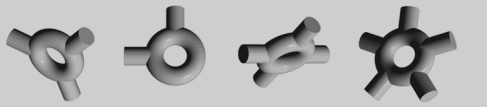
elements



Program Package

TARMS Inc.

September 07, 2000

Typeset in LaTeX.

Contents

List of Figures

List of Tables

Package Description

This package enables user-defined Turing-complete programs to be represented. These user defined programs will be predominantly used to provide unlimited customization of various parts of the system. It is important to realize that this model describes what is effectively the parse tree of a program. It does not describe how this model will be constructed from user interaction.

This user-defined program package could be incorporated into the following areas: date rolling, filtering, modeling the cashflows of a bond deal and in the creation of positions.

1 Interfaces

1.1 Expression

The Expression interface describes objects that can be evaluated. One or more expressions in the form of a parse tree are used to create a program. The Expression interface is used to capture the common behavior between all elements that can make up a parse tree.

Expressions can be of the form “x AND y”, or compound versions thereof. Atomic expressions, such as the value 1, are also considered expressions. In this

case the expression, “1”, evaluates to itself. The behavior of evaluating to the same object is considered the default behavior for an Expression.

Assignment statements, function calls, sequences of expressions, conditional statements, and repetitive statements are all considered Expressions because they can evaluate themselves and return an answer.

1.1.1 Relationships

	Class	Description	Notes
⇑	Validatable		
⇓	ConditionalAction §1.4		
⇓	Assign §1.2		
⇓	Sequence §1.9		
⇓	FunctionCall §1.6		
⇓	BinaryOperation §1.3		
⇓	Not §1.8		
⇓	Documentation §1.5		
⇓	Value §1.10		
↔	SequenceModel §2.17	collection	◊
↔	ConditionalActionModel §2.6	condition	
↔	ConditionalActionModel §2.6	ifTrueExpression	
↔	FunctionModel §2.14	instruction	
↔	IfThenElseModel §2.8	ifFalseExpression	
↔	BinaryOperationModel §2.2	expression2	
↔	BinaryOperationModel §2.2	expression1	
↔	AssignModel §2.1	expression	
↔	NotModel §2.16	expression	

⇑:Inherits ⇓:Inherited by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

1.1.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object> An ordered collection of parameters that this expression is being evaluated with.

This method evaluates “this” expression. The default evaluation of an expression is to return itself (i.e: “this” object.). This is the behavior of atomic expressions, such as numbers.

Boolean validate() validate

An expression can be checked for internal consistency. One example of validation is to check for consistent use of variable types.

1.2 Assign

This interface represents assignment statements of the form “variable = expression”. The assignment will occur when this object is asked to evaluate itself.

1.2.1 Relationships

	Class	Description	Notes
⇑	Expression §1.1		
↓	AssignModel §2.1		

⇑:Inherits ↓:Realized by

1.2.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

This method will evaluate the associated “expression”, assigns it to the associated “variable”, and returns the new value of the variable. Note: It is unlikely that the returned value of an assignment will be used.

Object variable() variable

Returns the Variable object that this Assign object is set-up to assign to.

Expression expression() expression

The expression is the right hand side of the assignment statement. Note that this expression will be evaluated before being assigned to the “variable”.

1.3 BinaryOperation

A binary operation is a simple function with two arguments (called operands). Operands required for the operation are evaluated before being used.

1.3.1 Relationships

	Class	Description	Notes
⇑	Expression §1.1		
↓	BinaryOperationModel §2.2		

⇑:Inherits ↓:Realized by

1.3.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Performs the binary action that this object represents.

Expression expression1() expression1

The first expression statement used by the binary operation.

Expression expression2() expression2

The second expression statement used by the binary operation.

1.4 ConditionalAction

The ConditionalAction interface describes an expression whose evaluation is dependent upon the result of evaluating a “conditional” expression.

1.4.1 Relationships

	Class	Description	Notes
⇑	Expression §1.1		
↓	ConditionalActionModel §2.6		

⇑:Inherits ↓:Realized by

1.4.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Performs the conditional action that this object represents.

