Capitolo 4
Implementazione delle nuove funzionalità

4.1 Implementazione del confronto per le variabili intere

Come detto nel secondo capitolo, per l'implementazione della funzionalità del confronto per le variabili intere è stato necessario modificare principalmente la classe *BooleanExpression* di *UnisimClassLibrary*. Questa classe si occupa del parsing, ossia la verifica della correttezza sintattica, e della valutazione dell'espressione logica che esprime la condizione legata ad una fase.

Le modifiche consistono nell'aggiunta, a livello sintattico, dei nuovi operatori di confronto che devono essere messi a disposizione dell'utente: "<", ">", "=", "<=", ">=" e nell'adattamento della struttura dati che rappresenta l'espressione e la valutazione in modo da tener conto della nuova funzionalità. Un ulteriore modifica riguarda il metodo *MakeString()* che ricrea la stringa contenente l'espressione logica a partire dalla struttura. Questa modifica però non verrà discussa in quanto triviale.

4.1.1 Modifica della parte riguardante il parsing dell'espressione logica

Il metodo *Parse()* di *BooleanExpression* scorre carattere a carattere la stringa che contiene l'espressione logica e controlla, per ognuno di esso, che il carattere successivo sia uno di quelli attesi. Il seguente codice è un estratto del metodo *Parse()*:

```
[...]
Expected = "c![("
 For i = 1 To Exp.Length
     NewChar = Mid(Exp, i, 1)
     If NewCharOk(Expected, NewChar) Then
         If Not DefiningStepMakerCondition Then
             'Non si sta definendo un StepMakerCondition
             Select Case NewChar
                 Case "+", "*"
                     Expected = "c![("
                 Case "!"
                     Expected = "c[("
                 Case "("
                     Expected = "c!(["
                     NumOpenPar = NumOpenPar + 1
                 Case ")"
                     Expected = "o)"
```

```
                    NumOpenPar = NumOpenPar - 1
                Case "["
                    DefiningStepMakerCondition = True
                    Expected = "c"
                Case ">", "<", "="
                    Expected = "c>"
                Case Else
                    Expected = "co)>"


            End Select

[...]
Else
        'Ha trovato un carattere non valido
        'Memorizza l'errore
        m_Error = "Unexpected char at " & i & ": '" & NewChar & "'!"
        Exit Function
    End If

       Next i

[...]
```

Il metodo *NewCharOk()* prende in ingresso il carattere *NewChar* e la stringa *Expected*, che contiene i caratteri attesi, e ritorna il valore “true” se *NewChar* è uno dei caratteri attesi specificati in *Expected*. Quest’ultima può contenere sia il singolo carattere atteso, sia un insieme di caratteri appartenenti alla stessa tipologia e rappresentati da un carattere speciale. Le corrispondenze tra i caratteri speciali e l’insieme di caratteri che essi rappresentano sono le seguenti:

Carattere speciale	**Corrispondenze**
c	Carattere appartenente al nome di una variabile
o	Operatore + (or) e * (and)
n	Numero
u	Unità di tempo (m, s, h, g)
>	Operatore di confronto (<, >, =)
T	Riferimento al tempo di attivazione di una fase (T,t)
X	Riferimento all’attivazione di una fase (X,x)

Come si può vedere nel codice estratto dal metodo Parse() è stato aggiunto un ulteriore Case nel costrutto Select che, nel caso *NewChar* sia uno degli operatori di confronto, definisce come caratteri attesi un carattere normale o un altro operatore di confronto (è il caso in cui l'utente utilizza l'operatore ">=" o "<=").

Analizzando il codice del metodo *NewCharOk()*:

```
Private Function NewCharOk(ByVal Expected As String,
                           ByVal NewChar As Char) As Boolean
 Dim i As Integer
 Dim ExpectedChar As Char
 'Controlla che il test sia vero almeno per uno caratteri attesi
 For i = 0 To Expected.Length - 1
     If CharTest(Expected.Chars(i), NewChar) Then
         NewCharOk = True
         Exit For
     End If
 Next i

End Function
```

e l'estratto del metodo *CharTest()*:

```
Select Case ExpectedChar
[...]
     Case ">"
           If NewChar = ">" Or NewChar = "<" Or NewChar = "=" Then
           CharTest = True
           End If
[...]
```

Si nota l'ulteriore modifica fatta in *CharTest()* che aggiunge la corrispondenza del carattere di confronto.

4.1.2 Modifica della parte riguardante la creazione della struttura ad albero

Dopo aver verificato la correttezza sintattica dell'espressione logica, viene creata una struttura ad albero dove ogni nodo rappresenta o l'operatore logico (and e or), o un riferimento al marker di fase, o la variabile booleana. Nel caso la variabile sia di tipo intero, viene considerato il suo valore logico che, come detto nel secondo capitolo, assume il valore "true" se il valore attuale intero ha raggiunto il valore finale.

Nella seguente figura è riportato un esempio di espressione logica con la relativa struttura ad albero.

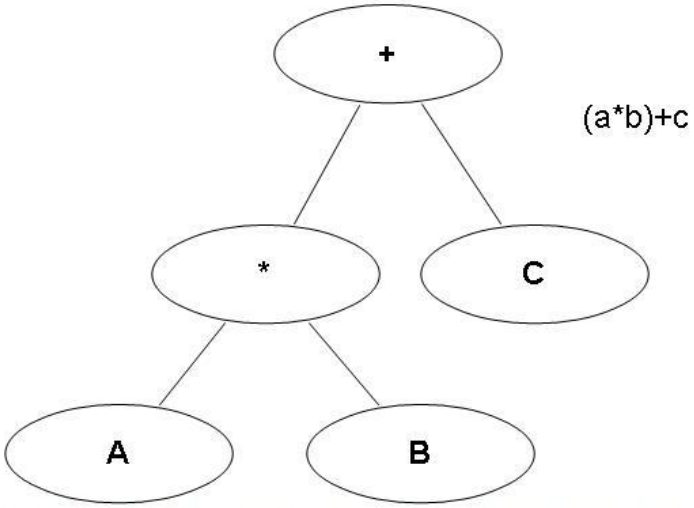

Figura 17: Esempio di una struttura ad albero

La classe *BooleanExpression*, quindi, contiene le classi *PlusNode*, *MultNode*, *stepMakerConditionNode* e *VariableNode* che implementano rispettivamente il nodo or, and, riferimento al marker di fase e variabile. Ognuna di queste classi, tranne *VariableNode* che rappresenta sempre il nodo più esterno dell'albero, contengono l'attributo *NextNodes* che contiene la lista dei sottonodi.

