NISTIR 8096

A metamodel for optimization problems

Ibrahim Assouroko
Peter Denno

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.8096

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 8096

A metamodel for optimization problems

Ibrahim Assouroko
Peter Denno
Systems Integration Division
Engineering Laboratory

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.8096

January 2016

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

Abstract: In mathematics, computer science, and operations research, *mathematical optimization* is the process of finding best feasible values (oftentimes maxima or minima) of a given objective that is modeled mathematically. This paper introduces the concept of a metamodel for optimization problems and describes an exploratory implementation of such a metamodel. The metamodel is (1) a conceptual model for capturing, in abstract terms, essential characteristics of a given optimization problem, and (2) a schema of sufficient formality to enable the problem modeled to be serialized to statements in a concrete optimization language. The exploratory metamodel presented in this paper addresses a majority of abstractions found in the OPL language. We present prototype software to compile OPL programs to populations of the metamodel and to serialize these populations back to OPL code. The paper describes the motivation for this work and a detailed specification of the exploratory implementation.

Keywords: metamodel, optimization problem, optimization model specification, Optimization Programming Language, UML diagram.

Contents

1 Introduction

This paper describes a metamodel of optimization problems. The sense of "metamodel" used here is consistent with usage of the term in information standards from the Object Management Group (OMG) including the Meta-Object Facility (MOF) [1] used to describe the Unified Modeling Language (UML) [2]. Roughly speaking, in these standards, a metamodel is a conceptual model where the domain of discourse is some aspect of information technology. For example, for UML, the domain of discourse is object-oriented software; here, the UML metamodel provides for the description of things such as classes and

methods. In this paper, the domain of discourse is optimization, of problems and software systems that solve those problems. The metamodel provides for the description of things such as objective functions and constraints. Ultimately, the metamodel should provide the breadth of detail necessary to serialize an instance optimization-metamodel to many algebraic modeling languages, such as OPL "Optimization Programing Language" [3], AMPL "A Mathematical Programing Language" [4], GAMS "General Algebraic Modeling System" [5] [6] and AIMMS "Advanced Interactive Multidimensional Modeling System" [7]. Initially and in this paper we focus on the OPL language.

Metamodels provide capabilities beyond ordinary conceptual models. Specifically, a metamodel can serve as a storage form for models, and useful artifacts may be generated directly from model content. For example, instances of the UML metamodel can describe classes (model content) with sufficient fidelity that automated tools can generate Java class definitions from the class descriptions. These capabilities of metamodels are due to their more formal method of description.

Motivation for developing a metamodel of optimization problems are the resulting capabilities just described; the metamodel can serve as the storage form for optimization problems, from which can be generated actual, executable, optimization problems. The idea of a conceptual model for optimization problems is not new; languages such as AMPL [4] and GAMS [6] also provide such models along with mappings from the language to many executable tools (optimization code). An optimization metamodel provides the ability to programmatically develop optimization problems. This ability is important when using optimization technology in operational settings. For example, in manufacturing operations, information from production monitoring systems can be transformed to elements of an optimization problem using the metamodel. The metamodel instance can then be transformed to an executable optimization.

NIST is developing two tools to read OPL[1] models into metamodel populations and to serialize those populations into OPL code. The former tool can be used to produce a "template" population that can be completed during problem formulation. (See Figure 1). The latter is used to produce an OPL model.

An example use of the metamodel in manufacturing is depicted in Figure 1. There a population conforming to the optimization metamodel is formulated from domain data and mapped into metamodel form, allowing serialization as an executable optimization. The top portion of the figure depicts a collection of manufacturing viewpoints relevant to some optimization problem such as finding optimal process parameters for a unit manufacturing process. Production information systems provide information in these viewpoints. The acquired information is used in problem formulation (middle third of Figure 1). Problem formulation concerns the development of design space constraints (typically mathematical inequalities) and objective function (typically an equation with weighted terms).

There are many possible technical means of formulating optimization problems. In this paper, we use *engineering notebooks*, an open notebook technology [8] integrated with semantic web technology [9] [10] [11]. The equations formulated by the user's work in the notebook are furnished to a partially completed metamodel population, thereby completing it. The completed metamodel population is then serialized into inputs to an optimization solver as depicted in the lower third.

[1] Optimization Programming Language (OPL) is an algebraic modeling language for mathematical optimization models. It is part of IBM's ILOG CPLEX Optimization Studio, an optimization software package. (https://en.wikipedia.org/wiki/Optimization_Programming_Language)

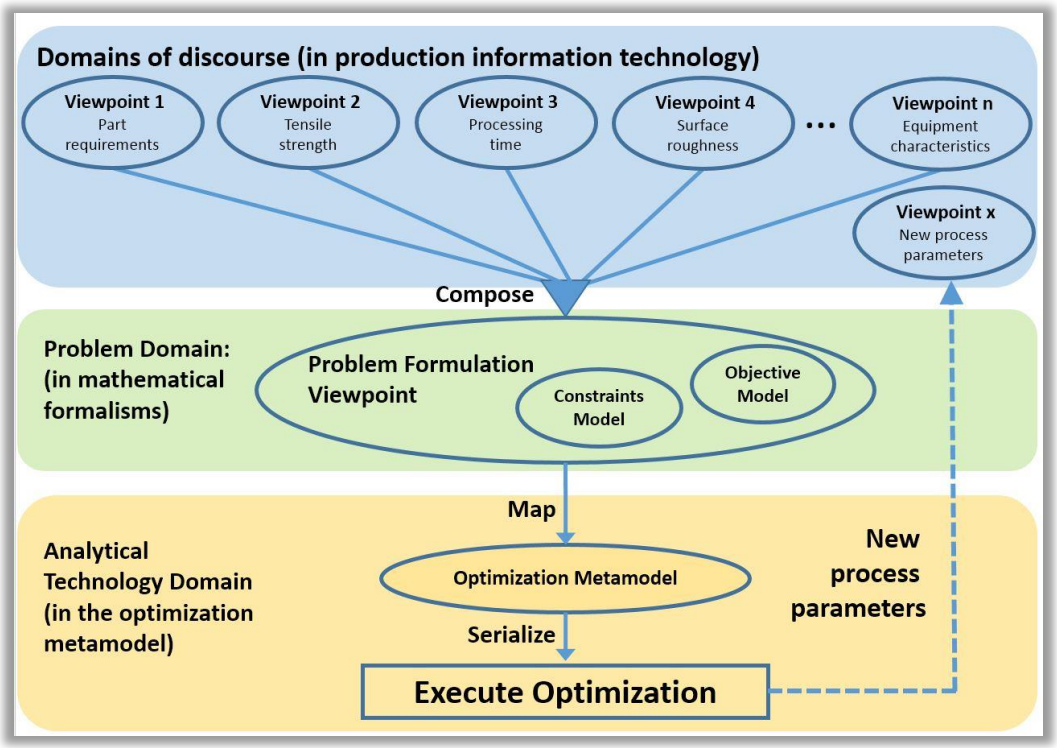

Figure 1: An example of using the metamodel in manufacturing.

The current metamodel can only address capabilities of OPL. NIST plans to extend the metamodel to address capabilities of other mathematical modeling languages. The objective in this paper is to begin to identify common elements of optimization problem specifications. The model specifications expressed in XMI "XML Metadata Interchange" standard can be found on the NIST GitHub site [12]. XMI is an OMG standard for exchanging metadata information via XML. It is commonly used as an interchange format for UML models, although it can also be used for serialization of metamodels.

The paper describes a specification of the proposed metamodel for optimization problems. Section 2 presents the form of the metamodel specification and the modeling technique used to make representation of the metamodel. Section 3 specifies the formal concepts in a set of UML diagrams. Each UML diagram represents a set of Classes corresponding to a set of optimization problem concepts.