Expression condition() condition

An expression which will be used to determine whether other associated expressions are evaluated.

Expression ifTrueExpression() ifTrueExpression

The expression to evaluate if the condition evaluates to false.

1.5 Documentation

The Documentation interface provides a way for various parts of a function to be commented. The presence of documentation is not required, and its presence will not interfere with execution of the program. The Documentation interface enables comments to be stored in the program parse tree. This enables the parse tree to be decompiled into any independent source-code representation.

1.5.1 Relationships

	Class	Description	Notes
⇑	Expression §1.1		
↓	DocumentationModel §2.11		

⇑:Inherits ↓:Realized by

1.5.2 Operations

Nil evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Does nothing and returns nil.

String documentation() documentation

Returns a string containing user documentation.

1.6 FunctionCall

A function call provides a way of executing another function. Every instance of a FunctionCall will be associated with a particular number of variables according to the number of parameters that its associated function requires. When a function call is evaluated, the associated variables will be evaluated and passed as parameters to the associated function, which is then executed.

Note: Variables are passed “by value” only. Allowances could be made to pass “by reference” if such a use was necessary.

1.6.1 Relationships

	Class	Description	Notes
⇑	Expression §1.1		
⇓	OOFunctionCall §1.7		
↓	FunctionCallModel §2.12		

⇑:Inherits ⇓:Inherited by ↓:Realized by

1.6.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object> An ordered collection of parameters that this expression is being evaluated with.

This method evaluates the associated “parameters”, passes them to the associated function as parameters and executes the function.

Function function() function
The function to be executed.

Collection parameters() parameters
A collection of the parameters to be passed to the function upon execution.

1.7 OOFunctionCall

This interface provides a way of representing an object-oriented function call, whereby a function is sent to a target object.

1.7.1 Relationships

	Class	Description	Notes
⇑	FunctionCall §1.6		
↓	OOFunctionCallModel §2.13		

⇑:Inherits ↓:Realized by

1.7.2 Operations

Variable targetObject() targetObject

This method returns the variable whose evaluation will result in the object to which a function call will be sent.

1.8 Not

Represents the boolean negation of an expression.

1.8.1 Relationships

	Class	Description	Notes
⇑	Expression §1.1		
↓	NotModel §2.16		

⇑:Inherits ↓:Realized by

1.8.2 Operations

Object evaluate(OrderedCollection context) evaluate
context: OrderedCollection

Evaluates the expression. If the value returned in not boolean then a NonBooleanValueReturned Exception is raised. Otherwise the negation of the boolean value returned is returned.

Expression expression() expression

The expression to be evaluated.

1.9 Sequence

The sequence interface represents an ordered sequence of expressions. A sequence of expressions is itself considered to be an expression.

1.9.1 Relationships

	Class	Description	Notes
⇑	Expression §1.1		
↓	SequenceModel §2.17		

⇑:Inherits ↓:Realized by

1.9.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

The collection of expressions returned by "collection" are evaluated in order. The return value of the last expression is used as the return value of this method.

Collection collection() collection

The collection of expressions that forms this sequence.

1.10 Value

Value represents a variable or a constant value used within a user defined program.

This interface has an identifier which can be used to enable full decompilation into any independent source-code representation. This name is used to identify the name of the value (variable or constant) to a human reader and to hopefully infer the purpose of the value.

1.10.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
⇑	Expression §1.1		
⇓	Variable §1.12		
⇓	Constant §1.11		
↔	FunctionCallModel §2.12	parameters	
↔	OOFunctionCallModel §2.13	targetObject	

⇑:Inherits ⇓:Inherited by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

1.10.2 Operations

String identifier() identifier

This method returns the name of this value. This name provides a way of identifying a value to a human user. The name is not required for the value to function correctly inside a program, however the use of names enables the parse tree (compiled code) to be decompiled into source code with meaningful constant and variable names.

Object evaluate(OrderedCollection<Object> context) evaluate

context: OrderedCollection<Object>

Returns the value represented by this object.

