Case-Study: Modelling a Library System

This case-study involves modelling an information system to support a simple *Lending Library Service*. This will be approached as an exercise in *Requirements Engineering* where the aim is to develop a *model*, capturing high-level requirements for such a system to produce an agreed *formal specification*. We make use of *Formal Methods*, exploiting the power of discrete mathematics to model, and to reason, *abstractly* over a succession of separate *levels* producing a *modular* de-composition of the system.

Membership Class: LM

We begin by defining an invariant for a simple membership class. Members belong to a finite, initially-empty, set named *Member* and each member has some associated *information*:

LM

$$\begin{array}{|l} Member : \mathsf{set}\ \mathbb{M} \\ info : Member \to \mathbb{I} \\ \hline Member' = \emptyset \end{array}$$

The *types* for both a member identifier and its associated information are specified here as two given, *fully-abstract*, sets named $\mathbb{M}$ and $\mathbb{I}$; the definition of their concrete data-representations is left to later *refinements*. The *initial-condition* for class LM, together with its *invariant*, imply that all components are empty when the class is first *instantiated*.

A *query* is an operation that provides information about the state of objects in the system. In this class, we can define a query that shows the information associated with a member:

$$\begin{array}{|l} LM?showInfo(m \to i) \\ \hline i := info(m) \end{array}$$

$$\equiv \begin{array}{|l} LM?showInfo(m \to i) \\ \hline m : Member\,;\ i : \mathbb{I} \\ i = info(m) \end{array}$$

and another that lists all the members, together with their information:

$$\begin{array}{|l} LM?showMembers(\to M) \\ \hline M := info \end{array}$$

$$\equiv \begin{array}{|l} LM?showMembers(\to M) \\ \hline M : Member \to \mathbb{I}\,;\ M = info \end{array}$$

In both cases their *pre-conditions* have been expanded to clarify these specifications by providing type information implicit from the invariant.

An *event* is an operation that changes the state of the system. We obviously require at least one event for this class, an operation to 'enrol' a new member, since the set *Member* is initially empty:

$$
\begin{array}{|l}
\hline
LM!NewMember(i \to m) \\
\hline
i : \mathbb{I};\ m : \mathbb{M};\ m \notin Member \\
\hline
m \in Member';\ info'(m) = i \\
\hline
\end{array}
$$

An operation to 'update' the recorded information for existing members can also be defined:

$$
\begin{array}{|l}
\hline
LM!UpdateInfo(m, i) \\
\hline
m : Member;\ i : \mathbb{I};\ i \neq info(m) \\
\hline
info'(m) = i \\
\hline
\end{array}
$$

Exercise 1
Provide a *simple* specification of an operation to 'remove' a current member.

Catalog Class: LT

The library's catalog can be modelled as an initially-empty set of *titles*, where each title has an associated *description*:

$$
\begin{array}{|l}
\hline
LT \\
\hline
Title : \mathsf{set}\ \mathbb{T} \\
desc : Title \to \mathbb{D} \\
\hline
Title' = \emptyset \\
\hline
\end{array}
$$

Observe that class LT has a *specification* that is very similar to the class LM – only the *names* differ. The operations we need for LT are also similar to those provided for LM.

We can provide a query that shows the description of a particular title:

$$
\begin{array}{|l}
\hline
LT?showDesc(t \to d) \\
\hline
d := desc(t) \\
\hline
\end{array}
$$

and one that lists the full catalog of titles together with their descriptions:

$$
\begin{array}{|l}
\hline
LT?showCatalog(\to T) \\
\hline
T := desc \\
\hline
\end{array}
$$

We require at least one event for LT – an operation to 'enter' a new title into the system, since the set *Title* is initially empty:

$LT!NewTitle(d \to t)$

$$d : \mathbb{D};\ t : \mathbb{T};\ t \notin Title$$

$$t \in Title';\ desc'(t) = d$$

Exercise 2
Suppose that an operation to 'update' the description for a known title along with one to 'remove' such a title were required at the level of class LT. Give a *simple* specification for each event.