2 Form of the specification

In this paper, metamodel elements are depicted as classes and associations in UML class diagrams. In UML, sets of concepts that are closely related are grouped into UML package constructs. Classes are depicted as boxes with partitions, and associations (called *objectProperties*) as labeled lines that connect the class boxes. Lines between class boxes with hollow arrow point on one end represent generalization/specialization relationships between the classes at the end of the line. Figure 2 shows an example UML diagram taken from the *OptimizationTopLevelObjects* UML package diagram. The upper partition of a class box provides the name of the class and sometimes special characteristics of the class

itself. The lower partition of a class box shows the attributes (called *dataProperties*) associated with members of the class, if any. Figure 2 shows five classes: Function, StateFunction, PiecewiseLinearFunction, CumulativeFunction and StepFunction.

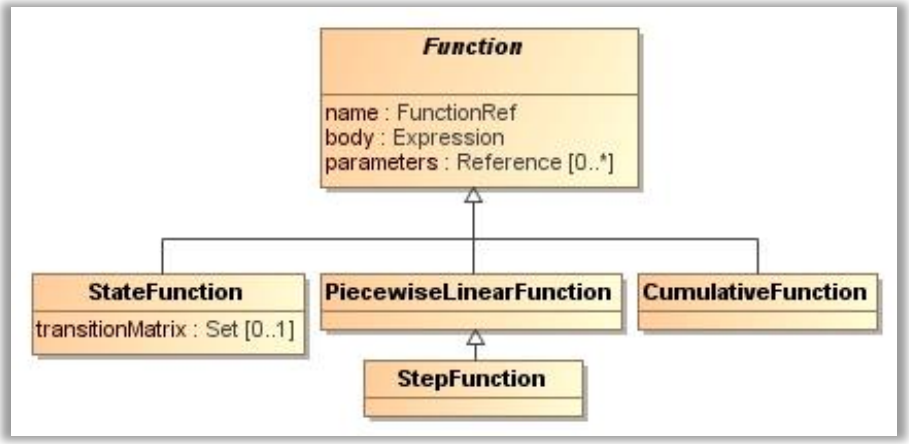

Figure 2: Example of UML classes arranged into a generalization or specialization hierarchy. Hollow arrows on the lines point to the more general class.

The sets of concepts represented in the metamodel are depicted as UML package diagrams in the form of classifications of concepts showing the hierarchy existing between those concepts. Some concepts are shown in more than one diagram to make it easier to understand dependencies between concepts not belonging to the same package. DataProperties are said to have multiplicities called *cardinality constraints* in UML that appear as numbers next to the property. The individual cardinality expressions are interpreted as follows:

- The cardinality "1" means "exactly one"
- The cardinality "1...*" means "at least one"
- The cardinality "0...1" means "at most one"
- The multiplicity "0...*" means "any number, zero or more"

In Figure 2, each "unnamed arrow" with open "head" between Function and StateFunction, PiecewiseLinearFunction, CumulativeFunction is called a *generalization* in UML. This means that StateFunction, PiecewiseLinearFunction, CumulativeFunction are sub classes of Function. In other words, every StateFunction, PiecewiseLinearFunction or CumulativeFunction is a Function, has a name, a body and may have zero or more parameters. The full model can be found on the NIST GitHub site.

3 Specification of the optimization metamodel concepts

This section of the paper describes the modeling of common concepts of the OPL language. As noted, these concepts are organized into UML package diagrams. Each model element is associated with one of the four packages defined in the metamodel: *OptimizationTopLevel*, *OptimizationDataTypes*, *OptimizationExpressions*, or *OptimizationOperators*. The main element in the *OptimizationTopLevel* package is the class **Model**. The class **Model** describes the optimization problem and serves as a container for the declaration of objective functions, constraints, and data types. The *OptimizationExpressions* package

contains specifications of the optimization problem formulation through objective functions and constraints. *OptimizationDataTypes* contains specifications of the different data types used in an optimization problem. Common operators (See section 3.4) used in optimization problem formulation and data declaration are specified in *OptimizationOperators* package.

3.1 Top level concepts

The *OptimizationTopLevel* package contains classes describing the most fundamental elements of an optimization problem. These classes are depicted in Figure 3, Figure 4 and Figure 5.

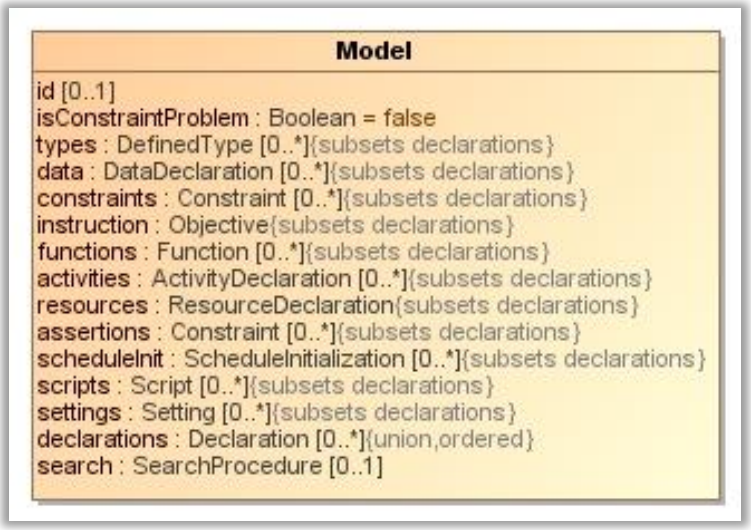

Figure 3: Optimization model specification

3.1.1 Class: Model

Definition: Model is an element that describes the optimization problem or a container for a complete optimization problem. A model consists of a set of declarations, such as data declaration, activity declaration, resource declaration, and can define, among other attributes, such attributes as types, data, constraints, instructions for objective functions, activities and resources.

Properties

Attribute: id	**Type: identifier**

Definition: the unique (optional) identification of the model.

Cardinality: at most one

Attribute: isConstraintProblem	**Type: Boolean**

Definition: a Boolean indicating whether or not this is a constraint satisfaction problem.

Attribute: types	**Type: DefinedType**

Definition: a collection of defined types used in the model.

Cardinality: any number, zero or more

Attribute: data	**Type: DataDeclaration**

Definition: a collection of data objects declared in the model.

Cardinality: any number, zero or more

Attribute: constraints	**Type: Constraint**

Definition: a collection of constraints defining the limits of the value produced by the objective.

Cardinality: any number, zero or more

Attribute: instruction	**Type: Objective**

Definition: the Objective of the model; OR an expression to be satisfied by performing the analysis defined from the model.

Attribute: functions	**Type: Function**

Definition: a collection of functions used in the model.

Cardinality: any number, zero or more

Attribute: activities	**Type: ActivityDeclaration**

Definition: a unit of work constrained to occur within a certain defined bounds.

Cardinality: any number, zero or more

Attribute: resources	**Type: ResourceDeclaration**

Definition: a collection of equipment, labor or material being used to perform an activity.

Cardinality: at least one

Attribute: assertions	**Type: Constraint**

Definition: an expression used to check if condition for the optimization model that need to be valid is respected.

Cardinality: at most one

Attribute: scripts	**Type: Script**

Definition: a collection of objects of type Script defined and embedded in the optimization model.

Cardinality: any number, zero or more

Attribute: settings	**Type: Setting**

Definition: a collection of settings defined over properties of the optimization model.

Cardinality: any number, zero or more

Attribute: declarations	**Type: Declaration**

Definition: an ordered collection of the subtypes of Declaration.

Cardinality: any number, zero or more

Attribute: search	Type: SearchProcedure

Definition: an optional search procedure defined to find a feasible solution in the scope of the objective.

