Guideline Definition Language (GDL)

Revision	0.91
Editors	Rong Chen[a], Iago Corbal[a]
Date of issue	19 December 2013

[a] Cambio Healthcare Systems

FUNDED BY: Cambio Healthcare Systems (http://www.cambio.se)

Contents

Address for correspondence

Rong Chen, MD, PhD
Cambio Healthcare Systems
Ringvägen 100
SE-118 60 Stockholm
Sweden
rong.chen@cambio.se

1 Introduction

1.1 Background

Expressing and sharing computerized clinical decision support (CDS) content across languages and technical platforms has been an evasive goal for a long time. Lack of commonly shared clinical information models and flexible support for various terminology resources have been identified as two main challenges for sharing detailed clinical rules between sites.

1.2 Purpose

This document contains the design specifications of the Guideline Definition Language (GDL). GDL is a formal language designed to represent clinical knowledge for computerized decision support. GDL is designed to be natural language- and reference terminology- agnostic by leveraging the designs of openEHR Reference Model and Archetype Model.

1.3 Scope

The scope of the GDL is to represent clear-cut clinical knowledge for single-decision making. Discrete GDL rules, artifacts written in a self-contained document in GDL format, can be combined together to support complex decision making. It does not cover the process aspects of the clinical guidelines.

1.4 Related Documents

- *open*EHR Reference Model Data Types Information Model (1.0.2)
- *open*EHR Reference Model Data Structures Information Model (1.0.2)
- *open*EHR Reference Model EHR Information Model (1.0.2)
- *open*EHR Reference Model Common Information Model (1.0.2)
- *open*EHR Archetype Model Archetype Object Model (AOM) (1.0.2)
- *open*EHR Archetype Model Archetype Definition Language (ADL) (1.0.2)

1.5 Requirements

1. It must be possible to express CDS rules using archetypes both as input and output for the rule execution.
2. It must be natural language-agnostic and able to support multiple language translations without changing the rule definitions.
3. It must be reference terminology-agnostic so different terminologies can be used to support reasoning.
4. It should be straight-forward to convert the CDS rules to main-stream general-purpose rule languages for execution.
5. There must be sufficient meta-information about the CDS rules, e.g. authorship, purpose, versions and relevant references.

6. It must be possible to reuse the CDS rules in different clinical contexts.
7. It should be possible to group a set of related CDS rules in order to support complex decision making.

1.6 Design Principles

In response to the above mentioned requirements, the following principles are applied in the GDL design.

1.6.1 Archetypes both as Input and Output of Rules

This is achieved by creating bindings between data elements defined by archetypes and variables used by the CDS rules. Each CDS rule variable is uniquely identified in the context of a guideline and bound to a specific data element defined by an archetype using Archetype ID and a path. Once defined, the variable can be used inside the ***when*** and ***then*** statements as input or output during rule execution.

1.6.2 Natural Language Neutrality

Several design ideas from openEHR archetype formalism are used to achieve natural language neutrality. First of all, all language-dependent meta-information about the purpose, use, misuse and references of the rules are grouped together under ***description*** and indexed by ISO language codes inside the guideline. Secondly, all natural language-dependent labels and descriptions, e.g. the name of a rule variable, are defined in the ***term_definitions*** section of the guideline and indexed by ISO language codes. Thirdly, unique identifiers for variables and rules are used in rule expressions instead of their names, which are language dependent.

1.6.3 Reference Terminology Neutrality

When IS_A operator is used in the evaluation statements for subsumption relationship checking, a locally defined term is used instead of an external code. This indirection makes it possible to modify the code or to add new codes from other terminologies without changing the rule definitions. The bindings between locally defined codes and external reference terminologies are maintained in the ***term_bindings*** section of the GDL document.

1.6.4 Rule Language Neutrality

GDL only uses a set of common rule language features, like ***when*** and ***then***. The expressions in the when and then statements support common arithmetic calucations, logic operator and functions.

1.6.5 Grouping and Reuse of Rules

A GDL document (guideline) may constain several rules that relate to each other. Each guideline is self-containing and should be reusable across different clinical contexts. Different guidelines can be chained together to support complex decision support. This is achieved by selecting the output of a rule, as a specific element of an archetype, as the input of anther rule.

1.6.6 Meta-information of the CDS rules

Authoring information, lifecycle state and various meta-information are supported by reuse of RESOURCE_DESCRIPTION class from the openEHR design.

