PROPRIETARY RIGHTS STATEMENT

THIS DOCUMENT CONTAINS INFORMATION, WHICH IS PROPRIETARY TO THE DANSE CONSORTIUM. NEITHER THIS DOCUMENT NOR THE INFORMATION CONTAINED HEREIN SHALL BE USED, DUPLICATED OR COMMUNICATED BY ANY MEANS TO ANY THIRD PARTY, IN WHOLE OR IN PARTS, EXCEPT WITH THE PRIOR WRITTEN CONSENT OF THE DANSE CONSORTIUM THIS RESTRICTION LEGEND SHALL NOT BE ALTERED OR OBLITERATED ON OR FROM THIS DOCUMENT. THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE SEVENTH FRAMEWORK PROGRAMME UNDER GRANT AGREEMENT n° 287716.

SEVENTH FRAMEWORK PROGRAMME

DANSE

Designing for adaptability and evolution in system of systems engineering

GCSL syntax, semantics and meta-model

D_6.3.3

Deliverable Information			
Nature	N/A	**Dissemination Level**	PU
Project	DANSE	**Contract Number**	INFSO-ICT-287716
Deliverable ID	D_6.3.3	**Date**	15/02/2015
Status	Final	**Version**	1.00
Contact Person	Valerio Senni	**Organisation**	ALES S.r.l.
Phone		**E-Mail**	valerio.senni@utsce.utc.com

AUTHORS TABLE

Name	Company	E-Mail
Benoît Boyer	INRIA	Benoit.Boyer@inria.fr
Jean Quilbeuf	INRIA	jean.quilbeuf@inria.fr
Christoph Etzien	OFFIS	christoph.etzien@offis.de
Marco Marazza	ALES S.r.l.	marco.marazza@utsce.utc.com
Valerio Senni	ALES S.r.l.	valerio.senni@utsce.utc.com
Francesca Stramandinoli	ALES S.r.l.	francesca.stramandinolil@utsce.utc.com
Thomas Peikenkamp	OFFIS	peikenkamp@offis.de

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
1.00	May 15, 2014	Initial version	all
2.00	January 16, 2015	New GCSL patterns to support MTBF	
3.00	February 6, 2015	Final version, transformed into a Deliverable	

CONTENT

1 Introduction

This working document reports on the work done by ALES, INRIA and OFFIS for the refinement of the GCSL syntax and semantics to support the integration of GCSL in the tool-chain. This document is not a replacement for the official Deliverable D6.3.2, which remains the reference document. This document contains only a refined version of the GCSL syntax and semantics, to be considered as the reference for tools implementation and users, thus replacing the syntax and semantics of D6.3.2.

1.1 Overview, Purpose and Scope

GCSL is a very expressive logic [ref. D6.3.2] parameterized by a language for expressing atomic propositions, based on OCL [http://www.omg.org/spec/OCL/]. Depending on the type of application and the version of the OCL language, it is possible to express various forms of atomic properties. The purpose of this document is to fix a reference version of GCSL syntax and semantics, including the fragment of OCL upon which it is based. For the purpose of the tool-chain integration, this document also reports about the development of a Meta-Model of GCSL. The final version of the GCSL language is going to be presented in full details in the forthcoming deliverable D6.3.3. The current document serves as a support to the development and integration of the tool-chain and as a reference for users to write contracts for use-cases.
Within DANSE, the GCSL language is supported by various analyses techniques. The set of GCSL operators supported by each analysis may vary, depending on the kind of analysis and the maturity of the underlying tools. The exact fragment of GCSL supported by the specific tool is going to be specified within the deliverable describing that tool.

2 GCSL syntax

2.1 GCSL contracts

Here we provide the formal grammar for the GCSL logic, based on the grammar from the PLASMA website https://project.inria.fr/plasma-lab/gcsl/, which has been extended with additional behavioural patterns in order to cover the GCSL deliverable D6.3.2. Note that the GCSL grammar does not specify which fragment of OCL it covers. In particular, it does not provide a specification for the terms OCL-coll, OCL-prop, and OCL-expr below. This is addressed in the next section.

```
GCSL ::=
   OCL-coll -> forAll ( variableName | pattern ) |
   OCL-coll -> exists ( variableName | pattern ) |
15 MEAN(OCL-expr,interval)  a-relop  OCL-expr  |
16 SUM(OCL-expr,interval)  a-relop  OCL-expr  |
17 PROD(OCL-expr,interval)  a-relop  OCL-expr  |
   pattern

pattern ::=
 1          [ OCL-prop ] implies [ OCL-prop ] holds forever |
 2         always [ OCL-prop ] |
 3         whenever [ OCL-prop ] occurs [ OCL-prop ] holds |
 4         [ OCL-prop ] implies [ OCL-prop ] during following interval |
 5         [ OCL-prop ] during interval raises [ OCL-prop ] |
 6         [ OCL-prop ] during interval implies [ OCL-prop ] during interval then [ OCL-prop ] during  interval  |
 7         [ OCL-prop ] occurs int times during interval raises [ OCL-prop ] |
 8         [ OCL-prop ] occurs  at most int times during interval |
 9         whenever [ OCL-prop ] occurs [ OCL-prop ] holds during following interval  |
10         whenever [ OCL-prop ] occurs [ OCL-prop ] implies [ OCL-prop ] during following  interval  |
11         whenever [ OCL-prop ] occurs [ OCL-prop ] does not occur during following interval  |
12         whenever [ OCL-prop ] occurs [ OCL-prop ] occurs within interval  |
13         always during interval [ OCL-prop ] has been true at least [ OCL-expr ] % of time
14         at the end of interval [ OCL-prop ] has been true at least [ OCL-expr ] % of time

interv ::= lp ( time ,)? time rp
lp ::=  [  |  (
rp ::=  ]  |  )
time ::= ℕ unit | inf
unit ::= s | min | hour | day | year   // year =365 days
```

