Library System extension

Richard Goodman
13842540

Detailed Library Class: LD

For the extension we begin by creating a new invariant which introduces new functionalities for the Library system, as well as inheriting the modelling choices in the previous invariants LL and LR. The concept of this extension is to introduce dates into the system, because *loan* and *reservation* were already defined as relations, we create new relations mapping said loan or reservation to a date. This allows us to then append and query dates onto these relations.

With this we can keep a record of key date variables, for example, *ls (loan start date)* and *le (loan end date)*, and a similar case for reservations *rs (reservation start date)* and *re (reservation end date)*.

We create a variable of the type DATE calling it today, this allows us to store today's date in a very abstract way. Finally, there are some more sets which define the states of *Loans* and *reservations*.

LD

$$
\begin{array}{|l}
\hline
LL\,;\ LR \\
today : \mathsf{DATE} \\
ls, le : loan \rightarrow \mathsf{DATE} \\
rs, re : reserve \rightarrow \mathsf{DATE} \\
DueToday := \{l : loan \bullet le(l) = today\} \\
OverDue := \{l : loan \bullet le(l) > today\} \\
ResExpire := \{r : reserve \bullet re(r) > today\} \\
\hline
\end{array}
$$

We begin by introducing simple queries that help retrieve information. In this case, two queries are presented returning either date for a given loan. Having these kind of queries defined can help later on when creating more complex events or queries.

$LD?showLoanStartDate(l \rightarrow d)$

$$
\begin{array}{|l}
\hline
d := ls(l) \\
\hline
\end{array}
$$

$LD?showLoanDueDate(l \rightarrow d)$

$$
\begin{array}{|l}
\hline
d := le(l) \\
\hline
\end{array}
$$

These two queries are replica's of the queries presented above, except return the dates for any given reservation.

$LD?showResStartDate(r \rightarrow d)$

$$d := rs(r)$$

$LD?showResEndDate(r \rightarrow d)$

$$d := re(r)$$

Dates can become a confusing and difficult property to work with. In this case, today's *today* is different to tomorrow's *today*, likewise the outcome of *today + 3* is different to the outcome of a different 'today' + 3. Nevertheless, with this model being a high-level abstract representation, we don't have to focus on the background logic of *'today'*, as long as we understand the basic behaviour for the variable.

In this modelling choice, we need to establish an event for when today moves onto the next day. With this however, side affects are apparent. With a new day being introduced, loans could now become due for return that day, and loans could also become over due *(if they were due what is now yesterday)*, similarly reservations could also expire.

This particular event features a case analysis, this is because not all loans could be *DueToday* before the day changes. With this aside, for all cases, there can be instances of reservations where the reservation end date is today, making it expire when the day changes.

$LD!nextDay()$

$$\exists r : reserve \bullet \{re(r) = today \land r \notin ResExpire\}$$

$$l \in DueToday$$

$$l \in OverDue'$$

$$today' = today + 1$$
$$r \in ResExpire'$$

The invariant declaration for reservations now comes with a duration before it is classed *'expired'*. This specific event inherits the method defined in the previous model. Once cancelled it is important to make sure this reservation is not a member of expired reservations anymore.

$LD!expired(t, m \rightarrow p)$

$$r \in ResExpire$$

$$LR!Cancel(t, m \rightarrow p)$$
$$r \notin ResExpire'$$

This following query produces a set which shows all *Title's* and their due dates in a tuple for a given member. By checking if the relation of the Title matches the member in a loan, we can determine the right loan for that given title and acquire the due date for that loan.

$LD?dueDatesForMember(l, m \rightarrow L)$

$$L : Title \nrightarrow \mathsf{DATE}$$
$$L = \{(t, d) : Title \times \mathsf{DATE} \bullet t \mapsto m \in loan \land d = le(l)\}$$

The next two queries follow the behaviour of the previous, just altered slightly. It can become an obvious request to have a set of all Members loaning a Title that is due today.