1.11 Constant

Constant represents a constant value used within a user defined program. Constant value are set when the method is defined and can not be changed by the user defined program.

1.11.1 Relationships

	Class	Description	Notes
⇑	Value §1.10		
↓	ConstantModel §2.10		

⇑:Inherits ↓:Realized by

1.11.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Returns the result of the value operation.

Object value() value

This is the value of the constant. The value can be any object.

1.12 Variable

The Variable interface provides a way for expressions to refer to variables. All the parameters and variables of a program are passed around via the "context". The main purpose of this interface is used to return a particular variable within the context as determined by the "variableNumber". The value of the variable that this interface represents can be set, and read at any time. The setting of a variable can only be achieved through the Assign interface.

1.12.1 Relationships

	Class	Description	Notes
⇑	Value §1.10		
↓	VariableModel §2.19		
↔	FunctionModel §2.14	parameters	
↔	AssignModel §2.1	variable	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

1.12.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Returns the element of the context collection determined by the "index".

Integer index() index

This index is used to refer to a particular element of the context collection. This element is the variable that this object represents.

1.13 Function

The Function interface represents a function that can be called by a user defined program. This Function can be called by a procedural function call.

1.13.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
⇑	Validatable		
⇓	SystemFunction §1.14		
↓	FunctionModel §2.14		
↓	FunctionReferenceDataModel §2.15		
↔	FunctionCallModel §2.12	function	
↔	FunctionReferenceDataModel §2.15	model	

⇑:Inherits ⇓:Inherited by ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

1.13.2 Operations

String identifier() identifier

The name of the function is used to identify the function to a human reader and hopefully infer the purpose of the function. This method also enables the parse tree to be decompiled with meaningful names.

Object executeWith(Collection<Object> passedParameters) executeWith
passedParameters: Collection<Object>

This method initiates the execution of the function that this object represents. The function that this object represents can be called in a procedural manner. A collection of "passedParameters" are used to facilitate the execution of the function that this object represents. This method will return the return-value of the function.

1.14 SystemFunction

The SystemFunction interface describes a function that is part of the elements system and consequently one that is not user definable. A SystemFunction can be called by either a procedural function call or an object-oriented function call.

1.14.1 Relationships

	Class	Description	Notes
⇑	Function §1.13		
↓	SystemFunctionModel §2.18		
↔	OOFunctionCallModel §2.13	function	

⇑:Inherits ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

1.14.2 Operations

String identifier() identifier

This method returns the name of the external function that this object represents. This name is used by the user program, and should corrsepond to a function name in the system.

Object executeWith(Collection<Object> passedParameters) executeWith

passedParameters: Collection<Object>

This method initiates the execution of the external function that this object represents. This method is used to execute external functions that have been called in a procedural manner. An implementation of the elements object model is anticipated to be done in an object-oriented language, and as such all function calls are expected to be directed towards an object. To enable a user to call an external (object-oriented) function in a procedural manner we provide a default target object to which we direct the function calls. This default target object is a single instance of a particular class. This class will have defined all the functions that a user can call in a procedural fashion.

The executeWith method forwards its collection of "passedParameters" onto the executeWithOn method accompanied by the default targetObject. The executeWithOn method will then execute the function that this object represents on the supplied object.

Alternatively this method can be overridden on a subclass to achieve the desired behavior directly.

Object executeWithOn(Collection<Object> passedParameters, Object targetObject) executeWithOn

passedParameters: Collection<Object>

targetObject: Object

Initiates the execution of the external function that this object represents. This method is used to execute functions that have been called in an object oriented manner (as opposed to a procedural manner). This method first verifies the existence of the function X on the targetObject, where X is the string returned by the name method. If this method does not exist then a MessageNotUnderstood exception is raised. Otherwise the function is executed upon the supplied targetObject with the passedParameters and its return value is returned.

Alternatively this method can be overridden on a subclass to achieve the desired behavior directly.

2 Classes

2.1 AssignModel

This is a concrete realization of the Assign interface.