2.2 OCL atomic propositions

The grammar below describes the fragment of OCL 2.0 (http://www.omg.org/spec/OCL/) that we support within GCSL. The grammar defines OCL propositions that are Boolean formulas, among which there are patterns that allow to quantify over collections of objects. It defines also OCL expressions, that are arithmetic expressions over integer and real values.

```
OCL-prop ::= true  |  false  |  path  |  not OCL-prop  |  OCL-prop  b-relop  OCL-prop  |
          OCL-expr  a-relop  OCL-expr  |
          if OCL-prop then OCL-prop else OCL-prop |
          OCL-coll -> forAll( variableName | OCL-prop ) |
          OCL-coll -> exists( variableName | OCL-prop ) |
          OCL-coll -> empty() |
          OCL-coll -> notempty()

b-relop ::= and | or | xor | implies | iff

a-relop ::= > | >= | = | <> | =< | <

OCL-expr ::= path | OCL-coll -> sum() | OCL-coll -> size() |
          if OCL-prop then OCL-expr else OCL-expr |
          const | - OCL-expr | ( OCL-expr ) |
          OCL-expr  a-binfn  OCL-expr

a-binfn ::= + | - | * | /

OCL-coll ::= itsClassNames  |  its(ClassName)  |
           connectedClassNames  |  connected(ClassName)  |
           basicTypeAttributeName  |
           objectName . OCL-coll  |  variableName . OCL-coll  |
           itsClassNames . OCL-coll  |  its(ClassName) . OCL-coll

path ::= basicTypeAttributeName  |  objectName . path  |  variableName . path

const ::= integers | reals
```

REMARKS:

basicTypeAttributeName is a name of an attribute of type Boolean, Int, Float, … (i.e. Primitive Data Types.)

itsClassName**s / its(**ClassName**)** are (equivalent) expressions used to navigate the relation of *containment*

connectedClassName**s** / **connected(**ClassName**)** are (equivalent) expressions used to navigate the relation of *being-connected-to (through a port)*

3 GCSL Semantics

From GCSL deliverable D6.3.2: "*Let* $\tau\colon \mathbb{N} \to \mathbb{R}$ *be a monotonically increasing time sequence, i.e. for all* $i \geq 0\colon \tau_i \leq \tau_{i+1}$ *and let* Σ *be an alphabet. A (infinite) sequence* $\sigma_0\sigma_1\sigma_2$ *... of symbols of* Σ *is called a trace. A trace* $(\sigma, \tau) = (\sigma_0, \tau_o)(\sigma_1, \tau_1)(\sigma_2, \tau_2)$ *where* σ *is a trace and* τ *is a time sequence is called a timed trace*." Behaviors are timed traces. The semantics of a GCSL formula is a (possibly infinite) set of behaviours. In this section we define the semantics of the GCSL temporal patterns by translation to corresponding B-LTL formulas. This is done following the paper [AiSoS2013] but with some modifications to B-LTL formulas corresponding to patterns 4,5,6 plus the addition of pattern 1 which is present in D6.3.2 but not in [AiSoS2013]. Below we assume that k is the current simulation bound: this bound constraints the patterns intervals bounds as indicated in the table below.

