COMP1216 - CW2 - Online Library System using Event B modelling

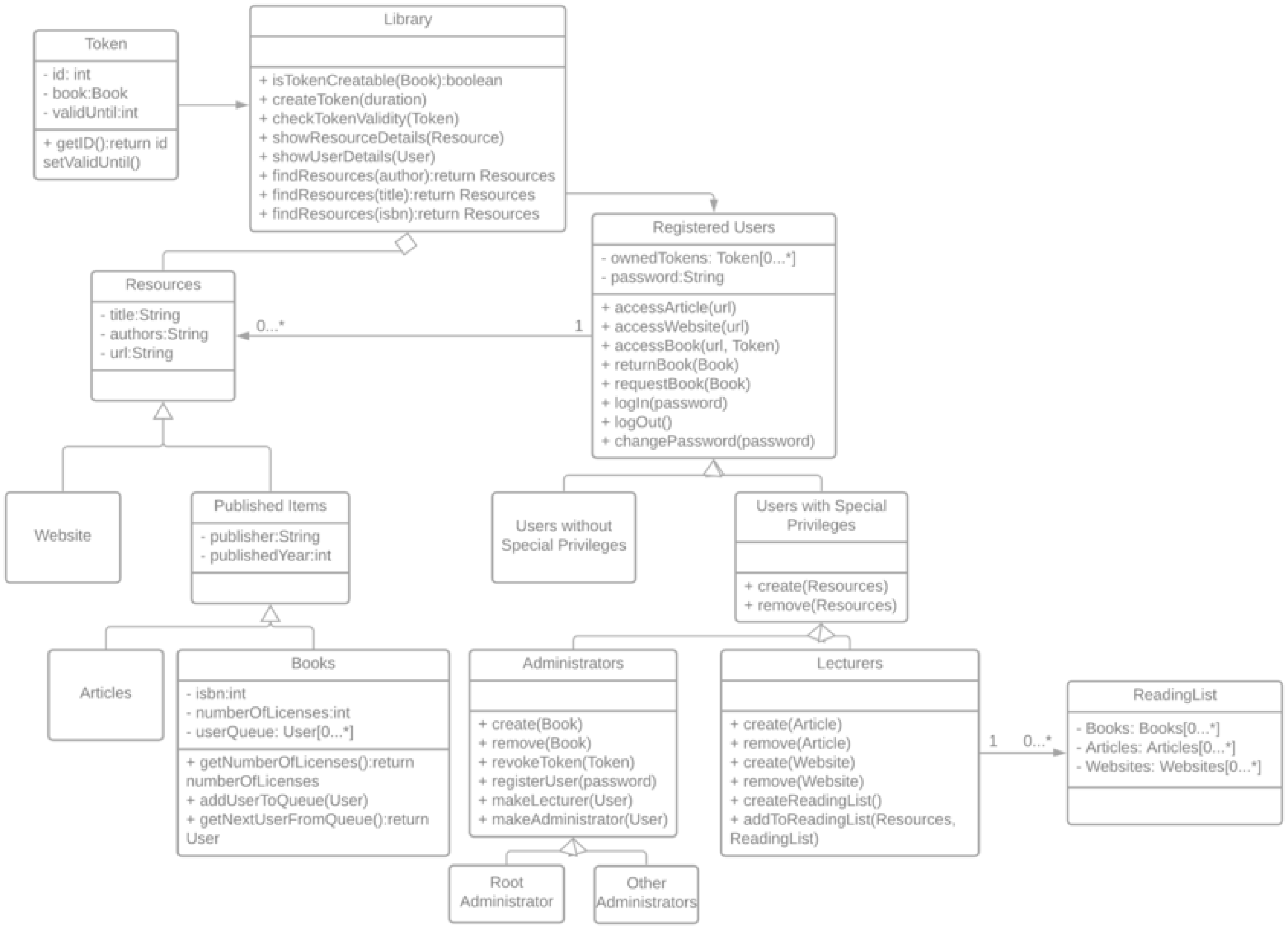
Patrik-Tibor Csanyi, Sandor Kovacs, Charlie Williams,
Zachariah Ridzuan-Allen
ptc1g20(ID: 31574602), sk10g20(ID: 31773478), cw2g19(ID: 431023274),
zra1u19(ID: 30870186)

Group 20

1 Introduction

The purpose of this report is to present the formal modelling of an online library from Coursework 1. The tasks were split among us based on the difficulty and effort required but we helped each other to reduce the repetition of invariants and variables. The first 6 requirements were done my Patrik-Tibor Csanyi(ptc1g20) and Sandor Kovacs(sk10g20) as well as the last 6 requirements. The remaining part, requirements from 7 to 16 were done by Zachariah Ridzuan-Allen(zra1u19), Charles Williams(cw2g19), Patrik-Tibor Csanyi(ptc1g20) and Sandor Kovacs(sk10g20) as these parts required the most effort. Lastly the class diagram was done by Patrik-Tibor Csanyi, and the report was written by Sandor Kovacs.

2 Diagram



3 Users

3.1 Context

```
context OnlineLibrarySystemContext
sets
USER
PASSWORD
constants
rootUser
defaultPassword
```

```
axioms
 @rootUser: rootUser ∈ USER
 @DEFAULT_PASSWORD−def: defaultPassword ∈ PASSWORD
end
```

In the context we determine the sets we are going to use as well as we set some axioms. These axioms help us to set a rootUser and a defaultPassword.