1.7 An Example

The following is a simple GDL example that allows us to calculate CHA2DS2VASc Score. The definition for each one of the keywords used here can be found on the next section. The GDL header follows the same specifications as the openEHR's ADL Description Section. The GDL source illustrates the current version of the guideline, the natural languages, to which it has been translated, authors, lifecycle state, keywords and information about the purpose, use and misuse of the guideline.

```
(GUIDE) <
  gdl_version = <"0.1">
  id = <"CHA2DS2VASc_Score_calculation.v1-Revised function">
  concept = <"gt0036">
  language = (LANGUAGE) <
    original_language = <[ISO_639-1::en]>
  >
  description = (RESOURCE_DESCRIPTION) <
    details = <
      ["en"] = (RESOURCE_DESCRIPTION_ITEM) <
        copyright = <"">
        keywords = <"Atrial Fibrillation", "Stroke", "CHA2DS2-VASc">
        misuse = <"">
        purpose = <"Calculates stroke risk for patients with atrial fibrillation, possibly better than
        the CHADS2 score.">
        use = <"Calculates stroke risk for patients with atrial fibrillation, possibly better than the
        CHADS2 score.">
      >
      ["sv"] = (RESOURCE_DESCRIPTION_ITEM) <
      >
    >
    lifecycle_state = <"Author draft">
    original_author = <
      ["date"] = <"2012/12/03">
      ["email"] = <"rong.chen@cambio.se">
      ["name"] = <"Rong Chen">
      ["organisation"] = <"Cambio Healthcare Systems">
    >

    other_contributors = <"Carlos Valladares",...>

    >
```

The following block contains the definition section, where the bindings to the archetypes and elements used inside the guideline are defined.

```
definition = (GUIDE_DEFINITION) <
    archetype_bindings = <
        [1] = (ARCHETYPE_BINDING) <
            archetype_id = <"openEHR-EHR-EVALUATION.problem-diagnosis.v1">
            domain = <"EHR">
            elements = <
                ["gt0107"] = (ELEMENT_BINDING) <
                    path = <"/data[at0001]/items[at0002.1]">
                >
            >
        >
    >
```

GDL provides also a section to define a set of conditions that have to be met before the rules inside the guide can be executed. In the case of CHA_2DS_2VASc Score, the guideline will not be executed unless the patient has been diagnosed with atrial fibrillation. In the example below, a pre-condition checks all the diagnosis of the patient (*gt0107*) for the existence of atrial fibrillation. Using predicate in the definition section the precondition is set to check against a local code (*gt0105*) which represents the meaning of atrial fibrillation. This code can be bound with external terminologies, e.g. SNOMED CT, later on in the ***term_bindings*** section.

```
definition = (GUIDE_DEFINITION) <
   archetype_bindings = <
      [1] = (ARCHETYPE_BINDING) <
         archetype_id = <"openEHR-EHR-EVALUATION.problem-diagnosis.v1">
         domain = <"EHR">
         elements = <
            ["gt0107"] = (ELEMENT_BINDING) <
               path = <"/data[at0001]/items[at0002.1]">
            >
         >
         predicates = <"/data[at0001]/items[at0002.1] is_a local::gt0105|Atrial fibrillation|",...>
         template_id = <"diagnosis_chadvas_icd10">

      >
   >
   pre_conditions = <"$gt0107!=null",...>
   >
>
```

The ***rule*** section makes exclusive use of the locally defined elements to express the core logic of the guides. Each rule has a local gt code as an identifier, with which its language-dependent name and description are indexed in the ***term_definitions*** section. Also a ***priority*** can be assigned to ensure execution of the rules can be prioritized. This example illustrates rules that inspect different diagnoses relevant to CHA_2DS_2VASc Score and set the values of the DV_ORDINALs inside a CHA_2DS_2VASc Score Archetype, the rule gt0026 (Calculate total score) sums up all the values and sets the total score.

```
rules = <
   ["gt0018"] = (RULE) <
        when = <"$gt0108!=null",...>
        then = <"$gt0014=1|local::at0031|Present|",...>
        priority = <11>
   >
   ["gt0019"] = (RULE) <
        when = <"$gt0109!=null",...>
        then = <"$gt0010=1|local::at0034|Present|",...>
        priority = <9>
   >
   ["gt0026"] = (RULE) <
        then = <"$gt0016.magnitude=((((((($gt0009.value+$gt0010.value)+$gt0011.value)+$gt0015.value)
        +$gt0012.value)+$gt0013.value)+$gt0014.value)",...>
        priority = <1>
   >
```

Finally we have the ontology block (see *open*EHR's ADL Ontology Section), where all the terms are translated into the natural languages supported by the guideline ***term_definitions***.

```
    term_definitions = <
      ["en"] = (TERM_DEFINITION) <
        terms = <
          ["gt0003"] = (TERM) <
            text = <"Diagnosis">
          >
          ["gt0014"] = (TERM) <
            text = <"Hypertension">
          >
          ["gt0102"] = (TERM) <
            text = <"Diabetes">
          >
          ["gt0105"] = (TERM) <
            text = <"Atrial fibrillation">
          >
          ["gt0018"] = (TERM) <
            text = <"Set hypertension">
          >
          ["gt0019"] = (TERM) <
            text = <"Set diabetes">
          >
          ["gt0026"] = (TERM) <
            text = <"Calculate total score">
          >
```

Inside the ***term_bindings*** local defined terms are bound to external terminologies. In this sample, the diagnosis of atrial fibrillation is bound to an external ICD10 terminology.

```
  term_bindings = <
     ["ICD10"] = (TERM_BINDING) <
        bindings = <
           ["gt0105"] = (BINDING) <
                codes = <[ICD10::I48],...>
                uri = <"">
           >
        >
     >
  >
```

2 Guide Object Model

2.1 Design Background

The underpinning of GDL design is the *open*EHR archetypes, both as input and output of CDS rules. This is the key to achieve natural language-independence and reference terminology-independence. Because of this design choice, *open*EHR specification plays a major role in the GDL design. In other words, the GDL design is aimed to make substantial reuse of existing *open*EHR specifications. In areas where existing *open*EHR design is not sufficient, additional designs are introduced.