Collection Class: LC

We model a library's 'collection' (of hard copies) by *extending* the catalog class. We add a function *nc* to record the current 'number of copies' for every title and define a *partition* over the set of such titles, based on the value of *nc*; titles are *in-collection* if there are copies of the title in the collection, otherwise they are *ex-collection*:

LC

$$LT$$
$$nc : Title \to \mathsf{NAT}$$
$$\{InColl, ExColl\} : \mathsf{part}\ Title$$
$$InColl := \{t : Title \bullet nc(t) > 0\}$$

$\equiv LC$

$$Title : \mathsf{set}\ \mathbb{T}$$
$$desc : Title \to \mathbb{D}$$
$$nc : Title \to \mathsf{NAT}$$
$$InColl := \{t : Title \bullet nc(t) > 0\}$$
$$ExColl := \{t : Title \bullet nc(t) = 0\}$$

$$Title' = \emptyset$$

Other state-components, and the initial-condition, are "inherited" from LT. That (intermediate) class specification is said to be *imported* into class LC: its invariant must always hold there, but this may be consistently *extended*; the operations it provides may also be applied or extended in such contexts.

Required queries at this level might be similar to those provided for a catalog. Indeed, we will directly *promote* the query *showDesc* from class LT:

$LC?showDesc(t \to d)$

$$LT?showDesc(t \to d)$$

and add a query showing the number of copies for a given title:

$LC?showNoCopies(t \to n)$

$$n := nc(t)$$

However, we will redefine the query *showCatalog* to include information only about titles that are in-collection and extend the query to include the number of copies for each title as well as its description:

$LC?showCollection(\rightarrow C)$

$$C : Title \nrightarrow \mathsf{POS} \times \mathbb{D}$$
$$C = \{(t, n, d) : InColl \times \mathsf{POS} \times \mathbb{D} \bullet n = nc(t) \land d = desc(t)\}$$

We extend the *NewTitle* event from LT, so as to 'fix' (as zero) an *initial* number of copies:

$LC!NewTitle(d \rightarrow t)$

$$LT!NewTitle(d \rightarrow t)$$
$$nc'(t) = 0$$

$\equiv LC!NewTitle(d \rightarrow t)$

$$d : \mathbb{D};\ t : \mathbb{T};\ t \notin Title$$
$$t \in Title' \cap ExColl'$$
$$Desc'(t) = d;\ nc'(t) = 0$$

When a new title is added it is ex-collection. We then clearly need some way to 'modify' the number of copies as well

$LC!AddCopies(t, n)$

$$t : Title;\ n : \mathsf{POS}$$
$$nc'(t) = nc(t) + n$$

$LC!RemoveCopy(t)$

$$t : InColl$$
$$nc'(t) = nc(t) - 1$$

The operations specified at this level involve modelling choices. These decisions should satisfy the relevant requirements of our intended *client*.

Exercise 3
Expand the LC events *AddCopies* and *RemoveCopy* to show all *changes-of-state* imposed by the state-invariant.

Exercise 4
Suppose that operations for class LC do not fully satisfy requirements of a client. Explore some possible alternatives, then specify that query or event:

1. there should be a way to show descriptions for titles having *no* copies;
2. it should be possible to fix the initial number of copies for a new title;
3. it should be possible to remove *more than one* copy at the same time;
4. one operation should serve to *increase* or *decrease* the number of copies.