I metodi *CreatePluseNode()*, *CreateMultNode()* e *CreateStepMarkerConditionNode()* creano la struttura ad albero con una serie di chiamate innestate.

La modifica per l'implementazione del confronto tra variabili intere ha comportato la creazione della classe *CompareNode* che rappresenta il nodo di confronto e del metodo *CreateCompareNode()* per la collocazione dei nodi di confronto nella struttura.

La classe *CompareNode* contiene, oltre all'attributo *NextNodes*, l'attributo *Op* di tipo String che specifica l'operatore di confronto:

```
Public Class CompareNode
      Public NextNodes As ArrayList
      Public Op As String
      Public Neg As Boolean = False

  End Class
```

L'attrubuto *Neg*, presente anche nelle altre classi che implementano i nodi, indica se il valore logico del nodo deve essere negato ed è inserito nella classe CompareNode per motivi di compatibilità.

Un esempio di una struttura ad albero che rappresenta una espressione logica contenente il confronto tra variabili intere è il seguente:

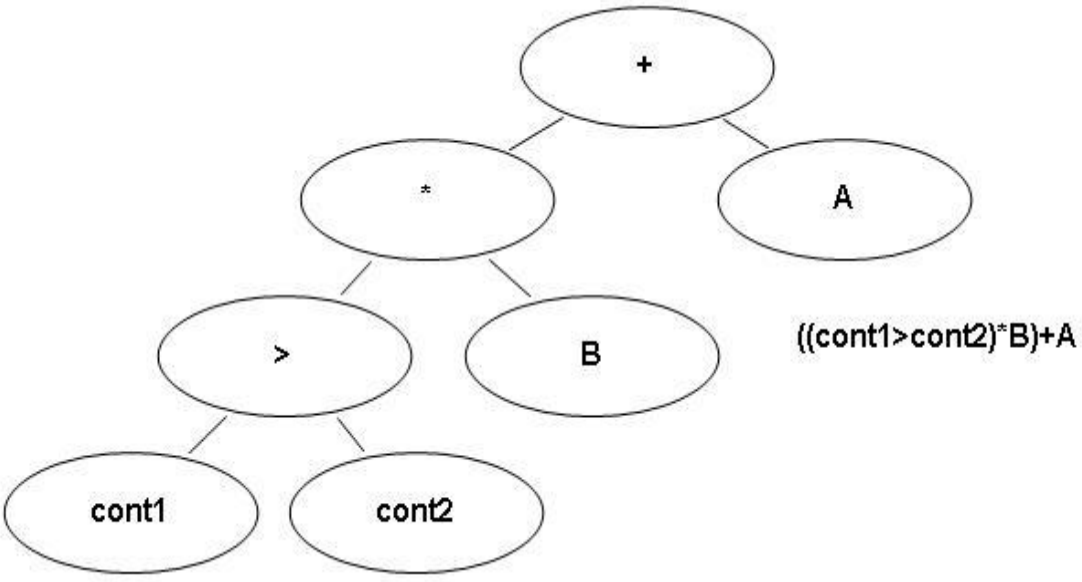

Figura 18: Esempio di una struttura ad albero con il confronto tra due variabili intere

L'implementazione della classe *CompareNode* è la seguente:

```
Private Function CreateCompareNode(ByVal Exp As String) As CompareNode
        Dim RootNode As New CompareNode
        RootNode.NextNodes = New ArrayList
        Dim i, NumOpenPar, FirstPosPar As Integer
        Dim VarName As String
        Dim NewChar As Char
        Dim go As Boolean = True
        Dim ParFind As Boolean
        While (go)
            For i = 1 To Exp.Length
                Select Case Mid(Exp, i, 1)
                    Case "("
                        NumOpenPar = NumOpenPar + 1
                        If Not ParFind Then
                            FirstPosPar = i
                            ParFind = True

                        End If
                    Case ")"
                        NumOpenPar = NumOpenPar - 1
                       If ParFind And NumOpenPar = 0 And Exp.IndexOf("+") >= 0 Then
                            RootNode.NextNodes.Add(CreatePlusNode(
                            Mid(Exp,FirstPosPar + 1, i - FirstPosPar - 1), False))
                            'Rimuove l'espressione dalla stringa
                            Dim j As Integer
                      'Elimina gli eventuali operatori prima della parentesi aperta
                            If Not FirstPosPar = 1 Then
                                j = 1
                            End If

                            Exp = Exp.Remove(FirstPosPar - 1 - j, i - FirstPosPar +
                                             1 + j)
                            i = 0
                            ParFind = False
                            Exit For
                        End If
                End Select
            Next i
            If i > Exp.Length Then   'Non ha trovato più ( )
                go = False
            End If
        End While
        For i = 1 To Exp.Length
            NewChar = Mid(Exp, i, 1)
            Select Case NewChar
                Case ">", "<", "="
                    If VarName <> "" Then
                        'Deve aggiungere un nodo  variabile
                        Dim NewNode As New VariableNode 'Aggiunge un nodo variabile
                        NewNode.Var = FindVarByName(VarName)
                        If (Not IsNothing(NewNode.Var)) Then
                            If (NewNode.Var.dataType = "BOOL") Then
                                m_Error = "Variable " & VarName & " is of type
                                          BOOL"
                            End If
                        End If
                        RootNode.NextNodes.Add(NewNode)
                        RootNode.Op = NewChar
                        If Not IsNothing(NewNode.Var) Then
                            AddToUsedVariablesList(NewNode.Var)
                     'Aggiunge la variabile alla lista delle variabili utilizzate
                        Else : Exit Function
                        End If
                    Else
                        Dim tmpChar = Mid(Exp, i - 1, 1)
                        If tmpChar = "<" And NewChar = "=" Then
                            RootNode.Op = "<="
                        End If
                        If tmpChar = ">" And NewChar = "=" Then
                            RootNode.Op = ">="
```

```
                    End If
                End If
                VarName = ""
                Case "(", ")"

            Case Else
                VarName = VarName & NewChar
        End Select
    Next i
    'Aggiunge l'ultima variabile rimasta
    If Exp.Length > 0 Then
        'E un nodo variabile
        Dim isFakeVar As Boolean = False
        Dim NewNode As New VariableNode 'Aggiunge un nodo variabile
        NewNode.Var = FindVarByName(VarName)
        If (Not IsNothing(NewNode.Var)) Then
            If (NewNode.Var.dataType = "BOOL") Then
                m_Error = "Variable " & VarName & " is of type BOOL"
            End If
        Else
            If (IsNumeric(VarName)) Then
                Dim intVar As IntegerVariable = New IntegerVariable
                intVar.SetActValue(VarName)
                intVar.Name = VarName
                NewNode.Var = intVar
                m_Error = ""
                isFakeVar = True
            End If
        End If
        RootNode.NextNodes.Add(NewNode)
        If Not IsNothing(NewNode.Var) And Not isFakeVar Then
            AddToUsedVariablesList(NewNode.Var)
         'Aggiunge la variabile alla lista delle variabili utilizzate
        End If

    End If
    CreateCompareNode = RootNode

End Function
```