3.2 Code

```
machine OnlineLibrarySystem
sees OnlineLibrarySystemContext

/∗Declaring the variables we are going to use ∗/
variables
  registeredUsers//Set of registered users
  administrators//Set of administrators
  lecturers
  loggedInUsers
  loggingPassword

/∗Handling the invariants ∗/
invariants
  @inv1: registeredUsers ⊆ USER
  @inv2: administrators ⊆ registeredUsers
  @inv3: lecturers ⊆ registeredUsers
  @inv4: administrators ∩ lecturers = ∅
  @inv5: rootUser ∈ administrators
  @inv6: loggingPassword ∈ registeredUsers → PASSWORD
  @inv7: loggedInUsers ⊆ registeredUsers
events

/∗Initializing our sets ∗/
event INITIALISATION
begin
  @beg1:registeredUsers := ∅ ∪ {rootUser}
  @beg2:administrators := ∅ ∪ {rootUser}
  @beg3:lecturers := ∅
  @beg4:loggedInUsers := ∅
  @beg5:loggingPassword := ∅ ∪ {rootUser ↦ defaultPassword}
end

/∗The event to log in a user ∗/
event LogIn
any u p where
  @grd1:u ∈ registeredUsers//Check if u is a registered user
  @grd2: p ∈ PASSWORD //Check if p is the set of PASSWORD
  @grd3:loggingPassword(u) = p //Check if the user and its password are correct
then
  @act1:loggedInUsers := loggedInUsers ∪ {u}//Add the user to the logged in user set
end

/∗The event to log out ∗/
event LogOut
```

```
any u where
  @grd1:u ∈ loggedInUsers//Check if u is logged in
then
  @act1:loggedInUsers := loggedInUsers \ {u}//Remove u from the logged in users
end

/*The even to register a new user */
event RegisterUser
any a u p where
  @grd1: u ∉ registeredUsers//Check if u is not part of the registeredUsers
  @grd2: u ∈ USER//Check if u is part of USER
  @grd3: a ∈ administrators//Check if a is administrator to allow registering a user
  @grd4: p ∈ PASSWORD//Check is p is part of PASSWORD
then
  /*Register the new user u and the password to this user */
  @act1: registeredUsers := registeredUsers ∪ {u}
  @act3: loggingPassword(u) := p
end

/*The event to change the password */
event ChangePassword
any u p where
  @grd1: u ∈ loggedInUsers//Check if u is a logged in user
  @grd2: p ∈ PASSWORD//Check if p is part of PASSWORD
then
  @act1: loggingPassword(u) := p//Change the password to p
end

/*The event to change to administrator */
event ChangeToAdministrator
 any a u where
  @grd1: a ∈ administrators//Check if a is and administrator
  @grd2: u ∈ registeredUsers \ (lecturers ∪ administrators)//Check if u is part of
       registered users who are not lecturers or administrators
 then
  @act1: administrators := administrators ∪ {u}//Add u to the administrators
 end

/*The event to change to lecturer */
event ChangeToLecturer
 any a u where
  @grd1: a ∈ administrators//Check if a is an administrator
  @grd2: u ∈ registeredUsers \ (administrators ∪ lecturers)//Check if u is a registered
       user but not an administrator and not a lecturer
 then
  @act1: lecturers := lecturers ∪ {u}//Add u to the lecturers
 end

end
```

3.3 Examining the code

Firstly, we declare the variables we are going to use :

```
variables
  registeredUsers//Set of registered users
  administrators//Set of administrators
  lecturers
  loggedInUsers
  loggingPassword
```

After this we are going to handle the invariants:

```
/*Handling the invariants */
invariants
  @inv1: registeredUsers ⊆ USER
  @inv2: administrators ⊆ registeredUsers
  @inv3: lecturers ⊆ registeredUsers
  @inv4: administrators ∩ lecturers = ∅
  @inv5: rootUser ∈ administrators
  @inv6: loggingPassword ∈ registeredUsers → PASSWORD
  @inv7: loggedInUsers ⊆ registeredUsers
```

After declaring the variables and invariants we are going to begin our events. Firstly, we write the INITIALISATION event which will let us initialise all of our variables/

```
events

/*Initializing our sets */
event INITIALISATION
begin
  @beg1:registeredUsers := ∅ ∪ {rootUser}
  @beg2:administrators := ∅ ∪ {rootUser}
  @beg3:lecturers := ∅
  @beg4:loggedInUsers := ∅
  @beg5:loggingPassword := ∅ ∪ {rootUser ↦ defaultPassword}
end
```

After the initialisation we do the log in event to allow a user to log in to the the online library. In order to log in there has to be a registered user and to enter his password correctly. If all of these were met than we add the user to the logged in users set.

```
/*The event to log in a user */
event LogIn
any u p where
  @grd1:u ∈ registeredUsers//Check if u is a registered user
  @grd2: p ∈ PASSWORD //Check if p is the set of PASSWORD
  @grd3:loggingPassword(u) = p //Check if the user and its password are correct
then
  @act1:loggedInUsers := loggedInUsers ∪ {u}//Add the user to the logged in user set
end

```

```
/*The event to log out */
```

We similarly write the log out method which allows a logged in user to log out of the system.

```
/*The event to log out */
event LogOut
any u where
  @grd1:u ∈ loggedInUsers//Check if u is logged in
then
  @act1:loggedInUsers := loggedInUsers \ {u}//Remove u from the logged in users
end
```

The following event is to allow an administrator to register a new user, only administrators can register new users with a new password.

```
/*The even to register a new user */
event RegisterUser
any a u p where
  @grd1: u ∉ registeredUsers//Check if u is not part of the registeredUsers
  @grd2: u ∈ USER//Check if u is part of USER
  @grd3: a ∈ administrators//Check if a is administrator to allow registering a user
  @grd4: p ∈ PASSWORD//Check is p is part of PASSWORD
then
  /*Register the new user u and the password to this user */
  @act1: registeredUsers := registeredUsers ∪ {u}
  @act3: loggingPassword(u) := p
end
```

A user can change its password if he is logged in to the system.

```
/*The event to change the password */
event ChangePassword
any u p where
  @grd1: u ∈ loggedInUsers//Check if u is a logged in user
  @grd2: p ∈ PASSWORD//Check if p is part of PASSWORD
then
  @act1: loggingPassword(u) := p//Change the password to p
end
```

An administrator can change a users status to administrator as well as to lecturer. A user is not allowed to be an administrator and lecturer at the same time.

```
/*The event to change to administrator */
event ChangeToAdministrator
 any a u where
  @grd1: a ∈ administrators//Check if a is and administrator
  @grd2: u ∈ registeredUsers \ (lecturers ∪ administrators)//Check if u is part of
      registered users who are not lecturers or administrators
 then
  @act1: administrators := administrators ∪ {u}//Add u to the administrators
 end

/*The event to change to lecturer */
event ChangeToLecturer
 any a u where
  @grd1: a ∈ administrators//Check if a is an administrator
  @grd2: u ∈ registeredUsers \ (administrators ∪ lecturers)//Check if u is a registered
      user but not an administrator and not a lecturer
 then
  @act1: lecturers := lecturers ∪ {u}//Add u to the lecturers
 end

end
```

4 Resources

4.1 Context

```
context ResourceContext
sets
RESOURCE
TITLES
URLS
AUTHORS
ISBN
PUBLISHER
PUBLISHED_YEAR
end
```

In this context we determine the sets that we are going to use.

