so far been encountered: note the use of a $\not\prec$ in the pre-condition to indicate

1. In the case that c is on hold, u must be the user associated with the reservation for which c is on hold.

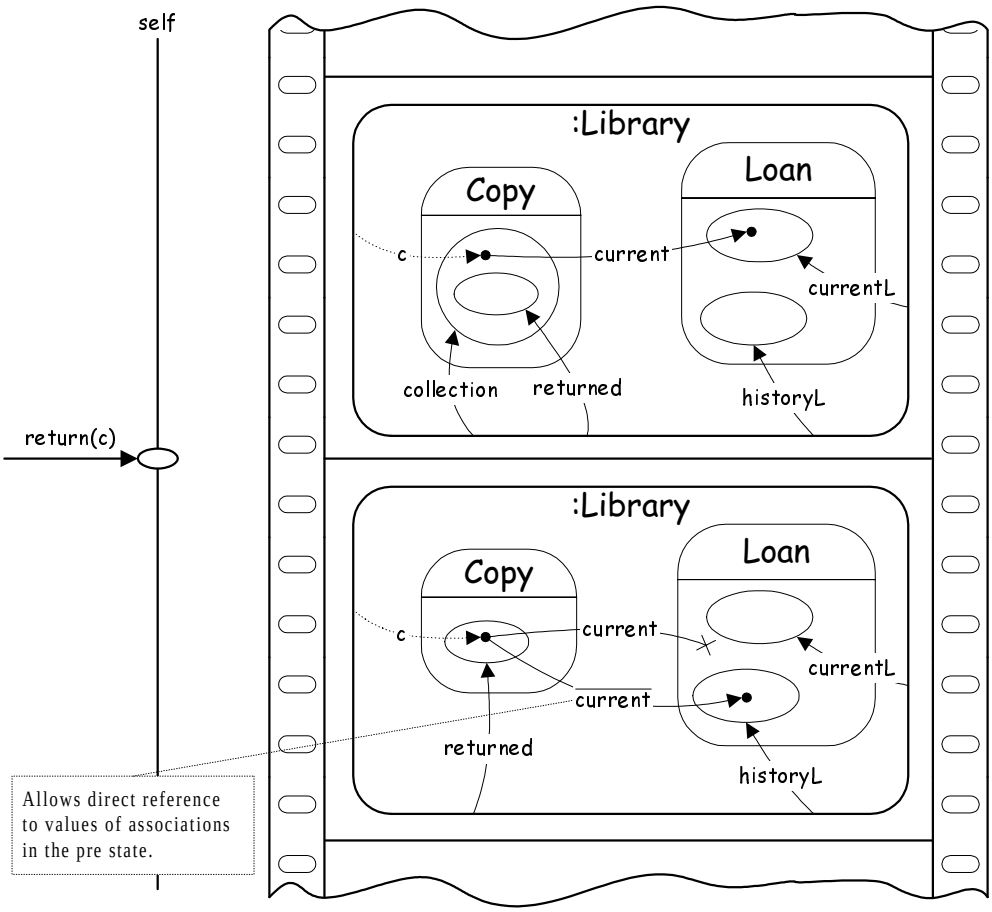


Figure 14: Generalised filmstrip for return

that there is no existing current reservation of p for u, and the use of Venn diagram notation (disjoint sets) to indicate that there are no copies of p available for lending to u. The textual specification derived from the diagram is:

reserve(p:Publication, u:User)

pre

p is not currently reserved by u.

$u.reservations \cap currentR \cap p.reservations = \varnothing$

there is no copy of p available for lending to u

$u.availableTo \cap p.copies = \varnothing$

post

The reservation of p to u is recorded and marked as current.

$\exists r : Reservation,\ r \in new \wedge r \in currentR \wedge$
$r.user = u \wedge r.publication = p$

4.5 States (again) and Set Counting: clearReturns

clearReturns is significantly more difficult to specify than the actions so far described. Its specification illustrates how relationships between set counts may be depicted, and shows a refinement of the notation for referring to objects in a particular state.