Il metodo, in modo analogo agli altri metodi che creano la struttura ad albero, scorre la stringa che contiene l'espressione cercando prima le coppie di parentesi tonde. Per ogni coppia di parentesi trovata invoca *CreatePlusNode()* passando in ingresso la parte dell'espressione contenuta in quest'ultima. Il nodo creato da *CreatePlusNode()* viene aggiunto alla lista dei sottonodi. Terminate le parentesi resta solo il confronto tra variabili intere. Si estraggono,quindi, le variabili e l'operatore di confronto e si crea il nodo che verrà assunto come valore di ritorno.

CreateCompareNode() deve essere invocato da *CreateMultNode()* o da *CreatePlusNode()* nel caso il confronto sia messo in and o or con una variabile booleana. Ad esempio, se l'espressione è *cont>2*ok* , con cont variabile intera e ok variabile booleana, CreateMultNode() deve invocare CreateCompareNode() per creare un sottonodo contenente il confronto. Sono state necessarie, quindi, delle modifiche a tali metodi. Di

seguito si riportano gli estratti del codice rispettivamente di *CreatePlusNode()* e di *CreateMultNode()* interessati dalle modifiche con evidenziate le parti di codice aggiunte:

[…]

```
    While (go)
        For i = 1 To Exp.Length
            Select Case Mid(Exp, i, 1)
                Case "+"
                    If MultFind And NumOpenPar = 0 Then
                        RootNode.NextNodes.Add(CreateMultNode(
                        Mid(Exp, FirstPosPlus + 1, i - FirstPosPlus - 1)))
                        MultFind = False
                        CompareOpFind = False
                        'Rimuove l'espressione dalla stringa
                        Dim j As Integer
                        'Elimina gli eventuali operatori dopo la parentesi chiusa
                        If Not i = Exp.Length Then
                            j = 1
                        End If
                        Exp = Exp.Remove(FirstPosPlus, i - FirstPosPlus - 1 + j)
                        i = 0
                        Exit For
                    ElseIf CompareOpFind And NumOpenPar = 0 Then
                        RootNode.NextNodes.Add(CreateCompareNode(Mid(Exp, FirstPosPlus
                        + 1, i - FirstPosPlus - 1)))
                        CompareOpFind = False
                        'Rimuove l'espressione dalla stringa
                        Dim j As Integer
                        'Elimina gli eventuali operatori dopo la parentesi chiusa
                        If Not i = Exp.Length Then
                            j = 1
                        End If
                        Exp = Exp.Remove(FirstPosPlus, i - FirstPosPlus - 1 + j)
                        i = 0
                        Exit For

                    Else
                        If NumOpenPar = 0 Then
                            FirstPosPlus = i
                        End If
                    End If
                Case "*"
                    MultFind = True
                Case ">", "<", "="
                    If (Not squareParFind) Then
                        CompareOpFind = True
                    ndE If
                Case "("
                    NumOpenPar = NumOpenPar + 1
                Case ")"
                    NumOpenPar = NumOpenPar - 1
                Case "["
                    squareParFind = True
                Case "]"
                    squareParFind = False

            End Select

        Next i
```

[…]

[...]

```
        'Exp ora contiene solo * e >
                Dim VarName As String
                Dim NewChar As Char
                'Cerca le singole variabili ,confronti tra interi e stepmaker tra i *
                For i = 1 To Exp.Length
                    NewChar = Mid(Exp, i, 1)
                    Select Case NewChar
                        Case "*"
                            If VarName <> "" Then
            'Deve aggiungere un nodo o variabile o stepmaker o una comparazione tra interi
                                If (VarName.IndexOf(">") >= 0 Or VarName.IndexOf("<") >= 0
                                        Or VarName.IndexOf("=") >= 0) Then
                                    If (Mid(VarName, 1, 1) <> "[") Then
                                    RootNode.NextNodes.Add(CreateCompareNode(VarName))
                                    VarName = ""
                                        Exit Select
                                    End If
                                End If
                                If Mid(VarName, 1, 1) <> "[" Then
                                    'E un nodo variabile
                                    Dim NewNode As New VariableNode
                                    'Aggiunge un nodo variabile
                                    NewNode.Var = FindVarByName(VarName)
                                    If NextNodeNeg Then 'Nega la variabile se richiesto
                                        NewNode.Neg = True
                                        NextNodeNeg = False
                                    End If
                                    RootNode.NextNodes.Add(NewNode)
                                    If Not IsNothing(NewNode.Var) Then
                                        AddToUsedVariablesList(NewNode.Var)
                            'Aggiunge la variabile alla lista delle variabili utilizzate
                                    End If
                                Else
                                    'E' un nodo stepmaker
                                    'Aggiunge un nodo stepmaker
                                    Dim NewNode As StepMakerConditionNode =
                                            CreateStepMakerConditionNode(VarName)
                                    If NextNodeNeg Then 'Nega il nodo se richiesto
                                        NewNode.Neg = True
                                        NextNodeNeg = False
                                    End If
                                    RootNode.NextNodes.Add(NewNode)
                                End If
                                VarName = ""
                            End If
                        Case "!"
                                NextNodeNeg = True
                        Case Else
                                VarName = VarName & NewChar
                    End Select
                Next i
                'Aggiunge l'ultima variabile rimasto
                If Exp.Length > 0 Then
                    If ((VarName.IndexOf(">") >= 0 Or VarName.IndexOf("<") >= 0 Or
                            VarName.IndexOf("=") >= 0) And Mid(VarName, 1, 1) <> "[") Then
                        RootNode.NextNodes.Add(CreateCompareNode(VarName))
                        VarName = ""
                    ElseIf Mid(VarName, 1, 1) <> "[" Then
                        'E un nodo variabile
                        Dim NewNode As New VariableNode 'Aggiunge un nodo variabile
                        NewNode.Var = FindVarByName(VarName)
                        If NextNodeNeg Then 'Nega la variabile se richiesto
                            NewNode.Neg = True
                            NextNodeNeg = False
                        End If
                        RootNode.NextNodes.Add(NewNode)
                        If Not IsNothing(NewNode.Var) Then
                            AddToUsedVariablesList(NewNode.Var)
                         'Aggiunge la variabile alla lista delle variabili utilizzate
                        End If
                    Else
                        'E' un nodo stepmaker
                        'Aggiunge un nodo stepmaker
                        Dim NewNode As StepMakerConditionNode =
                CreateStepMakerConditionNode(VarName)
                        If NextNodeNeg Then 'Nega il nodo se richiesto
```

```
                NewNode.Neg = True
                NextNodeNeg = False
            End If
            RootNode.NextNodes.Add(NewNode)
        End If
        End If
        CreateMultNode = RootNode

    End Function
```

4.1.3 Modifica della parte riguardante la valutazione dell'espressione logica

Il metodo *EvaluateNode()*, utilizzando una formula ricorsiva, scorre la struttura dell'albero rappresentante l'espressione logica valutandola.