4.2 The code

```
machine OnlineLibrarySystem1
refines OnlineLibrarySystem
sees OnlineLibrarySystemContext ResourceContext

/*Declaring the variables we are going to use */
variables
 registeredUsers
 administrators
 lecturers
 loggedInUsers
 loggingPassword
 resource
 book
 article
 website
 authors
```

```
urls
titles
getAuthor
getURL
getTitle
publisher
publishedYear
isbn
getPublisher
getPublishedYear
getISBN

/*Declaring the invariants we are going to use */
invariants
  @inv8:resource ⊆ RESOURCE
 @inv9:partition(resource, book, website, article)
 @inv10:authors ⊆ AUTHORS
 @inv11:titles ⊆ TITLES
 @inv12:urls ⊆ URLS
 @inv13:getAuthor ∈ (resource ∪ book ∪ website ∪ article) ↔ authors // we need to
      change this one
 @inv14:getURL ∈ (resource) ⤖ urls
 @inv15:getTitle ∈ (resource) ⤖ titles
 @inv16:isbn ⊆ ISBN
 @inv17:publisher ⊆ PUBLISHER
 @inv18:publishedYear ⊆ PUBLISHED_YEAR
 @inv19:getPublisher ∈ ((resource) \ (website)) → publisher
 @inv20:getISBN ∈ book ⤖ isbn
 @inv21:getPublishedYear ∈ ((resource) \ (website)) → publishedYear

events
/* We initialize everything what we are going to use (our variables) */
 event INITIALISATION extends INITIALISATION
 begin
 @beg6:resource := ∅
 @beg7:book := ∅
 @beg8:article := ∅
 @beg9:website := ∅
 @beg10:authors := ∅
 @beg11:urls := ∅
 @beg12:titles := ∅
 @beg13:getAuthor := ∅
 @beg14:getURL := ∅
 @beg15:getTitle := ∅
 @beg16:publisher := ∅
 @beg17:publishedYear := ∅
 @beg18:isbn := ∅
 @beg19:getPublisher := ∅
 @beg20:getPublishedYear := ∅
 @beg21:getISBN := ∅
 end

 event LogIn extends LogIn
 end

 event LogOut extends LogOut
```

```
end

event RegisterUser extends RegisterUser
end

event ChangePassword extends ChangePassword
end

event ChangeToAdministrator extends ChangeToAdministrator
end

event ChangeToLecturer extends ChangeToLecturer
end

/∗ The event to create a new book, this event can only be done by a logged in
     administrator ∗/
event CreateBook
 any a b publish publishedY tit ur is auth where
  @grd1:a ∈ administrators // Check if a is an administrator
  @grd2:a ∈ loggedInUsers // Check if a is logged in
  @grd3:b ∈ RESOURCE // Check if b is part of the RESOURCE set
  @grd4:b ∉ (book ∪ resource) // Check if b is not part of the resource ∪ book
  /∗ We check if is, publishedY, publish, auth, ur, tit is part or not part of the
     appropriate sets ∗/
  @grd5:is ∈ ISBN
  @grd6: is ∉ isbn
  @grd7:publishedY ∈ PUBLISHED_YEAR
  @grd8:publish ∈ PUBLISHER
  @grd9:auth ∈ AUTHORS
  @grd10:ur ∈ URLS
  @grd11: ur ∉ urls
  @grd12:tit ∈ TITLES
  @grd13: tit ∉ titles
 then
 /∗ We add b to book and resources ∗/
   @act1:resource := resource ∪ {b}
  @act2:book := book ∪ {b}
  @act3:isbn := isbn ∪ {is}
  @act4:publishedYear := publishedYear ∪ {publishedY}
  @act5:publisher := publisher ∪ {publish}
  @act6:authors := authors ∪ {auth}
  @act7:urls := urls ∪ {ur}
  @act8:titles := titles ∪ {tit}
  @act9: getISBN(b) := is
  @act10: getPublishedYear(b) := publishedY
  @act11: getPublisher(b) := publish
  @act12: getAuthor(b) := auth
  @act13: getURL(b) := ur
  @act14: getTitle(b) := tit
 end

 /∗ The event to remove a book, can only be done by a logged in administrator ∗/
 event RemoveBook
 any b a where
   @grd1: a ∈ administrators // Check if a is administrator
   @grd2: a ∈ loggedInUsers // Check if a is logged in
```

```
    @grd3: b ∈ (resource ∩ book) // Check if the b (book) is not part of the resource ∩
      book
  then
  /* We remove everything that is connected to b from everywhere */
    @act1: book := book \ {b}
    @act2: resource := resource \ {b}
    @act3: isbn := isbn \ {getISBN(b)}
    @act4: titles := titles \ {getTitle(b)}
    @act5: urls := urls \ {getURL(b)}
    @act6:getISBN := {b} ⩤ getISBN
    @act7:getTitle := {b} ⩤ getTitle
    @act8:getAuthor := {b} ⩤ getAuthor
    @act9:getPublisher := {b} ⩤ getPublisher
    @act10:getPublishedYear := {b} ⩤ getPublishedYear
    @act11:getURL := {b} ⩤ getURL
  end

/*The event to  create a new website which can only be done by a logged in lecturer*/
event CreateWebsite
  any l w auth ur tit where
   @grd1: l ∈ (lecturers ∩ loggedInUsers) // Check if the l is a lecturer and if it is logged
      in
   @grd2: w ∉ (resource ∪ website) // Check if w is not part of our resource ∪ website
   @grd3: w ∈ RESOURCE // Check if w is part of RESOURCE
   @grd4: auth ∈ AUTHORS //Check if auth is part of the AUTHORS
    @grd5: ur ∈ URLS // Check if ur is part of URLS
    @grd6: ur ∉ urls // Check if ur is not part of our urls
    @grd7: tit ∈ TITLES // Check if tit is part of TITLE
    @grd8: tit ∉ titles// Check if tit is not part of our titles
   then
   /*If all the grds are true we add the website with the author title and url */
    @act1: resource := resource ∪ {w}
    @act2: website := website ∪ {w}
    @act3: authors := authors ∪ {auth}
    @act4: urls := urls ∪ {ur}
    @act5: titles := titles ∪ {tit}
    @act6: getAuthor(w) := auth
    @act7: getURL(w) := ur
    @act8: getTitle(w) := tit
   end

  /*The event to create a new article which can only be done by a logged in lecturer */
  event CreateArticle
   any a l auth ur tit publish publishedY where
    @grd1: l ∈ (lecturers ∩ loggedInUsers) //Check if l is a lecturer and if it is logged in
    @grd2: a ∉ (resource ∪ article) // Check if a is not part of the resource ∪ article
    @grd3: a ∈ RESOURCE // Check if a is part of RESOURCE
    @grd4: auth ∈ AUTHORS// Check is auth is part of AUTHORS
    @grd5: ur ∈ URLS //Check if ur is part of the URLS
    @grd6: ur ∉ urls // Check if ur is not part of our urls
    @grd7: tit ∈ TITLES //Check if tit is part of all TITLES
    @grd8: tit ∉ titles// Check if tit is not part of our titles
    @grd9:publishedY ∈ PUBLISHED_YEAR //Check if publishedY is part of
      PUBLISHED_YEAR
    @grd10:publish ∈ PUBLISHER//Check if publish is part of PUBLISHER
   then
  /∗We are going to add the article to the resource and article and we add all properties
      related to this article ∗/
    @act1: resource := resource ∪ {a}
    @act2: article := article ∪ {a}
    @act3: authors := authors ∪ {auth}
    @act4: urls := urls ∪ {ur}
    @act5: titles := titles ∪ {tit}
    @act6: getAuthor(a) := auth
    @act7: getURL(a) := ur
    @act8: getTitle(a) := tit
    @act9:publishedYear := publishedYear ∪ {publishedY}
    @act10:publisher := publisher ∪ {publish}
    @act11: getPublishedYear(a) := publishedY
    @act12: getPublisher(a) := publish
  end

 /∗The event to remove an existing website ∗/
 event RemoveWebsite
  any w l where
    @grd1: l ∈ (lecturers ∩ loggedInUsers)//Check if the user who wants to remove it is
       logged in and a lecturer as well
    @grd2: w ∈ (resource ∩ website)//Check if w is part or resource ∩ website
  then
  /∗We remove everything that is related to website w ∗/
    @act1: website := website \ {w}
    @act2: resource := resource \ {w}
     @act3: titles := titles \ {getTitle(w)}
     @act4: urls := urls \ {getURL(w)}
     @act5:getTitle := {w} ⩤ getTitle
     @act6:getAuthor := {w} ⩤ getAuthor
     @act7:getURL := {w} ⩤ getURL
 end

 /∗The event to remove an existing article ∗/
 event RemoveArticle
  any a l where
    @grd1: l ∈ (lecturers ∩ loggedInUsers)//Check if the user l trying to remove the
       article is a leturer and if it is logged in
    @grd2: a ∈ (resource ∩ article)//Check if a is part of resource ∩ article
  then
  /∗We remove everything related to article a ∗/
    @act1: article := article \ {a}
    @act2: resource := resource \ {a}
     @act3: titles := titles \ {getTitle(a)}
     @act4: urls := urls \ {getURL(a)}
     @act10:getTitle := {a} ⩤ getTitle
     @act6:getAuthor := {a} ⩤ getAuthor
     @act7:getURL := {a} ⩤ getURL
     @act8:getPublisher := {a} ⩤ getPublisher
     @act9:getPublishedYear := {a} ⩤ getPublishedYear
    end

/∗The event to search by a title ∗/
 event SearchByTitle
  any r t u where
     @grd1:r ∈ resource
```

```
      @grd2:t ∈ titles
      @grd3: u ∈ loggedInUsers
      @grd4:getTitle(r) = t
  end

  /*The event to search by ISBN */
  event SearchByISBN
   any b i u where
      @grd1:b ∈ book
      @grd2:i ∈ isbn
      @grd3: u ∈ loggedInUsers
      @grd4:getISBN(b) = i
  end

end
```