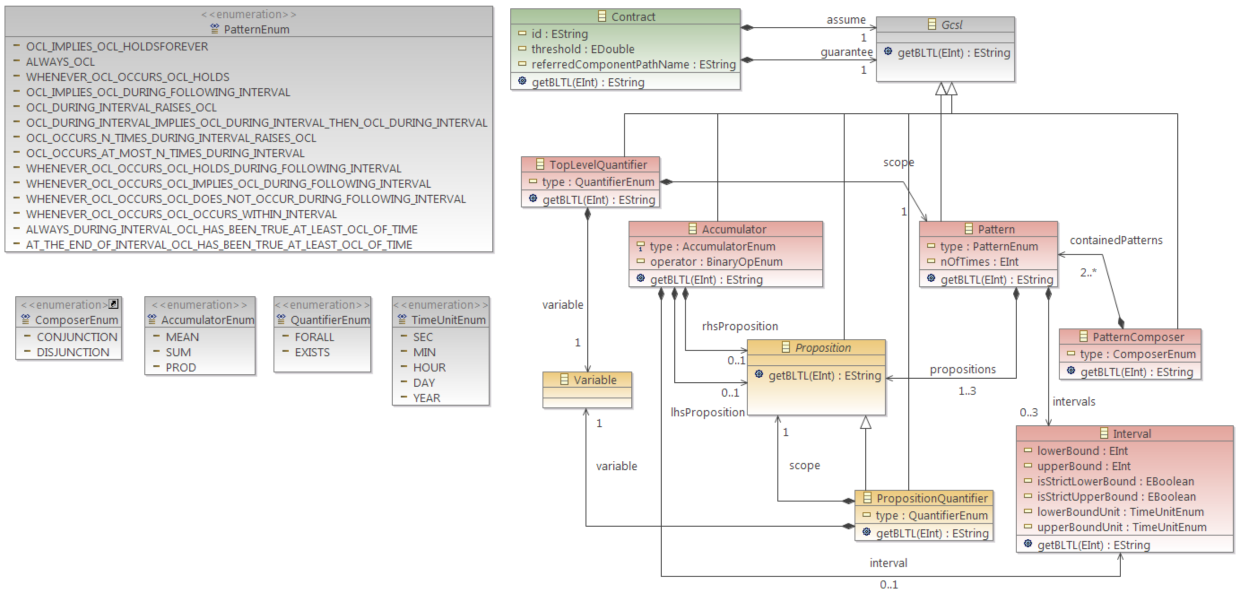
P.id	Patterns	B-LTL Semantics	Consistency
1	Ψ_1**implies** Ψ_2**holds forever**	$G_{\leq k}\ (\Psi_1 \rightarrow\ G_{\leq k}\ (\Psi_2))$	-
2	**always** Ψ	$G_{\leq k}\Psi$	-
3	**whenever** Ψ_1 **occurs** Ψ_2 **holds**	$G_{\leq k}(\Psi_1 \rightarrow \Psi_2)$	-
4	Ψ_1 **implies** Ψ_2 **during following** $[a, b]$	$X_{\leq a}G_{\leq b-a}(\Psi_1 \rightarrow\ \Psi_2)$	$a \leq b \leq k$
5	Ψ_1 **during** $[\mathrm{a}, \mathrm{b}]$ **raises** Ψ_2	$(X_{\leq a}G_{\leq b-a}\Psi_1) \rightarrow X_{\leq b}F_{\leq k-b}\Psi_2$	$a \leq b \leq k$
6	Ψ **during** $[a, b]$ **implies** Ψ_1 **during** $[a, c]$ **then** Ψ_2 **during** $[c, b]$	$(X_{\leq a}G_{\leq b-a}\Psi) \rightarrow (X_{\leq a}G_{\leq c-a}\Psi_1 \& X_{\leq c}G_{\leq b-c}\Psi_2)$	$a \leq c \leq b \leq k$
	EXTENDED Patterns		
7	Ψ_1 **occurs** n **times during** $[a, b]$ **raises** Ψ_2	$occ(\Psi_1, a, b) \geq n \rightarrow X_{\leq b}F_{\leq k-b}\Psi_2$	$a \leq b \leq k$
8	Ψ **occurs at most** n **times during** $[a, b]$	$occ(\Psi_1, a, b) \leq n$	$a \leq b \leq k$
13	**always during** $[a, b]$, Ψ **has been true at least** E **% of time**	$G_{\leq b}\ (TimeOf(\Psi) \geq \frac{E}{100} * \#Time \vee \#Time < a)$	$a \leq b \leq k$
14	**at the end of** $[a, b]$, Ψ **has been true at least** E **% of time**	$F_{\leq b}\ (TimeOf(\Psi) \geq \frac{E}{100} * b)$	$a \leq b \leq k$
	Patterns with SLIDING INTERVALS		
9	**whenever** Ψ_1 **occurs** Ψ_2 **holds during following** $[a, b]$	$G_{\leq k-b}(\Psi_1 \Rightarrow\ X_{\leq a}G_{\leq b-a}\Psi_2)$	$a \leq b \leq k$
10	**whenever** Ψ **occurs** Ψ_1 **implies** Ψ_2 **during following** $[a, b]$	$G_{\leq k-b}\big(\Psi \Rightarrow\ X_{\leq a}G_{\leq b-a}(\Psi_1 \Rightarrow \Psi_2)\big)$	$a \leq b \leq k$
11	**whenever** Ψ_1 **occurs** Ψ_2 **does not occur during following** $[a, b]$	$G_{\leq k-b}(\Psi_1 \Rightarrow\ X_{\leq a}G_{\leq b-a}\ \neg\Psi_2)$	$a \leq b \leq k$
12	**whenever** Ψ_1 **occurs** Ψ_2 **occurs within** $[a, b]$	$G_{\leq k-b}(\Psi_1 \Rightarrow\ X_{\leq a}F_{\leq b-a}\Psi_2)$	$a \leq b \leq k$

GCSL Meta-model

The latest version of the GCSL Meta Model can be retrieved from the DANSE SVN repository:

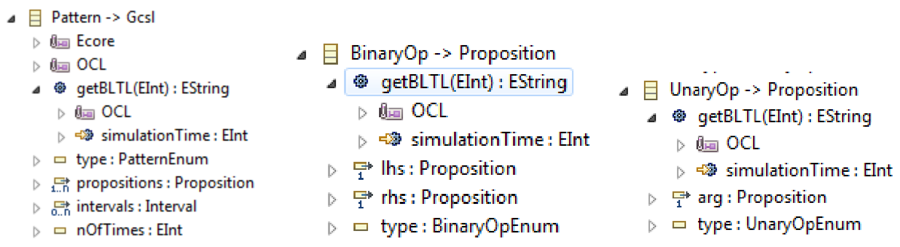
https://www.danse-ip.eu/redmine/projects/danse/repository/show/WP6/GCSL/software/eu.danse.gcslmm

The GCSL Meta Model supports the integration of the DANSE Analysis tools, in particular the integration of the Plasma Statistical Model Checker. For this reason, every class of this model contains a getBLTL function that allows translating the entire sub-tree starting from that class into an equivalent BLTL formula as indicated in Section 3. The getBLTL function has an integer argument containing the simulation time, which is required for proper translation into BLTL.