L'implementazione del metodo è riportata di seguito con evidenziata la parte di codice aggiunta:

```
Private Function EvaluateNode(ByRef EvNode As Object) As Boolean
        'Utilizza una formula ricorsiva
        Select Case EvNode.GetType.Name
            Case "CompareNode"

                If (EvNode.NextNodes.item(0).GetType.Name <> "VariableNode") Then
                    For Each N As Object In EvNode.NextNodes
                        EvaluateNode = False
                        EvaluateNode = EvaluateNode Or EvaluateNode(N)
                    Next N
                    Exit Function
                End If

                Dim firstVar = EvNode.NextNodes.item(0).Var
                Dim secondVar = EvNode.NextNodes.item(1).Var
                EvaluateNode = False
                Select Case EvNode.op
                    Case ">"
                        If (firstVar.ReadActValue > secondVar.ReadActValue) Then
                            EvaluateNode = True
                        End If
                    Case "<"
                        If (firstVar.ReadActValue < secondVar.ReadActValue) Then
                            EvaluateNode = True
                        End If
                    Case "="
                        If (firstVar.ReadActValue = secondVar.ReadActValue) Then
                            EvaluateNode = True
                        End If
                    Case "<="
                        If (firstVar.ReadActValue <= secondVar.ReadActValue) Then
                            EvaluateNode = True
                        End If
                    Case ">="
                        If (firstVar.ReadActValue >= secondVar.ReadActValue) Then
                            EvaluateNode = True
                        End If
                End Select
            Case "PlusNode" 'E' un nodo operatore
                'Fa la Or tra il valore attuale e tutti i figli di EvNode
                For Each N As Object In EvNode.NextNodes
                    EvaluateNode = EvaluateNode Or EvaluateNode(N)
                Next N
                'Nega il risultato se richiesto
                If EvNode.Neg Then
                    EvaluateNode = Not EvaluateNode
                End If
```

```
                Case "MultNode"
                    'Fa la And tra il valore attuale e tutti i figli di EvNode
                    For Each N As Object In EvNode.NextNodes
                        EvaluateNode = True
                        EvaluateNode = EvaluateNode And EvaluateNode(N)
                        If EvaluateNode = False Then
                            'Se c'e un nodo false esce dal ciclo
                            Exit For
                        End If
                    Next N
                    'Nega il risultato se richiesto
                    If EvNode.Neg Then
                        EvaluateNode = Not EvaluateNode
                    End If
                Case "VariableNode" 'E' un nodo variabile
                    'Legge il valore della variabile
                    'Monitor
                    EvaluateNode = EvNode.Var.ReadValue
                    'La nega se richiesto
                    If EvNode.Neg Then
                        EvaluateNode = Not EvaluateNode
                    End If
                Case "StepMakerConditionNode"   'E' un nodo stepmaker
                    Select Case EvNode.Type
                        Case False  'è di tipo S0.X
                            EvaluateNode = EvNode.StepMaker.ReadActive
                        Case True
                            Select Case EvNode.Op
                                Case "<"
'Confronta l'istante di attivazione della fase più il tempo Time con quello attuale
                                If DateTime.Compare
                                 (EvNode.StepMaker.ReadTimeActivation.Add(EvNode.Time),
                                       Now) > 0 And EvNode.StepMaker.ReadActive Then
                                        EvaluateNode = True
                                    End If
                                Case ">"
'Confronta l'istante di attivazione della fase più il tempo Time con quello attuale
                                If DateTime.Compare
                                (EvNode.StepMaker.ReadTimeActivation.Add(EvNode.Time)
                                  , Now) < 0 And EvNode.StepMaker.ReadActive Then
                                        EvaluateNode = True
                                    End If
                            End Select
                    End Select
                    'La nega se richiesto
                    If EvNode.Neg Then
                        EvaluateNode = Not EvaluateNode
                    End If
            End Select
        End Function

    End Class
```

Se il nodo di confronto contiene due sottonodi variabile, confronta queste due tramite l'operatore di confronto definito e assegna come valore di ritorno della funzione il risultato del confronto. Nel caso in cui il nodo non contenga due sottonodi variabile, l'unico caso che si può verificare è che abbia un solo sottonodo or. E' il caso di una espressione del tipo *((cont>2)+var)*var2)* : il metodo *CreatePlusNode()* invoca *CreateCompareNode()* fornendogli come espressione la stringa *((cont>2)+var)* . Il primo if nel codice evidenziato tiene conto di questa evenienza.

4.2 Implementazione delle espressioni aritmetiche

L'implementazione delle espressioni aritmetiche consiste in una serie di modifiche all'interfaccia grafica, al motore di simulazione e all'import/export del progetto, più l'aggiunta di una nuova classe che si occupa del calcolo dell'espressione. Le espressioni aritmetiche permettono di aggiungere un nuovo tipo di azione a quelle già messe a disposizione dell'utente da UniSim e, di conseguenza, rendono necessaria la modifica dello XML Schema. In questo lavoro di tesi non è stata seguita la specifica di PLCOpen per il salvataggio del progetto, rimandando a sviluppi futuri questo aspetto.