2.2. Packages Structure

The Guide Object Model, the object model of the GDL, consists of two packages, the guide package and the expressions package described in detail through the next two sections.

3 Guide Package

3.1 Overview

The overview of the guide package is illustrated in Figure 1. Note that classes in blue color are loosely based on the original design from the openEHR specifications.

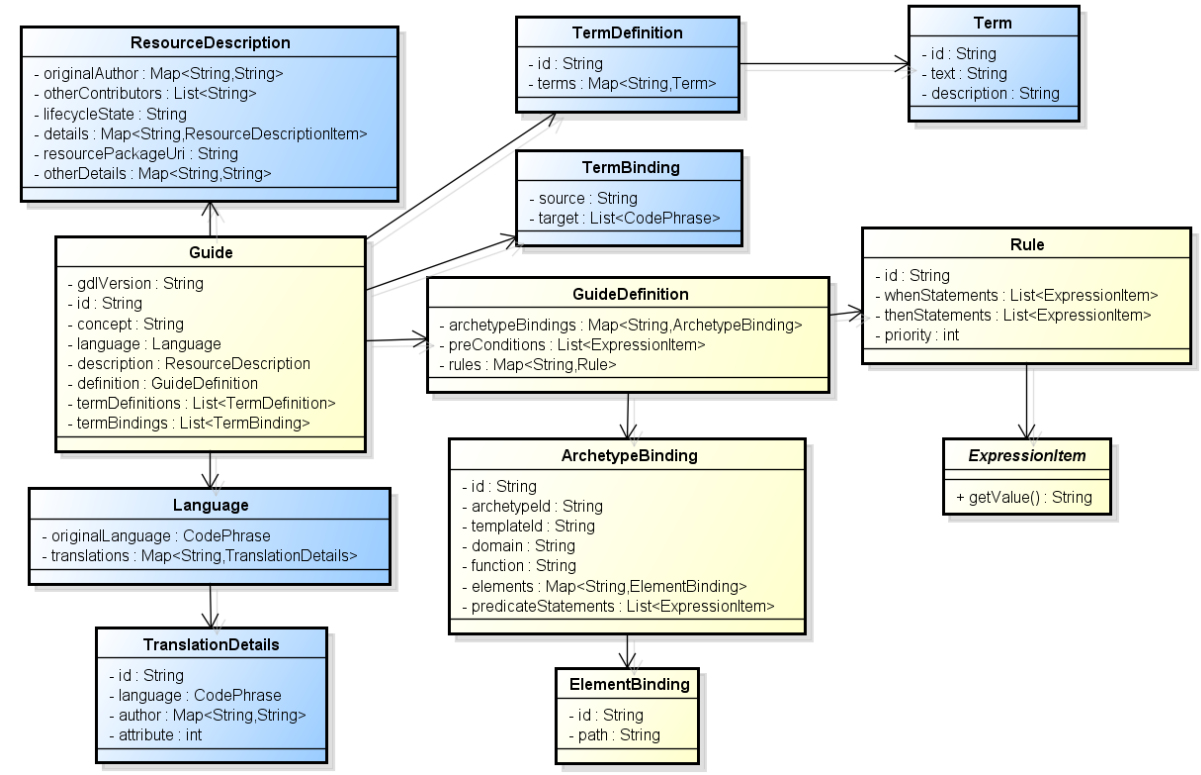

Figure 1 – The Guide Package

3.2 Class Definitions

3.2.1 GUIDE Class

CLASS	GUIDE	
Purpose	Main class of a discrete guide, which defines archetype bindings, rules and meta-information.	
Attributes	**Signature**	**Meaning**
0..1	**gdl_version:** String	the version of the GDL the guide is written in.
1..1	**id**: String	Identification of this guide
1..1	**concept**: String	The normative meaning of the guide as whole. Expressed as a local guide code.
1..1	**language**: Language	Natural language resources of this guide. It includes an original language and optional list of translations.
1..1	**description**: RESOURCE_DESCRIPTION	Resources description of this guide including authorship, use/misuse, life-cycle and references.
1..1	**definition**: GUIDE_DEFINITION	The main definition part of the guide. It consists of archetype bindings and rule definitions.
1..1	**ontology**: GUIDE_ONTOLOGY	The ontology of the guide.

3.2.2 GUIDE_DEFINITION

CLASS	GUIDE_DEFINITION	
Purpose	The definition of the guide includes a list of archetype bindings and a list of rule definitions.	
Attributes	**Signature**	**Meaning**
1..1	**archetype_bindings:** List<ARCHETYPE_BINDING>	List of archetype bindings, which define specific elements to be used by rules.
1..1	**rules**: Map<String, Rule>	Map of rules indexed by local gt codes.
0..1	**pre_conditions**: List<EXPRESSION_ITEM>	List of pre-conditions to be met before the guide should be executed.