Loan Class: LL

We model a simple loan class by *composing* the classes LM and LC. These two classes are independent so composing them will produce a consistent invariant. We then extend their combined state, to define the set of all current 'loans', which is expressed mathematically as a *relation* ($loan$), and the numbers of 'available' and 'loaned' copies (the functions na and nl, respectively) for each title:

LL

$$
\begin{array}{|l}
\hline
LM \,;\; LC \\
loan : Title \pfun Member \\
na, nl : Title \fun \mathsf{NAT} \\
\forall t : Title \bullet \\
\quad nc(t) = na(t) + nl(t) \land \\
\quad nl(t) = \#\{m : Member \bullet t \mapsto m \in loan\} \\
\hline
\end{array}
$$

$\equiv$ LL

$$
\begin{array}{|l}
\hline
Member : \mathsf{set}\ \mathbb{M} \\
info : Member \fun \mathbb{I} \\
Title : \mathsf{set}\ \mathbb{T} \\
desc : Title \fun \mathbb{D} \\
loan : Title \pfun Member \\
nc, na, nl : Title \fun \mathsf{NAT} \\
InColl := \{t : Title \bullet nc(t) > 0\} \\
ExColl := \{t : Title \bullet nc(t) = 0\} \\
\forall t : Title \bullet \\
\quad nc(t) = na(t) + nl(t) \land \\
\quad nl(t) = \#\{m : Member \bullet t \mapsto m \in loan\} \\
\hline
Member' = \emptyset \,;\; Title' = \emptyset \\
\hline
\end{array}
$$

Class LL embodies a significant assumption: *a member may borrow at most one copy of any particular title at the same time.*

The membership queries are needed at this level so they are directly *promoted*, as are two of the collection queries:

$LL?showInfo(m \fun i)$

$$
\begin{array}{|l}
\hline
LM?showInfo(m \fun i) \\
\hline
\end{array}
$$

$LL?showMembers(\fun M)$

$$
\begin{array}{|l}
\hline
LM?showMembers(\fun M) \\
\hline
\end{array}
$$

$LL?showDesc(t \fun d)$

$$
\begin{array}{|l}
\hline
LC?showDesc(t \fun d) \\
\hline
\end{array}
$$

$LL?showNoCopies(t \fun n)$

$$
\begin{array}{|l}
\hline
LC?showNoCopies(t \fun n) \\
\hline
\end{array}
$$

We add a query to show the number of available copies for a given title:

$LL?availableCopies(t \rightarrow n)$

$$n := na(t)$$

and update the *showCollection* query from the collection class:

$LL?showCollection(\rightarrow C)$

$$C : Title \nrightarrow \mathsf{POS} \times \mathsf{NAT} \times \mathbb{D}$$
$$C = \{(t, n, l, d) : InColl \times \mathsf{POS} \times \mathsf{NAT} \times \mathbb{D} \bullet n = nc(t) \land l = nl(t) \land d = desc(t)\}$$

Finally, we add a query that shows the set of members borrowing copies of a given title:

$LL?showLoans(t \rightarrow M)$

$$M := \{m : Member \bullet t \mapsto m \in loan\}$$

Many of the events at this level are also promotions:

$LL!NewMember(i \rightarrow m)$

$$LM!NewMember(i \rightarrow m)$$

$LL!UpdateInfo(m, i)$

$$LM!UpdateInfo(m, i)$$

$LL!NewTitle(d \rightarrow t)$

$$LC!NewTitle(d \rightarrow t)$$

$\equiv LL!NewTitle(d \rightarrow t)$

$$d : \mathbb{D};\ t : \mathbb{T};\ t \notin Title$$
$$t \in Title' \cap ExColl';\ desc'(t) = d$$
$$nc'(t) = na'(t) = nl'(t) = 0$$

The *NewTitle* event has been expanded to show that the number of available and loaned copies is initially zero. *AddCopies* is also promoted but expanded to show how the number of available copies is increased:

$LL!AddCopies(t, n)$

$$LC!AddCopies(t, n)$$

$\equiv LL!AddCopies(t, n)$

$$t : Title;\ n : \mathsf{POS}$$
$$nc'(t) = nc(t) + n$$
$$na'(t) = na(t) + n$$

RemoveCopy is extended to clarify the pre-condition; the removed copy must be an available copy:

