Describing UML Use Cases as Contracts

R. J. R. Back
(joint work with Luigia Petre and Ivan Porres Paltor)

Turku Centre for Computer Science and
Åbo Akademi University

UML

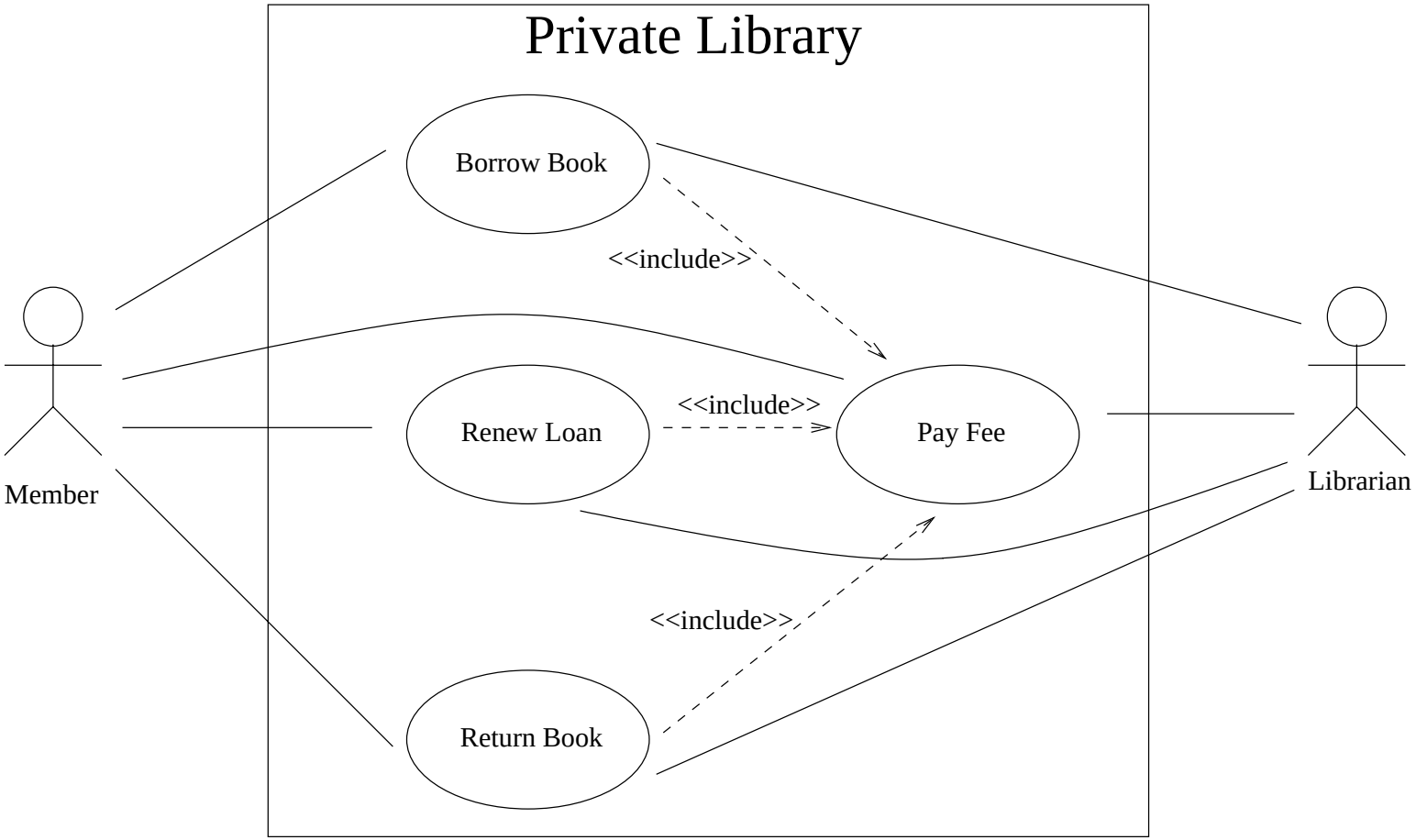
***Unified Modeling Language (UML)*:**

- **Standard *diagrammatic notation* for describing object-oriented software systems**
  - **class diagrams, use case diagrams, state charts, sequence diagrams, collaboration diagrams, etc**
- **Used in the *modelling phase* of the software engineering process**
- **The *consistency* of the UML model with the user requirements is important**
  - **checking usually *postponed* to later phases**

Use cases

- **A use case is a sequence of transactions in a system, whose task is to yield a measurable value to an individual actor of the system**
- **The set of use case descriptions specifies the complete functionality of the system (Ivar Jacobson, 1987)**
- **Use cases described informally, in English, or by sequence (collaboration) diagrams.**