Cardinality: at most one

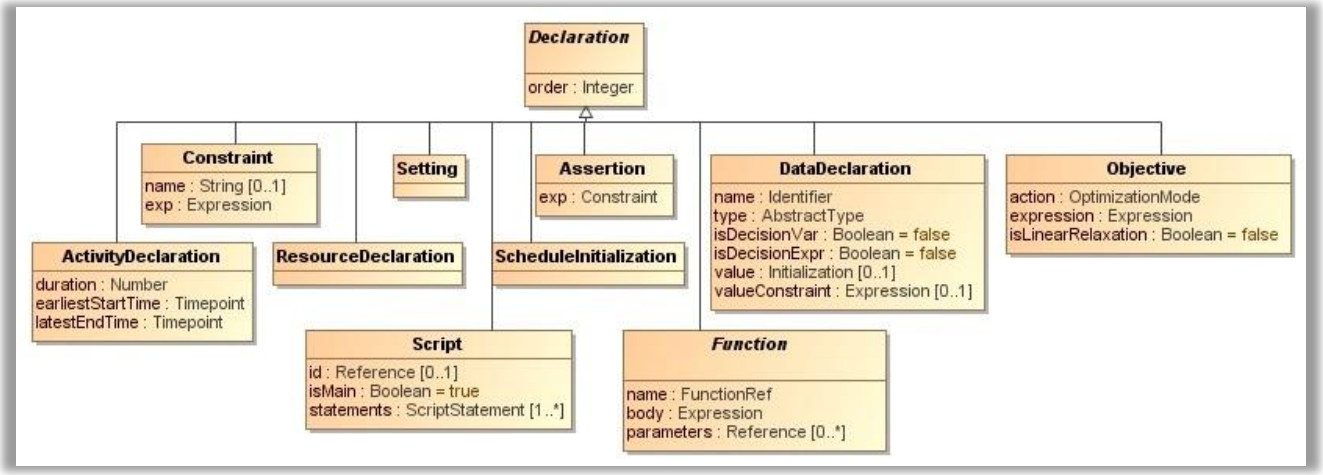

Figure 4: Optimization problem declaration

3.1.2 Class: Declaration

Definition: declaration of variables or elements of the model. Declaration is an abstract class defining an ordered collection of subtypes (or subsets declarations) such as DefinedType, DataDeclaration, Constraint, Objective, Function, ActivityDeclaration, ResourceDeclaration, Script, etc.

Properties

Attribute: order	Type: Integer

Definition: specifies the order in which declarations are made inside an optimization model.

3.1.3 Class: ActivityDeclaration

Definition: a unit of work constrained to occur within the bounds defined.

Properties

Attribute: duration	Type: Number

Definition: the quantity of time required to perform the activity associated with this declaration.

Attribute: earliestStartTime	Type: Timepoint

Definition: the earliest time point at which the activity may start.

Attribute: latestEndTime	Type: Timepoint

Definition: the latest time point at which the activity may end.

3.1.4 Class: Constraint

Definition: a Constraint in an optimization model is represented by an expression specifying the limit on the combinations of values for a given objective.

Properties

Attribute: name	**Type: String**

Definition: the name of the Constraint.

Cardinality: at most one

Attribute: exp	**Type: Expression**

Definition: the expression evaluated in the scope of the Constraint definition.

3.1.5 Class: Assertion

Definition: a collection of Boolean expressions that must be evaluated to true; OR an expression used to check if condition for the optimization model that needs to be valid is respected.

Properties

Attribute: exp	**Type: Constraint**

Definition: the expression defining the Assertion that must be checked in the scope of the model verification.

3.1.6 Class: ResourceDeclaration

Definition: declaration of a collection of equipment, labor or material being used to perform an activity.

Properties

none.

3.1.7 Class: DataDeclaration

Definition: declaration of a data object.

Properties

Attribute: name	**Type: Identifier**

Definition: the name used to refer to the data object.

Attribute: type	**Type: AbstractType**

Definition: the type of the data object.

Attribute: isDecisionVar	**Type: Boolean**

Definition: a Boolean indicating whether or not the data object may be modified in the course of execution; AND the value expresses a dimension of the decision space.

Attribute: isDecisionExpr	**Type: Boolean**

Definition: a Boolean indicating whether or not the data object is specified by a lexical structure producing its value.

Attribute: value	Type: Initialization

Definition: the initial value of the data object.

Cardinality: at most one.

Attribute: valueConstraint	Type: Expression

Definition: an optional expression governing the valid values of the data object.

Cardinality: at most one.

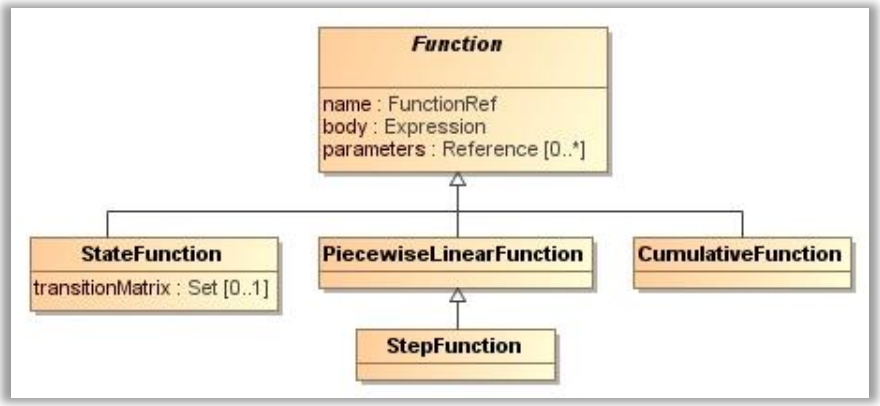

Figure 5: Functions in optimization problems

3.1.8 Class: Function

Definition: a function binds a set of parameters (.parameters) in the scope of an expression (.body) and evaluates the Expression producing a value; OR a scoped block of code that can be called with parameters and returns a value.

Properties

Attribute: name	Type: Identifier

Definition: the name of the function.

Attribute: body	Type: Expression

Definition: the Expression evaluated in the scope of the parameters binding.

Attribute: parameters	Type: Reference

Definition: identifiers bound in the call to the Function.

Cardinality: at least one

3.1.9 Class: StateFunction

Definition: a function describing the evolution of a given feature (of the optimization environment) constrained by interval variables of the optimization problem.

Properties

Attribute: transitionMatrix	Type: Set

Definition: a matrix containing a set of transition time (defined as set of integer numbers) necessary for a resource to switch from a state to another state in a StateFunction.

Cardinality: at most one

3.1.10 Class: PiecewiseLinearFunction

Definition: a function that is not purely linear but consist of linear segments with different slopes limited by breakpoints.

Properties

none.

3.1.11 Class: StepFunction

Definition: StepFunction is a special case of PiecewiseLinearFunction where all slopes are equal to 0 and the function domain is integer; AND a step function (or stepwise function) is used, in scheduling, to model the efficiency of a resource over time.

Properties

none.

3.1.12 Class: CumulativeFunction

Definition: a cumulative function expression can be used to model a quantity that varies over time and whose value depends on other decision variables of the problem.

Properties

none.

3.1.13 Class: Objective

Definition: it is the objective function of the optimization problem; AND its value measures the goodness of the solution.

Properties

Attribute: action	Type: OptimizationMode

Definition: the specification of whether the objective function is to be minimized, maximized or solved.

Attribute: expression	Type: Expression

Definition: Expression of the objective function.

Attribute: isLinearRelaxation	Type: Boolean

Definition: a Boolean indicating whether or not the objective is required to be solved by a linear relaxation.

3.1.14 Class: Setting

Definition: various options allowing the customization of the behavior of the optimization model.

Properties

none

3.1.15 Class: Script

Definition: a script is a program or function expression that can be embedded in the optimization model and executed from it.

Properties

Attribute: id	**Type: Reference**

Definition: the unique (optional) identification of the Script.

Cardinality: at most one

Attribute: isMain	**Type: Boolean**

Definition: a Boolean indicating whether or not the Script is defined as the main method of an optimization model.

Attribute: statements	**Type: ScriptStatement**

Definition: the statement describing the content of the Script.

Cardinality: at least one

3.1.16 Class: SearchProcedure

Definition: an optional search procedure defined to find feasible solution in the scope of the objective.

Properties

none.