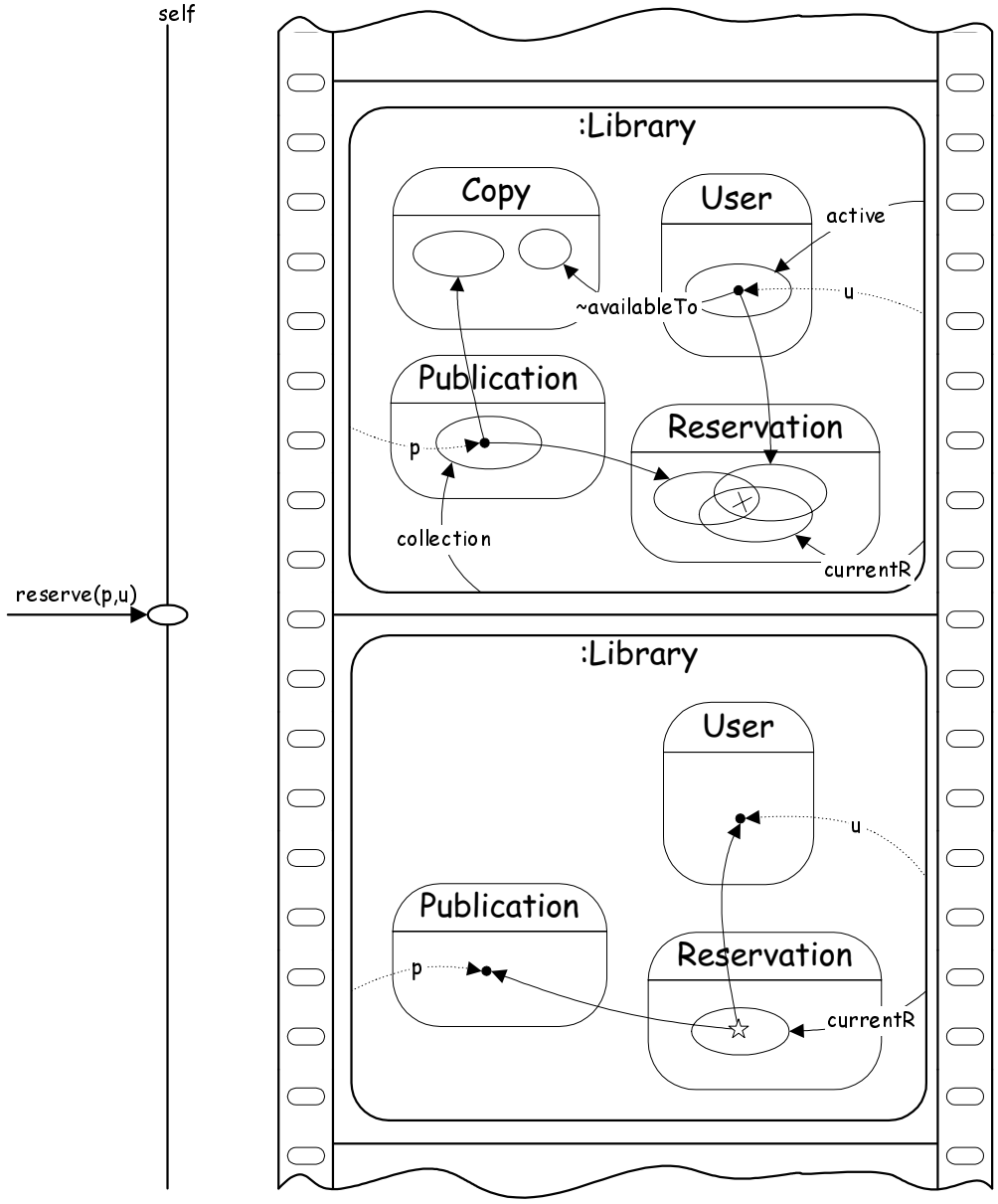


Figure 15: Generalised filmstrip for reserve

When a copy is returned it is marked as returned, which by invariants given in §3.3, p.10, means that it is unavailable for lending - it is waiting to be put on hold or put back on the shelf. The clear-Returns action takes all those copies that have been returned and matches them up with pending reservations, those that still require copies (one each) to be put on hold for them. As this is a specification, we do not wish to fix on any particular algorithm for matching copies to reservations; all we wish to ensure is that the correct number of copies are put on hold for the appropriate reservations.