Use case diagram: Private Library



Use case: Borrow a book

Use Case	Borrow a book
Actors	Member, Librarian
Summary	The Member borrows a book from the system
Precondition	The Member has no old debts to the system
Description	The Member chooses a book that is not already lent and that s/he does not own. The Librarian assigns the Member as the borrower of that book and also states a deadline for returning the book.
Postcondition	The Member has successfully borrowed the book
Exceptions	1. If the Member has a due to pay, the Member can pay it (see *Pay the fee* use case) and try again. 2. The Member owns the book. 3. The book is already lent.
Used use cases	Pay the fee

Use case: Renew the lending

Use Case	Renew loan
Actors	Member, Librarian
Summary	The Member renews the loan of a book from the system
Precondition	The Member has no dues to the system and has borrowed books
Description	The Member chooses a book that is already lent by her/himself. The Librarian assigns a new deadline for returning that book.
Postcondition	The Member has successfully renewed the loan
Exceptions	1. If the Member has old debts to pay, the Member can pay it (see *Pay the fee* use case) and try again. 2. The Member has not borrowed the book before.
Used use cases	Pay the fee

Use case: Return a book

Use Case	Return book
Actors	Member, Librarian
Summary	The Member returns a book to the system
Precondition	The Member has borrowed the book from the system
Description	The book is returned and the Member is no more the borrower of the book. If the Member has debts to pay, the Member can pay them (see 'Pay the fee' use case).
Postcondition	The Member has successfully returned the book
Exceptions	1. The Member has not borrowed the book from the system. 2. The book is already returned.
Used use cases	Pay the fee

Use case: Pay the fee

Use Case	Pay the fee
Actors	Member, Librarian
Summary	The Member pays a certain amount of his/her debts to the System
Precondition	The Member has a debt to the System
Description	The Member chooses a certain amount of debt, and pays it. The Librarian substracts the sum from the Member's debt.
Postcondition	The Member has successfully payed the sum
Exceptions	The Member has no debts to pay.

Advantages of UML use cases

Use cases:

- **capture the externally-required functionality of the system.**
- **identify the different goals for individual actors.**
- **identify candidate objects for the problem domain.**
- **gain an understanding of the problem domain.**
- **gain an understanding of the proposed solution.**

Another benefit of use cases comes from the fact that they are *accountable*, i.e. they are part of the agreement between the users and the developers of the system.

Disadvantages of UML use cases

- They are *informal*. This is an advantage at an earlier stage in the development process, but later on, informal requirements can be easily misinterpreted.
- It is *difficult* to check whether the system provides the functionality expected by the actors.
- Sequence and collaboration diagrams only provide*examples* of how the system should behave, they are only possible scenarios, not complete specifications.
- They are essentially *functional* in character, even though they are used to develop *object-oriented systems*. There is a missing link between functional use case diagrams and object-oriented class diagrams.

Complementing use cases with contracts

- We need a *systematic method* of verifying that the use case model is consistent with user requirements
- This requires that use cases are described *precisely* and *unambiguously*
- We propose the notion of a *contract* as the formal counterpart of the use case model
- We use the contract to *verfy* the consistency of the use case model
- Contracts *complement* use cases, they do not replace use cases

Describing the state

The state of a contract models the problem domain of the system, and is graphically represented in UML as a class diagram.

For the example, we need to model two notions within our system: a *person* and a *book*. We give a UML class for each of them.

Class and contract notation

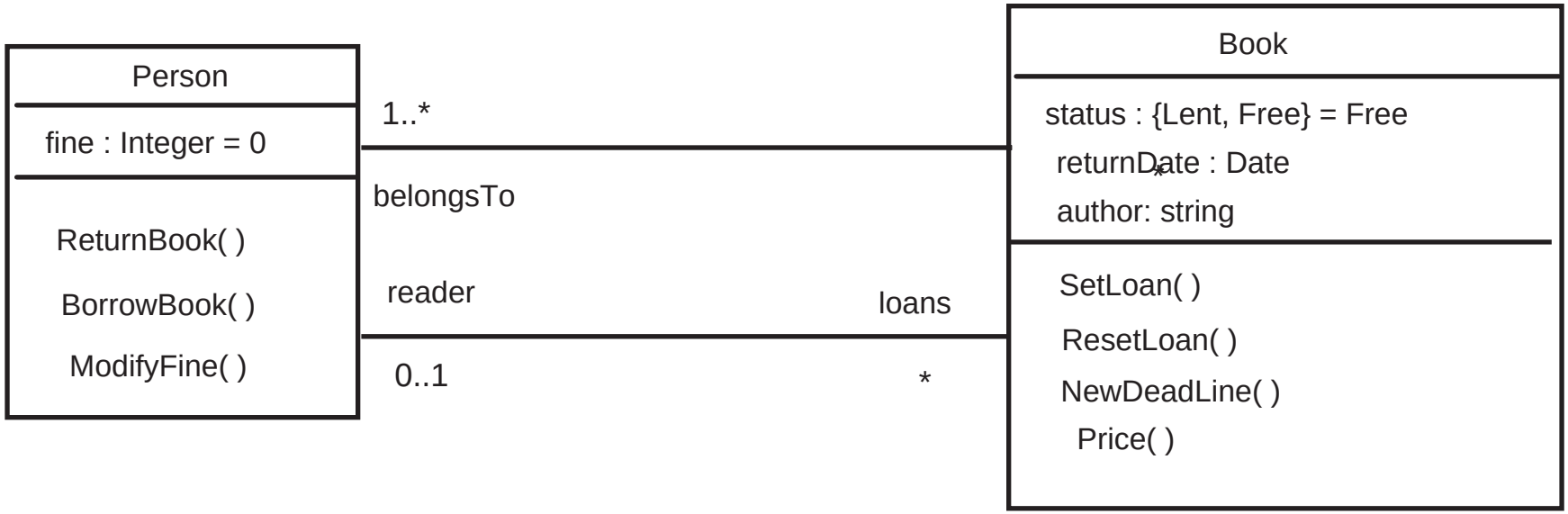
```
class      ClassName
  var      x
  init     x := x_0
  proc     p
end
```