<<enumeration>> PatternEnum
OCL_IMPLIES_OCL_HOLDSFOREVER
ALWAYS_OCL
WHENEVER_OCL_OCCURS_OCL_HOLDS
OCL_IMPLIES_OCL_DURING_FOLLOWING_INTERVAL
OCL_DURING_INTERVAL_RAISES_OCL
OCL_DURING_INTERVAL_IMPLIES_OCL_DURING_INTERVAL_THEN_OCL_DURING_INTERVAL
OCL_OCCURS_N_TIMES_DURING_INTERVAL_RAISES_OCL
OCL_OCCURS_AT_MOST_N_TIMES_DURING_INTERVAL
WHENEVER_OCL_OCCURS_OCL_HOLDS_DURING_FOLLOWING_INTERVAL
WHENEVER_OCL_OCCURS_OCL_IMPLIES_OCL_DURING_FOLLOWING_INTERVAL
WHENEVER_OCL_OCCURS_OCL_DOES_NOT_OCCUR_DURING_FOLLOWING_INTERVAL
WHENEVER_OCL_OCCURS_OCL_OCCURS_WITHIN_INTERVAL
ALWAYS_DURING_INTERVAL_OCL_HAS_BEEN_TRUE_AT_LEAST_OCL_OF_TIME
AT_THE_END_OF_INTERVAL_OCL_HAS_BEEN_TRUE_AT_LEAST_OCL_OF_TIME
<<enumeration>> ComposerEnum
CONJUNCTION
DISJUNCTION
<<enumeration>> AccumulatorEnum
MEAN
SUM
PROD
<<enumeration>> QuantifierEnum
FORALL
EXISTS
<<enumeration>> TimeUnitEnum
SEC
MIN
HOUR
DAY
YEAR
Contract
id : EString
threshold : EDouble
referredComponentPathName : EString
getBLTL(EInt) : EString
assume
1
guarantee
1
Gcsl
getBLTL(EInt) : EString
TopLevelQuantifier
type : QuantifierEnum
getBLTL(EInt) : EString
scope
1
Accumulator
type : AccumulatorEnum
operator : BinaryOpEnum
getBLTL(EInt) : EString
Pattern
type : PatternEnum
nOfTimes : EInt
getBLTL(EInt) : EString
containedPatterns
2..*
PatternComposer
type : ComposerEnum
getBLTL(EInt) : EString
variable
1
rhsProposition
0..1
0..1
lhsProposition
Proposition
getBLTL(EInt) : EString
propositions
1..3
0..3
intervals
Variable
1
variable
1
scope
PropositionQuantifier
type : QuantifierEnum
getBLTL(EInt) : EString
Interval
lowerBound : EInt
upperBound : EInt
isStrictLowerBound : EBoolean
isStrictUpperBound : EBoolean
lowerBoundUnit : TimeUnitEnum
upperBoundUnit : TimeUnitEnum
getBLTL(EInt) : EString
interval
0..1

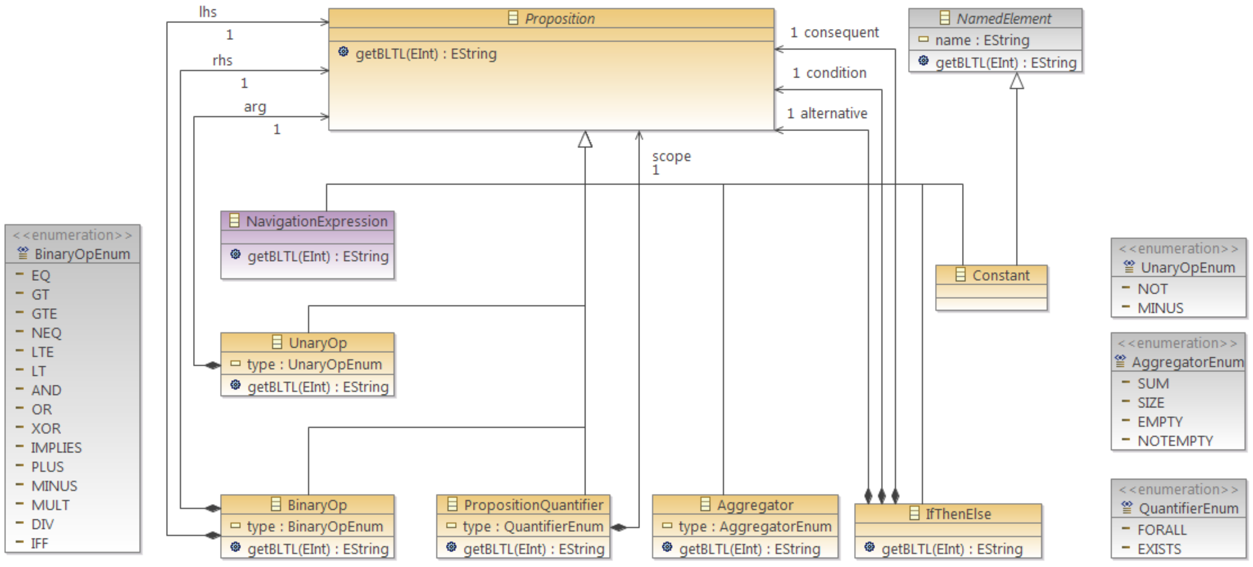
The picture above illustrates the modelling of a GCSL contract, having an ID of type string, a threshold of type double indicating the minimal probability of the contract truth, the referredComponentPathName of type string that contains the path to the Component to which the contract is attached and containing two GCSL assertions that are the Assume and the Guarantee parts of the contract. GCSL assertions are marked in red. They can be either TopLevelQuantifier or Accumulator (MEAN/SUM/PROD) or PatternComposer (AND/OR composition). They correspond to the syntactic elements provided in the grammar in Section 2.1. Then, nested, there are OCL Propositions, OCL quantifiers and variables. For some classes (Pattern, Binary Op and UnaryOp) the implementation of the getBLTL function was provided by using OCL constraints. This choice proved to be useful to allow effortless extension of the Patterns to support the needs of the users.



```
Pattern -> Gcsl
  Ecore
  OCL
  getBLTL(EInt) : EString
    OCL
    simulationTime : EInt
  type : PatternEnum
  propositions : Proposition
  intervals : Interval
  nOfTimes : EInt
```