3.2 Data type concepts

The *OptimizationDataTypes* concepts package identifies and specifies different types of data existing in an optimization problem model. The concepts are depicted in Figure 6, Figure 7, Figure 8 and Figure 9.

3.2.1 Class: AbstractType

Definition: a type descriptor for any type from which sub-classes may be derived, but which itself cannot be instantiated directly; AND instances of AbstractType are type descriptors.

Properties

none.

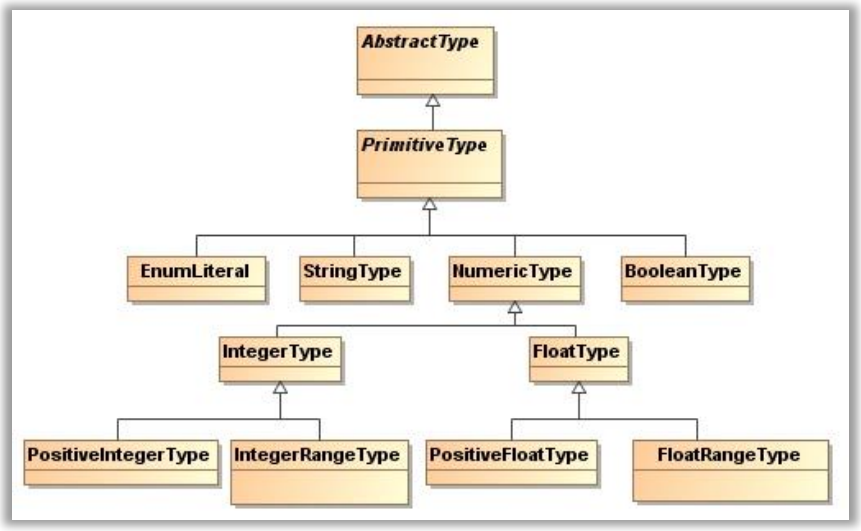

Figure 6: Optimization AbstractPrimitive data types

3.2.2 Class: PrimitiveType

Definition: a type descriptor for any type where the value provides identity.

Properties

none.

3.2.3 Class: EnumLiteral

Definition: an element of an EnumerationType.

Properties

none.

3.2.4 Class: BooleanType

Definition: a type whose values are True and False.

Properties

none.

3.2.5 Class: StringType

Definition: a data object type that represents sequences of characters.

Properties

none.

3.2.6 Class: NumericType

Definition: a type used to handle numbers in various representations.

Properties

none.

3.2.7 Class: IntegerType

Definition: an integer numeric type defined by a positive or negative counting number (that can be written without a fractional component) or zero.

Properties

none.

3.2.8 Class: PositiveIntegerType

Definition: a positive integer type.

Properties

none.

3.2.9 Class: IntegerRangeType

Definition: an ordered set of integer types limited by a lower and an upper bounds, and whose values can be stored inside a variable of type IntegerType.

Properties

none.

3.2.10 Class: FloatType

Definition: a float numeric type defined by a positive or negative number written with a fractional component (value); OR a number having digits on both sides of a decimal point.

Properties

none.

3.2.11 Class: PositiveFloatType

Definition: a positive numeric type written with fractional parts.

Properties

none.

3.2.12 Class: FloatRangeType

Definition: an ordered set of float types limited by a lower and an upper bounds.

Properties

None.

3.2.13 Class: DefinedType

Definition: an AbstractType defined in the scope of the model.

Properties

none.

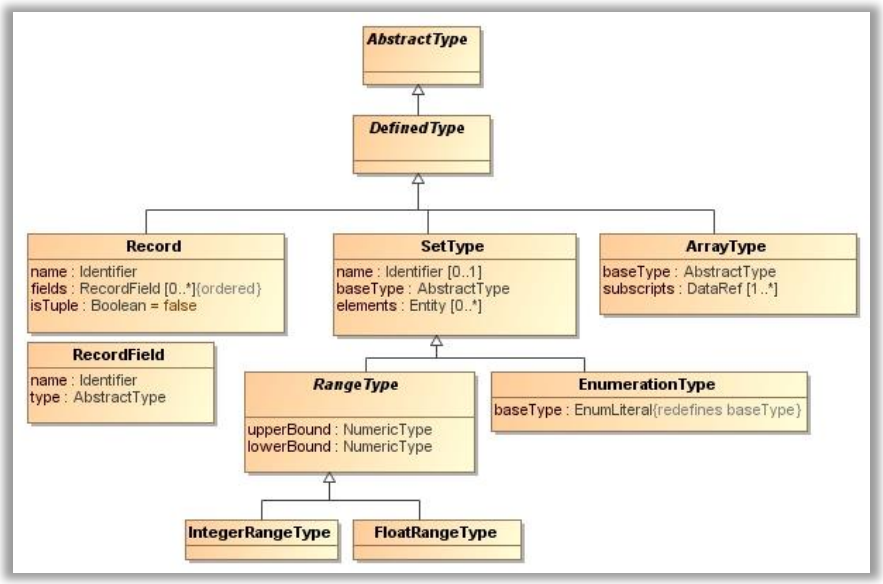

Figure 7: Optimization AbstractDefined data types

3.2.14 Class: Record

Definition: a DefinedType that defines a record structure, that is, named and ordered Record fields.

Properties

Attribute: name	**Type: Identifier**

Definition: a name identifying the record structure.

Attribute: fields	**Type: RecordField**

Definition: an ordered collection of the RecordFields of the record.

Cardinality: any number, zero or more

Attribute: isTuple	**Type: Boolean**

Definition: a data structure containing a given number of elements in a given order.

3.2.15 Class: RecordField

Definition: a field allocated to the record that being referred.

Properties

Attribute: name	**Type: String**

Definition: the name of the RecordField.

Attribute: type	**Type: AbstractType**

Definition: the type of the RecordField.

3.2.16 Class: SetType

Definition: a DefinedType that describes a set by enumerating its elements.

Properties

Attribute: name	Type: String

Definition: an optional name for the set.

Cardinality: at most one

Attribute: baseType	Type: AbstractType

Definition: the type of the elements of the set.

Attribute: elements	Type: Entity

Definition: the elements of the set.

Cardinality: any number, zero or more

3.2.17 Class: RangeType

Definition: a SetType whose BaseType is NumericType.

Properties

Attribute: upperBound	Type: NumericType

Definition: the largest element of the set.

Attribute: lowerBound	Type: NumericType

Definition: the smallest element of the set.

3.2.18 Class: IntegerRangeType

Definition: a RangeType whose BaseType is IntegerType

Properties

none

3.2.19 Class: FloatRangeType

Definition: a RangeType whose BaseType is FloatType.

Properties

none

3.2.20 Class: EnumerationType

Definition: a SetType whose BaseType is EnumLiteral

Properties

Attribute: baseType	Type: EnumLiteral

Definition: the type of the elements of the EnumerationType.

3.2.21 Class: ArrayType

Definition: a DefinedType constructed from an indexed collection of its .baseType.

Properties

Attribute: baseType	**Type: AbstractType**

Definition: the type of the elements of the array.

Attribute: subscripts	**Type: DataRef**

Definition: similar to the array index, subscripts can be defined as the inferior (symbol or number) used to identify elements in an array.

Cardinality: at least one

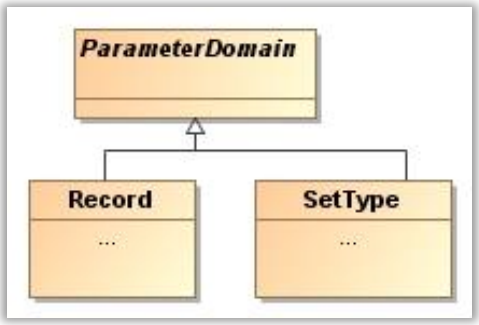

Figure 8: Optimization Parameters domain

3.2.22 Class: ParameterDomain

Definition: the set of possible values that can be assigned to an element (or variable) being referred in a Record, in the scope of an objective.