(a) Syntax of Class

```
contract   ContractName
  agent    a
  var      y
  proc     q
begin
  S
end
```

(b) Syntax of Contract

Class diagram for the system



Book class

class $Book$
  var $belongsTo$: set of $Person$; $returnDate : Date$; $author : String$;
    $status : \{Lent,\ Free\}$; $reader : Person \cup \{\text{None}\}$
  init $status, reader := Free$, None
  proc $SetLoan$(val $person : Person$) :
    pre $status = Free$;
    $reader, status, returnDate := person, Lent, Today() + 4\ weeks$
  proc $ResetLoan$:
    pre $status = Lent$;
    $reader, status :=$ None, $Free$
  proc $NewDeadLine$:
    $returnDate := Today() + 4\ weeks$
  proc $Price$(var $sum : Natural$) :
    if $returnDate \geq Today()$ then $sum := 0$
    else $sum := (Today() - returnDate) * n$ fi
end

Person class

class $Person$
  var $loans$: set of $Book$; $\boldsymbol{fine} : Natural$
  init $loans, \boldsymbol{fine} := \emptyset, 0$
  proc $BorrowBook$(val $book : Book$) :
    pre $book \notin loans$;
    $loans := loans \cup \{book\}$
  proc $ReturnBook$(val $book : Book$) :
    pre $book \in loans$;
    $loans := loans \setminus \{book\}$
  proc $\boldsymbol{ModifyFine}$(val $\boldsymbol{diff} : Integer$) :
    pre $\boldsymbol{fine} + \boldsymbol{diff} \geq 0$;
    $\boldsymbol{fine} := \boldsymbol{fine} + \boldsymbol{diff}$
end

The contract

contract $Private\ Library$
  agent M, L
  var $book : Book, person : Person, amount : Natural;$
  proc $BorrowBook :$ // *use case* **Borrow Book**
    $[book := b' \mid b' \in Book \wedge b'.status = Free \wedge person \notin b'.belongsTo]_M$;
    if $person.\boldsymbol{fine} \neq 0$ then $PayFee_M$ fi ;
    if $person.\boldsymbol{fine} = 0$ then
      $book.SetLoan(person)_L$; $person.BorrowBook(book)_M$ fi
  proc $RenewLoan :$ // *use case* **Renew Loan**
    $[book := b' \mid b' \in Book \wedge b' \in person.loans]_M$;
    if $person.\boldsymbol{fine} \neq 0$ then $PayFee_M$ fi ;
    if $person.\boldsymbol{fine} = 0$ then $book.NewDeadLine()_L$ fi
  proc $ReturnBook :$ // *use case* **Return Book**
    $[book := b' \mid b' \in Book \wedge b' \in person.loans]_M$;
    $book.Price(amount)$;
    $person.\boldsymbol{ModifyFine}(amount)_L$;
    if $person.\boldsymbol{fine} \neq 0$ then $PayFee_M$ fi ;
    $person.ReturnBook(book)_M$;
    $book.ResetLoan()_L$

The contract, cont.

$$
\begin{array}{l}
\quad \text{proc } PayFee: \qquad\qquad // \ use\ case\ \textbf{Pay Fee} \\
\qquad \{person.\textbf{\textit{fine}} \neq 0\}_M; \\
\qquad [amount := pay \mid 0 \leq pay \leq person.\textbf{\textit{fine}}]_M; \\
\qquad person.\textbf{\textit{ModifyFine}}(-amount)_L \\
\text{begin} \\
\quad [person := p' \mid p' \in Person]_M \ ; \\
\quad BorrowBook \sqcup_M RenewLoan \sqcup_M ReturnBook \sqcup_M PayFee \\
\text{end}
\end{array}
$$

Analyzing the contract

We analyse the conditions in which a member of the library can borrow a book written by a certain author.