$LL!RemoveCopy(t)$

$$LC!RemoveCopy(t)$$
$$na(t) > 0$$

$\equiv LL!RemoveCopy(t)$

$$t : Title;\ nc(t) \geq na(t) > 0$$
$$nc'(t) = nc(t) - 1$$
$$na'(t) = na(t) - 1$$

We require an event to allow members to make loans:

$LL!LoanCopy(t, m)$
$t : Title\,;\ m : Member$ $t \mapsto m \notin loan$ $na(t) > 0$
$t \mapsto m \in loan'$

$\equiv$

$LL!LoanCopy(t, m)$
$t : Title\,;\ m : Member$ $t \mapsto m \notin loan$ $na(t) > 0$
$t \mapsto m \in loan'$ $na'(t) = na(t) - 1$ $nl'(t) = nl(t) + 1$

Finally at this level, we define an event to return a copy:

$LL!Return(t, m)$
$t \mapsto m : loan$
$t \mapsto m \notin loan'$

$\equiv$

$LL!Return(t, m)$
$t : Title\,;\ m : Member$ $t \mapsto m \in loan$ $nl(t) > 0$
$t \mapsto m \notin loan'$ $na'(t) = na(t) + 1$ $nl'(t) = nl(t) - 1$

The expanded versions of *LoanCopy* and *Return* show how the numbers of available and loaned copies change.

Throughout this specification we have defined *minimal* versions of the operations. We are able to do this because of the assumption that *"the rest remains unchanged"*. In some cases we used expansion to explicitly present interesting derived properties.

Exercise 5
Show that the assumption *a member may borrow at most one copy of any particular title at the same time* is implicit from the invariant of class *LL*.

Exercise 6
Define a query to show titles borrowed by a member.

Exercise 7
Suppose that operations to 'remove' members or titles were provided for class *LL*. Specify these events, which are *applicable* only when the respective member or title does not figure in any current loan.

Reservation Class: LR

We specify LR as a class parallel to LL to allow titles to be reserved. Eventually we will compose LL and LR to specify a library system able to provide both loans and reservations. To specify LR, we again export and compose LM and LC as we did for LL. We define a 'request queue' ($requestQ$) to be a function that delivers, for each title t, the sequence of members currently requesting a copy of t. Similarly to *loans* in class (LL), we define the current reservations as a relation (*reserve*). We also define the number of requests (nQ) for each title.

LR

$$
\begin{array}{|l}
\hline
LM\ ;\ LC \\
requestQ : Title \rightarrow \mathsf{inj} \cdot \mathsf{seq}\ Member \\
reserve : Title \nrightarrow Member \\
\mathsf{cf}\ reserve = \mathsf{cod} \circ requestQ \\
nQ := (\#) \circ requestQ \\
\hline
\end{array}
$$

$\equiv$ LR

$$
\begin{array}{|l}
\hline
Member : \mathsf{set}\ \mathbb{M} \\
info : Member \rightarrow \mathbb{I} \\
Title : \mathsf{set}\ \mathbb{T} \\
desc : Title \rightarrow \mathbb{D} \\
requestQ : Title \rightarrow \mathsf{inj} \cdot \mathsf{seq}\ Member \\
reserve : Title \nrightarrow Member \\
nc, nQ : Title \rightarrow \mathsf{NAT} \\
InColl := \{t : Title \bullet nc(t) > 0\} \\
ExColl := \{t : Title \bullet nc(t) = 0\} \\
\forall t : Title \bullet \\
\quad \{m : Member \bullet t \mapsto m \in reserve\} = \{m : Member \bullet m \in \mathsf{cod}\ requestQ(t)\} \wedge \\
\quad nQ(t) = \#\{m : Member \bullet t \mapsto m \in reserve\} \wedge \\
\quad nQ(t) = \#requestQ(t) \\
\hline\hline
Member' = \emptyset\ ;\ Title' = \emptyset \\
\hline
\end{array}
$$