4.2.1 Implementazione della classe ArithmeticExpression

La classe *ArithmeticExpression*, come detto nel terzo capitolo, deve mettere a disposizione il metodo *Parse()* per la verifica della correttezza sintattica e il metodo *calculateExp()* per il calcolo dell'espressione.
Il costruttore accetta come parametri di ingresso la lista contenente le liste delle variabili globali e l'interfaccia al Pou per accedere alla lista delle variabili locali e assegna a due variabili globali della classe i due parametri in ingresso:

```
Public Sub New(ByRef ResGlobalVariables As VariablesLists, ByRef pouInterface As
                  pouInterface)

        m_pouInterface = pouInterface
        m_ResGlobalVariables = ResGlobalVariables

End Sub
```

Il metodo *Parse()* è stato implementato in maniera analoga a quello visto per la classe *BooleanExpression* cambiando soltanto le corrispondenze tra caratteri speciali e insieme di caratteri e le definizioni dei caratteri attesi per ogni tipo di carattere trovato. La seguente tabella illustra le corrispondenza tra i caratteri speciali e l'insieme di caratteri rappresentati:

Carattere speciale	Corrispondenze
c	Carattere appartenente al nome di una variabile
o	Operatore + (somma), – (sottrazione) * (moltiplicazione), / (modulo)
n	Numero

Si seguito si riporta il codice dei metodi *Parse()*, *NewCharOk()* e *CharTest()*.

```
Public Function Parse(ByVal Exp As String) As Boolean

        Dim Expected, NewChar As String
        Dim i, NumOpenPar As Integer

        Parse = True
        Expected = "co("
        For i = 1 To Exp.Length
            NewChar = Mid(Exp, i, 1)
            If NewCharOk(Expected, NewChar) Then
                Select Case NewChar
                    Case "+", "*", "-", "/"
                        Expected = "c(n"
                    Case "("
                        Expected = "c(n"
                        NumOpenPar = NumOpenPar + 1
                    Case ")"
                        Expected = "o)"
                        NumOpenPar = NumOpenPar - 1
                    Case Else
                        Expected = "co)n"

                End Select
            Else
                m_Error = "Unexpected char at " & i & ": '" & NewChar & "'!"
                Parse = False
                Exit Function
            End If
        Next i

        If NumOpenPar > 0 Then
            m_Error = "Expected ')'"
            Parse = False
            Exit Function
        Else
            If NumOpenPar < 0 Then
                m_Error = "Expected '('"
                Parse = False
                Exit Function
            End If
        End If

    End Function
```

```
Private Function NewCharOk(ByVal Expected As String, ByVal NewChar As Char) As Boolean
        Dim i As Integer
        Dim ExpectedChar As Char
        'Controlla che il test sia vero almeno per uno caratteri attesi
        For i = 0 To Expected.Length - 1
            If CharTest(Expected.Chars(i), NewChar) Then
                NewCharOk = True
                Exit For
            End If
        Next i
    End Function


Private Function CharTest(ByVal ExpectedChar As Char, ByVal NewChar As Char) As Boolean

        Select Case ExpectedChar
            Case "c"
               If Not (NewChar Like "[+|""'<>().,:; {}~/\]^%@" Or NewChar = "-" Or NewChar
                     = "*" Or NewChar = "#" Or NewChar = "?" Or NewChar = "!" Or NewChar =
                     "[" Or NewChar = "]" Or NewChar = "*" Or NewChar = "/" Or NewChar =
                     "+") Then
                     CharTest = True
                End If

            Case "o"
                If NewChar = "+" Or NewChar = "*" Or NewChar = "-" Or NewChar = "/" Then
                    CharTest = True
                End If
            Case "n"
                If IsNumeric(NewChar) Then
                    CharTest = True
                End If
            Case "("
                If NewChar = "(" Then
                    CharTest = True
                End If
            Case ")"
                If NewChar = ")" Then
                    CharTest = True
                End If
        End Select

 End Function
```

Il metodo *calculateExp()* prende in ingresso la stringa contenente l'espressione aritmetica e fornisce in uscita il risultato del calcolo sotto forma di stringa. Considerato che le espressioni aritmetiche che verranno utilizzate nella definizione delle azioni sugli interi non saranno particolarmente complesse, che l'uso di questo tipo di azioni non sarà particolarmente frequente all'interno di un programma SFC e che UniSim non è un programma pensato per il controllo realtime, si è deciso di implementare un algoritmo che lavora direttamente sulla stringa contenente l'espressione, traducendo al momento dell'esecuzione dell'azione i nomi delle variabili intere in indirizzi fisici di memoria per la conseguente lettura del loro valore attuale.

L'implementazione del metodo *calculateExp()* e del metodo privato *calculate()* usato per rendere più leggibile il codice è la seguente:

```
Public Function calculateExp(ByVal exp As String) As Integer
        'MsgBox("1 " & exp)
        Dim numPar As Integer = 0
        Dim parPos As Integer
        Dim c As String
        Dim i As Integer = 1
        Dim ParFound As Boolean = False

        While (True)
            c = Mid(exp, i, 1)
            While (c <> "")
                Select Case c
                    Case "("
                        ParFound = True
                        If numPar = 0 Then
                            parPos = i
                        End If
                        numPar = numPar + 1
                    Case ")"
                        numPar = numPar - 1
                      If numPar = 0 Then
                   Dim tmp As String = Mid(exp, parPos + 1, i - parPos - 1)
                            ' MsgBox("mid: " & tmp)

                            exp = exp.Remove(parPos - 1, i - parPos + 1)
                           exp = exp.Insert(parPos - 1, calculateExp(tmp))
                            ' MsgBox(exp)

                        End If

                End Select
                i = i + 1
                c = Mid(exp, i, 1)
            End While
            If (Not ParFound) Then
                Exit While
            End If
            ParFound = False
            i = 1
            numPar = 0
        End While
        calculateExp = calculate(exp)
    End Function
```

```
Private Function calculate(ByVal exp As String) As String
        ' MsgBox(exp)

        Dim c As String, tmp As String, tmp2 As String
        Dim i As Integer = 1
        Dim opCount As Integer = 0
        Dim result As Integer

        While i < exp.Length

            c = Mid(exp, i, 1)
            If (c = "*" Or c = "/" Or c = "+" Or c = "-") Then
                opCount = opCount + 1
                tmp = c
            End If
            i = i + 1
        End While

        If (opCount = 0) Then
            If (IsNumeric(exp)) Then
                calculate = exp
                Exit Function
            Else
                Dim var As IntegerVariable
                var = m_pouInterface.FindVariableByName(exp)
                If IsNothing(var) Then
                    var = m_ResGlobalVariables.FindVariableByName(exp)
                End If
                If IsNothing(var) Then
                    Throw New Exception("Variable " & exp & " not found")
                    Exit Function
                End If
                exp = var.ReadActValue
                calculate = exp
                Exit Function
            End If

        End If



        If opCount = 1 Then
            Dim firstVar As IntegerVariable
            Dim secondVar As IntegerVariable
            Dim first As String = exp.Substring(0, exp.IndexOf(tmp))
            If Not IsNumeric(first) Then
                firstVar = m_pouInterface.FindVariableByName(first)
                If IsNothing(firstVar) Then
                    firstVar = m_ResGlobalVariables.FindVariableByName(first)
                End If
                If IsNothing(firstVar) Then
                    Throw New Exception("Variable " & first & " not found")
                    Exit Function
                End If
                first = firstVar.ReadActValue
            End If
            Dim second As String = exp.Substring(exp.IndexOf(tmp) + 1)
            If Not IsNumeric(second) Then
                secondVar = m_pouInterface.FindVariableByName(second)
                If IsNothing(secondVar) Then
                    secondVar = m_ResGlobalVariables.FindVariableByName(second)
                End If
                If IsNothing(secondVar) Then
                    Throw New Exception("Variable " & second & " not found")
                    Exit Function
                End If
```

```
                second = secondVar.ReadActValue
            End If
            ' MsgBox(first & " " & second)
            Dim firstOp As Integer = first
            Dim secondOp As Integer = second
            Select Case tmp
                Case "*"
                    Dim res As Integer = firstOp * secondOp
                    calculate = res
                Case "/"
                    Dim res As Integer = firstOp / secondOp
                    calculate = res
                Case "+"
                    Dim res As Integer = firstOp + secondOp
                    calculate = res
                Case "-"
                    Dim res As Integer = firstOp - secondOp
                    calculate = res
            End Select
            Exit Function
        End If

        i = 1
        c = Mid(exp, i, 1)

        If (exp.IndexOf("+") > 0 Or exp.IndexOf("-") > 0) Then
            While (c <> "")
                Select Case c
                    Case "+", "-"
                        tmp = exp.Substring(0, i - 1)
                        tmp2 = exp.Substring(i)

                        exp = exp.Remove(i, exp.Length - i)
                        exp = exp.Insert(i, calculate(tmp2))
                        exp = exp.Remove(0, i - 1)
                        exp = exp.Insert(0, calculate(tmp))
                End Select

                i = i + 1
                c = Mid(exp, i, 1)
            End While

        Else
            While (c <> "")
                Select Case c
                    Case "*", "/"
                        tmp = exp.Substring(0, i - 1)
                        ' tmp2 = exp.Substring(i)

                        'exp = exp.Remove(i, exp.Length - i)
                        'exp = exp.Insert(i, calculate(tmp2))
                        exp = exp.Remove(0, i - 1)
                        exp = exp.Insert(0, calculate(tmp))
                End Select

                i = i + 1
                c = Mid(exp, i, 1)
            End While

        End If



        calculate = calculate(exp)


    End Function
```