4.3 Examining the code

This machine is refined from the OnlineLibrarySystem. First we determine the variables that we are going to use and we handle the invariants as well, and we allow the machine to see the context.

```
machine OnlineLibrarySystem1
refines OnlineLibrarySystem
sees OnlineLibrarySystemContext ResourceContext

/*Declaring the variables we are going to use */
variables
 registeredUsers
 administrators
 lecturers
 loggedInUsers
 loggingPassword
 resource
 book
 article
 website
 authors
 urls
 titles
 getAuthor
 getURL
 getTitle
 publisher
 publishedYear
 isbn
 getPublisher
 getPublishedYear
 getISBN

 /*Declaring the invariants we are going to use */
invariants
  @inv8:resource ⊆ RESOURCE
```

```
@inv9:partition(resource, book, website, article)
@inv10:authors ⊆ AUTHORS
@inv11:titles ⊆ TITLES
@inv12:urls ⊆ URLS
@inv13:getAuthor ∈ (resource ∪ book ∪ website ∪ article) ↔ authors // we need to
     change this one
@inv14:getURL ∈ (resource) ⤖ urls
@inv15:getTitle ∈ (resource) ⤖ titles
@inv16:isbn ⊆ ISBN
@inv17:publisher ⊆ PUBLISHER
@inv18:publishedYear ⊆ PUBLISHED_YEAR
@inv19:getPublisher ∈ ((resource) \ (website)) → publisher
@inv20:getISBN ∈ book ⤖ isbn
@inv21:getPublishedYear ∈ ((resource) \ (website)) → publishedYear

```

After handling the invariants we begin our events. The first one is the INITIALISATION, we initialise every set and variables that we are going to use.

```

events
/* We initialize everything what we are going to use (our variables) */
 event INITIALISATION extends INITIALISATION
 begin
 @beg6:resource := ∅
 @beg7:book := ∅
 @beg8:article := ∅
 @beg9:website := ∅
 @beg10:authors := ∅
 @beg11:urls := ∅
 @beg12:titles := ∅
 @beg13:getAuthor := ∅
 @beg14:getURL := ∅
 @beg15:getTitle := ∅
 @beg16:publisher := ∅
 @beg17:publishedYear := ∅
 @beg18:isbn := ∅
 @beg19:getPublisher := ∅
 @beg20:getPublishedYear := ∅
 @beg21:getISBN := ∅
 end
```

After the initialisation we extend the already written events from the OnlineLibrarySystem.

```

 event LogIn extends LogIn
 end

 event LogOut extends LogOut
 end

```

```
event RegisterUser extends RegisterUser
end

event ChangePassword extends ChangePassword
end

event ChangeToAdministrator extends ChangeToAdministrator
end

event ChangeToLecturer extends ChangeToLecturer
end

```

The first event we write in this machine is the CreateBook event which allows an administrator to create a new book. We need to check if the title, authors published year, publisher are valid. Thank we add them to the relevant functions and sets.

```
/∗ The event to create a new book, this event can only be done by a logged in
     administrator ∗/
event CreateBook
 any a b publish publishedY tit ur is auth where
  @grd1:a ∈ administrators // Check if a is an administrator
  @grd2:a ∈ loggedInUsers // Check if a is logged in
  @grd3:b ∈ RESOURCE // Check if b is part of the RESOURCE set
  @grd4:b ∉ (book ∪ resource) // Check if b is not part of the resource ∪ book
  /∗ We check if is, publishedY, publish, auth, ur, tit is part or not part of the
     appropriate sets ∗/
  @grd5:is ∈ ISBN
  @grd6: is ∉ isbn
  @grd7:publishedY ∈ PUBLISHED_YEAR
  @grd8:publish ∈ PUBLISHER
  @grd9:auth ∈ AUTHORS
  @grd10:ur ∈ URLS
  @grd11: ur ∉ urls
  @grd12:tit ∈ TITLES
  @grd13: tit ∉ titles
 then
 /∗ We add b to book and resources ∗/
   @act1:resource := resource ∪ {b}
  @act2:book := book ∪ {b}
  @act3:isbn := isbn ∪ {is}
  @act4:publishedYear := publishedYear ∪ {publishedY}
  @act5:publisher := publisher ∪ {publish}
  @act6:authors := authors ∪ {auth}
  @act7:urls := urls ∪ {ur}
  @act8:titles := titles ∪ {tit}
  @act9: getISBN(b) := is
  @act10: getPublishedYear(b) := publishedY
  @act11: getPublisher(b) := publish
  @act12: getAuthor(b) := auth
  @act13: getURL(b) := ur
  @act14: getTitle(b) := tit
 end
```