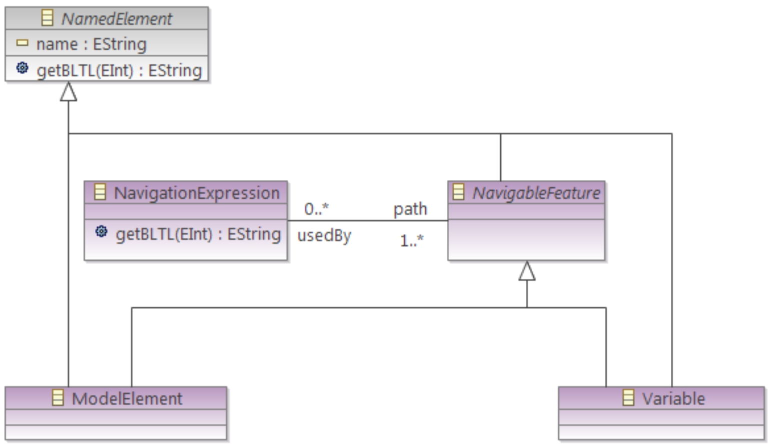
```
BinaryOp -> Proposition
  getBLTL(EInt) : EString
    OCL
    simulationTime : EInt
  lhs : Proposition
  rhs : Proposition
  type : BinaryOpEnum
```

```
UnaryOp -> Proposition
  getBLTL(EInt) : EString
    OCL
    simulationTime : EInt
  arg : Proposition
  type : UnaryOpEnum
```

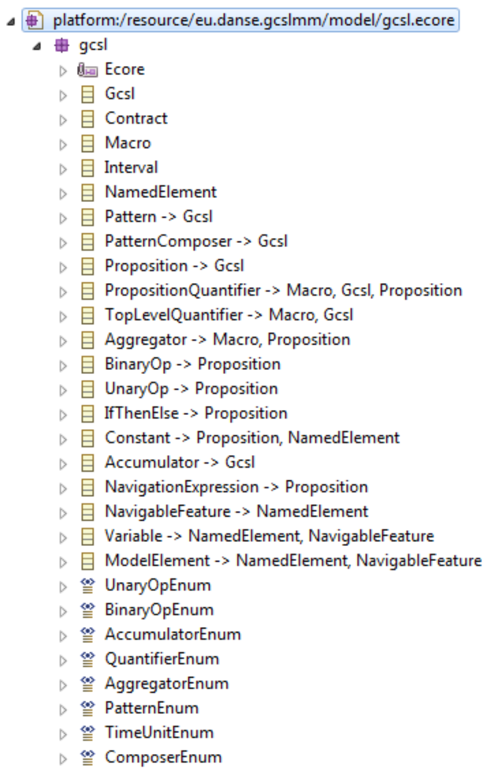


The picture above illustrates how OCL Propositions and Expressions are modelled. This part of the Meta Model is a little less restrictive than the grammar provided in Section 2.2. However, the GCSL grammar is tested by the GCSL Parser plugin and it will therefore produce correct instances of the Meta Model. The GCSL Parser plugin can be found in the DANSE repository at the following address:

https://www.danse-ip.eu/svn/danse/WP8/ToolsNetComponents/GCSLPlugins

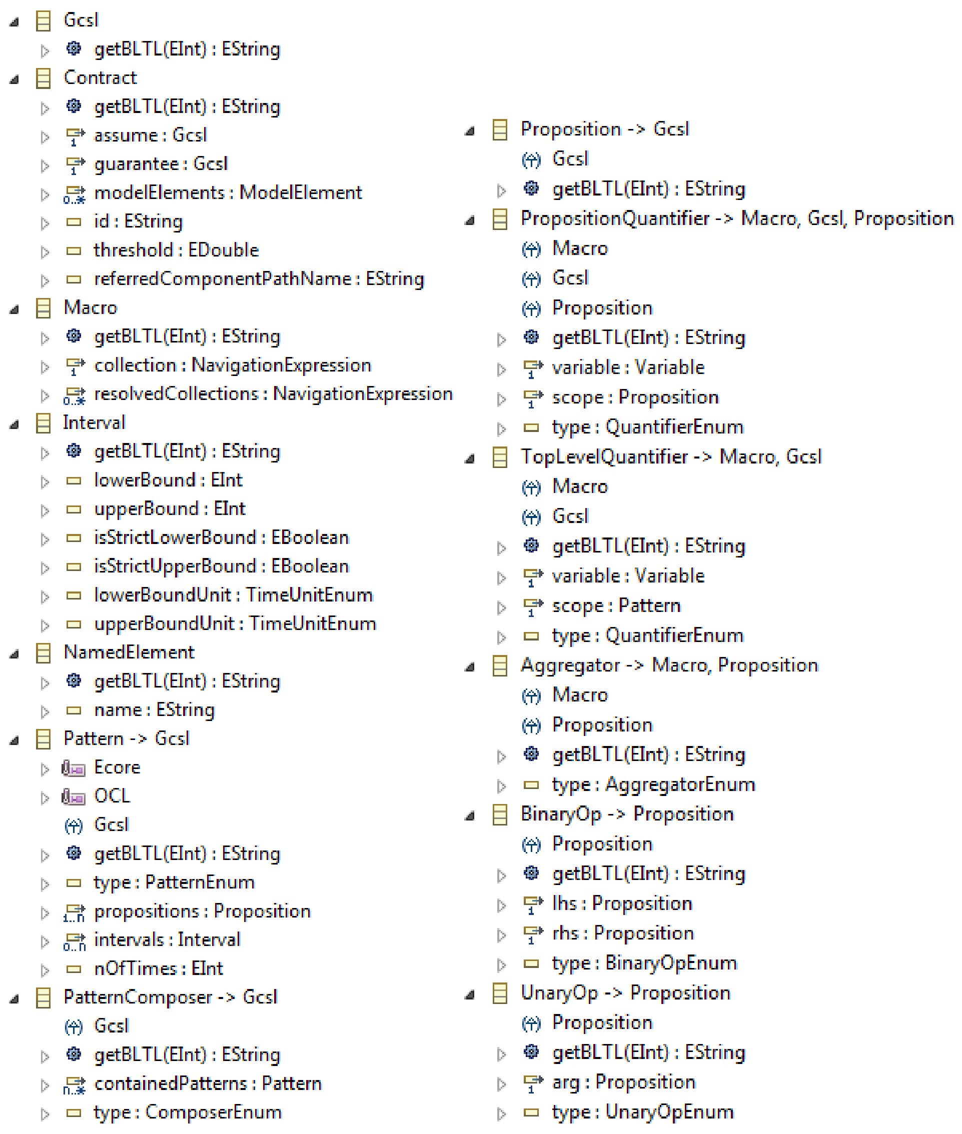


Finally, the picture above illustrates how Navigation Expressions are modelled: paths are sequences of navigable features that can be either model elements (i.e. Components and subcomponents identifiers) or variables (i.e. Components' and subcomponents' attributes).



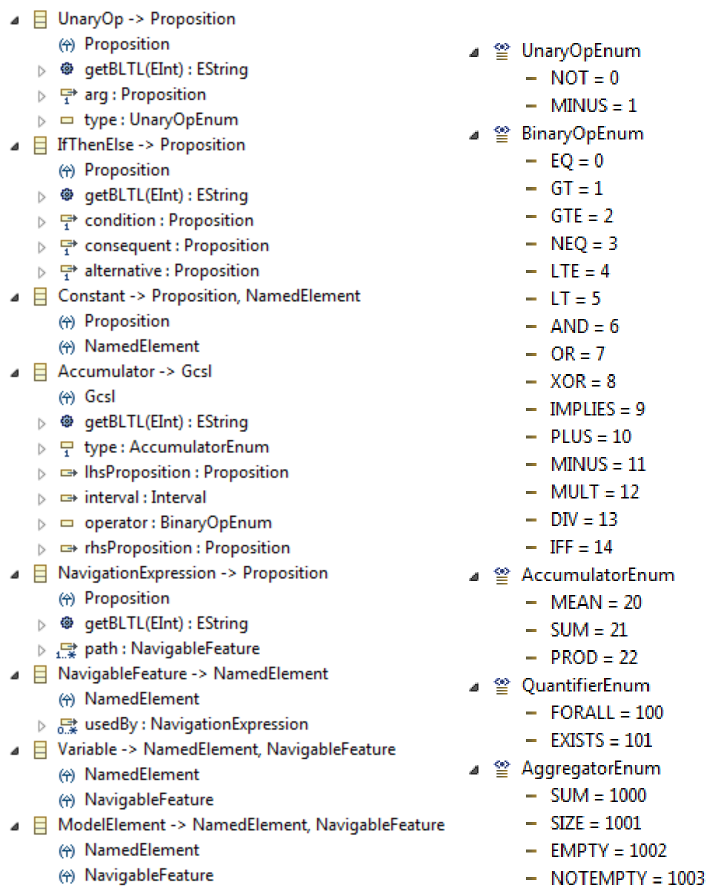
```
platform:/resource/eu.danse.gcslmm/model/gcsl.ecore
  gcsl
    Ecore
    Gcsl
    Contract
    Macro
    Interval
    NamedElement
    Pattern -> Gcsl
    PatternComposer -> Gcsl
    Proposition -> Gcsl
    PropositionQuantifier -> Macro, Gcsl, Proposition
    TopLevelQuantifier -> Macro, Gcsl
    Aggregator -> Macro, Proposition
    BinaryOp -> Proposition
    UnaryOp -> Proposition
    IfThenElse -> Proposition
    Constant -> Proposition, NamedElement
    Accumulator -> Gcsl
    NavigationExpression -> Proposition
    NavigableFeature -> NamedElement
    Variable -> NamedElement, NavigableFeature
    ModelElement -> NamedElement, NavigableFeature
    UnaryOpEnum
    BinaryOpEnum
    AccumulatorEnum
    QuantifierEnum
    AggregatorEnum
    PatternEnum
    TimeUnitEnum
    ComposerEnum
```