calculateExp() scorre la stringa contenente l'espressione invocando se stessa per ogni coppia di parentesi trovate, fino ad arrivare alla coppia di parentesi più interna, e sostituendo la parte di espressione passata in ingesso con il risultato fornito in uscita. Quando non ci sono altre parentesi, viene invocato il metodo *calculate()* che cerca gli operatori e invoca se stesso per ogni operatore trovato, sostituendo anch'esso la parte di espressione passata in ingresso con il risultato fornito in uscita. Quando *calculate()* riceve in ingresso una parte di espressione contenente solo la variabile, ne viene letto il valore attuale e sostituito al nome di quest'ultima.

Con questo algoritmo vengono calcolate prima le parti dell'espressione contenute tra le parentesi più interne sostituendole con il valore restituito dal calcolo, risalendo via via fino alla coppia di parentesi più esterna e, dunque, all'espressione completa che a questo punto contiene solo numeri e operatori. L'espressione, infine, viene calcolata.

Per dare la priorità agli operatori di moltiplicazione e divisione *calculate()*, quando trova l'operaore di addizione o sottrazione, chiama se stessa passando in ingresso e sostituendo sia la parte a sinistra dell'operatore che quella a destra:

```
[…]
If (exp.IndexOf("+") > 0 Or exp.IndexOf("-") > 0) Then
            While (c <> "")
                Select Case c
                    Case "+", "-"
                        tmp = exp.Substring(0, i - 1)
                        tmp2 = exp.Substring(i)

                        exp = exp.Remove(i, exp.Length - i)
                        exp = exp.Insert(i, calculate(tmp2))
                        exp = exp.Remove(0, i - 1)
                        exp = exp.Insert(0, calculate(tmp))
                End Select

                i = i + 1
                c = Mid(exp, i, 1)

            End While

[…]
```

Supponendo di avere, ad esempio, l'espressione aritmetica: *((5+7)/2)-4* i passi che verranno compiuti nel calcolo dell'espressione sono i seguenti:

- *((5+7)/2)-4*
- *(12)/2)-4*
- *6-4*
- *2*

4.2.2 Modifica dell'interfaccia grafica

Le modifiche più importanti apportate al codice per l'interfaccia grafica riguardano la classe *ActionDialogForm*, che implementa la finestra di dialogo per l'aggiunta di una nuova azione e la classe *GraphicalAction*, che si occupa, oltre all'esecuzione dell'azione, anche del disegno dell'elemento grafico dell'SFC che rappresenta l'azione.

Nella classe *ActionDialogForm* sono state aggiunte, con l'utilizzo dell'editor visuale, l'opzione "operation" nel combobox che permette di scegliere quale azione deve essere operata sulla variabile intera, la casella di testo per l'immissione dell'espressione e il bottone "Validate" per la sua validazione. L'utente non può aggiungere l'azione se non sono state verificate la correttezza sintattica e l'esistenza delle variabili utilizzate. L'implementazione del metodo che gestisce l'evento di click del bottone "Validate" è riportata di seguito:

```
Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button8.Click
        Dim arithExp As arithmeticExpression = New
arithmeticExpression(m_ResGlobalVariables, m_pouInterface)
        Try

            If (arithExp.Parse(TextBox4.Text)) Then
                arithExp.calculateExp(TextBox4.Text)
                Button1.Enabled = True
            Else
                MsgBox(arithExp.m_Error)
                Button1.Enabled = False
            End If
        Catch ex As Exception
```

```
            If ex.GetType.Name <> "OverflowException" Then
                Button1.Enabled = False
                MsgBox(ex.Message)
            End If
        End Try

    End Sub
```

L'oggetto *TextBox4* rappresenta la casella di testo per l'immissione dell'espressione mentre l'oggetto *Button1* rappresenta il bottone "Ok" che consente di aggiungere l'azione.

Il metodo crea una istanza della classe *arithmeticExpression*, verifica la correttezza sintattica dell'espressione invocando il metodo *Parse()* e l'esistenza delle variabili utilizzate nell'espressione invocando il metodo *CalculateExp()*. Quest'ultimo genera una eccezione nel caso non sia stato possibile risolvere il nome della variabile in indirizzo fisico di memoria (variabile inesistente). L'eccezione di overflow, generata quando, ad esempio, si ha una divisione per zero, è ignorata in quanto non interessa in questo ambito il valore attuale delle variabili, nè il risultato del calcolo dell'espressione.