3.2.3 ARCHETYPE_BINDING

CLASS	ARCHETYPE_BINDING	
Purpose	The binding of list of elements from a selected archetype or template to local gt codes	
Attributes	**Signature**	**Meaning**
1..1	**archetype_id:** String	The ID of the archetype, from where the list of elements is selected.
0..1	**template_id:** String	The ID of an optional template to be used for selecting elements.
0..1	**domain:** String	The space in which the rule variables reside. The value can either be “EHR” meaning the value is retrieved from the EHR, or “CDS: meaning the value is derived in the CDS engine. When missing, the assumption is either “EHR” or “CDS”.
1..1	**Elements:** Map<String, ELEMENT_BINDING>	Map of element binding indexed by local gt codes.
0..1	**predicate_statements:** List<EXPRESSION_ITEM>	List of predicates (constraints) that need to be fulfilled before the EHR queries can be performed

3.2.4 ELEMENT_BINDING

CLASS	ELEMENT_BINDING	
Purpose	The binding between a specific element in an archetype and a local variable in the guide.	
Attributes	**Signature**	**Meaning**
1..1	**id:** String	The local gt code of this element
1..1	**path**: String	The path to reach this element in the archetype.

3.2.5 RULE

CLASS	RULE	
Purpose	A single rule defined in a guide	
Attributes	**Signature**	**Meaning**
1..1	**id:** String	The local gt code of this element
1..1	**when_statements**: List<EXPRESSION_ITEM>	List of expressions to be evaluated before the rule can be fired.
1..1	**then_statements**: List<ASSIGNMENT_EXPRESSION	List of expressions to generate output of the rule.

3.3 Syntax Specification

The grammar and lexical specification for the standard GDL is entirely based on dADL and driven by the guide object model.

4 Expressions Package

4.1 Overview

The overview of the expressions package is illustrated by figure 2.

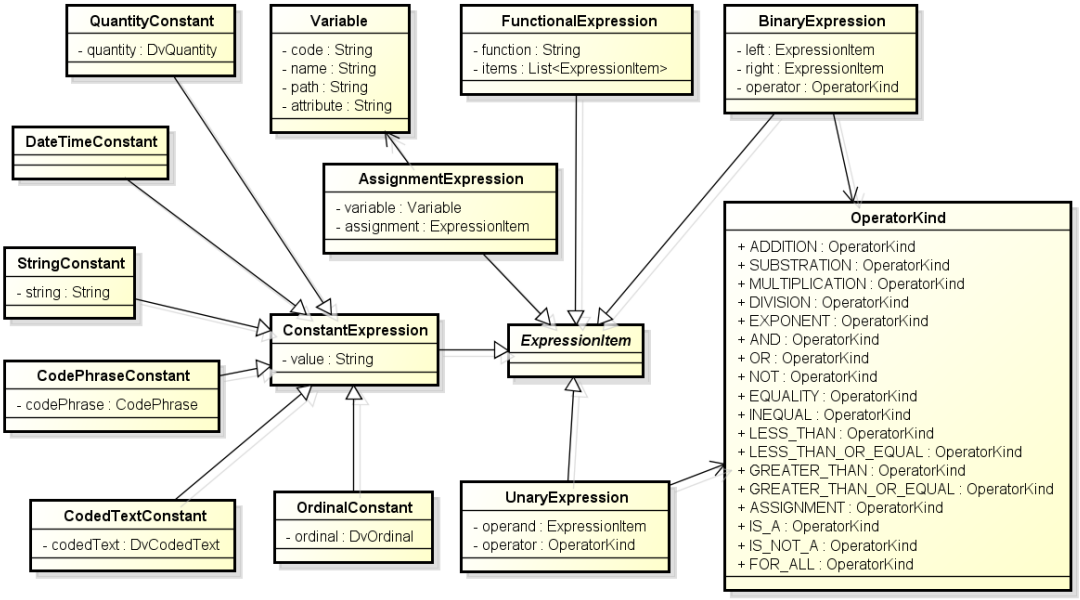

Figure 2 – The Expression Package

4.2 Class Definitions

4.2.1 EXPRESSION_ITEM

CLASS	*EXPRESSION_ITEM* (abstract)
Purpose	Abstract model of an expression item in the rule.

4.2.2 UNARY_EXPRESSION

CLASS	UNARY_EXPRESSION	
Purpose	Abstract model of an expression item in the rule.	
Inherit	EXPRESSION_ITEM	
Attributes	**Signature**	**Meaning**
1..1	**operand:** EXPRESSION_ITEM	The operand of this unary expression.
1..1	**operator**: OPERATOR_KIND	The operator of this unary expression.

4.2.3 BINARY_EXPRESSION

CLASS	BINARY _EXPRESSION	
Purpose	Concrete model of a binary expression item.	
Inherit	EXPRESSION_ITEM	
Attributes	**Signature**	**Meaning**
1..1	**left:** EXPRESSION_ITEM	The left operand of this binary expression.
1..1	**right:** EXPRESSION_ITEM	The right operand of this binary expression.
1..1	**operator**: OPERATOR_KIND	The operator of this binary expression.