Properties

none.

3.2.23 Class: Record

Definition: a DefinedType that defines a record structure, that is, named and ordered Record fields (See section 3.2.1.2.1).

3.2.24 Class: SetType

Definition: a DefinedType describing a set by enumerating its elements (See section 3.2.1.2.2)

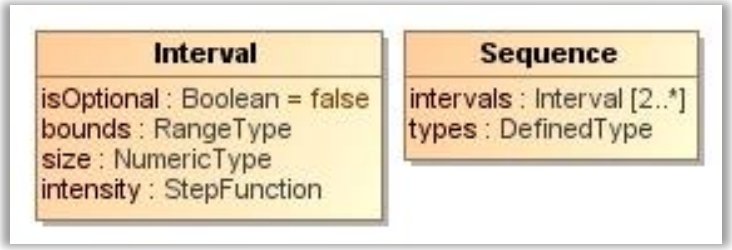

Figure 9: Interval in optimization data types

3.2.25 Class: Sequence

Definition: describes a set of data objects lying within an interval.

Properties

Attribute: intervals	**Type: Interval**

Definition: the interval of the sequence being referred.

Cardinality: at least two

Attribute: types	**Type: DefinedType**

Definition: the types of the data object in the scope of the sequence being referred.

3.2.26 Class: Interval

Definition: represents an interval of time during which something happens, and whose position in time is an unknown to the scheduling problem.

Properties

Attribute: isOptional	**Type: Boolean**

Definition: a Boolean indicating whether or not the interval is mandatory in the solution definition.

Attribute: bounds	**Type: RangeType**

Definition: the bounds of the interval defined by a start time and an end time within a certain range.

Attribute: size	**Type: Number**

Definition: the size of the interval.

Attribute: intensity	**Type: StepFunction**

Definition: the intensity of the interval variables defined by a stepwise linear function.

3.3 Expression concepts

The *OptimizationExpressions* package provides the context for identifying different types of expressions used in an optimization problem model. The concepts are depicted in Figure 10, Figure11, Figure 12, Figure13, Figure14 and Figure15.

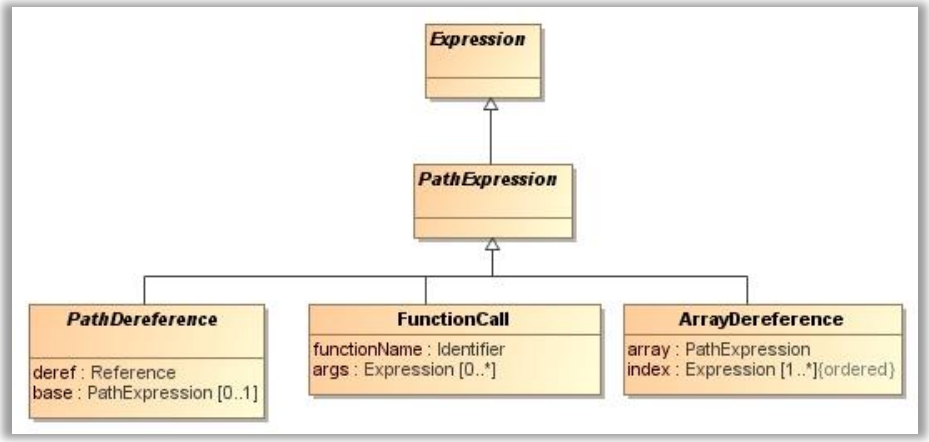

Figure 10: PathExpression in optimization problem

3.3.1 Class: Expression

Definition: an abstract class representing a lexical structure that may be evaluated to produce a value conforming to AbstractType.

Properties

none.

3.3.2 Class: PathExpression

Definition: an abstract Expression defining the path to a value.

Properties

none

3.3.3 Class: PathDereference

Definition: a PathExpression (may be partial) that navigates to a Reference.

Properties

Attribute: deref	**Type: Reference**

Definition: the partial PathExpression navigating to a Reference specified by PathExpression.deref.

Attribute: base	**Type: PathExpression**

Definition: the partial PathExpression navigating to the base of the PathExpression.

Cardinality: at most one

3.3.4 Class: FunctionCall

Definition: a PathExpression that takes some arguments as input and whose execution (call) returns a result.

Properties

Attribute: functionName	**Type: FunctionRef**

Definition: the name of the Function being called.

Attribute: args	**Type: Expression**

Definition: the expression of the arguments being input to the Function.

Cardinality: any number, zero or more

3.3.5 Class: ArrayDereference

Definition: a reference into an array.

Properties

Attribute: array	**Type: PathExpression**

Definition: the partial PathExpression navigating to an array.

Attribute: index	Type: Expression

Definition: the offset into the array referenced by ArrayDereference.array.

Cardinality: at least one

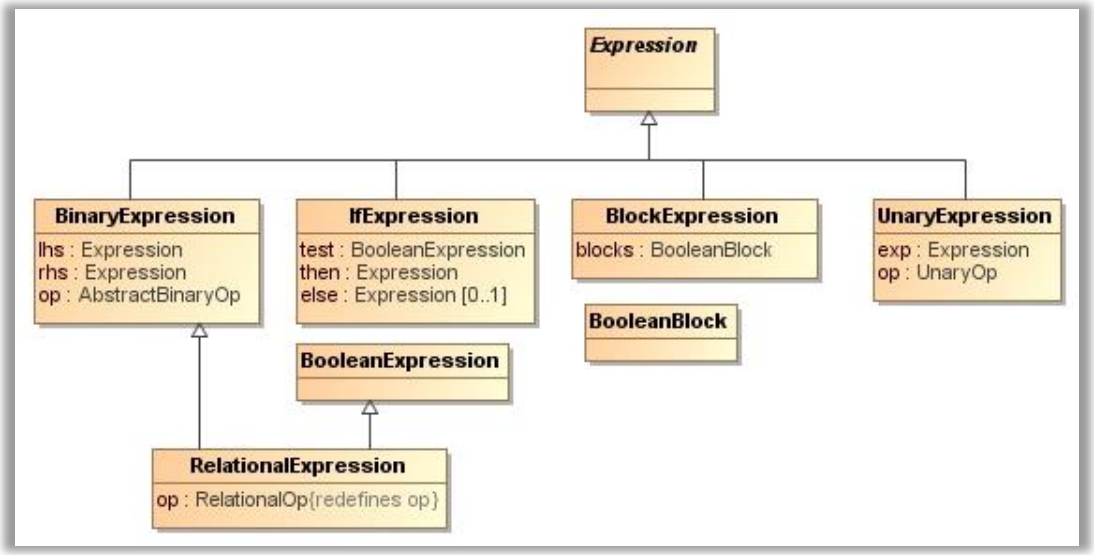

Figure 11: Expressions in optimization problem

3.3.6 Class: BinaryExpression

Definition: an expression formed from the application of a BinaryOp to two subexpressions.

Properties

Attribute: lhs	Type: Expression

Definition: an Expression on the left-hand side of a binary operator specified by BinaryExpression.op.

Attribute: rhs	Type: Expression

Definition: an Expression on the right-hand side of a binary operator specified by BinaryExpression.op.

Attribute: op	Type: AbstractBinaryOp

Definition: the operator of the BinaryExpression, of type AbstractBinaryOp.

3.3.7 Class: RelationalExpression

Definition: an expression built using operators of type RelationalOp and returning a value of type BooleanType.

Properties

Attribute: op	Type: RelationalOp

Definition: the operator of the RelationalExpression of type RelationalOp.

3.3.8 Class: BlockExpression

Definition: an expression formed from a set of block.

Properties

Attribute: blocks	Type: BooleanBlock

Definition: the blocks describing the condition inside the BlockExpression.

3.3.9 Class: BooleanBlock

Definition: the block containing a condition which execution returns a Boolean.

Properties

none

3.3.10 Class: IfExpression

Definition: an Expression describing if/then /else statement.