Se i controlli vanno a buon fine il bottone "Ok" viene abilitato.

Nel metodo che gestisce l'evento di click del bottone "Ok" la modifica permette di assegnare alla variabile che contiene il qualificatore dell'azione il valore "PO", che identifica l'azione sulla variabile intera definita con l'uso di una espressione aritmetica, e alla variabile arithExp la stringa contenente l'espressione, nel caso in cui l'utente abbia selezionato dal combobox l'opzione "Operation":

```
[…]
      If m_type = "INT" Then
                  Select Case ComboBox2.Text
                      Case "Increase"
                          m_qualifier = "PI"
                      Case "Decrease"
                          m_qualifier = "PD"
                      Case "Reset"
                          m_qualifier = "PR"
                      Case "Operation"
                          m_qualifier = "PO"
                          ArithExp = TextBox4.Text
                  End Select

              End If

[…]
```

Nella classe *GraphicalAction* è stato modificato il metodo *Draw()* per consentire la corretta visualizzazione delle informazioni relative all'azione all'interno dell'elemento grafico che la rappresenta :

[…]

```
      Case "INT"
                    Select Case m_Qualifier
                        Case "PI"
                            Rect = m_GraphToDraw.MeasureString(m_Name & "++", m_TypeChar)
                            'Controlla se il nome è più largo
                            If Rect.Width < m_Width / 2 Then
                                m_GraphToDraw.DrawString(m_Name & "++", m_TypeChar, Br,
                                    StartXPoint + m_Width / 4 - (Rect.Width / 2) + 1,
                                    m_position.Y - (Rect.Height / 2))
                            Else
                                Dim NewRect As New Drawing.RectangleF(StartXPoint + m_Width
                                    / 20, CInt(m_position.Y - (Rect.Height / 2)),
                                    m_Width / 2 - (m_Width / 10), Rect.Height)
                                m_GraphToDraw.DrawString(m_Name & "++", m_TypeChar, Br,
                                    NewRect)
                            End If
                        Case "PD"
                            Rect = m_GraphToDraw.MeasureString(m_Name & "--", m_TypeChar)
                            'Controlla se il nome è più largo
                            If Rect.Width < m_Width / 2 Then
                                m_GraphToDraw.DrawString(m_Name & "--", m_TypeChar, Br,
                                    StartXPoint + m_Width / 4 - (Rect.Width / 2) + 1,
                                     m_position.Y - (Rect.Height / 2))
                            Else
                                Dim NewRect As New Drawing.RectangleF(StartXPoint + m_Width
                                    / 20, CInt(m_position.Y - (Rect.Height / 2)),
                                     m_Width / 2 - (m_Width / 10), Rect.Height)
                                m_GraphToDraw.DrawString(m_Name & "--", m_TypeChar, Br,
                                    NewRect)
                            End If
                        Case "PR"
                            Rect = m_GraphToDraw.MeasureString("R " & m_Name, m_TypeChar)
                            'Controlla se il nome è più largo
                            If Rect.Width < m_Width / 2 Then
                                m_GraphToDraw.DrawString("R " & m_Name, m_TypeChar, Br,
                                    StartXPoint + m_Width / 4 - (Rect.Width / 2) + 1,
                                    m_position.Y - (Rect.Height / 2))
                            Else
                                Dim NewRect As New Drawing.RectangleF(StartXPoint + m_Width
                                    / 20, CInt(m_position.Y - (Rect.Height / 2)),
                                    m_Width / 2 - (m_Width / 10), Rect.Height)
                                   m_GraphToDraw.DrawString("R " & m_Name, m_TypeChar, Br,
                                    NewRect)
                            End If
                        Case "PO"
                            Dim tmpString As String
                            tmpString = m_Name & ":=" & m_ArithExp
                            Rect = m_GraphToDraw.MeasureString(tmpString, m_TypeChar)
                            'Controlla se il nome è più largo
                            If Rect.Width < m_Width / 2 Then
                                m_GraphToDraw.DrawString(tmpString, m_TypeChar, Br,
                                    StartXPoint + m_Width / 4 - (Rect.Width / 2) + 1,
                                    m_position.Y - (Rect.Height / 2))
                            Else
                                Dim NewRect As New Drawing.RectangleF(StartXPoint + m_Width
                                    / 20, CInt(m_position.Y - (Rect.Height / 2)),
                                    m_Width / 2 - (m_Width / 10), Rect.Height)
                                m_GraphToDraw.DrawString(tmpString, m_TypeChar, Br,
                                    NewRect)
                            End If

                End Select
```

[…]

L'attributo *m_ArithExp* di *GraphicalAction* contiene l'espressione aritmetica che viene assegnata al momento dell'aggiunta o della modifica dell'azione (ad esempio l'utente sceglie di modificare l'espressione o di cambiare una azione di tipo diverso in una che utilizza l'espressione). Quando l'utente sceglie di aggiungere una azione, la classe *FormSfc*, che implementa l'editor del programma SFC, utilizzando la classe *ActionDialogForm*, preleva le informazioni fornite dall'utente e invoca il metodo privato *StoreActions()* passando queste informazioni come parametri di ingresso. *StoreActions()* invoca il metodo *AddActionToSelectedSteps()* della classe *Sfc* che contiene il corpo (body) del programma SFC. È stato necessario, quindi, aggiungere un ulteriore parametro ai metodi *StoreActions()* di *FormSfc* e *AddActionToSelectedSteps()* di *Sfc* che tiene traccia dell'espressione aritmetica:

```
Private Function StoreActions(ByVal Nome As String, ByVal Qual As String,
                              ByVal Var As BaseVariable, ByVal VarInd As
                              BaseVariable, ByVal Time As TimeSpan, ByRef RefSfc
                             As Sfc, ByVal StepToForces As GraphicalStepsList,
                             ByVal ArithExp As String)
    m_Sfc.AddActionToSelectedSteps(Nome, Qual, Var, VarInd, Time, RefSfc,
                                   StepToForces, ArithExp)

    RefreshTreeStruct()

End Function
```

Modifiche analoghe sono state fatte per le classi *GraphicalStepList*, che contiene la lista delle fasi, *GraphicalStep*, che rappresenta la fase e *ActionBlock* che contiene le azioni legate ad una fase. Quest' ultima crea l'oggetto *GraphicalAction* passando al suo costruttore, tra le altre informazioni, anche la stringa contenente l'espressione aritmetica, nel caso sia stata definita.

