The Stakeholder Modeling Language Reference Grammar Version 1.1

John Benjamin Cassel

> "...a computer language is not just a way of getting a computer to perform operations but rather that it is a novel formal medium for expressing ideas about methodology."
> -from *The Structure and Interpretation of Computer Programs* by Harold Abelson, Gerald Jay Sussman, and Julie Sussman [Abelson et al., 1996]

The Stakeholder modeling language was originally developed for Cassel [2011]. After this document, the primary change is to use 'policy' instead of 'anticipation', to better reflect the conventions in decision theory and reinforcement learning. This grammar uses ANTLR notation [Parr, 2007] and was developed in ANTLRWorks [Bovet and Parr, 2008].

The Grammar

```
model : 'model' ('current' absolutetime)?
      (structure|observation|stakeholder|sense|action
      |event|impact|deference|policy|criteria|dependence)+
```

Each program is a model. A model consists of one or more structures, observations, stakeholders, sensings, actions, events, impacts, deferences, policies, criteria, and dependences. It may be specified in reference to a particular time, taken as the time when the model was elicited.

```
absolutetime : 'absolutetime' '[' (integerstring)+ ']'
```

An absolute time is a date given by a sequence of one or more decreasingly large calendar units, where

the first corresponding absolute time is assumed. For example, `absolutetime [2005 11]` would be November 1, 2005, midnight.

```
criteria : 'criteria' criteriaName=STRING (desc=description)?
                      ('minimize'|'maximize') '.'
```

A criteria is identified by a string, and a description can optionally be provided. Criteria either correspond to rewards and other positive notions, or losses and other negative notions, and thus should be maximized or minimized, respectively.

```
structure : 'structure' stateName=STRING (structuredcloud)?
            (description)? ('current' stakeholderClause )? '.'
```

A structure is identified by a name, and a description can optionally be provided. Structures correspond to various aspects of overall states-of-affairs that are, could be, or could have been. Structures can be as simple as their name, or can be described by more complicated structures called structured clouds. Structures can be designated as currently the case according to one or more stakeholders.

```
observation : 'observation' observationName=STRING
              (desc=description)? ('current' stakeholderClause)? '.'
```

An observation is identified by a name, and a description can optionally be provided. Observations describe aspects of the world which could be directly observable. Observations can be designated as currently the case according to one or more stakeholders.

```
impact : 'impact' impactName=STRING (desc=description)?
         magnitude crit=STRING
         'to' stakeholderName=STRING 'if' ce=cloudexpression
         ('when' te=temporalExpression)?
         stakeholderClause '.'
```

An impact is some magnitude along a particular criteria felt by a stakeholder under particular conditions (where such descriptions would logically match a cloud expression). This may not be their own assessment, but the assessment of another stakeholder.

```
temporalExpression : temporalOp=STRING absolutetime
```

A temporal expression is a temporal operator compared against an absolute time.

```
magnitude : (qualMagnitude|weight)
```

Magnitudes can be expressed qualitatively or quantitatively.

```
stakeholder : 'stakeholder' stakeholderName=STRING (structuredcloud)?
              (description)? ('current' stakeholderClause )? '.'
```

A stakeholder is an individual in the overall state of affairs that may be impacted by events and have assertions and opinions about what is the case.

```
action : 'action' actionName=STRING  (desc=description)?
         ('typically' duration)? '.'
```

An action is something a stakeholder can do, sometimes with a characteristic duration which will carry to events the action is undertaken within, unless otherwise specified.

```
sense : 'observe' name=STRING observationName=STRING 'when'
        stateName=STRING ('with' prob=likelihood)?
        'according' 'to' stakeholderName=STRING '.'
```

A particular observation is sensed when a given state occurs, possibly only with a given likelihood, according to a particular stakeholder. The default likelihood for all units is almost certain, reflecting the convention of speaking deterministically unless otherwise qualified.

```
event : 'event' eventName=STRING  'if' orc=orclause
        'then' var c1=consequence ('and'  var c2=consequence )*
         ('with' prob=likelihood )?
         'according' 'to' stakeholderName=STRING
                     (',' skName2=STRING)*
        '.'
```

If some events happen or stakeholders undertake actions, given the current state of affairs, then some new state of affairs comes into effect, possibly only with a given likelihood, according to a particular stakeholder.

```
dependence : 'depending' name=STRING 'on'
             (compositeVaryingTerm=jointDependenceTerm
                  | singleVaryingTerm=STRING)
     'dependent' (compositeVaryingTerm=jointDependenceTerm
                              | singleVaryingTerm=STRING)
      ('mutually')? ('exclusive'|'independent' | likelihood) '.'
```

For all units for which likelihood may be expressed, it may also be appropriate to express a conditional likelihood between them and units of a similar type. These conditional likelihoods can be mutually exclusive, mutually independent, or dependent to a given degree. The current default is mutual independence.

```
jointDependenceTerm : 'joint' '[' term=STRING
                      (',' term2=STRING)* ']'
```

Joint terms reflect a composite condition over which conditional likelihoods can be expressed.

```
consequence : (('becomes' result=structuredcloud
                ('from' from=structuredcloud )? )
               | 'stops' 'being' result2=structuredcloud
               | 'shifts' cloudMathExpr (',' cloudMathExpr)*)
              ('within' duration)?
```