Figure 1 Overview of the GCSL Meta-Model components



```
Gcsl
    getBLTL(EInt) : EString
Contract
    getBLTL(EInt) : EString
    assume : Gcsl
    guarantee : Gcsl
    modelElements : ModelElement
    id : EString
    threshold : EDouble
    referredComponentPathName : EString
Macro
    getBLTL(EInt) : EString
    collection : NavigationExpression
    resolvedCollections : NavigationExpression
Interval
    getBLTL(EInt) : EString
    lowerBound : EInt
    upperBound : EInt
    isStrictLowerBound : EBoolean
    isStrictUpperBound : EBoolean
    lowerBoundUnit : TimeUnitEnum
    upperBoundUnit : TimeUnitEnum
NamedElement
    getBLTL(EInt) : EString
    name : EString
Pattern -> Gcsl
    Ecore
    OCL
    Gcsl
    getBLTL(EInt) : EString
    type : PatternEnum
    propositions : Proposition
    intervals : Interval
    nOfTimes : EInt
PatternComposer -> Gcsl
    Gcsl
    getBLTL(EInt) : EString
    containedPatterns : Pattern
    type : ComposerEnum
Proposition -> Gcsl
    Gcsl
    getBLTL(EInt) : EString
PropositionQuantifier -> Macro, Gcsl, Proposition
    Macro
    Gcsl
    Proposition
    getBLTL(EInt) : EString
    variable : Variable
    scope : Proposition
    type : QuantifierEnum
TopLevelQuantifier -> Macro, Gcsl
    Macro
    Gcsl
    getBLTL(EInt) : EString
    variable : Variable
    scope : Pattern
    type : QuantifierEnum
Aggregator -> Macro, Proposition
    Macro
    Proposition
    getBLTL(EInt) : EString
    type : AggregatorEnum
BinaryOp -> Proposition
    Proposition
    getBLTL(EInt) : EString
    lhs : Proposition
    rhs : Proposition
    type : BinaryOpEnum
UnaryOp -> Proposition
    Proposition
    getBLTL(EInt) : EString
    arg : Proposition
    type : UnaryOpEnum
```

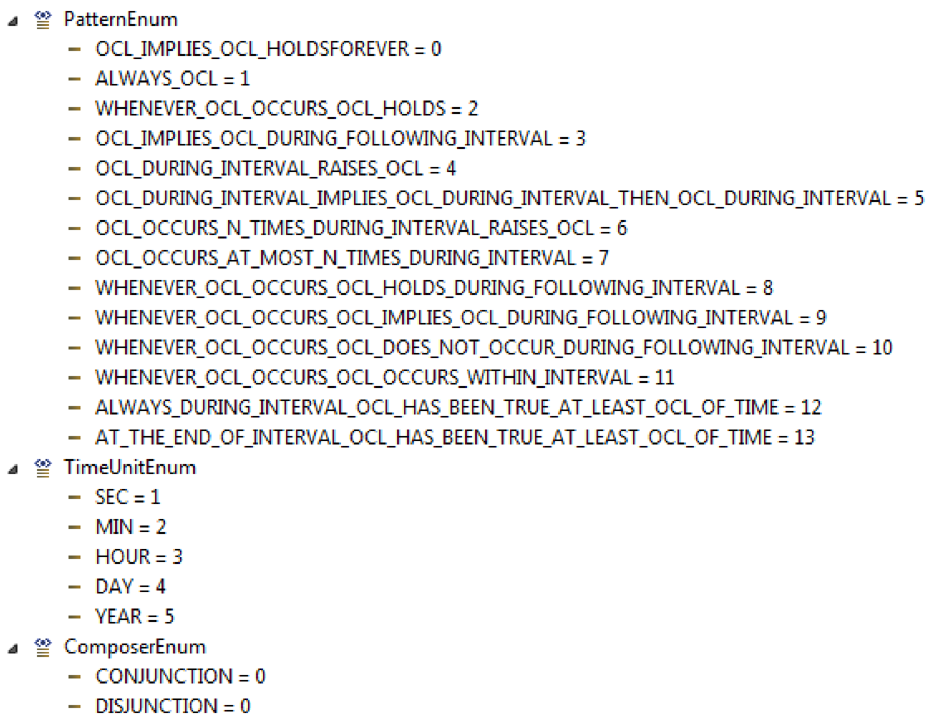
Figure 2. GCSL Meta Model Exploded View – part 1



```
UnaryOp -> Proposition
    Proposition
    getBLTL(EInt) : EString
    arg : Proposition
    type : UnaryOpEnum
IfThenElse -> Proposition
    Proposition
    getBLTL(EInt) : EString
    condition : Proposition
    consequent : Proposition
    alternative : Proposition
Constant -> Proposition, NamedElement
    Proposition
    NamedElement
Accumulator -> Gcsl
    Gcsl
    getBLTL(EInt) : EString
    type : AccumulatorEnum
    lhsProposition : Proposition
    interval : Interval
    operator : BinaryOpEnum
    rhsProposition : Proposition
NavigationExpression -> Proposition
    Proposition
    getBLTL(EInt) : EString
    path : NavigableFeature
NavigableFeature -> NamedElement
    NamedElement
    usedBy : NavigationExpression
Variable -> NamedElement, NavigableFeature
    NamedElement
    NavigableFeature
ModelElement -> NamedElement, NavigableFeature
    NamedElement
    NavigableFeature
```

```
UnaryOpEnum
    NOT = 0
    MINUS = 1
BinaryOpEnum
    EQ = 0
    GT = 1
    GTE = 2
    NEQ = 3
    LTE = 4
    LT = 5
    AND = 6
    OR = 7
    XOR = 8
    IMPLIES = 9
    PLUS = 10
    MINUS = 11
    MULT = 12
    DIV = 13
    IFF = 14
AccumulatorEnum
    MEAN = 20
    SUM = 21
    PROD = 22
QuantifierEnum
    FORALL = 100
    EXISTS = 101
AggregatorEnum
    SUM = 1000
    SIZE = 1001
    EMPTY = 1002
    NOTEMPTY = 1003
```