The agent M chooses one identity ($person$) and invokes the use case *BorrowBook*.

The agent can successfully borrow a book written, say, by Steven King, for the chosen person, if

$$\exists b \in Book \cdot b.author = \texttt{StevenKing} \wedge b \in person.loans$$

holds after invoking the use case.

Computing the precondition

We can determine the weakest precondition to achieve this goal by computing:

$$wp_M(BorrowBook_M, (\exists b \in Book \cdot b.author = \texttt{StevenKing} \wedge b \in person.loans))$$

If we obtain a predicate different from false**, then we have shown that the *Member* can borrow a new book authored by Steven King.**

At the same time, we compute the conditions under which this is possible.

Result

Applying the rules of the weakest predicate transformer given above, we get:

$$
\begin{aligned}
& wp_M(BorrowBook_M, \\
& \qquad (\exists b \in Book \cdot b.author = \texttt{StevenKing} \wedge b \in person.loans)) \\
= & \\
& (\exists b' \in Book \cdot b'.status = Free \wedge person \notin b'.belongsTo \ \wedge \\
& \qquad\qquad b'.author = \texttt{StevenKing}) \\
& \vee \\
& (\exists b \in Book \cdot b.author = \texttt{StevenKing} \wedge b \in person.loans) \ \wedge \\
& (\exists b' \in Book \cdot b'.status = Free \wedge person \notin b'.belongsTo)
\end{aligned}
$$

Interpretation

We can interpret this result in the following way:

There is a free book written by Steven King in the library, which is not owned by the person.

Alternatively, the person already has borrowed a book by Steven King. In this case, there should also exist at least one free book not belonging to the person.

Unexpected result

This latter conjunction is unexpected and was not intended. It reveals an error in our design of the use case.

Here the error is easy to detect: the *Member* is forced to choose a book even in the case that he already has the desired book.

Improved design

An improved design of the use case, which avoids this error, is given below:

proc $BorrowBook$: // *use case* **Borrow Book**
  if $person.\boldsymbol{fine} \neq 0$ then $PayFee_M$ fi ;
  $(\{person.fine = 0\}_M ;$
   $[book := b' \mid b' \in Book \wedge b'.status = Free \wedge person \notin b'.belongsTo]_M$;
   $book.SetLoan(person)_L$; $person.BorrowBook(book)_M$
)
  $\sqcup_M$
  $skip$

Revised result

$$
\begin{array}{l}
precondition = \\
\\
either \\
\quad \exists book \in Book \bullet \\
\qquad book.status = Free \\
\qquad book.author = \texttt{Steven King} \\
\qquad person \notin book.belongsTo \\
or \\
\quad \exists b \in Book \bullet \\
\qquad b.author = \texttt{Steven King} \\
\qquad b \in person.loans
\end{array}
$$

Use a clearer format for logical formula, necessary when the formulas become more complicated.

Member and librarian co-operate

We can check the weakest precondition for a member to borrow a book by Steven King, when the librarian co-operates with the member, i.e., we choose both the member and the librarian to be angels.

It turns out that this gives the same precondition. In other words, from the members point of view, it does not matter if the librarian co-operates or not.

The librarians viewpoint

Let us finally check the weakest precondition for reaching the same goal as above, with the same contract, but now from the *Librarian*'s perspective.

That means that we compute the conditions in which a book written by a certain author can be borrowed to a person, represented by the *Member*, when the latter might be against this action.

The result

$$
\begin{array}{l}
precondition \\
= \\
person.fine = 0 \\
\textbf{\textit{if}} \\
\quad \neg(\exists b \in Book \bullet \\
\qquad b.author = \text{"}StevenKing\text{"} \\
\qquad b \in person.loans) \\
then \\
\quad \forall book \in Book \bullet \\
\qquad \textbf{\textit{if}} \\
\qquad\quad book.status = Free \\
\qquad\quad person \notin book.belongsTo \\
\qquad then \\
\qquad\quad book.author = \text{"}StevenKing\text{"}
\end{array}
$$

Interpreting the result

The result can be interpreted as follows:

The person should have no debts to the library and moreover, when a book written by Steven King is not already lent to the person represented by the *Member*, then all the free books not owned by the person should be written by Steven King.

This is a substantially stronger condition compared to the precondition computed from the *Member*'s perspective, because the *Librarian* cannot influence the payment of the person's fees and the choice of a book. He/she has therefore to assume beforehand that there are no such debts and that all the possible choices for the book are correct.