The specification is given by Figure 16 on page 20. It introduces some new notation which needs explaining:

- Two ways are shown for “counting” sets or placing restrictions on the size of a set: annotating an association targeted on the set, as in the pre-condition, and placing a box on the boundary of the

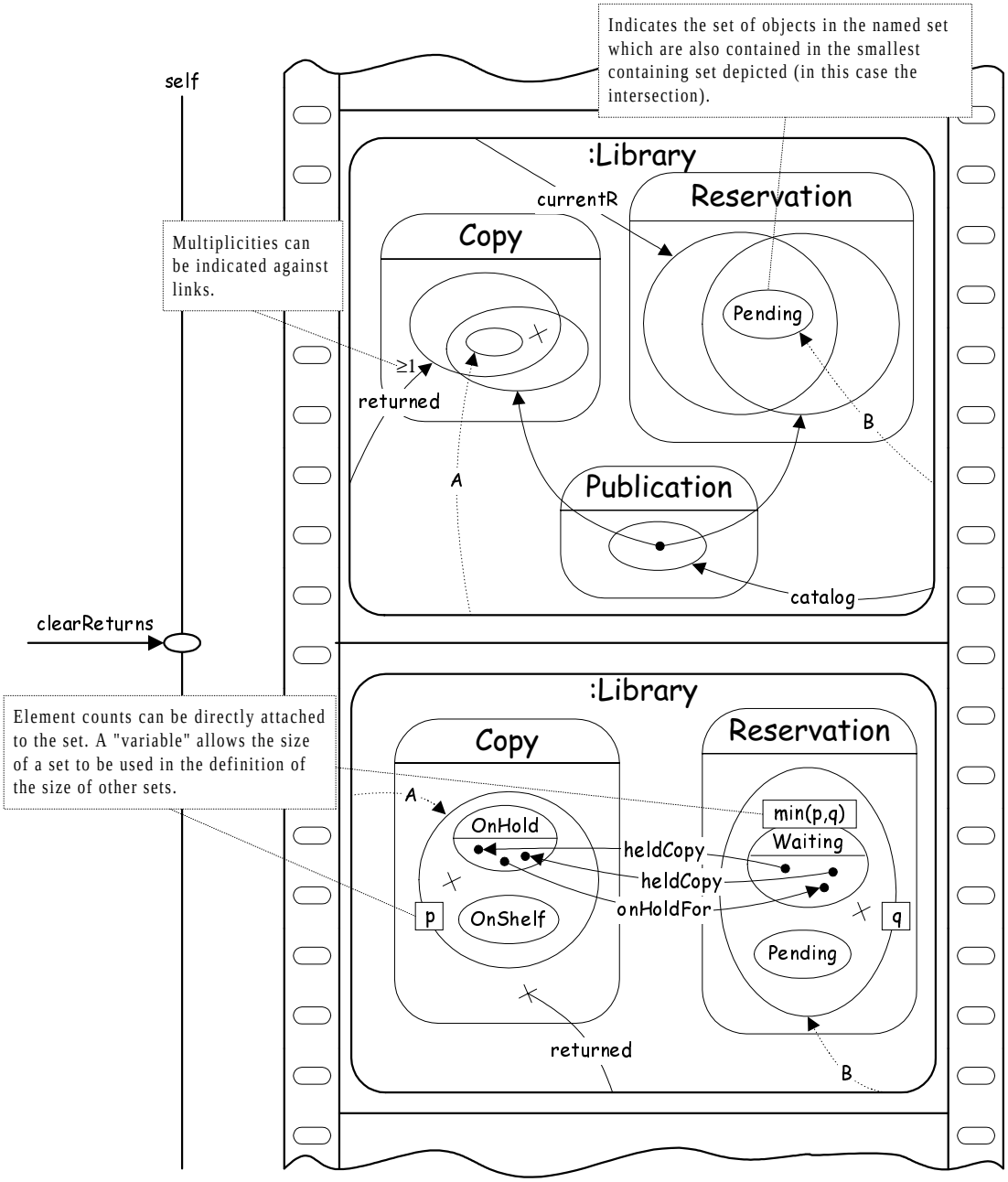


Figure 16: Generalised filmstrip for clearReturns

set, which contains the counting restriction or defines an integer variable whose value is the size of the set, as in the post-condition.

- The use of notation like (Pending) to indicate the set of objects in a particular state which are contained within the smallest containing set, in this case the set of reservations which are pending and are reservations for the publication depicted.
- Arbitrary temporary associations have been introduced (here A and B) for allowing reference to a particular set, derived from other sets. A represents all those copies that have been returned for the particular publication depicted; B represents all the pending reservations for that publication.

The diagram then shows the sets A and B (defined in the pre frame) being partitioned, respectively, between the copies that are put on hold and those that are put on the shelf, and between the reservations which are awaiting collection, and those which remain pending. The copies put on hold are matched to the reservations awaiting collection on a one-one basis. The size indicators on sets ensure that the number of copy/reservation pairs matched in this way is the minimum of

the number of reservations for the depicted publication that are pending

the number of returned copies for that publication

The textual specification derived from the diagram is given below. It is perhaps worth mentioning that this specification was first attempted without the aid of the diagram and found to be very difficult to write; we only arrived at (what we think is) the correct specification by constructing the constraint diagram; previous attempts were either incomplete or inconsistent.

clearReturns

pre

There are copies waiting to be shelved.