Properties

Attribute: test	Type: BooleanExpression

Definition: the condition deciding whether the 'then' or 'else' branch is to be evaluated.

Attribute: then	Type: Expression

Definition: an expression to be evaluated when IfExpression.condition evaluates to true.

Attribute: else	Type: Expression

Definition: an optional expression to be evaluated when IfExpression.condition evaluates to false.

Cardinality: at most one

3.3.11 Class: UnaryExpression

Definition: an expression formed as the application of a unary operator to an expression.

Properties

Attribute: exp	Type: Expression

Definition: the subexpression to which the unary operation, UnaryExpression.op, is applied.

Attribute: op	Type: UnaryOp

Definition: the operator of the UnaryExpression, of type UnaryOp.

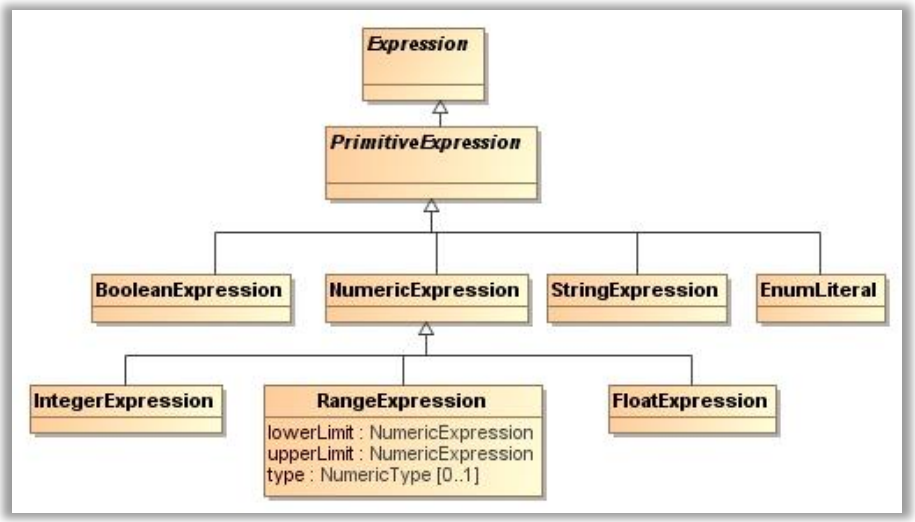

Figure 12: Primitive expressions in optimization problem

3.3.12 Class: PrimitiveExpression

Definition: an expression that evaluates to a primitive.

Properties

none.

3.3.13 Class: BooleanExpression

Definition: a PrimitiveExpression that evaluates to a Boolean.

Properties

none.

3.3.14 Class: NumericExpression

Definition: a PrimitiveExpression that evaluates to a Numeric.

Properties

none.

3.3.15 Class: IntegerExpression

Definition: a NumericExpression that evaluates to an Integer.

Properties

none.

3.3.16 Class: RangeExpression

Definition: an AllRange defining the lower and upper bounds on the Range.

Properties

Attribute: lowerLimit	Type: NumericExpression

Definition: the smallest element of the Range.

Attribute: upperLimit	Type: NumericExpression

Definition: the largest element of the Range.

Attribute: type	Type: NumericType

Definition: the (optional) type of the element defined within the Range.

Cardinality: at least one

3.3.17 Class: FloatExpression

Definition: a PrimitiveExpression that evaluates to a Float.

Properties

none.

3.3.18 Class: StringExpression

Definition: a PrimitiveExpression that evaluates to a String.

Properties

none.

3.3.19 Class: EnumLiteral

Definition: similar to the one defined in section 3.2.1.1.1 under PrimitiveAbstractType.

Properties

none.

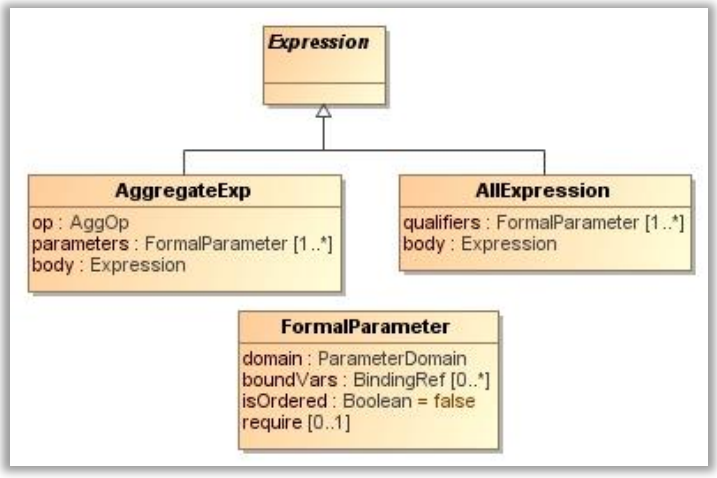

Figure 13: AllExpression and AggregateExpression

3.3.20 Class: AllExpression

Definition: an expression formed as the application of the keyword “all” that allows the usage of only part of an array with functions that take array parameters.

Properties

Attribute: qualifiers	**Type: FormalParameter**

Definition: the formal parameter that is being referred in the expression.

Cardinality: at least one

Attribute: body	**Type: Expression**

Definition: the body of the Expression.

3.3.21 Class: AggregateExpression

Definition: an Expression formed from the application of an AggOp to two subexpressions.

Properties

Attribute: op	**Type: AggOp**

Definition: the operator of the AggregateExpression, of type AggOp.

Attribute: parameters	**Type: FormalParameters**

Definition: the parameters evaluated inside the AggregateExpression.

Cardinality: at least one

Attribute: body	**Type: Expression**

Definition: the body of the AggregateExpression.

3.3.22 Class: FormalParameter

Definition: An abstract representation (usually composed of a name and a type) of a parameter expected to be called in the scope of an expression.

Properties

Attribute: domain	**Type: ParameterDomain**

Definition: the set of possible values over which the FormalParameter may range.

Attribute: boundVars	**Type: BindingRef**

Definition: a list of BindingRef (variables) used in the body of an AggregateExpression or AllExpression.

Cardinality: at least one

Attribute: isOrdered	**Type: Boolean**

Definition: a Boolean indicating whether or not the collection of values in the range of the FormalParameter is ordered or not.

Attribute: require	**Type: Expression**

Definition: a constraint governing valid values of the .boundVars.

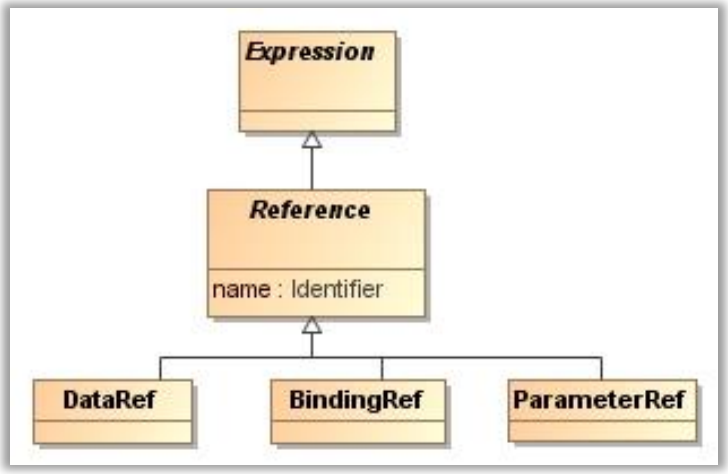

Figure 14: References in optimization problems

3.3.23 Class: Reference

Definition: a relational that takes a Binary and returns a Boolean.

Properties

Attribute: name	**Type: Identifier**

Definition: the name that identifies the reference.

3.3.24 Class: DataRef

Definition: a reference to a DataObject.

Properties

none.

3.3.25 Class: BindingRef

Definition: a reference to a VariableBinding.

Properties

none.

3.3.26 Class: ParameterRef

Definition: a reference introduced in a FormalParameter.

Properties

none.

