Maude specification of the cinema model. Multi-threaded execution

Multi-threaded execution requires some way to activate new execution threads when necessary. We use a `newThread` message which should be thrown to create a new execution thread.

```
op newThread : Oid -> Msg [ctor] .
```

Each new thread is identified by a given `Oid` which should be used in `call`, `return` and `resume` messages to select the active thread where they are managed.

Furthermore, a different stack management is necessary to deal with the chaining of invocation in each active thread. As it is necessary to manage multiple execution stacks we provide a multi-threaded version of the stack management (`STACK-MT`) which should be included instead of the `STACK` module.

In the following we show the complete executable specification of the system behavior for the cinema example modeled for multi-threaded execution. In this case the Maude specification is similar to the one presented for the single-threaded case, however it includes two differences:

- Messages `call`, `return` and `resume` include the thread identifier.
- The specification of the `buyTicket` operation is modified to avoid that the number of sold ticked in a session exceeds its capacity. As the cinema version presented in Appendix **??** only considers one execution thread, it is not necessary to lock the tickets requested in a session until their payment is completed. However, now we have several execution threads, and several requests of tickets are possible for a given session. For this reason we include a new attribute `reserved` in the class `Session`, which helps us to control that the number of sold and pending of payment tickets does not exceed the capacity of a given session. The mentioned modification only requires to change the `BUY-TICKET-1-FREE-SEAT` and `BUY-TICKET-1-NO-FREE-SEAT` rules.

```
mod CINEMA-MT is
   pr CLASSES-CINEMA-MT .
   pr STACK-MT .

   subsort Nat < Oid .

   vars Self C S AC  : Oid .
   var T : Oid .
   vars AS-1 AS-2 AS-3 : AttributeSet .
   vars LC LC' LS LS' LT : List .
   var RESULT : OclType .
   var A : ArgsList .
```

```
op go-cinema : Oid Oid String Nat Nat -> Msg [msg] .
op next-goCinema-call : Oid Nat -> Msg [msg] .
op seq-goCinema :  Oid Nat -> Msg [msg] .
op seq-buyTicket :  Oid Nat -> Msg [msg] .
op seq-pay :  Oid Nat -> Msg [msg] .
op next-ticket : Nat -> Msg [msg] .

rl [SEQUENCE-GO] :
  go-cinema(T, Cl:Oid, Cn:String, St:Nat, I:Nat)
  next-goCinema-call(T, I:Nat)
  => next-goCinema-call(T, I:Nat)
     seq-goCinema(T, 1)
     call(goCinema, Cl:Oid, (arg(cinema, Cn:String),
                             arg(startTime, St:Nat)),Cl:Oid) .

rl [SEQUENCE-GO-RT] :
    resume(goCinema, RESULT, T)
    next-goCinema-call(T, I:Nat)
    => next-goCinema-call(T, s I:Nat) .

--- ------------------------------------------------------------
--- CLASS: Client
--- ------------------------------------------------------------
rl [OP-GO-CINEMA-1] :
  < ctx : Context | id : T, opN : goCinema, obj : Self ,
          args : (arg(cinema, Cn:String),arg(startTime,St:Nat)) >
  < Self : Client | cinemas : Set{LC', C, LC}, AS-1 >
  < C : Cinema | name : Cn:String,
                 sessions : Set{LS', S, LS}, AS-2 >
  < S : Session | startTime : St:Nat, AS-3 >
  seq-goCinema(T, 1)
    =>
      < Self : Client | cinemas : Set{LC', C, LC}, AS-1 >
      < C : Cinema | name : Cn:String,
                     sessions : Set{LS', S, LS}, AS-2 >
      < S : Session   | startTime : St:Nat, AS-3 >
      seq-goCinema(T, 2)
      seq-buyTicket(T, 1)
      < ctx : Context | id : T, opN : goCinema, obj : Self,
              args : (arg(cinema, Cn:String),
                      arg(startTime, St:Nat)) >
      call(buyTicket, C, (arg(startTime, St:Nat),
                          arg(aClient, Self)), T) .

crl [OP-GO-CINEMA-2-OK] :
  < ctx : Context | id : T, opN : goCinema, obj : Self, args : A >
  resume(buyTicket, RESULT, T)
  seq-goCinema(T, 2)
  < Self : Client | ticket : Set{LT, AS:AttributeSet >
  =>
```

```
    < Self : Client | ticket : Set{RESULT, LT}, AS:AttributeSet >
    < ctx : Context | id : T, opN : goCinema, obj : Self, args : A >
    return(true, T)
if RESULT =/= null .

rl [GO-CINEMA-2-FAIL] :
  < ctx : Context | id : T, opN : goCinema, obj : Self,
                    args : (arg(cinema, Cn:String),
                            arg(startTime, St:Nat)) >
  resume(buyTicket, null, T)
  seq-goCinema(T, 2)
  => < ctx : Context | id : T, opN : goCinema, obj : Self,
                       args : (arg(cinema, Cn:String),
                               arg(startTime, St:Nat)) >
     return(false, T) .

--- ------------------------------------------------------------------
--- CLASS: Cinema
--- ------------------------------------------------------------------