Consequences cause structures to transition from one state of affairs to another, usually as the result of an event. These consequences can optionally take some duration.

```
cloudMathExpr : tag=STRING '=' argtag=STRING
                ('+'|'-'|'*') value=weight
```

Cloud math expressions take the weight of a argument tag from a structure, change it according to some mathematical expression, and set the value of a tag in the structure with that new value.

```
duration: (weight calendarUnits ('varying' duration)?)
          | calendarUnits
```

A duration is a quantitative calendar unit, possibly with another duration as a usual variation.

```
calendarUnits : ('nanoseconds'|'milliseconds'|'seconds'|'minutes'
                |'hours'|'days'|'years'|'decades'|'centuries'
                |'millenia')
```

Calendar units are conventional measures of time.

```
structuredcloud : cloudchild
```

A structured cloud is a cloud-child with no parents.

```
cloudchild : '(' (wt)* (cloudchild )*  ')'
```

A cloud child is a set of weighted tags and a set of cloud children.

```
wt : STRING (':' weight )?

weight : weightstring

weightstring : ('-')?
       (((INTEGER )+ ('.'  (INTEGER )* )?)
        | ('.'  (INTEGER)+ ))
```

Weighted tags are pairs of names with numerical values.

```
orclause : andclause ('or' andclause)*

andclause : notclause ('and' notclause)*

notclause : 'not' notclause | parenclause

parenclause : '(' orclause ')' | condition
              | 'True' | 'False'
```

Boolean expressions are supported with precedence given to parenthesis, then logical *not*, then logical *and*, and finally logical *or*.

```
condition : conditionalvariable STRING
            '[' (value (',' value)* )? ']'
```

```
conditionalvariable  : VARIABLE

value : var  | constant

constant : STRING

var : VARIABLE
```

Conditions are predicates on variables and constants.

```
cloudexpression : '(' (var
                       | cloudexpression
                       | STRING)* ')'
```

Cloud expressions are logical conditions applicable to structured clouds.

```
deference : 'defer' 'to' knowledgeHolder=STRING
            'on' phenomena=STRING stakeholderClause
            ('with' prob=likelihood)? '.'
```

If a given state, observation, event, or action has taken place, one stakeholder trusts or distrusts the assessment of another stakeholder.

```
policy :
    'policy' name=STRING policyEvent
    ('and' policyEvent)*
    ('with' prob=likelihood)?
    ('when' consequence)?
    stakeholderClause '.'

policyEvent :
        'event' eventName=STRING   'happens'
        | 'stakeholder' stakeholderName=STRING
          'will' actionName=STRING
```

Some events will happen or some stakeholders undertake actions with a given likelihood if a given condition holds, according to a particular stakeholder or stakeholders.

```
stakeholderClause : 'according' 'to'
                    stakeholderName=STRING (',' STRING)*
```

Many constructs are according to the particular testimony of a stakeholder or stakeholders.

```
likelihood : qualProb|quantProb|'unknown'
```

The assessment of likelihood can either be qualitative or quantitative.

```
qualMagnitude : ('no'|'insignificant'|'low'|'moderate'
                     |'high'|'extreme')?
                ('declining'|'rising')
```

The magnitude and direction of an impact can be expressed in a qualitative way.

```
qualProb : ('extremely'|'very'|'somewhat'|'moderately')?
           ('impossible'|'low'|'moderate'|'high'|'certain'
                       |'almost' ('certain'|'impossible'))
```

Probabilities and related quantities can be expressed in a qualitative way. Almost certain is reserved for if it is ontologically possible or impossible, but the observer is certain it will not occur.

```
quantProb : 'between' weight 'and' weight
            | ('about'|'near'|'exactly') weight
```

Probabilities can be given by a range or approximated as near a particular value.

```
description :  '"' (STRING|punct)+ '"'
```

Descriptions are double-quoted strings.

```
COMMENT :  '/*' (options {greedy=false;} : . )* '*/'
```

Additional comments, not included with in the model, can be included in the model text if surrounded by C-style comment delimiters.

```
num  : ('-')? INTEGER+ ('.' INTEGER+)?

VARIABLE :'$' ('A'..'Z'|'a'..'z'|'0'..'9')+
```

```
INTEGER : '0'..'9'

STRING : ('A'..'Z'|'a'..'z')('A'..'Z'|'a'..'z'|INTEGER|'-'|'_')*

punct  : ('.' |',' )

WHITESPACE : (' '|'\t'|'\r'|'\n') {$channel=HIDDEN; }
```

Numbers, strings, and punctuation are defined in terms of particular allowed characters. Whitespace characters help delimit tokens but are otherwise insignificant.

References

Abelson, H., Sussman, G. J., and Sussman, J. (1996). *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, 2nd edition.

Bovet, J. and Parr, T. (2008). Antlrworks: an antlr grammar development environment. *Software: Practice and Experience*, 38:1305–1332.

Cassel, J. B. (2011). Addressing risk governance deficits through scenario modeling practices. Master's thesis, OCAD University. Available from http://john-benjamin-cassel.com/FinalProject.pdf.

Parr, T. (2007). *The Definitive ANTLR Reference: Building Domain-Specific Languages (Pragmatic Programmers)*. Pragmatic Bookshelf.