As we have written the creation of a book we need to write the removal of one as well. Only an administrator can remove a book and he needs to be logged in. We remove every function and relation that is related to this book.

```

/∗ The event to remove a book, can only be done by a logged in administrator ∗/
event RemoveBook
any b a where
  @grd1: a ∈ administrators // Check if a is administrator
  @grd2: a ∈ loggedInUsers // Check if a is logged in
  @grd3: b ∈ (resource ∩ book) // Check if the b (book) is not part of the resource ∩
    book
then
/∗ We remove everything that is connected to b from everywhere ∗/
  @act1: book := book \ {b}
  @act2: resource := resource \ {b}
  @act3: isbn := isbn \ {getISBN(b)}
  @act4: titles := titles \ {getTitle(b)}
  @act5: urls := urls \ {getURL(b)}
  @act6:getISBN := {b} ⩤ getISBN
  @act7:getTitle := {b} ⩤ getTitle
  @act8:getAuthor := {b} ⩤ getAuthor
  @act9:getPublisher := {b} ⩤ getPublisher
  @act10:getPublishedYear := {b} ⩤ getPublishedYear
  @act11:getURL := {b} ⩤ getURL
end
```

The next event is creating a website. Lecturers can create websites and a website needs a valid author url and title.

```
/∗The event to  create a new website which can only be done by a logged in lecturer∗/
event CreateWebsite
 any l w auth ur tit where
  @grd1: l ∈ (lecturers ∩ loggedInUsers) // Check if the l is a lecturer and if it is logged
     in
  @grd2: w ∉ (resource ∪ website) // Check if w is not part of our resource ∪ website
  @grd3: w ∈ RESOURCE // Check if w is part of RESOURCE
  @grd4: auth ∈ AUTHORS //Check if auth is part of the AUTHORS
   @grd5: ur ∈ URLS // Check if ur is part of URLS
   @grd6: ur ∉ urls // Check if ur is not part of our urls
   @grd7: tit ∈ TITLES // Check if tit is part of TITLE
   @grd8: tit ∉ titles// Check if tit is not part of our titles
  then
  /∗If all the grds are true we add the website with the author title and url ∗/
   @act1: resource := resource ∪ {w}
   @act2: website := website ∪ {w}
   @act3: authors := authors ∪ {auth}
   @act4: urls := urls ∪ {ur}
   @act5: titles := titles ∪ {tit}
   @act6: getAuthor(w) := auth
   @act7: getURL(w) := ur
   @act8: getTitle(w) := tit
 end
```

Now we need to create an article as well. Just in the case of creating a website only lecturers are allowed to create and besides the valid author, url and title it also needs a valid publisher and published year.

```
/*The event to create a new article which can only be done by a logged in lecturer */
event CreateArticle
 any a l auth ur tit publish publishedY where
  @grd1: l ∈ (lecturers ∩ loggedInUsers) //Check if l is a lecturer and if it is logged in
  @grd2: a ∉ (resource ∪ article) // Check if a is not part of the resource ∪ article
  @grd3: a ∈ RESOURCE // Check if a is part of RESOURCE
  @grd4: auth ∈ AUTHORS// Check is auth is part of AUTHORS
  @grd5: ur ∈ URLS //Check if ur is part of the URLS
  @grd6: ur ∉ urls // Check if ur is not part of our urls
  @grd7: tit ∈ TITLES //Check if tit is part of all TITLES
  @grd8: tit ∉ titles// Check if tit is not part of our titles
  @grd9:publishedY ∈ PUBLISHED_YEAR //Check if publishedY is part of
    PUBLISHED_YEAR
  @grd10:publish ∈ PUBLISHER//Check if publish is part of PUBLISHER
 then
 /*We are going to add the article to the resource and article and we add all properties
     related to this article */
  @act1: resource := resource ∪ {a}
  @act2: article := article ∪ {a}
  @act3: authors := authors ∪ {auth}
  @act4: urls := urls ∪ {ur}
  @act5: titles := titles ∪ {tit}
  @act6: getAuthor(a) := auth
  @act7: getURL(a) := ur
  @act8: getTitle(a) := tit
  @act9:publishedYear := publishedYear ∪ {publishedY}
  @act10:publisher := publisher ∪ {publish}
  @act11: getPublishedYear(a) := publishedY
  @act12: getPublisher(a) := publish
 end
```

The next event is responsible for removing a website. Logged in lecturers can remove websites, and it deletes all of the related information to it.

```
/*The event to remove an existing website */
event RemoveWebsite
 any w l where
  @grd1: l ∈ (lecturers ∩ loggedInUsers)//Check if the user who wants to remove it is
    logged in and a lecturer as well
  @grd2: w ∈ (resource ∩ website)//Check if w is part or resource ∩ website
 then
 /*We remove everything that is related to website w */
  @act1: website := website \ {w}
  @act2: resource := resource \ {w}
   @act3: titles := titles \ {getTitle(w)}
   @act4: urls := urls \ {getURL(w)}
```

```
    @act5:getTitle := {w} ⩤ getTitle
    @act6:getAuthor := {w} ⩤ getAuthor
    @act7:getURL := {w} ⩤ getURL
end
```

Same as in the case of removing a website only logged in lecturers can remove an article.

```
/*The event to remove an existing article */
event RemoveArticle
 any a l where
  @grd1: l ∈ (lecturers ∩ loggedInUsers)//Check if the user l trying to remove the
     article is a leturer and if it is logged in
  @grd2: a ∈ (resource ∩ article)//Check if a is part of resource ∩ article
 then
 /*We remove everything related to article a */
  @act1: article := article \ {a}
  @act2: resource := resource \ {a}
    @act3: titles := titles \ {getTitle(a)}
    @act4: urls := urls \ {getURL(a)}
    @act10:getTitle := {a} ⩤ getTitle
    @act6:getAuthor := {a} ⩤ getAuthor
    @act7:getURL := {a} ⩤ getURL
    @act8:getPublisher := {a} ⩤ getPublisher
    @act9:getPublishedYear := {a} ⩤ getPublishedYear
   end

