CMU 15-671 | Models of Software Systems | Fall 1997

# A Z Solution of the Library Problem

Garlan | Handout 8a | 8 October 1997

## 1 The Problem

Consider a small library database with the following transactions:[1]

1. Check out a copy of a book; Return a copy of a book;
2. Add a copy of a book to the library; Remove a copy of a book from the library;
3. Get the list of books by a particular author or in a particular subject area;
4. Find out the list of books currently checked out by a particular borrower;
5. Find out what borrower last checked out a particular copy of a book.

There are two types of users: staff users and ordinary borrowers. Transactions 1, 2, 4 and 5 are restricted to staff users, except that ordinary borrowers can perform transaction 4 to find the list of books currently borrowed by themselves. The database must also satisfy the following constraints:

- All copies in the library must be available for checkout or be checked out.
- No copy of the book may be both available and checked out at the same time.
- A borrower may not have more than a predefined number of books checked out at one time.

## 2 A Solution

I will divide my design into three parts. In the first part I describe the state space of the library. In the second part I define the operations on that space. In the third part I add access control to those operations.

**State Space:** I will assume that every book has an $AUTHOR$, a $TITLE$, a set of $SUBJECT$s, and an $BOOKID$, which I take to be primitive sets. [2]

$$[AUTHOR, TITLE, SUBJECT, BOOKID]$$



[1] From R.A. Kemmerer, "Testing formal specifications to detect design errors".

[2] In doing it this way, I am actually defining a "copy" of a book. If I wanted to separate these two notions, I would first define an abstract $Book$ as having an $AUTHOR$, a $TITLE$, and a set of $SUBJECT$s. Then I would define a $Copy$ as a new schema that includes $Book$, and adds an $id$ to distinguish the different copies.

$$\begin{array}{|l}\hline \textit{Book} \\ author : AUTHOR \\ title : TITLE \\ subjects : \mathbb{P}\, SUBJECT \\ id : BOOKID \\ \hline\end{array}$$

The basic design of the library system centers around a record keeping facility that associates with each book some data, which records the most recent user of each book and whether the book is checked out. The universe of people will be designated by the set $PERSON$, and the borrowing status of book is designated by the enumeration type $STATUS$.

$$[PERSON]$$

$$STATUS ::= out \mid in$$

$$\begin{array}{|l}\hline \textit{Data} \\ lastuser : PERSON \\ status : STATUS \\ \hline\end{array}$$

It will be handy to consider *nobody* a special kind of person. In our record keeping, I will use the convention that if a book was last checked out by *nobody*, then it has never been checked out before. In situations where *nobody* is inappropriate I will use the type $REAL\_PERSON$. The global number $MAX$ represents the maximum number of books that any user can have checked out of the library at one time.

$$\begin{array}{|l} MAX : \mathbb{N} \\ nobody : PERSON \end{array}$$

$$REAL\_PERSON == PERSON \setminus \{nobody\}$$

The *Library* itself is now defined as an entity consisting of a set of *books*, a set of *records*—one record for each book—and sets of *users* and *staff*. The *users* of the library are the ordinary borrowers; the *staff* run the library and (as we will see later) perform certain priviledged operations. The state invariant of the library guarantees that each book in the library has a unique *id*, that there is a record for exactly the books that are in the library, and that no user has checked out more than $MAX$ books.

$$\begin{array}{|l}\hline \textit{Library} \\ books : \mathbb{P}\, Book \\ records : Book \pfun Data \\ users, staff : \mathbb{P}\, REAL\_PERSON \\ \hline \forall\, b_1, b_2 : books \bullet b_1.id = b_2.id \Leftrightarrow b_1 = b_2 \\ \mathrm{dom}\, records = books \\ \forall\, user : users \bullet \\ \quad \#\{b : books \mid (records\ b).lastuser = user \land \\ \qquad (records\ b).status = out\} \leq MAX \\ \hline\end{array}$$

This definition of *Library* guarantees that the problem constraints are met. First, every book is either available or checked out, since every book has a unique record, and the status field of each such record is either *in* or *out*. Second, by the same argument no book may be both available and checked out. Third, the limit on checked out books is enforced through the state invariant of *Library*.