3.3.27 Class: VariableBinding

Definition: specifies the binding of a variable to a declared DataObject.

Properties

Attribute: vars	Type: BindingRef

Definition: the reference binding a variable to its DataObject.

Cardinality: at least one

Attribute: domain	Type: Expression

Definition: an expression that specifies the domain of the variables.

3.3.28 Class: DataObject

Definition: a data at meta-level below the OPL metamodel; AND it is an instance of an AbstractType whose value must conform to the type declared in the DataDeclaration.

Properties

none.

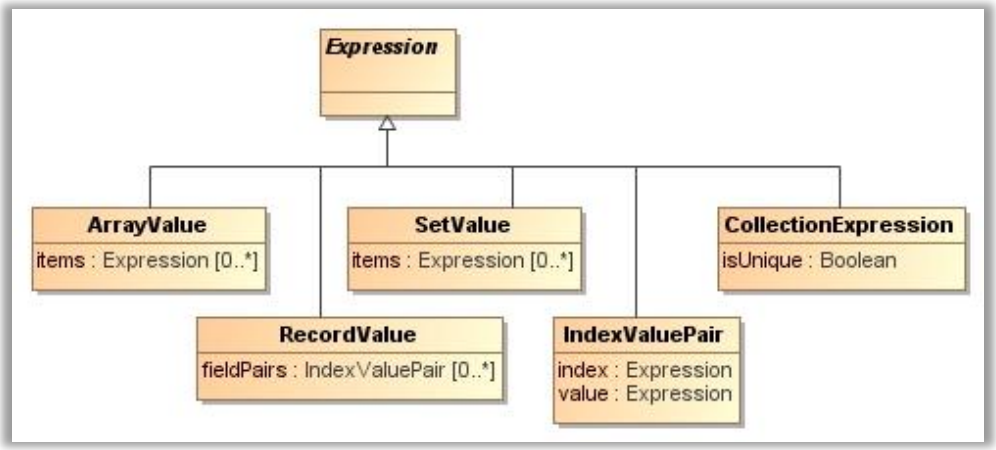

Figure 15: Expressions in optimization problems

3.3.29 Class: ArrayValue

Definition: an Expression that evaluates to an ArrayType indexed by offset into the ordered collection.

Properties

Attribute: items	Type: Expression

Definition: an ordered collection of Expression, each of which evaluates to an element of the Array; AND the resulting ArrayType object is indexed by offset into the ordered collection (i.e. 0, 1, 2…).

Cardinality: any number, zero or more

3.3.30 Class: RecordValue

Definition: an Expression that evaluates to a RecordType instance.

Properties

Attribute: fieldPairs	Type: IndexValuePair

Definition: a collection of pairs consisting of a record field name and a value.

Cardinality: any number, zero or more

3.3.31 Class: SetValue

Definition: an Expression that evaluates to a SetType instance.

Properties

Attribute: items	**Type: Expression**

Definition: the elements of the SetType.

Cardinality: any number, zero or more

3.3.32 Class: IndexValuePair

Definition: a pair of values consisting of a field name and value.

Properties

Attribute: index	**Type: Expression**

Definition: an Expression evaluating to the name of the field.

Attribute: value	**Type: Expression**

Definition: an Expression evaluating to the value of the field.

3.3.33 Class: CollectionExpression

Definition: an Expression formed from a collection of DataObject

Properties

Attribute: isUnique	**Type: Boolean**

Definition: a Boolean indicating whether or not the collection is unique.

3.4 Operator concepts

This section of the metamodel related to the *OptimizationOperators* package introduces the common concepts surrounding operator's usage in optimization problem definition. These concepts are depicted in Figure 16.

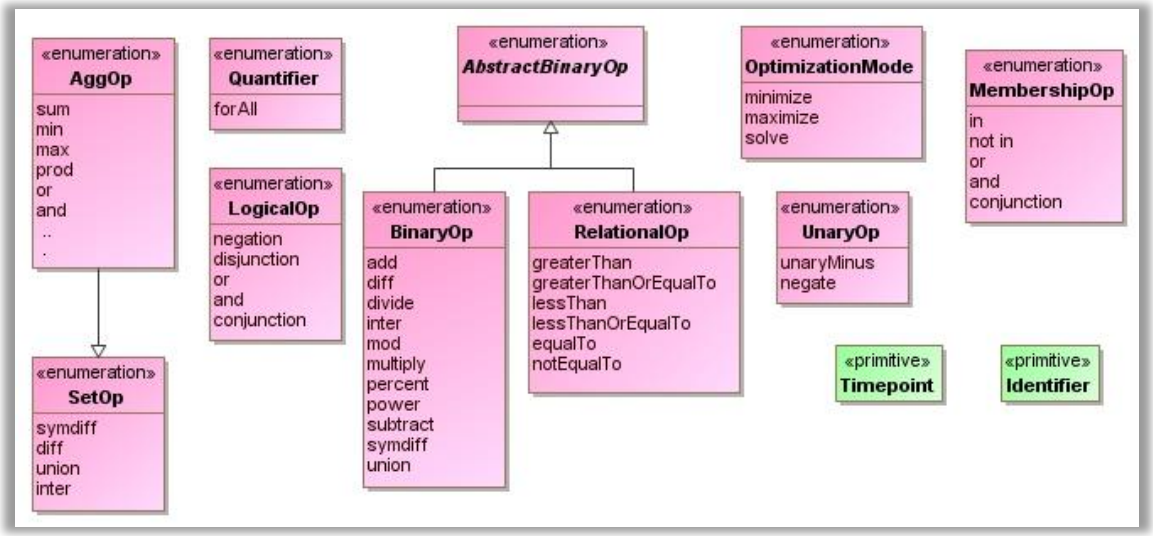

Figure 16: Operators in optimization problems

Concepts in the OptimizationOperators' package appear as Enumeration classes. In an enumeration class box, the lower partition is the list of the *Named Individuals* that are members. In Figure 14, the enumeration classes are: AggOp, SetOp, LogicalOp, AbstractBinaryOp, BinaryOp, RelationalOp, UnaryOp, MembershipOp, OptimizationMode, and Quantifier. In addition, Timepoint and Identifier are defined as primitive operators.

3.4.1 Enumeration Class: AggOp

Definition: specifies the set of operators used to construct expressions of type AggregateExpression

Named Individual Members

Member: sum

Definition: the operator used to aggregate, by the mathematical process of addition, two or more numbers, magnitudes, quantities, or expressions.

Member: min

Definition: the operator used to find the value of an Expression where a minimum occurs.

Member: max

Definition: the operator used to find the value of an Expression where a maximum occurs

Member: prod

Definition: the operator used to construct product of two or more expressions together.

Member: or

Definition: the operator used to connect statements or expressions representing alternative constraints; OR a Boolean operator that returns a positive result when either or both operands are positive.

Member: and

Definition: the operator used to connect statements or expressions representing cumulative constraints; OR a Boolean operator that returns a positive result when both operands are positive.

3.4.2 Enumeration Class: SetOp

Definition: specifies the set of operators used to construct collection or group of similar optimization data designed to be used together

Named Individual Members

Member: union

Definition: The operator used to construct a collection (or a set) whose members are elements from two or more other sets.

Member: inter

Definition: the operator used to compute a set that contains only elements share by two or more other sets.

Member: symdiff

Definition: the operator used to construct a set containing elements of the difference between the union and the intersection (inter) of two other sets.

Member: diff

Definition: the operator used to subtract elements of a given set from another one.

3.4.3 Enumeration Class: LogicalOp

Definition: specifies the set of operators used to construct logical statement in an optimization problem.

Named Individual Members

Member: negation

Definition: the operator used to make a negative statement, or to specify a logical operation of negating.

Member: inclusive disjunction

Definition: the operator used to construct a compound statement, which is true if and only if at least a number of alternatives is true.

Member: exclusive disjunction

Definition: the operator used to construct a compound statement, which is true if and only if one and only one of a number of alternatives is true.