Nel caso in cui l'utente decida di cambiare una o più informazioni che definiscono una azione, facendo doppio click sul rettangolo disegnato dall'istanza della classe *GraphicalAction* che la rappresenta, la classe *FormSfc* gestisce l'evento usando la classe *ActionDialogForm* e assegnado di nuovo all'istanza di *GraphicalAction* le informazioni fornite:

```
Private Sub Panel1_DoubleClick(ByVal sender As Object, ByVal e As
            System.EventArgs) Handles Panel1.DoubleClick
     If EditingMode Then
     If CurrentOperation = Operation.Selected And m_Sfc.CountSelectedElement = 1
     And Not MultipleSelection Then
               Dim Obj As Object = m_Sfc.ReadObjectSelected

               Select Case Obj.GetType.Name
```

[...]

```
Case "GraphicalAction"
     Dim ActionDlgForm As New ActionDialogForm(m_Sfc.ResGlobalVariables,
     m_Pou.PouInterface, m_Pou.Pous, m_Pou.Name)
     Dim a As GraphicalAction
     ActionDlgForm.m_name = Obj.Name
     ActionDlgForm.m_qualifier = Obj.Qualifier
     ActionDlgForm.m_time = Obj.Time

     If Obj.Qualifier = "PO" Then
           ActionDlgForm.ArithExp = Obj.ReadArithExp()
     Else : ActionDlgForm.ArithExp = ""

     End If
```

[...]

```
Dim ResultDialog As System.Windows.Forms.DialogResult =
     ActionDlgForm.ShowDialog()
     If ResultDialog = DialogResult.OK Then
           Obj.Qualifier = ActionDlgForm.m_qualifier
           Obj.Name = ActionDlgForm.m_name
           Obj.Time = ActionDlgForm.m_time
           Obj.SetArithExp(ActionDlgForm.ArithExp)
           Obj.Type = ActionDlgForm.m_type
```

[...]

Il metodo *setArithExp()* è stato aggiunto alla classe *GraphicalAction* per consentire l'aggiornamento della stringa contenente l'espressione:

```
Public Sub SetArithExp(ByRef ArithExp As String)
       m_ArithExp = ArithExp

    End Sub
```

4.2.3 Modifica del motore di simulazione

Per quanto riguarda la parte di simulazione l'unico metodo da modificare è *ExecuteIfIsPulseAction()* della classe *GraphicalAction* la cui implementazione è riportata di seguito con evidenziate le aggiunte di codice:

```
Public Sub ExecuteIfIsPulseAction()
        'Esegue l'azione se è  S, R o P
        If Not IsNothing(m_Var) Then 'La variabile può essere stata eliminata
            Try
                Select Case m_Qualifier
                    Case "S"    'Set
                        m_Var.SetValue(True)
                    Case "R"    'Reset
                        m_Var.SetValue(False)
                    Case "P"    'Pulse
                        m_Var.SetValue(True)
                        m_Var.SetValue(False)
                    Case "PI"
                        m_Var.IncreaseValue(m_Var.ReadActValue)
                    Case "PD"
                        m_Var.DecreaseValue(m_Var.ReadActValue)
                    Case "PR"
                        m_Var.ResetValue()

                    Case "PO"
                     Dim exp As New  arithmeticExpression(
                                     m_Sfc.ResGlobalVariables,
                                     m_Sfc.PouInterface)
                        m_Var.SetActValue(exp.calculateExp(m_ArithExp))


                End Select
            Catch ex As System.Exception
            End Try
        End If

        Select Case m_Qualifier
            '----------------------------
            'Blocco aggiuntivo alla norma

        Case "ifr"    'Forzature impulsiva
            If Not IsNothing(m_Sfc) Then
                m_Sfc.ForceMeImpulsively(m_StepToForces)
            End If 'Forzatura impulsiva
            '-------------------------
        End Select


    End Sub
```

Nel caso il qualificatore dell'azione è "PO" si tratta, come già visto, di una azione in cui è stata definita un espressione aritmetica. Il codice aggiunto assegna alla variabile intera di riferimento come valore attuale il risultato del calcolo dell'espressione.

4.2.4 Modifica dell'import/export del progetto

Per permettere l'importazione e l'esportazione dei progetti che utilizzano le espressioni aritmetiche è stato necessario modificare lo XML Schema aggiungendo all'interno dell'elemento "ActionBlock": il tipo di azione "PO", nella definizione dell'attributo "qualifier", e l'attributo "arithExp", che contiene l'espressione. L'estratto dello XML Schema interessato dalla modifica è il seguente:

```
[…]

<xsd:attribute name="qualifier" use="optional" default="N">
     <xsd:simpleType>
     <xsd:restriction base="xsd:NMTOKEN">
                        <xsd:enumeration value="N" />
                        <xsd:enumeration value="R" />
                        <xsd:enumeration value="S" />
                        <xsd:enumeration value="L" />
                        <xsd:enumeration value="D" />
                        <xsd:enumeration value="P" />
                        <xsd:enumeration value="PO" />
                        <xsd:enumeration value="DS" />
                        <xsd:enumeration value="DL" />
                        <xsd:enumeration value="SD" />
                        <xsd:enumeration value="SL" />
                        <xsd:enumeration value="fr" />
                        <xsd:enumeration value="ifr" />
                        <xsd:enumeration value="sus" />
                        <xsd:enumeration value="st" />
                  </xsd:restriction>
            </xsd:simpleType>
       </xsd:attribute>
 <xsd:attribute name="duration" type="xsd:string" use="optional" />
 <xsd:attribute name="indicator" type="xsd:string" use="optional" />
 <xsd:attribute name="refsfc" type="xsd:string" use="optional" />
 <xsd:attribute name="stepstoforces" type="xsd:string" use="optional" />
 <xsd:attribute name="arithExp" type="xsd:string" use="optional" />


 […]
```

Nel metodo *xmlExport()* della classe *GraphicalAction*, che si occupa di salvare le informazioni riguardanti l'azione nel file XML, è stata aggiunta la scrittura

dell'espressione invocando il metodo *writeAttributeString()* della classe *XMLProjectWriter*:

```
Public Sub xmlExport(ByRef RefXMLProjectWriter As XmlTextWriter)

[…]


If m_ArithExp <> "" Then
  RefXMLProjectWriter.WriteAttributeString("arithExp", m_ArithExp)
End If

[…]
```

Mentre nel metodo *xmlImport()* è stata aggiunta la lettura:

```
Public Sub xmlImport(ByRef RefXmlProjectReader As XmlTextReader)

[…]


If RefXmlProjectReader.MoveToAttribute("arithExp") Then
   m_ArithExp = (RefXmlProjectReader.Value)
End If

[…]
```