4.2.4 ASSIGNMENT_EXPRESSION

CLASS	ASSIGNMENT_EXPRESSION	
Purpose	Concrete model of an assignment expression.	
Inherit	EXPRESSION_ITEM	
Attributes	**Signature**	**Meaning**
1..1	**variable:** String	The gt code of the variable to assign the value to.
1..1	**assignment**: EXPRESSION_ITEM	The expression item, from which the value is derived from.

4.2.5 FUNCTIONAL_EXPRESSION

CLASS	FUNCTIONAL_EXPRESSION	
Purpose	Concrete expression models a function.	
Inherit	EXPRESSION_ITEM	
Attributes	**Signature**	**Meaning**
1..1	**function:** Kind	The kind of function used.
1..1	**items**: List<EXPRESSION_ITEM>	A list of parameters to the function.

4.2.6 OPERATOR_KIND

CLASS	OPERATOR_KIND	
Purpose	Enumeration containing all the operators used.	
Type	**Name**	**Symbol**
Arithmetic	**Addition**	+
Arithmetic	**Subtraction**	-
Arithmetic	**Multiplication**	*
Arithmetic	**Division**	/
Arithmetic	**Exponent**	^
Logical	**And**	&&
Logical	**Or**	||

Logical	**Not**	!
Relational	**Equal**	==
Relational	**Unequal**	!=
Relational	**Less than**	<
Relational	**Less than or equal**	<=
Relational	**Greater than**	>
Relational	**Greater than or equal**	>=
Assignment	**Assignment**	=
Terminological reasoning	**Is a**	is_a
Terminological reasoning	**Is not a**	!is_a

4.2.7 FUNCTION_KIND

Type	Function
Max	Use for getting the maximum value of an element.
Min	Use for getting the minimum value of an element.

4.3 Syntax Specification

The grammar and lexical specification for the expressions used by GDL is loosely based on the assertion syntax in the ADL specification. This grammar is implemented using javaCC specifications in the Java programming environment.

The full source code of the java GDL parser can be found below.