2.1.1 Relationships

	Class	Description	Notes
↑	Assign §1.2		
↔	Variable §1.12	variable	→
↔	Expression §1.1	expression	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.2 BinaryOperationModel

This class is an abstract class which realizes the BinaryOperation interface.

2.2.1 Relationships

	Class	Description	Notes
↑	BinaryOperation §1.3		
⇓	AndModel §2.3		
⇓	OrModel §2.5		
⇓	EqualsModel §2.4		
↔	Expression §1.1	expression2	→
↔	Expression §1.1	expression1	→

⇓:Inherited by ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.3 AndModel

When an instance of this class is evaluated it will return the logical “AND” of its two operands.

2.3.1 Relationships

	Class	Description	Notes
⇑	BinaryOperationModel §2.2		

⇑:Inherits

2.3.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Returns the logical “AND” of the two operands.

2.4 EqualsModel

Check to see whether two expressions evaluate to the same object.

2.4.1 Relationships

	Class	Description	Notes
⇑	BinaryOperationModel §2.2		

⇑:Inherits

2.4.2 Operations

Object evaluate() evaluate

Returns true if the two associated expressions are equal, otherwise returns false.

2.5 OrModel

When an instance of this class is evaluated it will return the logical “OR” of the evaluation of its two operands.

2.5.1 Relationships

	Class	Description	Notes
⇑	BinaryOperationModel §2.2		

⇑:Inherits

2.5.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Returns the logical “OR” of the two operands that it holds.

2.6 ConditionalActionModel

This is a concrete realization of the ConditionAction interface.

2.6.1 Relationships

	Class	Description	Notes
↑	ConditionalAction §1.4		
⇓	WhileModel §2.9		
⇓	IfThenModel §2.7		
↔	Expression §1.1	condition	→
↔	Expression §1.1	ifTrueExpression	→

⇓:Inherited by ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

2.7 IfThenModel

This model represents an expression whose evaluation is dependent upon the result of a conditional expression.

2.7.1 Relationships

	Class	Description	Notes
⇑	ConditionalActionModel §2.6		
⇓	IfThenElseModel §2.8		

⇑:Inherits ⇓:Inherited by

2.7.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Evaluates the associated "conditional" expression, and if that expression evaluates to true, then the "ifTrueExpression" is evaluated. Returns Nil.

2.8 IfThenElseModel

This model represents the evaluation of one of two expressions dependent upon the result of the evaluation of a conditional expression.

2.8.1 Relationships

	Class	Description	Notes
⇑	IfThenModel §2.7		
↔	Expression §1.1	ifFalseExpression	→

⇑:Inherits ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.8.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Evaluates the associated "conditional" expression, and if that expression evaluates to true, then the "ifTrueExpression" is evaluated, otherwise the "ifFalseExpression" is evaluated. Returns Nil.

Expression ifFalseExpression() ifFalseExpression

The expression to evaluate if the condition evaluates to false. Returns the associated "ifFalseExpression".

2.9 WhileModel

The WhileModel describes an expression that will be repeatedly evaluated while some "conditional" expression continues to evaluate to true.

2.9.1 Relationships

	Class	Description	Notes
⇑	ConditionalActionModel §2.6		

⇑:Inherits

2.9.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object>

Evaluates the "ifTrueExpression" while the "condition" expression continues to evaluate to true. Returns null.

2.10 ConstantModel

This class is a concrete realization of the Constant interface.

2.10.1 Relationships

	Class	Description	Notes
↑	Constant §1.11		

↑:Realizes

2.10.2 Attributes

identifier: String The name of the constant value.

value: Object The value of the constant.

2.11 DocumentationModel

A concrete realization of the Documentation interface.

2.11.1 Relationships

	Class	Description	Notes
↑	Documentation §1.5		

↑:Realizes

2.11.2 Attributes

documentation: String

2.12 FunctionCallModel

A concrete realization of the FunctionCall interface.