Class LR embodies another significant assumption: *a member may have at most one request for every title outstanding at the same time.*

As we plan to compose this class with LL, we won't promote or extend any of the existing queries to this level. However, we will define a query to show the queue of requests for a particular title:

$LR?showRequests(t \rightarrow Q)$

$$
\begin{array}{|l}
\hline
Q := requestQ(t) \\
\hline
\end{array}
$$

We need to extend the collection event *NewTitle* at this level, to create a new (initially-empty) request queue for the title that is introduced:

$$LR!NewTitle(d \to t) \quad \equiv \quad LR!NewTitle(d \to t)$$

$$\begin{array}{|l|}\hline LC!NewTitle(d \to t) \\ \hline\hline requestQ'(t) = \langle\rangle \\ \hline\end{array} \qquad \begin{array}{|l|}\hline d : \mathbb{D};\ t : \mathbb{T};\ t \notin Title \\ \hline\hline t \in Title' \cap ExColl' \\ desc'(t) = d;\ requestQ'(t) = \langle\rangle \\ nc'(t) = nQ'(t) = 0 \\ \hline\end{array}$$

Further events will be required in this context to update the pending requests for some particular title. An event to allow a member to reserve a particular title:

$$\begin{array}{|l|}\hline LR!Reserve(t, m \to p) \\ \hline t : Title;\ m : Member;\ p : \mathsf{POS} \\ t \mapsto m \notin reserve;\ p = nQ(t) + 1 \\ \hline\hline requestQ'(t) = (requestQ(t))\langle m\rangle \\ nQ'(t) = p \\ t \mapsto m \in reserve' \\ \hline\end{array}$$

and an event to cancel such a reservation:

$$\begin{array}{|l|}\hline LR!Cancel(t, m \to p) \\ \hline t : Title;\ m : Member;\ p : \mathsf{POS} \\ t \mapsto m \in reserve \\ Q_1\langle m\rangle Q_2 := requestQ(t) \\ p = \#\, Q_1 + 1 \\ \hline\hline requestQ'(t) = \ Q_1\langle\rangle\, Q_2 \\ nQ'(t) = nQ(t) - 1 \\ t \mapsto m \notin reserve' \\ \hline\end{array}$$

These two events preserve the *relative* ordering of all other pending requests whenever one for title t and member m is either 'reserved' or 'cancelled'; they also give its new or previous position p in the queue.

Exercise 8
Show that the assumption *a member may have at most one request for every title outstanding at the same time* is implicit from the invariant of class *LR*.

Exercise 9
Were we to remove a member, we'd need to consider their requests as well. Would an event to remove a title need to consider requests for that title? Develop specifications of these two events (remove member, remove title) in this class.

Simple Library Class: LS

We model a simple library class by *composing* the loan (LL) and reservation (LR) classes. The composed specification must be consistent:

LS

$$LL\,;\ LR$$
$$loan \cap reserve = \emptyset$$

$\equiv\ LS$

$$Member : \mathsf{set}\ \mathbb{M}$$
$$info : Member \to \mathbb{I}$$
$$Title : \mathsf{set}\ \mathbb{T}$$
$$desc : Title \to \mathbb{D}$$
$$requestQ : Title \to \mathsf{inj} \cdot \mathsf{seq}\ Member$$
$$loan, reserve : Title \nrightarrow Member$$
$$\mathsf{cf}\ reserve = \mathsf{cod} \circ requestQ$$
$$loan \cap reserve = \emptyset$$
$$nc, na, nl, nQ : Title \to \mathsf{NAT}$$
$$InColl := \{t : Title \bullet nc(t) > 0\}$$
$$ExColl := \{t : Title \bullet nc(t) = 0\}$$
$$\forall t : Title \bullet$$
$$nc(t) = na(t) + nl(t) \land$$
$$nl(t) = \#\{m : Member \bullet t \mapsto m \in loan\} \land$$
$$\{m : Member \bullet t \mapsto m \in reserve\} = \{m : Member \bullet m \in \mathsf{cod}\ requestQ(t)\} \land$$
$$nQ(t) = \#\{m : Member \bullet t \mapsto m \in reserve\} \land$$
$$nQ(t) = \#requestQ(t)$$

$$Member' = \emptyset\,;\ Title' = \emptyset$$

To maintain consistency, we have imposed a further constraint at this level: *for each title, a member may have at most one loan* or *one reservation at the same time.*