Figure 3. GCSL Meta Model Exploded View – part 2



```
PatternEnum
    OCL_IMPLIES_OCL_HOLDSFOREVER = 0
    ALWAYS_OCL = 1
    WHENEVER_OCL_OCCURS_OCL_HOLDS = 2
    OCL_IMPLIES_OCL_DURING_FOLLOWING_INTERVAL = 3
    OCL_DURING_INTERVAL_RAISES_OCL = 4
    OCL_DURING_INTERVAL_IMPLIES_OCL_DURING_INTERVAL_THEN_OCL_DURING_INTERVAL = 5
    OCL_OCCURS_N_TIMES_DURING_INTERVAL_RAISES_OCL = 6
    OCL_OCCURS_AT_MOST_N_TIMES_DURING_INTERVAL = 7
    WHENEVER_OCL_OCCURS_OCL_HOLDS_DURING_FOLLOWING_INTERVAL = 8
    WHENEVER_OCL_OCCURS_OCL_IMPLIES_OCL_DURING_FOLLOWING_INTERVAL = 9
    WHENEVER_OCL_OCCURS_OCL_DOES_NOT_OCCUR_DURING_FOLLOWING_INTERVAL = 10
    WHENEVER_OCL_OCCURS_OCL_OCCURS_WITHIN_INTERVAL = 11
    ALWAYS_DURING_INTERVAL_OCL_HAS_BEEN_TRUE_AT_LEAST_OCL_OF_TIME = 12
    AT_THE_END_OF_INTERVAL_OCL_HAS_BEEN_TRUE_AT_LEAST_OCL_OF_TIME = 13
TimeUnitEnum
    SEC = 1
    MIN = 2
    HOUR = 3
    DAY = 4
    YEAR = 5
ComposerEnum
    CONJUNCTION = 0
    DISJUNCTION = 0
```

Figure 4. GCSL Meta Model Exploded View – part 3

4 GCSL Tool chain development details and remarks

The purpose of this section is to collect the technical details that are relevant for the tool chain developers, *i.e.* ALES, OFFIS and INRIA. Therefore this section is not meant for the normal users, concerning mainly the GCSL compiler output alignment with the inputs of analysis tools.

4.1 The Time bounds in BLTL:

Each temporal operator has a time bound that define the duration on which it must hold. This time bound is always a **constant value** and it must be expressed in the **basic time unit** used in simulator, or in **number of states**.

In the DANSE settings, the basic time unit provided by Desyre is the ***second***. For instance, the property like '(G<=b P1)' means that P1 must hold during the next **b seconds** in the simulation. It means that every time bound used in a GCSL contract must be converted in seconds.

A GCSL contract may also be defined over interval using **steps** of simulation. This notion of steps match exactly the notion of **states** that can also be used to bound the temporal operators. In BLTL, the distinction between time of the simulation and the steps of simulation is done by prefixing the number of steps with the character **'#'**. For instance, the property like '(G<=#b P1)' means that P1 must hold during the next **b states** of the simulation.

4.2 BLTL encoding of operator MEAN, SUM and PROD

```
MEAN(OCL-expr,interval)          |
SUM(OCL-expr,interval)           |
PROD(OCL-expr,interval)
```

MEAN(OCL-expr, [low, up]) is translated into BLTL as follows:

(X<= low X<= len MEAN<= len (BLTL-expr))

where

1. BLTL-expr is OCL-EXPR encoded in BLTL
2. len is a constant defined as len = up – low
3. the time bound low, up and len are constants defined in seconds.

The encoding of SUM and PROD operators follows exactly the same construction.

4.3 The BLTL grammar used in Plasma

```
Expression ::= IfThenElse
                  |  IfThenElse ( U | W ) <= Bound Expression
                  |  ( F  | G) <= Bound Expression
                  | X ( <= Bound )? Expression
Bound ::= # integer | number
IfThenElse ::= Implication ? Implication : Implication
Implication ::= Disjunction (=> Disjunction)?
Disjunction ::= Conjunction (| Conjunction)*
Conjunction ::= Equality (& Equality)*

Equality ::= !? RelExp (Eop RelExp)?
Eop ::= = | !=

RelExp ::= NumExp (Rop NumExp)?
Rop =  < | <= | >= | >

NumExp ::= Term (Nop Term)*
Nop = + | -

Term ::= Factor (Top Factor)*
Top = * | /

Factor ::=   -?  ( | ( Expression )
                      | Function
                      | ident   | number
                      | true | false )

Function ::=   MEAN (<= Bound)? ( NumExp )
                 | FtoT  (<= Bound)? ( Implication )
                 | MIN ( NumExp )
                 | MAX ( NumExp )

ident = (letter | '_')(letter | digit | '_' | '.' )*
integer = '-'? digit+
float = '-'? digit+ '.' digit+ (('e' | 'E')('-' | '+')? digit+)?
number = float | integer
```

4.4 The operator precedence (top-down) in BLTL

- (unary minus)

*, / (multiplication, division)

+, - (addition, subtraction)

<, <=, >=, > (relational operators)

=, != (equality operators)

! (negation)

& (conjunction)

| (disjunction)

=> (implication)

? (ternary operator test ? a : b means "if test = true then a else b")

U W F G X (temporal operators)

5 References

[D 6.3.2] Specification of the goal contracts specification language, DANSE EU Project, 2013.