2.12.1 Relationships

	Class	Description	Notes
↑	FunctionCall §1.6		
⇓	OOFunctionCallModel §2.13		
↔	Function §1.13	function	→
↔	Value §1.10	parameters 0..n	→

⇓:Inherited by ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.13 OOFunctionCallModel

This is a concrete realization of the OOFunctionCall interface.

2.13.1 Relationships

	Class	Description	Notes
⇑	FunctionCallModel §2.12		
↑	OOFunctionCall §1.7		
↔	Value §1.10	targetObject	→
↔	SystemFunction §1.14	function	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.13.2 Operations

Object evaluate(OrderedCollection<Object> context) evaluate
context: OrderedCollection<Object> An ordered collection of parameters that this expression is being evaluated with.

This method evaluates both the associated "parameters" and the associated targetObject and sends them to the associated function.

2.14 FunctionModel

The FunctionModel class represents a function that has been defined by a user. This class is a concrete realization of the Function interface. The static state of a function can be validated.

2.14.1 Relationships

	Class	Description	Notes
↑	Function §1.13		
↔	Variable §1.12	parameters 0..n	→
↔	Expression §1.1	instruction	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.14.2 Attributes

identifier: String A string that is used to identify a particular function model.

parameterTypes: OrderedCollection<Object> The parameter types are used to check the type of the parameters upon entry to the function.

2.14.3 Operations

Object executeWith(OrderedCollection passedParameters) executeWith
passedParameters: OrderedCollection

This method constructs a "context" collection from a copy of the passed parameters plus one additional element (initialized to "null") for each local variable. The held "instruction" is then evaluated with the context.

Dynamic type checking of the incoming parameters can be performed by comparing them against a held collection of parameter types. The functionality of this interface can be extended to allow variables to be passed by reference.

Reportable validate() validate

An instance of a class that realizes this interface can be validated by performing a static type check of the program.

OrderedCollection<Object> parameterTypes() parameterTypes

The parameter types are used to check the type of the parameters upon entry to the function.

2.15 FunctionReferenceDataModel

2.15.1 Relationships

	Class	Description	Notes
⇑	ReferenceDataModel		
↑	Function §1.13		
↔	Function §1.13	model	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.16 NotModel

This class is a concrete realization of the Not interface.

2.16.1 Relationships

	Class	Description	Notes
↑	Not §1.8		
↔	Expression §1.1	expression	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.16.2 Operations

Expression expression(OrderedCollection context) expression
context: OrderedCollection

Returns the associated Expression.

2.17 SequenceModel

A concrete implementation of the Sequence interface.

2.17.1 Relationships

	Class	Description	Notes
↑	Sequence §1.9		
↔	Expression §1.1	collection	→

↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.18 SystemFunctionModel

A concrete implementation of the SystemFunction interface.

2.18.1 Relationships

	Class	Description	Notes
↑	SystemFunction §1.14		

↑:Realizes

2.18.2 Attributes

identifier: String

2.19 VariableModel

This class is a concrete realization of the Variable interface.

2.19.1 Relationships

	Class	Description	Notes
↑	Variable §1.12		

↑:Realizes

2.19.2 Attributes

identifier: String The name of the variable.

index: Integer The position of the variable in the context array.

3 Exceptions

3.1 MessageNotUnderstoodException

This exception will be raised when the object in an “OOFunctionCall” does not implement the function that has been called. A function call is considered to be a “message” that is sent to an object, and, in this case, the message is not understood by the target object.

3.2 TypeMismatchException

This exception will be raised when the program is tested for internal type consistency and is found to have mismatching types.