4.3.1 Grammar

```
List < ExpressionItem > expressions() :
{
  List < ExpressionItem > items = new ArrayList < ExpressionItem > ();
  ExpressionItem item = null;
}
{
  item = expression_item()
  {
    items.add(item);
  }
  (
    LOOKAHEAD(2)
    < SYM_COMMA > item = expression_item()
    {
      items.add(item);
    }
  )*
  {
    return items;
  }
}

ExpressionItem expression_item() :
{
  ExpressionItem item = null;
}
{
  (
    LOOKAHEAD(4)
    item = expression_node()
  | LOOKAHEAD(4)
    item = expression_leaf()
  )
  {
    return item;
  }
  {
    return item;
  }
}

CodePhrase code_phrase() :
{
  Token t;
  String lang = null;
  String langTerm = null;
  String langCode = null;
}
{
  t = < V_CODE_PHRASE >
  {
    lang = t.image;
    int i = lang.indexOf("::");
    langTerm = lang.substring(1, i);
    langCode = lang.substring(i + 2, lang.length() - 1);
  }
  {
    return new CodePhrase(langTerm, langCode);
  }
}

CodePhrase code_phrase_raw() :
{
  Token t;
  String lang = null;
  String langTerm = null;
  String langCode = null;
}
{
  t = < V_CODE_PHRASE_RAW >
  {
    lang = t.image;
    int i = lang.indexOf("::");
```

```
    langTerm = lang.substring(0, i);
    langCode = lang.substring(i + 2);
  }
  {
    return new CodePhrase(langTerm, langCode);
  }
}

/* ---------------------- expressions ---------------------- */
ExpressionItem expression_node() :
{
  ExpressionItem ret = null;
  ExpressionItem item = null;
  ExpressionItem item2 = null;
  OperatorKind op = null;
  boolean precedenceOverridden = false; // TODO
  Token t = null;
  String attrId = null;
}
{
  (
    < SYM_FOR_ALL > item = expression_leaf()
    {
      return new UnaryExpression(item, OperatorKind.FOR_ALL);
        }
  |
    (
      item = expression_leaf()
      (
        < SYM_EQ >
        {
          op = OperatorKind.EQUALITY;
        }
      | < SYM_NE >
        {
          op = OperatorKind.INEQUAL;
        }
      | < SYM_LT >
        {
          op = OperatorKind.LESS_THAN;
        }
      | < SYM_GT >
        {
          op = OperatorKind.GREATER_THAN;
        }
      | < SYM_LE >
        {
          op = OperatorKind.LESS_THAN_OR_EQUAL;
        }
      | < SYM_GE >
        {
          op = OperatorKind.GREATER_THAN_OR_EQUAL;
        }
      | < SYM_PLUS >
        {
          op = OperatorKind.ADDITION;
        }
      | < SYM_MINUS >
        {
          op = OperatorKind.SUBSTRATION;
        }
      | < SYM_STAR >
        {
          op = OperatorKind.MULTIPLICATION;
        }
      | < SYM_SLASH >
        {
          op = OperatorKind.DIVISION;
        }
      | < SYM_CARET >
        {
          op = OperatorKind.EXPONENT;
        }
      | < SYM_AND >
```

```
        {
          op = OperatorKind.AND;
        }
      | < SYM_OR >
        {
          op = OperatorKind.OR;
        }
      | < SYM_IS_A >
        {
          op = OperatorKind.IS_A;
        }
      | < SYM_IS_NOT_A >
        {
          op = OperatorKind.IS_NOT_A;
        }
      | LOOKAHEAD(4)
        < SYM_ASSIGNMENT >
        {
          op = OperatorKind.ASSIGNMENT;
        }
        item2 = expression_leaf()
        {
          return new AssignmentExpression((Variable) item, item2);
        }
      )
      item2 = expression_leaf()
      {
        ret = new BinaryExpression(item, item2, op);
      }
    )
  )
  {
    return ret;
  }
}

ExpressionItem expression_leaf() :
{
  ExpressionItem item = null;
  Token t = null;
}
{
  (
    < SYM_L_PARENTHESIS >
    (
      LOOKAHEAD(expression_node())
      item = expression_node()
    | LOOKAHEAD(variable())
      item = variable()
    | LOOKAHEAD(constant_expression())
      item = constant_expression()
    )
    < SYM_R_PARENTHESIS >
  | item = constant_expression()
  | item = variable()
  )
  {
    return item;
  }
}

ConstantExpression constant_expression() :
{
  Token t = null;
  CodePhrase code = null;
  String text = null;
  String units = null;
  Integer order = null;
}
{
  (
    t = < V_STRING >
    {
      String str = t.image;
```

```
      return new StringConstant(str.substring(1, str.length() - 1));
    }
  | t = < V_ORDINAL >
    {
      String value = "DV_ORDINAL," + t.image;
      DvOrdinal ordinal = (DvOrdinal) DataValue.parseValue(value);
      return new OrdinalConstant(ordinal);
    }
  | t = < V_REAL >
  | t = < V_INTEGER >
  | t = <V_DATE>
  | t = <V_DATE_TIME_Z>
  | t = <V_TIME>
  | t = <V_ISO8601_DURATION>
  | t = < SYM_NULL >
  | t = < SYM_TRUE >
  | t = < SYM_FALSE >
  | LOOKAHEAD(2)
    code = code_phrase_raw() [ text = label() ]
    {
      if (text != null)
      {
        return new CodedTextConstant(text, code);
      }
      else
      {
        return new CodePhraseConstant(code);
      }
    }
  | t = < V_QUANTITY >
    {
      text = t.image;
      text = text.replace("(","");
      text = text.replace(")","");
      DvQuantity q = new DvQuantity("m",1,0).parse(text);
      return new QuantityConstant(q);
    }
  )
  {
    return new ConstantExpression(t.image);
  }
}

Variable variable() :
{
  Variable v;
  Token t;
  String code = null;
  String path = null;
  String label = null;
  String attribute = null;
}
{
  (
    < SYM_DOLLAR >
    (
      t = < V_LOCAL_CODE >
    | t = < SYM_CURRENT_DATETIME >
    )
    {
      code = t.image;
    }
    [ label = label() ]
  | t = < V_ABSOLUTE_PATH >
    {
      path = t.image;
    }
  )
  [
    < SYM_DOT > t = < V_ATTRIBUTE_IDENTIFIER >
    {
      attribute = t.image;
    }
  ]
```

```
  {
    return new Variable(code, label, path, attribute);
  }
}

String label() :
{
  Token t;
  String label = null;
}
{
  t = < V_LABEL >
  {
    label = t.image;
    label = label.substring(1, label.length() - 1);
    return label;
  }
}

double real() :
{
  Token t;
  String value = null;
}
{
  t = < V_REAL >
  {
    value = t.image;
    return Double.parseDouble(value);
  }
}

int integer() :
{
  Token t;
  String value = null;
}
{
  t = < V_INTEGER >
  {
    value = t.image;
    return Integer.parseInt(value);
  }
}
```

