## *Projected implementation of atomic actions in Lemick*

The following interfaces and classes could be used as templates for programmers or for implementation of build-in classes and interfaces supporting AA.

Lemick supports interface as they are found in many OO languages

```
Interface AtomicAction
        Const AA_STAGE_REGISTER = 1, ...
            ' Allow members to register in AA
        Sub GatherMembers
            ' Query state of the AA
        Fun GetState As AtomicActionState
            ' Register member of AA
        Sub RegisterMember(member As AtomicActionMember)
            ' Toggle "member" as a finished
        Sub ReportComplete(member As AtomicActionMember)
            ' Recieve exception from "member"
        Sub ReportException(member As AtomicActionMember)
            ' Resolve concurrent exceptions
  Private Sub ResolveExceptions
            ' Do something about coordinated EH of conc. exceptions
  Private Sub CoHandleExceptions
End Interface


Interface AtomicActionMember
      Const AAM_OK = 1
            ' Get into AA
      Fun RegisterIn(group As AtomicAction) As Integer
            ' Normal activity inside AA
      Sub Action
      Sub Commit
      Sub Abort
End Interface
```

Sample implementation of an atomic action with three participants – two robot hands and some sensor. Code not related to AA is omitted.

```
Class Robot_AA Implements AtomicAction Signals AA_NoRegistration,
AA_Timeout
      Private Const MEM_NUM = 3 ' assume 3 participants
      Private members(MEM_NUM) As AtomicActionMember
      Private mth(MEM_NUM) As Thread
      Private actstate As AtomicActionState
      Private watchdog As Thread

        Sub Init
            actstate.register = 0 ' do not accept registration yet
            actstate.members  = 0 ' members registered so far
            actstate.complete = 0 ' members completed their job so far
            accstate.timeout  = 5 ' wait for all the participants to come
                                  ' in five seconds
```

```
        End Sub
        Sub GatherMembers
            actstate.register = 1 ' accept registration now
            accstate.stage = AA_STAGE_REGISTER
            WatchDog Concurrent watchdog   ' start up the watchdog
                                           ' in background
        End Sub
        Fun GetState As AtomicActionState
            me = actstate            ' return value of actstate
        End Sub
        Sub RegisterMember(member As AtomicActionMember) Synchronized

            ' check if accepting registration
            If actstate.register Then
                  members(actstate.members) = member
                  actstate.members = actstate.members + 1
                 Else
                  Signal AA_NoRegistration
            End If
            ' check if we have got all the needed participants.
            If actstate.members = MEM_NUM Then
                  actstate.register = 0   ' reject future registrations
                  accstate.stage = AA_STAGE_ACTION_INIT
                  ' start up the participants normal activity
                  For i = 0 To MEM_NUM
                        members(i).Action Concurrent mth(i)
                  Next
                  accstate.stage = AA_STAGE_ACTION
            End If
          End Sub
        Sub ReportComplete(member As AtomicActionMember) Synchronized
            actstate.complete = actstate.complete + 1
            If actstate.complete =  actstate.members Then
                  accstate.stage = AA_STAGE_COMMIT
                  ' call commit method of all the participants
                  For i = 0 To MEM_NUM
                        members(i).Commit
                  Next
                  accstate.stage = AA_STAGE_JOIN
                  ' join all the participants
                  For i = 0 To MEM_NUM
                        Join mth(i)
                  Next
                  accstate.stage = AA_STAGE_DONE
                  ' watchdog should be in suspended state now. Join it to free resources
                  Join watchdog
            End If
        End Sub
        Sub ReportException(member As AtomicActionMember) Synchronized
            ' accumulate exception for future exception resolution ?
        End Sub
    Private Sub ResolveExceptions
            ' find the best match for acc. concurrent exceptions ?
        End Sub
    Private Sub CoHandleExceptions
            ' tell the members to execute some handler ?
```

```
        End Sub
  Private Sub WatchDog
            Sleep accstate.timeout

            ' if we are still in registration stage then something
            ' went wrong. Reset to the initial stage
            If accstate.stage = AA_STAGE_REGISTER Then
                  actstate.register = 0 ' do not accept registration
                  actstate.members  = 0
                  actstate.complete = 0
                  Signal AA_Timeout
            End If
        End Sub
End Class
```

Implementation of the atomic action participants for the atomic action class described above will is based on the following abstract class.

```
Class RobotPart_AAM Implements AtomicActionMember
      Private aagroup As AtomicAction
      Sub Init(group As AtomicAction)
            aagroup = group
            t = RegisterIn(group)
            If t <> AAM_OK Then aagroup = Empty
      End Sub
      Private Sub ResgiterIn(group As AtomicAction)
            group.RegisterMember this
      End Sub
      Declare Private Sub Action ' virtual method
      Declare Private Sub Commit ' virtual method
      Declare Private Sub Abort  ' virtual method
End Class
```

Three following classes need to implement only Action, Commit and Abort. Other methods are inherited from class RobotPart_AAM.

```
Class LeftRobotHand Extends RobotPart_AAM
      Sub Action
            ... ' do something
            aagroup.reportComplete(this) ' tell that the work is done
      End Sub
      ' Implementation of Commit and Abort
End Class


Class RightRobotHand Extends RobotPart_AAM
      Sub Action
            ... ' do something
            aagroup.reportComplete(this) ' tell that the work is done
      End Sub
      ' Implementation of Commit and Abort
End Class
```

```
Class RobotLHSensor Extends RobotPart_AAM
      ' Implementation of Action, Commit and Abort
End Class

' ... and so on
```

## Sample program

```
Dim RobotAA   As Robot_AA
Dim LeftHand  As LeftRobotHand
Dim RightHand As RightRobotHand
Dim LHSensor  As RobotLHSensor


RobotAA    = New Robot_AA ' create a new atomic action

RobotAA.GatherMembers      ' allow the participants to register in the AA

LeftHand  = New LeftRobotHand (RobotAA)   ' create LeftHand and register
                                          ' it in RobotAA

RightHand = New RightRobotHand(RobotAA)   ' create RightHand and
                                          ' register it in RobotAA

LHSensor  = New RobotLHSensor (RobotAA)   ' create LHSensor and register
                                          ' it in RobotAA
' at this point all the members are found and
' Robot AA starts

' waiting for all the threads to finish
```