$LD?loansDueToday(\rightarrow L)$

$$L := \{l : DueToday\}$$

Likewise, it would be compulsory for a real-life scenario to know all the loans are overdue and the members who currently have an overdue loan. The major difference this query has compared to the previous is the tuple also containing the loans due date, to emphasise the meaning of having a loan overdue.

$LD?loansOverDue(\rightarrow L)$

$$L : Title \nrightarrow Member \times \mathsf{DATE}$$
$$L = \{(t, m, d) : Title \times Member \times \mathsf{DATE} \bullet \forall l : OverDue \Rightarrow$$
$$t = l[1] \land m = l[2] \land d = le(l)\}$$

We need to promote this event from a previous model as we are introducing new variables and relations. With the need to now store dates for any loan, we want to inherit the behaviour previously stated, but just incorporate dates in to said loan. As we are creating a new loan in the post condition, we need to have this *Title* and *Member* who are mapped in parenthesis for the dates as they act as the loan.

$LD!LoanCopy(t, m)$

$$t : Title\,;\ m : Member\,;\ n : \mathsf{NAT}\,;\ d1 : \mathsf{DATE}\,;\ d2 : \mathsf{DATE}$$
$$t \mapsto m \notin loan\,;\ na(t) > n\,;\ d2 = d1 + \mathsf{POS}$$

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

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

$$t \mapsto m \in loan'$$
$$(t \mapsto m) \mapsto d1 \in ls'$$
$$(t \mapsto m) \mapsto d2 \in le'$$

Here we are giving the option for a member to extend their loan. Presented are two possible scenarios for this event. We can either reinstate the date for a given loan, or simply increment the due date by a positive integer.

$LD!extendLoan(l, d)$

$l : loan\,;\ d : \mathsf{DATE}\,;\ d \neq le(l)$

$le'(l) = d$

$\equiv\ LD!extendLoan(l, d)$

$d = le(l)$

$le'(l) = d + \mathsf{POS}$

Here is another event that required a promotion with added functionality. As we are also introducing dates to reservations, we need to inherit the previous behaviour and apply dates to the new relations we have stated in the invariant. This particular event has similar post conditions to *LD!LoanCopy*, in the fact that because this is a new reservation we have the mapping of $t \mapsto m$ in parenthesis to act as the reservation for the new relations.

$LD!Reserve(t, m \to p)$

$t : Title\,;\ m : Member$
$p : \mathsf{POS}\,;\ d1 : \mathsf{DATE}\,;\ d2 : \mathsf{DATE}$
$t \mapsto m \notin reserve\,;\ t \mapsto m \notin loan$
$p = nQ(t) + 1\,;\ d2 = d1 + \mathsf{POS}$

$requestQ'(t) = (requestQ(t))\langle m \rangle$
$nQ'(t) = p$
$t \mapsto m \in reserve'$
$(t \mapsto m) \mapsto d1 \in rs'$
$(t \mapsto m) \mapsto d2 \in re'$

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

$LD?showCollection(\to C)$

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

Direct promotions

The remaining queries and events are just direct promotions from the previous invariants that do not need any additional changes for the extension of this system.

$LD?showInfo(m \to i)$

$LM?showInfo(m \to i)$

$LD?showMembers(\to M)$

$LM?showMembers(\to M)$

$LD?showDesc(t \to d)$	$LD?showNoCopies(t \to n)$
$LM?showDesc(t \to d)$	$LC?showNoCopies(t \to n)$
$LD?showLoans(t \to M)$	$LD?showRequests(t \to Q)$
$LL?showLoans(t \to M)$	$LR?showRequests(t \to Q)$
$LD!NewMember(i \to m)$	$LD!UpdateInfo(m, i)$
$LM!NewMember(m, i)$	$LM!UpdateInfo(m, i)$
$LD!AddCopies(t, n)$	$LD!RemoveCopy(t)$
$LL!AddCopies(t, n)$	$LL!RemoveCopy(t)$
$LD!Return(t, m)$	$LD!Cancel(t, m \to p)$
$LL!Return(t, m)$	$LR!Cancel(t, m \to p)$