$returned \neq nil$

post

For every publication p in the catalog

$\forall p:Publication, p \in catalog \Rightarrow ($

Let A represent all the returned copies for p in the pre state

$Let\ A = \overline{returned \cap p.copies}\ in$

Let B represent all the pending reservations for p in the pre state

$Let\ B = \overline{p.reservations \cap currentR \cap Pending}\ in$

Every copy in A put on hold must be put on hold for a reservation in B that has been marked as waiting

$\forall c:Copy, c \in OnHold \cap A \Rightarrow c.onHoldFor \in Waiting \cap B$

Any two reservations in B that have been marked as waiting must have different copies put on hold for them, and those copies must be in A and must have been put on hold

$\forall r_1, r_2:Reservation, (r_1 \in Waiting \cap B \wedge r_2 \in Waiting \cap B \wedge r_1 \neq r_2)$

$\Rightarrow (\ r_1.heldCopy \neq r_2.heldCopy$

$\wedge\ r_1.heldCopy \in A \wedge\ r_1.heldCopy \in A$

$\wedge\ r_2.heldCopy \neq nil \wedge\ r_2.heldCopy \neq nil)$

Copies in A have either been put on hold or put on the shelf

$(OnHold \cap A) \cup (OnShelf \cap A) = A$

Reservations in B have either been marked as waiting or left pending

$(Pending \cap B) \cup (Waiting \cap B) = B$

The number of reservations in B marked as waiting (which by earlier relationships is the same as the number of copies in A put on hold) is the minimum of the size of A and B, respectively.

$|Waiting \cap B| = min(|A|, |B|)$

There are no returned copies left

$returned = nil$

)

4.6 Diagram Composition: Impact of checkout on Reservations

When a copy which is on hold is checked out, then the reservation it is on hold for must be marked as fulfilled. This aspect of its specification was not dealt with in §4.2, p.16. There are two choices: go back and change the original specification, or, assuming appropriate notational and semantic support, deal with this part of the specification separately and then just “compose” it with the original specification. The latter approach is more appealing as, in general, it lets us split up a specification into more manageable pieces. Catalysis tells us how to compose textual specifications of actions, and this, in turn, has been taken from research in formal methods. Since we can derive the textual specification from the constraint diagrams, this provides a semantic underpinning to composition of generalised filmstrips. Intuitively it is just the overlaying of diagrams, as we would expect.

The constraint diagram showing the impact of checkout on reservations is given in Figure 17 on page 22. Comparing this with Figure 13 on page 16 we see that there are no mismatches: specifically