4 Associations

Table 1: Program— Associations

Association Role	Class	Card.	Notes
collection			
	Expression §1.1		→
	SequenceModel §2.17		◊
parameters			
	Variable §1.12	0..n	→
	FunctionModel §2.14		
condition			
	Expression §1.1		→

Table 1: . . . continued

Association Role	Class	Card.	Notes
	ConditionalActionModel §2.6		
ifTrueExpression			
	Expression §1.1		→
	ConditionalActionModel §2.6		
instruction			
	Expression §1.1		→
	FunctionModel §2.14		
variable			
	Variable §1.12		→
	AssignModel §2.1		
ifFalseExpression			
	Expression §1.1		→
	IfThenElseModel §2.8		
expression2			
	Expression §1.1		→
	BinaryOperationModel §2.2		
expression1			
	Expression §1.1		→
	BinaryOperationModel §2.2		
function			
	Function §1.13		→
	FunctionCallModel §2.12		
parameters			
	Value §1.10	0..n	→
	FunctionCallModel §2.12		
expression			
	Expression §1.1		→
	AssignModel §2.1		
expression			
	Expression §1.1		→
	NotModel §2.16		
model			
	Function §1.13		→
	FunctionReferenceDataModel §2.15		
targetObject			
	Value §1.10		→

Table 1: . . . continued

Association Role	Class	Card.	Notes
	OOFunctionCallModel §2.13		
function			
	SystemFunction §1.14		→
	OOFunctionCallModel §2.13		

→:Navigable ◊:Aggregate ♦:Composite

4.1 collection

Role: *Navigable* Expression.
Role: *Aggregate* SequenceModel.

4.2 parameters

Role: *Navigable* Variable, 0..n.
Role: FunctionModel.

4.3 condition

Role: *Navigable* Expression.
Role: ConditionalActionModel.

4.4 ifTrueExpression

Role: *Navigable* Expression.
Role: ConditionalActionModel.

4.5 instruction

Role: *Navigable* Expression.
Role: FunctionModel.

4.6 variable

Role: *Navigable* Variable.
Role: AssignModel.

4.7 ifFalseExpression

Role: *Navigable* Expression.
Role: IfThenElseModel.

4.8 expression2

Role: *Navigable* Expression.
Role: BinaryOperationModel.

4.9 expression1

Role: *Navigable* Expression.
Role: BinaryOperationModel.

4.10 function

Role: *Navigable* Function.
Role: FunctionCallModel.

4.11 parameters

Role: *Navigable* Value, 0..n.
Role: FunctionCallModel.

4.12 expression

Role: *Navigable* Expression.
Role: AssignModel.

4.13 expression

Role: *Navigable* Expression.
Role: NotModel.

4.14 model

Role: *Navigable* Function.
Role: FunctionReferenceDataModel.

4.15 targetObject

Role: *Navigable* Value.
Role: OOFunctionCallModel.

4.16 function

Role: *Navigable* SystemFunction.
Role: OOFunctionCallModel.

This association is a specialization of the association that is inherited by the OOFunctionCallModel class. This association is specialized because user defined functions do not support object-oriented function calls. The creation of an object-oriented development environment for used defined programs is not currently warranted.