Member: or

Definition: similar to the one defined in section 3.4.1 on AggOp.

Member: and

Definition: similar to the one defined in section 3.4.1 on AggOp.

Member: conjunction

Definition: the operator used to construct a compound statement, which is true if and only if all of its component statements are true.

3.4.4 Enumeration Class: AbstractBinaryOp

AbstractBinaryOp generalizes two types of operators identified as: BinaryOp and RelationalOp.

3.4.4.1 *Enumeration Class: BinaryOp*

Definition: specifies the set of operators used to construct statements or expressions relating two logical or mathematical elements; OR the set of operators used to construct expressions of type BinaryExpression.

Named Individual Members

Member: add

Definition: the operator used to join or combine (numbers, quantities…) by the arithmetic operation of addition; OR the operator used to find a sum in arithmetic.

Member: diff

Definition: similar to the one defined in section 3.4.1.1 on SetOp.

Member: divide

Definition: the operator used to separate an element into equal parts by the process of mathematical integer division.

Member: inter

Definition: similar to the one defined in section 3.4.1.1 on SetOp.

Member: mod

Definition: the operator that evaluates the remainder of a mathematical integer division in an optimization problem.

Member: multiply

Definition: the operator used to find a product of, by the process of multiplication.

Member: percent

Definition: the operator used to find a proportion in relation to a whole (usually defined as the amount per hundred).

Member: power

Definition: the operator used to indicate the number of times a quantity is multiplied by itself.

Member: subtract

Definition: the operator used to calculate the difference between two numbers or quantities by the arithmetic operation of subtraction.

Member: symdiff

Definition: similar to the one defined in section 3.4.1.1 on SetOp.

Member: union

Definition: similar to the one defined in section 3.4.1.1 on SetOp.

3.4.4.2 *Enumeration Class: RelationalOp*

Definition: this enumeration contains the set of operators used to indicate or specify some relations in the scope of mathematical expressions.

Named Individual Members

Member: greaterThan

Definition: an algebraic relational operator showing inequality between two subexpressions; AND a statement indicating that the value of a subexpression (or quantity) preceding the operator is greater than the value of the subexpression (or quantity) following the operator.

Member: greaterThanOrEqualTo

Definition: an algebraic relational operator showing inequality between two subexpressions; AND a statement indicating that the value of a subexpression (or quantity) preceding the operator is greater than or equal to the value of the subexpression (or quantity) following the operator.

Member: lessThan

Definition: an algebraic relational operator showing inequality between two subexpressions; AND a statement indicating that the value of a subexpression (or quantity) preceding the operator is less than the value of the subexpression (or quantity) following the operator.

Member: lessThanOrEqualTo

Definition: an algebraic relational operator showing inequality between two subexpressions; AND a statement indicating that the value of a subexpression (or quantity) preceding the operator is less than or equal to the value of the subexpression (or quantity) following the operator.

Member: equalTo

Definition: an algebraic relational operator showing equality between two subexpressions; AND a statement indicating that the value of a subexpression (or quantity) preceding the operator is equal to (or as great as) the value of the subexpression (or quantity) following the operator.

Member: notEqualTo

Definition: an algebraic relational operator showing inequality between two subexpressions; AND a statement indicating that the value of a subexpression (or quantity) preceding the operator is not equal to the value of the subexpression (or quantity) following the operator.

3.4.5 Enumeration Class: UnaryOp

Definition: specifies enumeration of operators used to construct statements or expressions consisting of or involving a single element or component; OR the type of operator used in mathematical operations where only one element is used to yield a single result.

Named Individual Members

Member: unaryMinus

Definition: specifies a unary operator which negates the arguments it applies to.

Member: negate

Definition: specifies the operator used to define negation, which is an operation on the value of a proposition that produces a value of true when its operand is false and a value of false when its operand is true.

3.4.6 Enumeration Class: MembershipOp

Definition: the list of operators used to define the state of an optimization data object being member of a set or a given range.

Named Individual Members

Member: in

Definition: the operator used to define the membership of an element to a set or a range within a certain limits, bounds or area.

Member: not in

Definition: the operator used to define the non-membership of an element to a set or a range within a certain limits, bounds or area.

Member: or

Definition: similar to the one defined in section 3.4.2 on LogicalOp.

Member: and

Definition: similar to the one defined in section 3.4.2 on LogicalOp.

Member: conjunction

Definition: similar to the one defined in section 3.4.2 on LogicalOp.

3.4.7 Enumeration Class: OptimizationMode

Definition: enumeration of the different modes by which an objective function is to be optimized or computed.

Named Individual Members

Member: minimize

Definition: the optimization mode used to find the smallest value of an objective function either within a given range or on the entire domain of the function.

Member: maximize

Definition: the optimization mode used to find the largest value of an objective function either within a given range or on the entire domain of the function.

Member: solve

Definition: the optimization mode used to find what values fulfill the condition stated in an objective function that is subject to a given (set of) constraint(s).

3.4.8 Enumeration Class: Quantifier

Definition: an operator that specifies for which values of a variable a proposition is true.

Member: forAll

Definition: a logical quantifier of a proposition (within an expression) asserting that the proposition is true for all members of a class of objects.

3.4.9 Primitive Class: Timepoint

Definition: the exact point in time at which something may happen.

3.4.10 Primitive Class: Identifier

Definition: a word, symbol or phrase use to provide the identity of the optimization element bearing it.

4 Conclusions

This document proposes a specification for an optimization metamodel, which is intended to be used as a common reference for optimization problem definitions. The proposed specification provides an alternative technology to replace the current, multiple APIs implemented in commercial optimization tools. It also is an easy-to-use solution technique because it does not require the advanced knowledge and skills commonly needed to use those tools.

The main idea behind the work is to use the optimization metamodel as part of a new standard. This new standard has the potential to provide 1) a catalyst for a new or improved uses of optimization tools in manufacturing and 2) a component of emerging CPS architectures and model-based engineering activities. We invite the participation of others from the optimization-modeling-tools industry and experts from the standardization organizations to improve the semantic precision and make this potential a reality.

As it is today, the specification only uses the vocabulary of OPL; but, we intend to integrate other optimization-modeling-language vocabularies in the near future.

Disclaimer

Certain products or services are identified in the paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products or services identified are necessarily the best available for the purpose.

References

[1] Object Management Group. 2011. OMG Meta Object Facility (MOF) Core Specification. Needham: Object Management Group.

[2] Object Management Group. 2011. "Unified Modeling Language (UML).", http://www.omg.org/spec/UML/2.4.1/.

[3] Van Hentenryck P., Lustig I., Michel L., Puget J.F. 1999. The OPL “Optimization Programming Language.” The MIT Press, Cambridge, Massachusetts, London, England

[4] Fourer R., Gay D.M., Kernighan B.W. 1990. A Modeling Language for Mathematical Programming. Management Science, Vol. 36, No 5, pp. 519-554.

[5] Rosenthal R.E. 2015. *GAMS -- A User's Guide.* Washington DC: GAMS Development Corporation.

[6] Bussieck M.R., Meeraus A. 2004. "General algebraic modeling system (GAMS)." In modeling languages in mathematical optimization, pp. 137-157.

[7] Bisschop J., Entriken R. 1993. AIMMS "Advanced Interactive Multidimensional Modeling System (AIMMS): The modeling system.", Paragon Decision Technology.

[8] Jupyter Project. 2015. *Jupyter Notebook.* https://jupyter.org/.

[9] W3C. 2012. "OWL 2 Web Ontology Language.", http://www.w3.org/TR/owl2-primer/.

[10] World Wide Web Consortium. 2012. "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition).", http://www.w3.org/TR/owl2-syntax/.

[11] Denno P. O., Kim D.B., to be published. "Integrating views of properties in models of unit manufacturing processes." *International Journal of Computers in Manufacturing*.

[12] NIST Modeling Methodology for Smart Manufacturing Systems. 2015, https://github.com/usnistgov/modelmeth/tree/master/models/optimization.