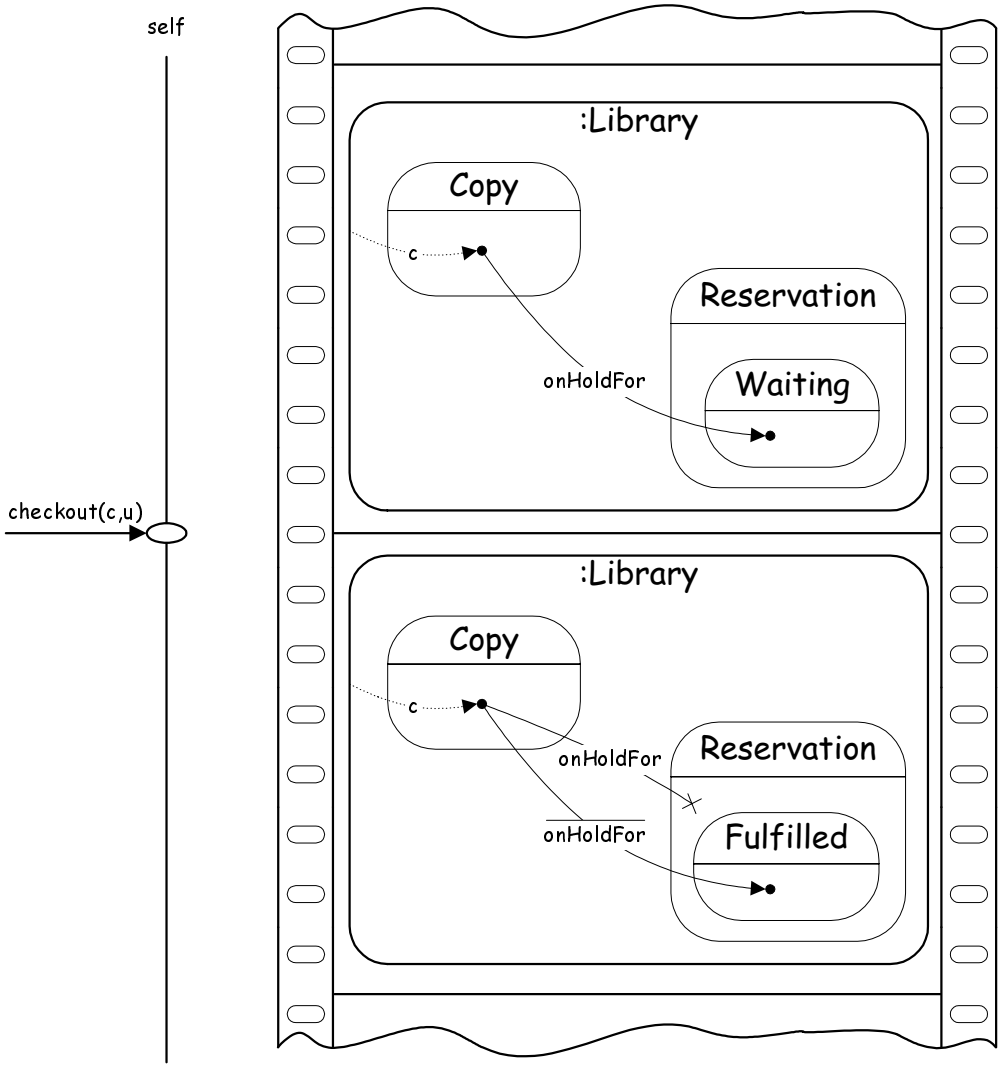


Figure 17: Impact of checkout on reservations

both pre and post-conditions in Figure 17 only consider associations between the copy c and Reservation objects, which are not mentioned in Figure 13.

5 Summary

The paper has introduced a new notation, ***constraint diagrams***, for visualising assertions in object-oriented modelling. The notation has similarities with informal diagrams used by mathematicians for illustrating relations and borrows much from Venn diagrams; it may be viewed as a generalisation of snapshot or instance diagram notation. We have shown, by means of a case study, how the notation may be used to express invariants and pre/post-conditions, and, in addition, how it may be used in conjunction with existing visual notations, such as type models and statecharts. The semantics of the notation has been informally sketched through the use of instance diagrams, informal description, and mappings into formal assertions.

6 Further Work

Constraint diagrams have opened up a number of avenues for further research. A few of them are listed below:

Relationships with Other Notations. The examples in the paper have illustrated to some extent the relationships between constraint diagrams and other visual notations. Further investigation is required to explore these relationships in more generality, in particular (a) whether constraint diagrams could actually be used as the sole notation, hence underpin other notations, and (b) how they could best be used in conjunction with other notations. So far we have observed:

- Multiplicity constraints on associations on type diagrams can be expressed in constraint diagrams by annotating links, annotating sets, or using different notation for different sets with different multiplicity: • = 1, ◦ = 0,1, ⬭ = 0 or more.
- State types on type diagrams, and constraints on them, can be intuitively represented on constraint diagrams. See §3.3, p.10 and §4.5, p.18.
- Types are represented as sets on constraint diagrams. Thus, as with substating, static subtyping and associated constraints (partitions, disjoint types, etc.) can be represented directly in constraint diagrams using Venn diagrams. This accords with explanations of subtyping e.g. in Wirfs-Brock et al. (1990).
- Statecharts could be simplified, with constraint diagrams used to show nesting and orthogonality of states. Then the primary focus of statecharts would be on describing state transition behaviour, providing an alternative way of visualising the specifications of actions e.g. as discussed in D'Souza and Wills (1997). Following this route might make it easier to split the description of transition behaviour for one type over many statecharts.
- Constraint diagrams can be overlayed with snapshots, as illustrated by Figure 18 on page 24. Similarly for filmstrips. Links can then be compared to establish if one is consistent with the other. Figure 18 on page 24 is inconsistent: look at the `current` links. Visually this may be useful in design, especially with appropriate tool support. A "logical" overlaying process could provide the basis for consistency checking algorithms.