**Operations:** Many of the operations on books in the library have a similar form: they involve some set of changes to the *Library*, but leave the set of *staff* and *users* unchanged. I encapsulate these aspects of an operation once and for all in the schema, *BookOp*.

$$
\begin{array}{|l}
\hline
\textit{BookOp} \\
\Delta \textit{Library} \\
\hline
\textit{staff}' = \textit{staff} \land \textit{users}' = \textit{users} \\
\hline
\end{array}
$$

Our approach to defining the checkout/checkin operations will be to first define operations on the book data, and then to "promote" them into the context of the full library. There are two data-changing operations to consider: one to mark the book data as checked out (*CheckoutData*), and one to mark it as checked in (*CheckinData*). In the former case we update the *lastuser* of the book to be the new borrower. Both operations change the status of the book.

$$
\begin{array}{|l}
\hline
\textit{CheckoutData} \\
\textit{data}, \textit{data}' : \textit{Data} \\
\textit{borrower}? : \textit{REAL\_PERSON} \\
\hline
\textit{data}.\textit{status} = \textit{in} \\
\textit{data}'.\textit{status} = \textit{out} \\
\textit{data}'.\textit{lastuser} = \textit{borrower}? \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{CheckinData} \\
\textit{data}, \textit{data}' : \textit{Data} \\
\hline
\textit{data}.\textit{status} = \textit{out} \\
\textit{data}'.\textit{status} = \textit{in} \\
\textit{data}.\textit{lastuser} = \textit{data}'.\textit{lastuser} \\
\hline
\end{array}
$$

To place these operations in the context of the library system I will use a technique sometimes referred to as "promotion". First I define a generic "frame" that sets up the *context* for incorporating the previous operations. This is the schema *DataFrame*, which defines an open-ended operation that modifies the library by updating the data record for a given library *book*?. It allows some (as yet unspecified) changes to be made to the data of *book*?, and then updates *records* to reflect the new data.

$$
\begin{array}{|l}
\hline
\textit{DataFrame} \\
\textit{BookOp} \\
\textit{data}, \textit{data}' : \textit{Data} \\
\textit{book}? : \textit{Book} \\
\hline
\textit{book}? \in \textit{books} \\
\textit{data} = (\textit{records}\ \textit{book}?) \\
\textit{records}' = \textit{records} \oplus \{\textit{book}? \mapsto \textit{data}'\} \\
\hline
\end{array}
$$

(As an aside, note that any operation constructed out of *DataFrame* leaves the set of library books unchanged. This is because the schema does not change the domain of *records*, which is defined to be the same as the set *books*. Exercise: prove this.)

Now for the operations themselves, I simply combine the previous data operations with the framing schema, and add any other constraints to relate the parameters of the operation.

$$
\begin{array}{|l}
\hline
\textit{Checkout0} \\
\textit{DataFrame} \\
\textit{CheckoutData} \\
\hline
\textit{borrower?} \in \textit{users} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Checkin0} \\
\textit{DataFrame} \\
\textit{CheckinData} \\
\hline
\end{array}
$$

We could stop here. But it would be cleaner to hide the components *data* and $data'$, which are, after all, simply local variables of the operation. We use the hiding operator of Z to do this:

$$\textit{Checkout} \mathrel{\widehat{=}} \textit{Checkout0} \setminus (\textit{data}, \textit{data}')$$
$$\textit{Checkin} \mathrel{\widehat{=}} \textit{Checkin0} \setminus (\textit{data}, \textit{data}')$$

Now consider the operations to add and delete books from the library. The *AddBook* operation makes use of a local variable, *data*, to represent the initial entry for a new book. *records* is updated with the new *book*? and its associated data. Note that the precondition on *book*? is required to guarantee that old records aren't destroyed.

$$
\begin{array}{|l}
\hline
\textit{AddBook} \\
\textit{BookOp} \\
\textit{book?} : \textit{Book} \\
\hline
\textit{book?} \notin \textit{books} \\
\exists\, d : \textit{Data} \mid d.\textit{lastuser} = \textit{nobody} \land d.\textit{status} = \textit{in} \bullet \\
\qquad \textit{records}' = \textit{records} \oplus \{\textit{book?} \mapsto d\} \\
\hline
\end{array}
$$