4.3.2 Symbols

```
<* >
SKIP : /* WHITE SPACE */
{
  " "
| "\t"
| "\n"
| "\r"
| "\f"
}

<* >
SPECIAL_TOKEN : /* COMMENTS */
{
  < SINGLE_LINE_COMMENT : "--" (~[ "\n", "\r" ])* >
}

<* >
TOKEN : /* SYMBOLS - common */
{
  < SYM_MINUS : "-" >
| < SYM_PLUS : "+" >
| < SYM_STAR : "*" >
| < SYM_SLASH : "/" >
| < SYM_CARET : "^" >
| < SYM_DOT : "." >
```

```
| < SYM_SEMICOLON : ";" >
| < SYM_COMMA : "," >
| < SYM_TWO_COLONS : "::" >
| < SYM_COLON : ":" >
| < SYM_EXCLAMATION : "!" >
| < SYM_L_PARENTHESIS : "(" >
| < SYM_R_PARENTHESIS : ")" >
| < SYM_DOLLAR : "$" >
| < SYM_QUESTION : "?" >

| < SYM_L_BRACKET : "[" >

| < SYM_R_BRACKET : "]" >
| < SYM_INTERVAL_DELIM : "|" >
| < SYM_EQ : "==" >
| < SYM_GE : ">=" >
| < SYM_LE : "<=" >
| < SYM_LT : "<" >
| < SYM_GT : ">" >
| < SYM_NE : "!=" >
| < SYM_NOT : "not" >
| < SYM_AND :
    "and"
  | "&&" >
| < SYM_OR :
    "or"
  | "||" >
| < SYM_FALSE : "false" >
| < SYM_TRUE : "true" >
| < SYM_NULL : "null" >
| < SYM_IS_A : "is_a" >
| < SYM_IS_NOT_A : "!is_a" >
| < SYM_FOR_ALL : "for_all" >
| < SYM_CURRENT_DATETIME : "currentDateTime" >
| < SYM_ASSIGNMENT : "=" >
| < SYM_MODULO : "\\" >
| < SYM_DIV : "//" >
| < SYM_ELLIPSIS : ".." >
| < SYM_LIST_CONTINUE : "..." >
}

<* >
TOKEN :
{
  < #V_LOCAL_CODE_CORE : "g" [ "c", "t" ] ([ "0"-"9", "." ])+ [ "0"-"9" ] >
| < V_LOCAL_CODE : < V_LOCAL_CODE_CORE > >
| < V_QUANTITY :
    (
      < V_REAL >
    | < V_INTEGER >
    )
    "," ([ "a"-"z", "A"-"Z", "µ", "°", "%", "0"-"9", "[", "]", "/" ])+
        ([ "a"-"z", "A"-"Z", "µ", "°", "%", "0"-"9", "[", "]", "/" ])*
        ([ "a"-"z", "A"-"Z", "µ", "°", "%", "0"-"9", "[", "]"])* >
| < V_INTEGER :
    (< DIG >)+
  |
    "(-" (< DIG >)+ ")"
  |
    (< DIG >)
    {
      1, 3
    }
    (
      "," (< DIG >)
      {
        3
      }
    )+ >
|
  < V_ISO8601_DURATION: ("-")? "P"((<DIG>)+["m","M"])?((<DIG>)+["w","W"])?
  ((<DIG>)+["d","D"])?("T"((<DIG>)+["h","H"])?((<DIG>)+["m","M"])?
  ((<DIG>)+["s","S"])?)?>
|
```

```
  < V_ISO8601_DURATION_CONSTRAINT_PATTERN: "P"(["y","Y"])?(["m","M"])?
  (["w","W"])?(["d","D"])?"T"(["h","H"])?(["m","M"])?(["s","S"])?
  |"P"(["y","Y"])?(["m","M"])?(["w","W"])?(["d","D"])?>
|
  < V_DATE: (["0"-"9"]){4} "-" ( "0"["1"-"9"] | "1"["0"-"2"] ) "-"
           ( "0"["1"-"9"] | ["1"-"2"]["0"-"9"]|"3"["0"-"1"] ) >
|
  < V_HHMM_TIME: <HOUR_MINUTE> >
|
  < V_HHMMSS_TIME: < HOUR_MINUTE> <SECOND> >
|
  < V_HHMMSSss_TIME: < HOUR_MINUTE> <SECOND> <MILLI_SECOND> >
|
  < V_HHMMSSZ_TIME: < HOUR_MINUTE> <SECOND> <TIME_ZONE> >
|
  < V_HHMMSSssZ_TIME: < HOUR_MINUTE> <SECOND> <MILLI_SECOND> <TIME_ZONE> >
|
  < V_TIME: <HOUR_MINUTE> <SECOND> >
|
  < V_DATE_TIME: <V_DATE>"T"<V_TIME> >
|
  < V_DATE_TIME_MS: <V_DATE_TIME> <MILLI_SECOND> >
|
  < V_DATE_TIME_Z: <V_DATE_TIME> <TIME_ZONE> >
|
  < V_DATE_TIME_MSZ: <V_DATE_TIME> <MILLI_SECOND> <TIME_ZONE> >
|
  < #TIME_ZONE: ["-","+"](["0"-"9"]){2}":"(["0"-"9"]){2} | "Z" >
|
  < #SECOND: ":" ["0"-"5"]["0"-"9"] >
|
  < #MILLI_SECOND: "."(["0"-"9"]){2, 3} >
|

  < #HOUR_MINUTE: ["0"-"9"]["0"-"9"] ":" ["0"-"5"]["0"-"9"] >

| < V_CODE_PHRASE : "[" (< LET_DIG_DUDSLR >)+ "::" (< LET_DIG_DUDS >)+ "]" >

| < V_CODE_PHRASE_RAW : (< LET_DIG_DUDSLR >)+ "::" (< LET_DIG_DUDS >)+ >

| < V_ORDINAL : < V_INTEGER > "|" < V_CODE_PHRASE_RAW > < V_LABEL > >
| < V_ATTRIBUTE_IDENTIFIER : [ "a"-"z" ] (< LET_DIG_U >)* >
| < V_LABEL : "|" ([ "a"-"z", "A"-"Z", "0"-"9", ",", "/", "\\", "*", " ", "-",
   "+", "&", "?", "@", "!", "#", "^", "~", ":", ";", ".", "[", "]", "(", ")",
   ">", "<", "=" ])+ "|" >
| < V_REAL :
    (< DIG >)+ "./" ~[ ".", "0"-"9" ]
  | (< DIG >)+ "." (< DIG >)* [ "e", "E" ] ([ "+", "-" ])? (< DIG >)+
  | (< DIG >)* "." (< DIG >)+
    (
      [ "e", "E" ] ([ "+", "-" ])? (< DIG >)+
    )?
  | "(-" (< DIG >)* "." (< DIG >)+
    (
      [ "e", "E" ] ([ "+", "-" ])? (< DIG >)+
    )? ")"
  | (< DIG >)
    {
      1, 3
    }
    (
      "_" (< DIG >)
      {
        3
      }
    )+
    "./" ~[ ".", "0"-"9" ]
  | (< DIG >)
    {
      1, 3
    }
    (
      "_" (< DIG >)
```

```
      {
        3
      }
    )*
    "."
    (
      (< DIG >)
      {
        1, 3
      }
      (
        "_" (< DIG >)
        {
          3
        }
      )*
    )?
    [ "e", "E" ] ([ "+", "-" ])? (< DIG >)
    {
      1, 3
    }
    (
      "_" (< DIG >)
      {
        3
      }
    )*
  |
    (
      (< DIG >)
      {
        1, 3
      }
      (
        "_" (< DIG >)
        {
          3
        }
      )*
    )?
    "." (< DIG >)
    {
      1, 3
    }
    (
      "_" (< DIG >)
      {
        3
      }
    )*
    (
      [ "e", "E" ] ([ "+", "-" ])? (< DIG >)
      {
        1, 3
      }
      (
        "_" (< DIG >)
        {
          3
        }
      )*
    )? >
| < V_STRING :
    "'"
    (
      (
        "\\\"" (~[ "\"", "\n", "\\" ])*
      )
    |
      (
        "\\\\" (~[ "\"", "\n", "\\" ])*
      )
    |
      (
```

```
        "\n" ([ "\r", " ", "\t" ])*
      )
    | (~[ "\\", "\n", "\"" ])*
    )*
    "'" >
}

<* >
TOKEN : /* LOCAL TOKENS */
{
  < #DIG : [ "0"-"9" ] >
| < #LET_DIG : [ "a"-"z", "A"-"Z", "0"-"9" ] >
| < #LET_DIG_DD :
    < LET_DIG >
  | "."
  | "_"
  | "-" >
| < #LET_DIG_U :
    < LET_DIG >
  | "_" >
| < #LET_DIG_DU :
    < LET_DIG_U >
  | "-" >
| < #LET_DIG_DUDS :
    < LET_DIG_DU >
  | "."
  | "\\" >
| < #LET_DIG_DUDSLR :
    < LET_DIG_DUDS >

  | "("
  | ")" >
| < V_LOCAL_TERM_CODE_REF : "[" < LET_DIG > (< LET_DIG_DD >)* "]" >
| < #PATH_SEGMENT : < V_ATTRIBUTE_IDENTIFIER > (< V_LOCAL_TERM_CODE_REF >)? >
| < V_ABSOLUTE_PATH : < SYM_SLASH > < PATH_SEGMENT > (< SYM_SLASH > < PATH_SEGMENT >)* >
}
```