Most required queries are now direct promotions:

$LS?showInfo(m \to i)$

$$LM?showInfo(m \to i)$$

$\equiv\ LS?showInfo(m \to i)$

$$m : Member\,;\ i := info(m)$$

$LS?showMembers(\to M)$

$$LM?showMembers(\to M)$$

$\equiv\ LS?showMembers(\to M)$

$$M := info$$

$LS?showDesc(t \to d)$

$$LT?showDesc(t \to d)$$

$\equiv\ LS?showDesc(t \to d)$

$$d := desc(t)$$

$LS?showNoCopies(t \to n)$

$LC?showNoCopies(t \to n)$

$\equiv LS?showNoCopies(t \to n)$

$n := nc(t)$

$LS?showLoans(t \to M)$

$LL?showLoans(t \to M)$

$\equiv LS?showLoans(t \to M)$

$M := \{m : Member \bullet t \mapsto m \in loan\}$

$LS?showRequests(t \to Q)$

$LR?showRequests(t \to Q)$

$\equiv LS?showRequests(t \to Q)$

$Q := requestQ(t)$

Finally we'll update the *showCollection* query from the loan class to include the number of reservations for each title in the collection:

$LS?showCollection(\to C)$

$$C : Title \pfun \mathsf{POS} \times \mathsf{NAT} \times \mathsf{NAT} \times \mathbb{D}$$
$$C = \{(t, n, l, q, d) : InColl \times \mathsf{POS} \times \mathsf{NAT} \times \mathsf{NAT} \times \mathbb{D} \bullet$$
$$n = nc(t) \land l = nl(t) \land q = nQ(t) \land d = desc(t)\}$$

Most events are also direct promotions:

$LS!NewMember(i \to m)$

$LM!NewMember(i \to m)$

$\equiv LS!NewMember(i \to m)$

$i : \mathbb{I}\,;\ m : \mathbb{M}\,;\ m \notin Member$

$m \in Member'\,;\ info'(m) = i$

$LS!UpdateInfo(m, i)$

$LM!UpdateInfo(m, i)$

$\equiv LS!UpdateInfo(m, i)$

$m : Member\,;\ i : \mathbb{I}\,;\ i \neq info\ m$

$info'(m) = i$

$LS!AddCopies(t, n)$

$LL!AddCopies(t, n)$

$\equiv LS!AddCopies(t, n)$

$t : Title\,;\ n : \mathsf{POS}$

$nc'(t) = nc(t) + n$
$na'(t) = na(t) + n$

$LS!RemoveCopy(t)$

$LL!RemoveCopy(t)$

$\equiv LS!RemoveCopy(t)$

$t : Title\,;\ nc(t) \geq na(t) > 0$

$nc'(t) = nc(t) - 1$
$na'(t) = na(t) - 1$

We inherit versions of NewTitle from both loan and reservation classes and combine them:

$$
\begin{array}{|l|}
\hline
LS!NewTitle(d \to t) \\
\hline
LL!NewTitle(d \to t) \\
LR!NewTitle(d \to t) \\
\hline
\end{array}
\quad \equiv \quad
\begin{array}{|l|}
\hline
LS!NewTitle(d \to t) \\
\hline
d : \mathbb{D}\,;\ t : \mathbb{T}\,;\ t \notin Title \\
\hline\hline
t \in Title'\,;\ t \in ExColl' \\
desc'(t) = d \\
nc'(t) = na'(t) = nl'(t) = 0 \\
nQ'(t) = 0 \\
requestQ'(t) = \langle\rangle \\
\hline
\end{array}
$$