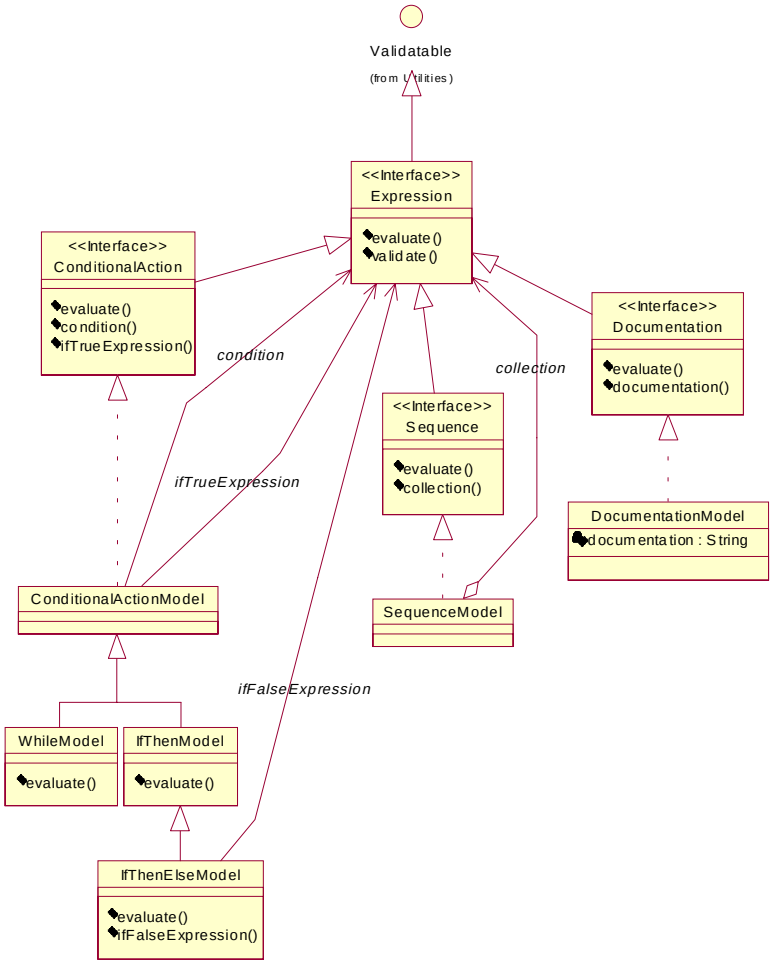


Figure 1: Class Diagram— Constructs

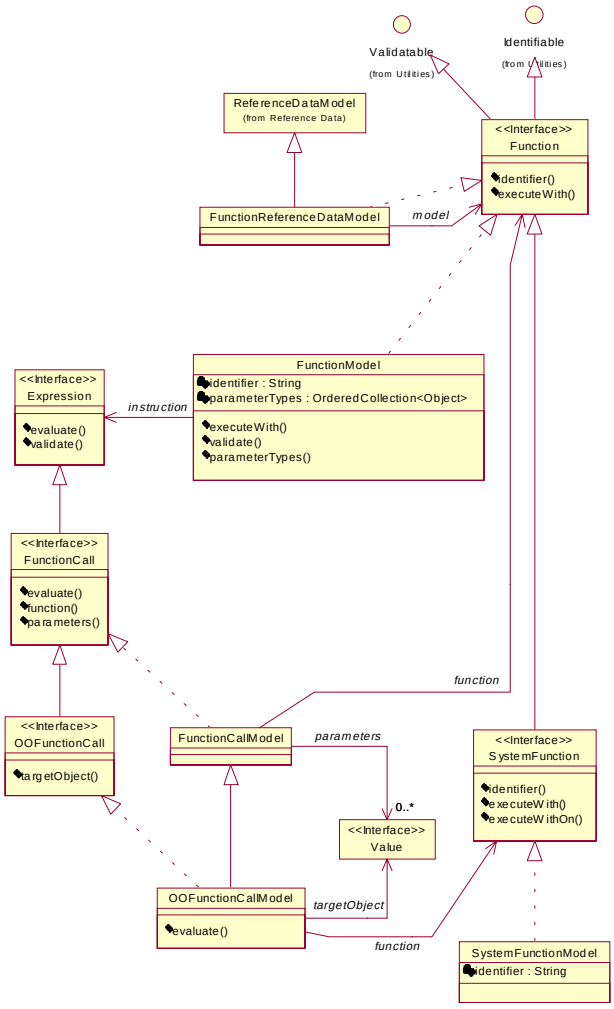


Figure 2: Class Diagram— Functions