crl [BUY-TICKET-1-FREE-SEAT] :
  < ctx : Context | id : T, opN : buyTicket, obj : Self,
                    args : (arg(startTime, St:Nat),
                            arg(aClient, Cl:Oid)) >
  < Self : Cinema | bank : B:Oid,
                    sessions : Set{LS', S, LS}, AS-1 >
  < S : Session | startTime : St:Nat, ticket : TS:Set,
                  reserved : Rsv:Int, capacity : Cp:Nat,
                  price : P:Nat, AS-2 >
  < Cl:Oid : Client | debitCard : Cn:Nat, AS-3 >
  seq-buyTicket(T, 1)
  => < Self : Cinema | bank : B:Oid,
                       sessions : Set{LS', S, LS}, AS-1 >
     < S : Session | startTime : St:Nat, ticket : TS:Set,
                     reserved : Rsv:Int + 1, capacity : Cp:Nat,
                     price : P:Nat, AS-2 >
     < Cl:Oid : Client | debitCard : Cn:Nat, AS-3 >
     < ctx : Context | id : T, opN : buyTicket, obj : Self,
                       args : (arg(startTime, St:Nat),
                               arg(aClient, Cl:Oid)) >
     seq-pay(T, 1)
     call(pay, B:Oid, (arg(debitCard, Cn:Nat),
                       arg(amount, P:Nat)), T)
     seq-buyTicket(T, 2)
if | TS:Set | + Rsv:Int < Cp:Nat .

crl [BUY-TICKET-1-NO-FREE-SEAT] :
  < ctx : Context | id : T, opN : buyTicket, obj : Self,
                    args : (arg(startTime, St:Nat),
                            arg(aClient, Cl:Oid)) >
  < Self : Cinema | sessions : Set{LS', S, LS}, AS-1 >
```

```
  < S : Session | startTime : St:Nat, ticket : TS:Set,
                  reserved : Rsv:Int, capacity : Cp:Nat, AS-2 >
  seq-buyTicket(T, 1)
  => < Self : Cinema | sessions : Set{LS', S, LS}, AS-1 >
     < S : Session | startTime : St:Nat, ticket : TS:Set,
                     reserved : Rsv:Int, capacity : Cp:Nat, AS-2 >
     < ctx : Context | id : T, opN : buyTicket, obj : Self,
                       args : (arg(startTime, St:Nat),
                               arg(aClient, Cl:Oid)) >
    return(null, T)
if | TS:Set | + Rsv:Int >= Cp:Nat .

rl [BUY-TICKET-2-CHARGE-OK] :
  resume(pay, true, T)
  < ctx : Context | id : T, opN : buyTicket, obj : Self,
          args : (arg(startTime, St:Nat), arg(aClient, Cl:Oid)) >
  next-ticket(TK:Nat)
  < Self : Cinema | sessions : Set{LS', S, LS}, AS-1  >
  < S : Session | startTime : St:Nat,
                  ticket : Set{LT}, reserved : Rsv:Int, AS-2 >
  seq-buyTicket(T, 2)
  => < TK:Nat : Ticket | seat : 0,  session : S, client : Cl:Oid >
     < Self : Cinema | sessions : Set{LS', S, LS}, AS-1  >
     < S : Session | startTime : St:Nat, ticket : Set{TK:Nat, LT},
                     reserved : Rsv:Int - 1, AS-2 >
     next-ticket(s TK:Nat)
    < ctx : Context | id : T, opN : buyTicket, obj : Self,
                      args : (arg(startTime, St:Nat),
                              arg(aClient, Cl:Oid)) >
    return(TK:Nat, T) .

rl [BUY-TICKET-2-CHARGE-FAIL] :
  resume(pay, false, T)
  < ctx : Context | id : T, opN : buyTicket, obj : Self,
          args : (arg(startTime, St:Nat), arg(aClient, Cl:Oid)) >
  < S : Session | startTime : St:Nat, reserved : Rsv:Int, AS-1 >
  seq-buyTicket(T, 2)
  => < S : Session | startTime : St:Nat, reserved : Rsv:Int - 1, AS-1 >
     < ctx : Context | id : T, opN : buyTicket, obj : Self,
                       args : (arg(startTime, St:Nat),
                               arg(aClient, Cl:Oid)) >
     return(null, T) .

--- ------------------------------------------------------------
--- CLASS: Bank
 --- ------------------------------------------------------------
crl [PAY] :
  < ctx : Context | id : T, opN : pay, obj : Self,
                    args : (arg(debitCard, Cn:Nat),
                            arg(amount, Amt:Nat)) >
```

```
    < Self : Bank | cards : qas(Cn:Nat,AC) $$ AA:Q-Assoc >
    < AC : Account | bal : B:Nat >
    seq-pay(T, 1)
    =>
       < Self : Bank | cards : qas(Cn:Nat,AC) $$ AA:Q-Assoc >
       < AC : Account | bal : sd(B:Nat,Amt:Nat) >
       < ctx : Context | id : T, opN : pay, obj : Self,
                         args : (arg(debitCard, Cn:Nat),
                                 arg(amount, Amt:Nat)) >
       return(true, T)
    if B:Nat >= Amt:Nat .

  crl [PAY] :
    < ctx : Context | id : T, opN : pay, obj : Self,
                      args : (arg(debitCard, Cn:Nat),
                              arg(amount, Amt:Nat)) >
    < Self : Bank | cards : qas(Cn:Nat,AC) $$ AA:Q-Assoc >
    < AC : Account | bal : B:Nat >
    seq-pay(T, 1)
    =>
       < Self : Bank | cards : qas(Cn:Nat,AC) $$ AA:Q-Assoc >
       < AC : Account | bal : B:Nat >
       < ctx : Context | id : T, opN : pay, obj : Self,
                         args : (arg(debitCard, Cn:Nat),
                                 arg(amount, Amt:Nat)) >
       return(false, T)
    if B:Nat < Amt:Nat .
endm
```