5 Implementation

5.1 Drools

GDL is technology independent, so it can be implemented using different rule engines. We've chosen JBoss Drools, an open source technology, to develop our first implementation of GDL execution engine. Drools allows us to refer directly to our Java objects inside the rules. It provides support for a powerful expression language (MVEL) and has proven robust and efficient enough for our purposes. Our drools module will be able to translate our GDL language into Drools Rule Language (DRL).

The following list summarizes the considerations for the drools-based implementation:

- Each rule in GDL will generate a new rule in DRL
- Each rule will have as a title the guide id and the GT code (*rule "CHADSVAS_Score.v1/gt0028"*)
- Priority will be mapped to *salience* in DRL language
- All rules will have a *no-loop* attribute, to disable the execution loops inside one rule
- Current time variable will have to be defined using the *DvDateTime* class
- All elements used inside the expressions will be checked for definition prior to execution
- Modification of the elements will use the *modify* method to propagate the changes inside the knowledge base

- General definition section: Drools does not support declaration of elements outside the rule's scope. Each one of the elements had to be declared inside each rule.
- Preconditions: there is no support for this type of conditions. All preconditions will be copied inside each one of the rules.

6 Tools

6.1 GDL Editor

An authoring tool for GDL is released as open-source software for the community. The GDL editor is a multiplatform desktop application and will allow users to create, edit and run GDL files. We implemented in the editor a feature capable of generating forms based on the archetype elements defined in the GDL. These forms can be used to take input from the user and trigger the rules. More information about this tool can be found in the GDL Editor Manual.

The home page of GDL Editor on GitHub is https://github.com/openEHR/gdl-tools.

The software binary download page is here: http://sourceforge.net/projects/gdl-editor/.