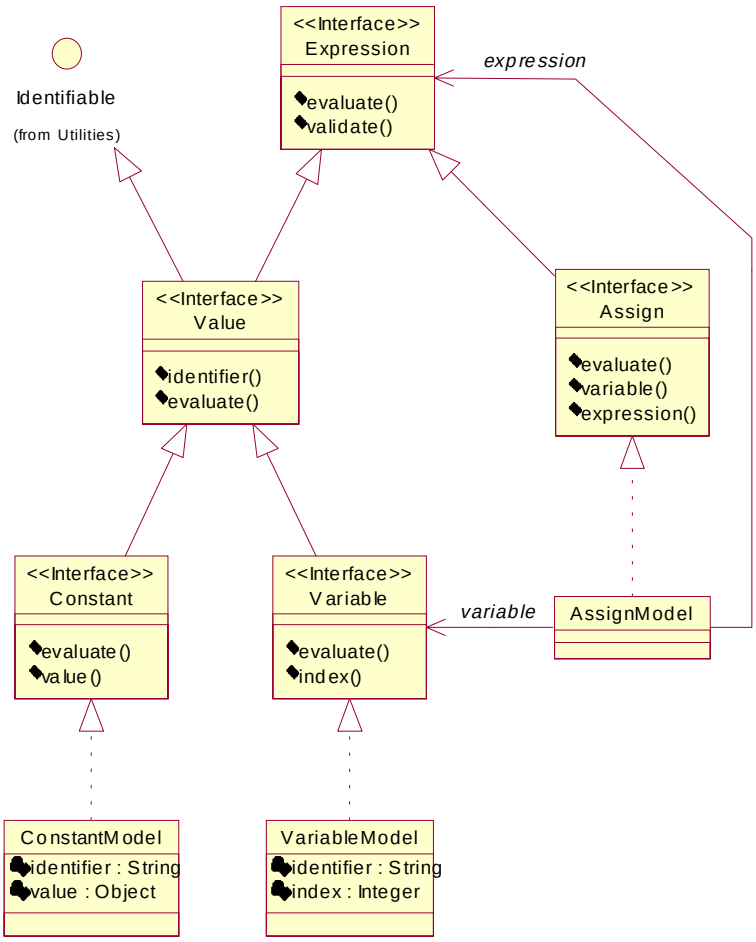


Figure 3: Class Diagram— Variables

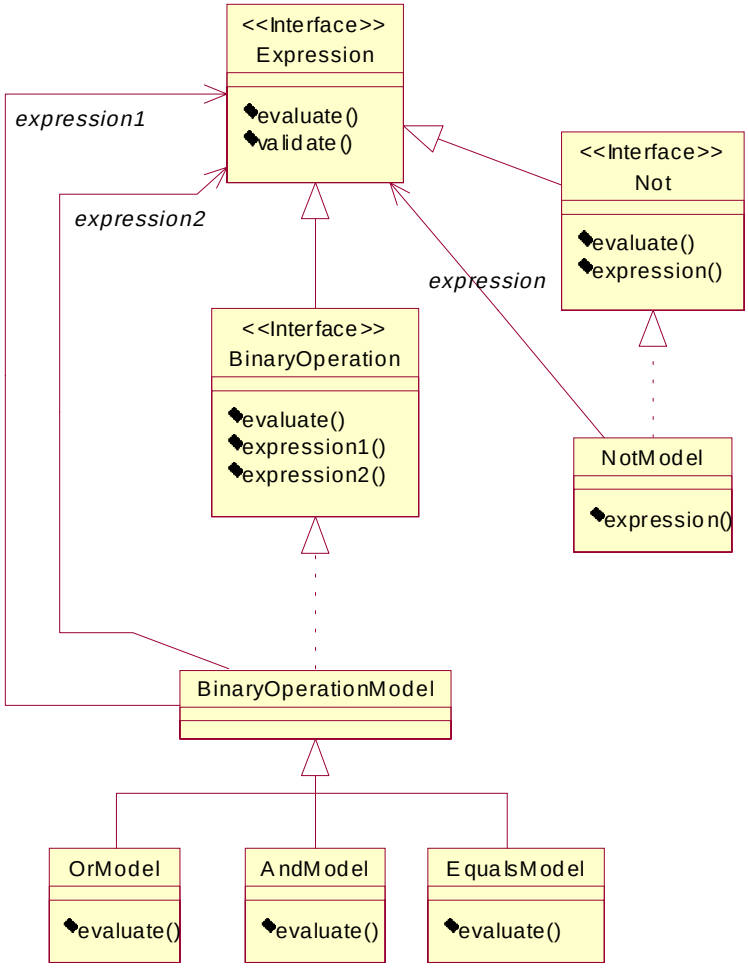


Figure 4: Class Diagram— Primitive Operations

References