Now we can perform some simulations by defining some initial configurations and using the rewrite command as usual in Maude. For example, we define the following module `TEST-CINEMA-MT`, which uses the `init-state` configuration defined in the `TEST-CINEMA` module and creates some execution threads to buy a couple of tickets.

```
mod TEST-CINEMA-MT is
  pr TEST-CINEMA .

  op init-state-1-mt : -> Configuration .
  eq init-state-1-mt
    = init-state

      newThread(bob)
      next-goCinema-call(bob, 1)
      go-cinema(bob, bob, "Coronet", 1100, 1)
      go-cinema(bob, bob, "Coronet", 1200, 2)

      newThread(ada)
      next-goCinema-call(ada, 1)
      go-cinema(ada, ada, "Coronet", 1100, 1)
```

```
    newThread(tom)
    next-goCinema-call(tom, 1)
    go-cinema(tom, tom, "Coronet", 1300, 1) .

  op init-state-2-mt : -> Configuration .
  eq init-state-2-mt
    = init-state
    newThread(bob)
    next-goCinema-call(bob, 1)
    go-cinema(bob, bob, "Coronet", 1100, 1)
    go-cinema(bob, bob, "Coronet", 1100, 2)
    go-cinema(bob, bob, "Coronet", 1200, 3)
    go-cinema(bob, bob, "Coronet", 1300, 4)

    newThread(ada)
    next-goCinema-call(ada, 1)
    go-cinema(ada, ada, "Coronet", 1100, 1)

    newThread(tom)
    next-goCinema-call(tom, 1)
    go-cinema(tom, tom, "Coronet", 1100, 1)
    go-cinema(tom, tom, "Coronet", 1200, 2)
    go-cinema(tom, tom, "Coronet", 1300, 3) .
endm
```