Removing a book is even easier, since we don't need to identify the data to be removed. I simply form a new set of records, which no longer has an entry for the removed *book*?. (Note that the invariant of *Library* implies that *book*? is removed from the library if it is removed from the library records.)

$$
\begin{array}{|l}
\hline
\textit{RemoveBook} \\
\textit{BookOp} \\
\textit{book?} : \textit{Book} \\
\hline
\textit{book?} \in \textit{books} \\
\textit{records}' = \{\textit{book?}\} \mathbin{\lhd} \textit{records} \\
\hline
\end{array}
$$

Finally, consider the operations for querying the library. Each of these is relatively straightforward: the resulting set of books is defined using an appropriate predicate together with set comprehension.

$$
\begin{array}{|l}
\hline
\textit{BooksByAuthor} \\
\Xi \textit{Library} \\
\textit{author?} : \textit{AUTHOR} \\
\textit{books!} : \mathbb{P}\, \textit{Book} \\
\hline
\textit{books!} = \{b : \textit{books} \mid b.\textit{author} = \textit{author?}\} \\
\hline
\end{array}
$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{BooksBySubject}} \\
\Xi Library \\
subject? : SUBJECT \\
books! : \mathbb{P}\, Book \\
\hline
books! = \{ b : books \mid subject? \in b.subjects \} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{BooksByBorrower}} \\
\Xi Library \\
borrower? : PERSON \\
books! : \mathbb{P}\, Book \\
\hline
books! = \{ b : books \mid (records\ b).lastuser = borrower? \} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{BorrowerByBook}} \\
\Xi Library \\
book? : Book \\
borrower! : PERSON \\
\hline
book? \in books \\
borrower! = (records\ book?).lastuser \\
\hline
\end{array}$$

Note that in the last two operations we allow the query to be performed with any person (even *nobody*).

**Access Control:** Thus far, I have not considered any of the restrictions that are placed on who actually performs the operations. I will now build in access control using the schema calculus. First I define the kinds of access that are permitted. Then I associate the appropriate access with each operation.

In terms of access control there are three kinds of operations:

1. *Restricted* operations, which can be performed only by library *staff*;
2. *UnRestricted* operations, which can be performed by any library *user* or *staff*; and
3. *SemiRestricted* operations, which can be performed either by *staff* or by a *user* that is the same as the *borrower* in the operation.

Each of these is defined in terms of the person who actually performs the operation. I refer to this person as the *doer?* of the operation; this is in contrast to the *borrower?*, for whom the operation is performed.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Restricted}} \\
\Delta Library \\
doer? : PERSON \\
\hline
doer? \in staff \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{UnRestricted}} \\
\Delta Library \\
doer? : PERSON \\
\hline
doer? \in staff \cup users \\
\hline
\end{array}$$

$$
\begin{array}{|l}
\hline
\textit{SemiRestricted} \\
\Delta Library \\
doer? : PERSON \\
borrower? : PERSON \\
\hline
doer? \in \mathit{staff} \cup \{borrower?\} \\
\hline
\end{array}
$$

Now I associate these restrictions with the operations.

$$
\begin{array}{l}
SafeCheckout \mathrel{\widehat=} Checkout \land Restricted \\
SafeCheckin \mathrel{\widehat=} Checkin \land Restricted \\
SafeAddBook \mathrel{\widehat=} AddBook \land Restricted \\
SafeRemoveBook \mathrel{\widehat=} RemoveBook \land Restricted \\
SafeBooksByAuthor \mathrel{\widehat=} BooksByAuthor \land UnRestricted \\
SafeBooksBySubject \mathrel{\widehat=} BooksBySubject \land UnRestricted \\
SafeBooksByBorrower \mathrel{\widehat=} BooksByBorrower \land SemiRestricted \\
SafeBorrowerByBook \mathrel{\widehat=} BorrowerByBook \land Restricted
\end{array}
$$