```

The last 2 events of this part is responsible fro searching a resource by title or in the case of a book by ISBN.

```
/*The event to search by a title */
event SearchByTitle
 any r t u where
    @grd1:r ∈ resource
    @grd2:t ∈ titles
    @grd3: u ∈ loggedInUsers
    @grd4:getTitle(r) = t
end

/*The event to search by ISBN */
event SearchByISBN
 any b i u where
    @grd1:b ∈ book
    @grd2:i ∈ isbn
    @grd3: u ∈ loggedInUsers
    @grd4:getISBN(b) = i
end
```

5 Reading Lists

5.1 Context

```
context ReadingListContext
sets
LIST
end
```

5.2 The code

```
machine OnlineLibrarySystem2
refines OnlineLibrarySystem1
sees OnlineLibrarySystemContext ResourceContext ReadingListContext
variables
 registeredUsers
 administrators
 lecturers
 loggedInUsers
 loggingPassword

 resource
 book
 article
 website
 authors
 urls
 titles
 getAuthor
 getURL
 getTitle
 publisher
 publishedYear
 isbn
 getPublisher
 getPublishedYear
 getISBN

 lists
 getResources
 getCreator

 /*Handling the invariants */
invariants
 @inv22:lists ⊆ LIST //Making lists part of LIST
 @inv23:getResources ∈ lists ↔ resource //Creating  getResources which is a relation
      between lists and resource
 @inv24:getCreator ∈ lists → lecturers //getCreator is a total function between lists and
      lecturers

/*The events that the machine is going to do */
events
 /*Initialising the invariants*/
```

```
event INITIALISATION extends INITIALISATION
begin
@beg22:lists := ∅
@beg23:getResources := ∅
@beg24:getCreator := ∅
end

event LogIn extends LogIn
end

event LogOut extends LogOut
end

event RegisterUser extends RegisterUser
end

event ChangePassword extends ChangePassword
end

event ChangeToAdministrator extends ChangeToAdministrator
end

event ChangeToLecturer extends ChangeToLecturer
end

event CreateBook extends CreateBook
end

event RemoveBook extends RemoveBook
then
 @act12:getResources := getResources ⩥ {b} //We remove b from the getResources
      when we remove a book
end

event CreateWebsite extends CreateWebsite
end

event CreateArticle extends CreateArticle
end

event RemoveWebsite extends RemoveWebsite
then
 @act8:getResources := getResources ⩥ {w} //We remove w from getResources when
      we remove a website
end

event RemoveArticle extends RemoveArticle
then
 @act10:getResources := getResources ⩥ {a} //We remove a from getResources when
      we remove an article
end

event SearchByTitle extends SearchByTitle
end

event SearchByISBN extends SearchByISBN
```

```
end

/*The event to create a new reading list*/
event CreateReadingList
any lect lis where
 @grd1:lect ∈ lecturers //Checking if lect is a lecturer
 @grd2:lect ∈ loggedInUsers//Checking if lect is logged in
 @grd3:lis ∈ LIST //Checking if lis is part of LIST
 @grd4:lis ∉ lists //Checking if lis is not part of lists
then
 @act1:lists := lists ∪ {lis}
 @act2:getCreator(lis) := lect
end

/*The event to remove a reading list*/
event RemoveReadingList
any lect lis where
 @grd1:lect ∈ lecturers//Checking if lect is a lecturer
 @grd2:lect ∈ loggedInUsers//Checking if lect is logged in
 @grd3:lis ∈ lists//Cheking if lis is part of lists
 @grd4:getCreator(lis) = lect//Cheking if the creator of the lis is lect
then
 @act1:getCreator := {lis} ⩤ getCreator
 @act2:getResources := {lis} ⩤ getResources
 @act3:lists := lists \ {lis}
end

/*The event to add a resource to the list */
event AddResourceToList
any lect lis res where
 @grd1:lect ∈ lecturers//Checking if lect is a lecturer
 @grd2:lect ∈ loggedInUsers//Cheking if lect is logged in
 @grd3:lis ∈ lists//Cheking if lis is part of lists
 @grd4:getCreator(lis) = lect//Cheking if the creator of this list is lect
 @grd5:res ∈ resource//Cheking if res is part of resource
then
 @act1:getResources(lis) := res
end

/*The event to remove a resource from the list */
event RemoveResourceFromList
any lect lis res where
 @grd1:lect ∈ lecturers//Cheking if lect is a lecturer
 @grd2:lect ∈ loggedInUsers//Cheking if lect is logged in
 @grd3:lis ∈ lists//Checking if lis is part of lists
 @grd4:getCreator(lis) = lect//Checking if the creator of lis is the lecturer lect
 @grd5:res ∈ resource//Checking if res is part of resource
then
 @act1:getResources := getResources \ {lis ↦ res}
end

end
```

5.3 Examining the code

Firstly, we refine the OnlineLibrarySystem1 and than we declare the variables that we are going to use.

```
machine OnlineLibrarySystem2
refines OnlineLibrarySystem1
sees OnlineLibrarySystemContext ResourceContext ReadingListContext
variables
 registeredUsers
 administrators
 lecturers
 loggedInUsers
 loggingPassword

 resource
 book
 article
 website
 authors
 urls
 titles
 getAuthor
 getURL
 getTitle
 publisher
 publishedYear
 isbn
 getPublisher
 getPublishedYear
 getISBN

 lists
 getResources
 getCreator

```

Than we handle the invariants with appropriate notations.

```
 /*Handling the invariants */
invariants
 @inv22:lists ⊆ LIST //Making lists part of LIST
 @inv23:getResources ∈ lists ↔ resource //Creating  getResources which is a relation
      between lists and resource
 @inv24:getCreator ∈ lists → lecturers //getCreator is a total function between lists and
      lecturers
```

The first event we are going to do is the INITIALISATION in which we initialise all our invariants.

```
/*The events that the machine is going to do */
events
```

```
/*Initialising the invariants*/
event INITIALISATION extends INITIALISATION
begin
@beg22:lists := ∅
@beg23:getResources := ∅
@beg24:getCreator := ∅
end
```