Stronger constraints are imposed for making a reservation, to preserve the invariant:

$$
\begin{array}{|l|}
\hline
LS!Reserve(t, m \to p) \\
\hline
LR!Reserve(t, m \to p) \\
t \mapsto m \notin loan \\
\hline
\end{array}
\quad \equiv \quad
\begin{array}{|l|}
\hline
LS!Reserve(t, m \to p) \\
\hline
t : Title\,;\ m : Member\,;\ p : \mathsf{POS} \\
t \mapsto m \notin reserve \cup loan \\
p = nQ(t) + 1 \\
\hline\hline
requestQ'(t) = (requestQ(t))\langle m\rangle \\
nQ'(t) = p \\
t \mapsto m \in reserve' \\
\hline
\end{array}
$$

The event *cancel* is just a promotion from the reservation class:

$$
\begin{array}{|l|}
\hline
LS!Cancel(t, m \to p) \\
\hline
LR!Cancel(t, m \to p) \\
\hline
\end{array}
\qquad
\begin{array}{|l|}
\hline
LR!Cancel(t, m \to p) \\
\hline
t : Title\,;\ m : Member\,;\ p : \mathsf{POS} \\
t \mapsto m \in reserve \\
Q_1\langle m\rangle Q_2 := requestQ(t) \\
p = \#\,Q_1 + 1 \\
\hline\hline
requestQ'(t) = \ Q_1\langle\rangle\, Q_2 \\
nQ'(t) = nQ(t) - 1 \\
t \mapsto m \notin reserve' \\
\hline
\end{array}
$$

The *LoanCopy* event becomes interesting when reservations are taken into account. There are two cases: the member has reserved the title or not. If not then the member can borrow a copy if there are more available copies than the number of reservations. Otherwise, that is the member has reserved the title, the member can borrow a copy if there are more available copies than the number of reservations prior to the member in the queue. In this case, the member's reservation is cancelled when the copy is loaned.

$LS!LoanCopy(t, m)$

$LL!LoanCopy(t, m)$
$n : \mathsf{NAT}\,;\ na(t) > n$

$t \mapsto m \notin reserve\,;\ n = nQ(t)$

$LR!Cancel(t, m \to p)$
$n = p - 1$

$\equiv LS!LoanCopy(t, m)$

$t : Title\,;\ m : Member\,;\ n : \mathsf{NAT}$
$t \mapsto m \notin loan\,;\ na(t) > n$

$t \mapsto m \notin reserve\,;\ n = nQ(t)$

$t \mapsto m \in reserve$
$Q_1\langle m\rangle\, Q_2 := requestQ(t)$
$n = \#\, Q_1$

$requestQ'(t) = \ Q_1\langle\rangle\, Q_2$
$nQ'(t) = nQ(t) - 1$
$t \mapsto m \notin reserve'$

$t \mapsto m \in loan'$
$na'(t) = na(t) - 1$
$nl'(t) = nl(t) + 1$

Finally, the event *Return* is a direct promotion from the loan class:

$LS!Return(t, m)$

$LL!Return(t, m)$

$\equiv LS!Return(t, m)$

$t : Title\,;\ m : Member$
$t \mapsto m \in loan$
$nl(t) > 0$

$t \mapsto m \notin loan'$
$na'(t) = na(t) + 1$
$nl'(t) = nl(t) - 1$

Exercise 10
We have promoted or re-defined all queries at this level; was it *necessary* to specify them earlier?

Exercise 11
Explain the extended *LoanCopy* event as specified for class LS, saying how its local variable n corresponds to the number of prior reservations in both sub-cases.

Exercise 12
Suggest ways in which the current Simple Library class can be extended or refined. (This is a very open-ended question.)