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

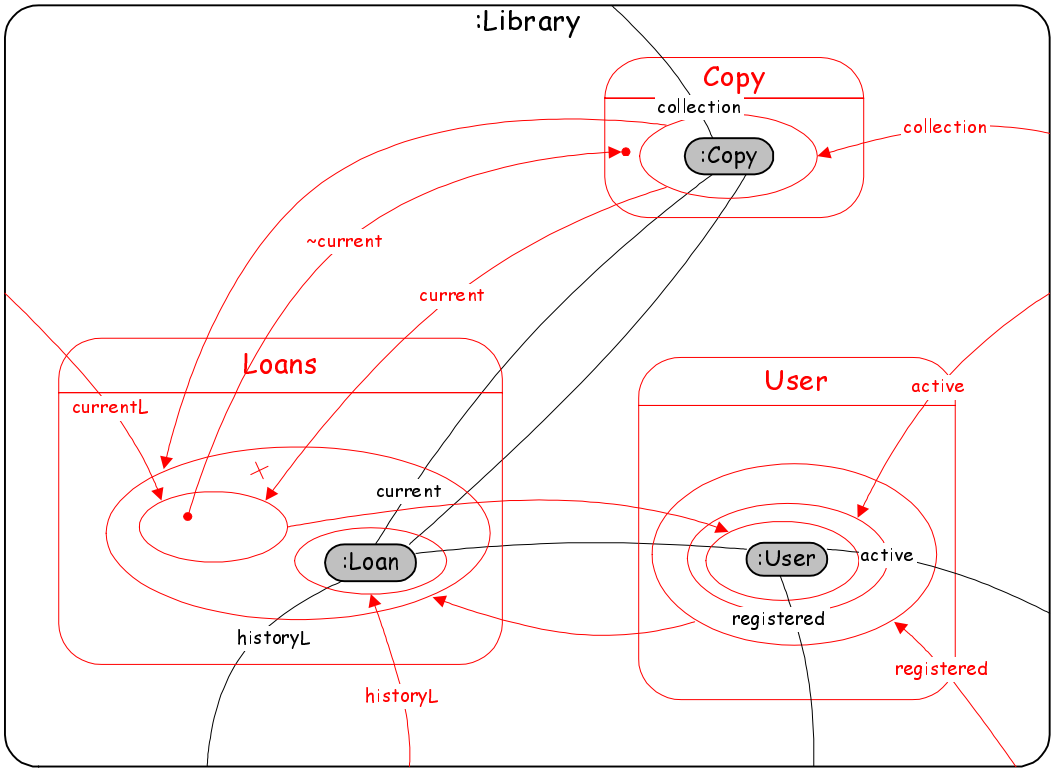


Figure 18: Integration with snapshot diagram

tion and conjunction. This may open up 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 visual notations. With its own formal semantics, we would then be in a position to provide formal, yet intuitive, semantic underpinnings to other OO modelling notations, such as those introduced by UML (UML, 1997).

Concepts. If the notation is able to express most constraints that one requires in OO modelling, then it is interesting to see if it is able to clarify certain concepts in that world. By exploring the various possibilities, a better and more precise characterisation of the intended semantics of new notations could be given. Specifically we have in mind the area of composite objects, where there seem to be many possible semantics (see e.g. Civello, 1993), and how this impacts on notations such as composite classes in UML.

Use of the notation. Further investigation is required into (a) whether the notation would be useful in practice and whether it really is more intuitive and easier to use than mathematical assertions; and (b) what are the most appropriate ways to use it, in particular in conjunction with other notations such as stachecharts and type models. 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).

Tools. We foresee interesting possibilities for providing sophisticated yet intuitive tool support for semantic checking of models. Some of this has already been hinted at e.g. in checking snapshots and filmstrips against constraint diagrams. Other areas to consider are the generation of constrainst diagrams from snapshots and filmstrips; animation of models, visualised through constraint diagrams; and consistency checking between notations by mapping into constraint diagrams and then matching the results.

Acknowledgements

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

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.

Civello F. (1993) "Roles of Composite Objects in Object-Oriented Analysis and Design", in *OOPSLA'93*, pp.376-393, ACM Press.

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

D'Souza D. and Wills A. (1995) *Catalysis: Practical Rigour 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.

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

Hamie A. and Howse J. (1997a) *A Larch-based Semantics for the Type Views of Syntropy*, submitted to FME97.

Hamie A and Howse J (1997b) *A Larch-based Semantics for the Statecharts of Syntropy*, submitted to ECOOP97.

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.

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

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