Due to refining of the OnlineLibrarySystem1 we get the following events extended. We add another act to removeWebsite, removeBook and removeArticle which allows us to remove the website, book or article from getResource if it was removed.

```

event LogIn extends LogIn
end

event LogOut extends LogOut
end

event RegisterUser extends RegisterUser
end

event ChangePassword extends ChangePassword
end

event ChangeToAdministrator extends ChangeToAdministrator
end

event ChangeToLecturer extends ChangeToLecturer
end

event CreateBook extends CreateBook
end

event RemoveBook extends RemoveBook
then
 @act12:getResources := getResources ⩥ {b} //We remove b from the getResources
     when we remove a book
end

event CreateWebsite extends CreateWebsite
end

event CreateArticle extends CreateArticle
end

event RemoveWebsite extends RemoveWebsite
then
 @act8:getResources := getResources ⩥ {w} //We remove w from getResources when
     we remove a website
end

event RemoveArticle extends RemoveArticle
```

```
then
 @act10:getResources := getResources ⩥ {a} //We remove a from getResources when
     we remove an article
end

event SearchByTitle extends SearchByTitle
end

event SearchByISBN extends SearchByISBN
end

```

The CreateReadingList event is responsible for creating a new reading list. Only lecturers can create reading list and by creating one they become its owner.

```
/*The event to create a new reading list*/
event CreateReadingList
any lect lis where
 @grd1:lect ∈ lecturers //Checking if lect is a lecturer
 @grd2:lect ∈ loggedInUsers//Checking if lect is logged in
 @grd3:lis ∈ LIST //Checking if lis is part of LIST
 @grd4:lis ∉ lists //Checking if lis is not part of lists
then
 @act1:lists := lists ∪ {lis}
 @act2:getCreator(lis) := lect
end
```

We also need to remove a reading list. Only the lecturer who created this reading list can delete it. It deletes every relation connected to this reading list.

```
/*The event to remove a reading list*/
event RemoveReadingList
any lect lis where
 @grd1:lect ∈ lecturers//Checking if lect is a lecturer
 @grd2:lect ∈ loggedInUsers//Checking if lect is logged in
 @grd3:lis ∈ lists//Cheking if lis is part of lists
 @grd4:getCreator(lis) = lect//Cheking if the creator of the lis is lect
then
 @act1:getCreator := {lis} ⩤ getCreator
 @act2:getResources := {lis} ⩤ getResources
 @act3:lists := lists \ {lis}
end
```

We need to be able to add resources to a reading list. Only the creator lecturer can add to this reading list.

```
/*The event to add a resource to the list */
event AddResourceToList
any lect lis res where
```

```
 @grd1:lect ∈ lecturers//Checking if lect is a lecturer
 @grd2:lect ∈ loggedInUsers//Cheking if lect is logged in
 @grd3:lis ∈ lists//Cheking if lis is part of lists
 @grd4:getCreator(lis) = lect//Cheking if the creator of this list is lect
 @grd5:res ∈ resource//Cheking if res is part of resource
then
 @act1:getResources(lis) := res
end

```

The RemoveResourceFromList event is responsible for removing a resource from a reading list. Only the creator can remove resources from a reading list.

```
/*The event to remove a resource from the list */
event RemoveResourceFromList
any lect lis res where
 @grd1:lect ∈ lecturers//Cheking if lect is a lecturer
 @grd2:lect ∈ loggedInUsers//Cheking if lect is logged in
 @grd3:lis ∈ lists//Checking if lis is part of lists
 @grd4:getCreator(lis) = lect//Checking if the creator of lis is the lecturer lect
 @grd5:res ∈ resource//Checking if res is part of resource
then
 @act1:getResources := getResources \ {lis ↦ res}
end
```

6 Borrowing books

6.1 Context

```
context BorrowingContext
sets
QUEUE
end
```

6.2 The code

```
machine OnlineLibrarySystem3
refines OnlineLibrarySystem2
sees OnlineLibrarySystemContext ResourceContext ReadingListContext
        BorrowingContext

/*We declare the variables that we are going to use */
variables
 registeredUsers
 administrators
 lecturers
 loggedInUsers
 loggingPassword

```

```
resource
book
article
website
authors
urls
titles
getAuthor
getURL
getTitle
publisher
publishedYear
isbn
getPublisher
getPublishedYear
getISBN
lists
getResources
getCreator

getTotalTokens
getCurrentTokens
getBooks

getReserves
queue
bookQueues
getStudent

/*Handling the invariants */
invariants
@inv25:getTotalTokens ∈ book → ℕ
@inv26:getCurrentTokens ∈ book → ℕ
@inv27:getBooks ∈ registeredUsers ↔ book
@inv28:getReserves ∈ book → ℕ
@inv29:getStudent ∈ ℕ ⇸ registeredUsers
@inv30:queue ⊆ QUEUE
@inv31:bookQueues ∈ book → queue
events
event INITIALISATION extends INITIALISATION
begin
/*Setting every invariant initially to an empty set
 */
@beg25:getTotalTokens := ∅
@beg26:getCurrentTokens := ∅
@beg27:getBooks := ∅
@beg28:getReserves := ∅
@beg29:getStudent := ∅
@beg30:queue := ∅
@beg31:bookQueues := ∅
end

event LogIn extends LogIn
end

event LogOut extends LogOut
```

```
end

event RegisterUser extends RegisterUser
end

event ChangePassword extends ChangePassword
end

event ChangeToAdministrator extends ChangeToAdministrator
end

event ChangeToLecturer extends ChangeToLecturer
end

event CreateBook extends CreateBook
end

event RemoveBook extends RemoveBook
end

event CreateWebsite extends CreateWebsite
end

event CreateArticle extends CreateArticle
end

event RemoveWebsite extends RemoveWebsite
end

event RemoveArticle extends RemoveArticle
end

event SearchByTitle extends SearchByTitle
end

event SearchByISBN extends SearchByISBN
end

event CreateReadingList extends CreateReadingList
end

event RemoveReadingList extends RemoveReadingList
end

event AddResourceToList extends AddResourceToList
end

event RemoveResourceFromList extends RemoveResourceFromList
end

/*The event to borrow a book onlu a registered and logged in user can borrow a book if
      there is still license
 * for that book
 */
event BorrowBook
any u b where
```

```
  @grd1:b ∈ book //Checking if b is part of book
  @grd2:getTotalTokens(b) > getCurrentTokens(b) // Checking if there are still tokens
      available to borrow a book
  @grd3:u ∈ registeredUsers // Checking if u is a registered user
  @grd4:u ∈ loggedInUsers //Checking if u is logged in
 then
 /*We increase the curretnTokens for this book by one and assign the book */
  @act1:getCurrentTokens(b) := getCurrentTokens(b) + 1
  @act2:getBooks(u) := b
 end

 /*The event to reserve a book only  a logged in user can reserve a book */
 event ReserveBook
 any u b where
  @grd1:b ∈ book//Checking if b is part of book
  @grd2:getTotalTokens(b) ≤ getCurrentTokens(b)//Ckecking if there are no more
      tokens available
  @grd3:u ∈ registeredUsers // Checking if u is a registered user
  @grd4:u ∈ loggedInUsers//Checking if u is logged in
 then
  @act1:getReserves(b) := getReserves(b) + 1
  @act2:getStudent(getReserves(b)) := u
 end

 /*The event to return a book that can be done by a user */
 event ReturnBook
 any u b where
  @grd1:b ∈ book//Checking if b is part of book
  @grd2:u ∈ registeredUsers//Checking if u is a registered user
  @grd3:u ∈ loggedInUsers//Checking if u is a logged in user
 then
 /*Returning the book and decreasing the used tokens*/
  @act1:getBooks := getBooks \ {u ↦ b}
  @act2:getCurrentTokens(b) := getCurrentTokens(b) − 1
 end

 /*The event to revoke a book and it can be done by the administrator */
 event RevokeBook
 any a u b where
  @grd1:b ∈ book//Checking if b is part of book
  @grd2:a ∈ administrators//Checking if a is an administrator
  @grd3:u ∈ registeredUsers//Checking if u is a registered user
  @grd4:a ∈ loggedInUsers//Checking if a is logged in
 then
 /*We revoke the book */
  @act1:getReserves(b) := getReserves(b) + 1
  @act2:getStudent(getReserves(b)) := u
 end

end
```