We use the rewrite command and we obtain the following results:

```
Maude> rewrite in TEST-CINEMA : init-state-1-mt . rewrites: 134 in
0ms cpu (7ms real) (~ rewrites/second)
    result Configuration: next-ticket(5)
    < stack : Stack | state : nil,id : bob >
    < stack : Stack | state : nil,id : ada >
    < stack : Stack | state : nil,id : tom >
    < cnm1 : Cinema | name : "Coronet",
                      bank : bbva,sessions : Set{s1, s2, s3} >
    < bbva : Bank | cards : (qas(111, acc1)
                             $$ qas(222, acc2) $$ qas(333, acc3)) >
    < s1 : Session | startTime : 1100,endTime : 1150,capacity : 10,
                     price : 5,ticket : Set{2, 1},reserved : 0 >
    < s2 : Session | startTime : 1200,endTime : 1250,capacity : 10,
                     price : 8,ticket : Set{4},reserved : 0 >
    < s3 : Session | startTime : 1300,endTime : 1350,capacity : 10,
                     price : 5,ticket : Set{3},reserved : 0 >
    < bob : Client | ticket : Set{4, 1},
                     cinemas : Set{cnm1},debitCard : 111 >
    < ada : Client | ticket : Set{2},
                     cinemas : Set{cnm1},debitCard : 222 >
    < tom : Client | ticket : Set{3},
                     cinemas : Set{cnm1},debitCard : 333 >
```

```
    < acc1 : Account | bal : 87 >
    < acc2 : Account | bal : 995 >
    < acc3 : Account | bal : 9995 >
    < 1 : Ticket | seat : 0,session : s1,client : bob >
    < 2 : Ticket | seat : 0,session : s1,client : ada >
    < 3 : Ticket | seat : 0,session : s3,client : tom >
    < 4 : Ticket | seat : 0,session : s2,client : bob >

Maude> rewrite in TEST-CINEMA : init-state-2-mt . rewrites: 280 in
0ms cpu (7ms real) (~ rewrites/second) result Configuration:
    < stack : Stack | state : nil,id : bob >
    < stack : Stack | state : nil,id : ada >
    < stack : Stack | state : nil,id : tom >
    < cnm1 : Cinema | name : "Coronet",
                      bank : bbva,sessions : Set{s1, s2, s3} >
    < bbva : Bank | cards : (qas(111, acc1)
                             $$ qas(222, acc2) $$ qas(333, acc3)) >
    < s1 : Session | startTime : 1100,endTime : 1150,capacity : 10,
                     price : 5,ticket : Set{4,3,2,1},reserved : 0 >
    < s2 : Session | startTime : 1200,endTime : 1250,capacity : 10,
                     price : 8,ticket : Set{6, 5},reserved : 0 >
    < s3 : Session | startTime : 1300,endTime : 1350,capacity : 10,
                     price : 5,ticket : Set{8, 7},reserved : 0 >
    < bob : Client | ticket : Set{8, 6, 4, 1},
                     cinemas : Set{cnm1},debitCard : 111 >
    < ada : Client | ticket : Set{2},
                     cinemas : Set{cnm1},debitCard : 222 >
    < tom : Client | ticket : Set{7, 5, 3},
                     cinemas : Set{cnm1},debitCard : 333 >
    < acc1 : Account | bal : 77 >
    < acc2 : Account | bal : 995 >
    < acc3 : Account | bal : 9982 >
    < 1 : Ticket | seat : 0,session : s1,client : bob >
    < 2 : Ticket | seat : 0,session : s1,client : ada >
    < 3 : Ticket | seat : 0,session : s1,client : tom >
    < 4 : Ticket | seat : 0,session : s1,client : bob >
    < 5 : Ticket | seat : 0,session : s2,client : tom >
    < 6 : Ticket | seat : 0,session : s2,client : bob >
    < 7 : Ticket | seat : 0,session : s3,client : tom >
    < 8 : Ticket | seat : 0,session : s3,client : bob >
```

The first case is a correct situation, however the second one violates some constraints as we will see later.