6.3 Examining the code

First, we have to declare the variables that we are going to use and we refined the OnlineLibrarySystem2.

```
machine OnlineLibrarySystem3
refines OnlineLibrarySystem2
sees OnlineLibrarySystemContext ResourceContext ReadingListContext
      BorrowingContext

/*We declare the variables that we are going to use */
variables
 registeredUsers
 administrators
 lecturers
 loggedInUsers
 loggingPassword

 resource
 book
 article
 website
 authors
 urls
 titles
 getAuthor
 getURL
 getTitle
 publisher
 publishedYear
 isbn
 getPublisher
 getPublishedYear
 getISBN
 lists
 getResources
 getCreator

 getTotalTokens
 getCurrentTokens
 getBooks

 getReserves
 queue
 bookQueues
 getStudent
```

After declaring the variables we handle the invariants and and initialise these invariants.

```
 /*Handling the invariants */
 invariants
 @inv25:getTotalTokens ∈ book → ℕ
 @inv26:getCurrentTokens ∈ book → ℕ
 @inv27:getBooks ∈ registeredUsers ↔ book
 @inv28:getReserves ∈ book → ℕ
 @inv29:getStudent ∈ ℕ ⇸ registeredUsers
 @inv30:queue ⊆ QUEUE
 @inv31:bookQueues ∈ book → queue
```

```
events
 event INITIALISATION extends INITIALISATION
 begin
 /*Setting every invariant initially to an empty set
  */
 @beg25:getTotalTokens := ∅
 @beg26:getCurrentTokens := ∅
 @beg27:getBooks := ∅
 @beg28:getReserves := ∅
 @beg29:getStudent := ∅
 @beg30:queue := ∅
 @beg31:bookQueues := ∅
 end
```

We extend the events from previous machines.

```

 event LogIn extends LogIn
 end

 event LogOut extends LogOut
 end

 event RegisterUser extends RegisterUser
 end

 event ChangePassword extends ChangePassword
 end

 event ChangeToAdministrator extends ChangeToAdministrator
 end

 event ChangeToLecturer extends ChangeToLecturer
 end

 event CreateBook extends CreateBook
 end

 event RemoveBook extends RemoveBook
 end

 event CreateWebsite extends CreateWebsite
 end

 event CreateArticle extends CreateArticle
 end

 event RemoveWebsite extends RemoveWebsite
 end

 event RemoveArticle extends RemoveArticle
 end

 event SearchByTitle extends SearchByTitle
 end
```

```

event SearchByISBN extends SearchByISBN
end

event CreateReadingList extends CreateReadingList
end

event RemoveReadingList extends RemoveReadingList
end

event AddResourceToList extends AddResourceToList
end

event RemoveResourceFromList extends RemoveResourceFromList
end

```

The first event we add to this machine is the BorrowBook event which allows a user to borrow a book if there are still available license for that book.

```
/*The event to borrow a book onlu a registered and logged in user can borrow a book if
      there is still license
 * for that book
 */
event BorrowBook
any u b where
 @grd1:b ∈ book //Checking if b is part of book
 @grd2:getTotalTokens(b) > getCurrentTokens(b) // Checking if there are still tokens
      available to borrow a book
 @grd3:u ∈ registeredUsers // Checking if u is a registered user
 @grd4:u ∈ loggedInUsers //Checking if u is logged in
then
/*We increase the curretnTokens for this book by one and assign the book */
 @act1:getCurrentTokens(b) := getCurrentTokens(b) + 1
 @act2:getBooks(u) := b
end

```

The ReserveBook method lets a user to reserve a book if there is no token available to borrow it.

```
/*The event to reserve a book only  a logged in user can reserve a book */
event ReserveBook
any u b where
 @grd1:b ∈ book//Checking if b is part of book
 @grd2:getTotalTokens(b) ≤ getCurrentTokens(b)//Ckecking if there are no more
      tokens available
 @grd3:u ∈ registeredUsers // Checking if u is a registered user
 @grd4:u ∈ loggedInUsers//Checking if u is logged in
then
 @act1:getReserves(b) := getReserves(b) + 1
 @act2:getStudent(getReserves(b)) := u
end
```

```
```

The ReturnBook method allows a user to return a book that was borrowed by him.

```
/*The event to return a book that can be done by a user */
event ReturnBook
any u b where
 @grd1:b ∈ book//Checking if b is part of book
 @grd2:u ∈ registeredUsers//Checking if u is a registered user
 @grd3:u ∈ loggedInUsers//Checking if u is a logged in user
then
/*Returning the book and decreasing the used tokens*/
 @act1:getBooks := getBooks \ {u ↦ b}
 @act2:getCurrentTokens(b) := getCurrentTokens(b) − 1
end

```

The RevokeBook method allows an administrator to take a book from a user, the administrator needs to be logged in to perform this action.

```
/*The event to revoke a book and it can be done by the administrator */
event RevokeBook
any a u b where
 @grd1:b ∈ book//Checking if b is part of book
 @grd2:a ∈ administrators//Checking if a is an administrator
 @grd3:u ∈ registeredUsers//Checking if u is a registered user
 @grd4:a ∈ loggedInUsers//Checking if a is logged in
then
/*We revoke the book */
 @act1:getReserves(b) := getReserves(b) + 1
 @act2:getStudent(getReserves(b)) := u
end
```