Appendix A:
Platform Specific Metamodels and Language Metamodels

The following metamodels are partially described:

A.1: PSM-Eiffel Metamodel
A.2: PSM-Java Metamodel
A.3: ISM-Eiffel Metamodel
A.4: ISM-Java Metamodel
A.5: ISM-C++ Metamodel

Metamodels are specified by using the UML notation:

• Abstract syntax: It consists of one or more UML class diagrams that show the metaclasses defining constructs and relationships. The shaded metaclasses belong to the UML metamodel.
• Metaclasses description: Natural language is used to describe metaclasses, generalizations and associations. Constraints are specified in OCL. Metaclasses are listed in alphabetic order.

A.1 PSM METAMODEL: EIFFEL PLATFORM

Description of Metaclases

AssociationEnd

Description
It represents the own association-ends of the class.

DOI: 10.4018/978-1-61520-649-0.ch015
Appendix A

Figure 1. PSM-Eiffel metamodel: Diagram of classes

Generalization

• Property (from Kernel)

Attributes
No additional attributes

Associations

• class: EiffelClass [0..1] It refers to the class of which this association-end is part

Attribute

Description
It represents the attributes declared in a class Eiffel.

Generalization

• Property (from Kernel)
Attributes

- `isFrozen: Boolean` [1] It specifies whether an attribute is frozen, i.e., constant. In this case, it must have an initial value. It redefines `RedefinableElement::isLeaf`.

Associations

- `class: EiffelClass` [1] It refers to the class declaring this attribute. It redefines `Property::class`.

Constraints

[1] An attribute is a property which is part of a class and but not member of any association. `self.class->size() = 1` and `self.association->isEmpty()` and `self.opposite->isEmpty()`

EiffelClass

Description

An Eiffel class describes a set of objects which share the same specifications of features, restrictions and semantics.
Appendix A

Generalizations

- Class (from Kernel), Classifier (from Templates).

Attributes

- isDeferred Boolean [1] It specifies whether a class is deferred, i.e., one or more features, that are specified but not implemented, are included in the class. It redefines Classifier::isAbstract.
- isExpanded: Boolean [1] It specifies whether the class is flattened, i.e., its instances are objects but no references to objects.

Associations

- associationEnd: AssociationEnd [*] It refers to the own association-ends of the class Eiffel. It is a subset of Class::ownedAttribute.
- attribute: Attribute [*] It refers to the own variables of the Eiffel class. It is a subset of Class::ownedAttribute.
- generalization: Generalization [*] It specifies generalizations.
- invariant: Constraint [*] It refers to class invariants. It redefines NameSpace::ownedRule.
- /parameters: TemplateParameter [*] It refers to the set of parameters of the class. It is derived.
- /parents: EiffelClass [*] It refers to supeclasses of an Eiffel class. It redefines Class::superclass. It is derived.
- routine: Routine [*] It refers the own operations of the class. It redefines Class::ownedOperation.
Constraints

[1] A class, with at least a deferred routine, must be declared deferred. self.ownedRoutine -> exists (r | r.isDeferred) implies self. isDeferred
[2] Private routines of a class can not be declared abstract. self.ownedRoutine -> forAll (r | r.visibility = #private implies not r.isAbstract)
[3] Frozen routines of a class can not be declared deferred. self.ownedRoutine -> forAll (r | r.isFrozen implies not r.isDeferred)
[4] An Eiffel class does not have nested classes. self.nestedClassifier -> isEmpty ()
[5] parents is derived from generalization. parents = self.generalization.parent
[6] parameters is derived from the parameters of the signature template that can be redefined. parameters = ownedSignature.parameter

Function

Description
It declares a function that can be called passing a fixed number of arguments.

Generalizations

• Routine

Attributes
No additional attributes.

Associations

• returnType: Type [1] It refers to the return type of the function. It redefines Operation::type.

Constraints

[1] A function must have a return type and therefore, its set of arguments includes one argument whose type is return. self.ownedParameter -> select (p | p.direction = #return) -> size = 1

Implementation

Description
It specifies a procedure that realizes a routine.

Generalization

• Element (from Kernel)
Appendix A

Attributes

- procedure: Expression [0..1] It refers to the procedure of the routine.

Associations

- invokedRoutine: Routine [*] It refers to the routines invoked in this implementation.
- referencedAttribute: Field [*] It specifies the variables referred in this implementation.
- signature: Routine [1] It refers to routine linked to this implementation.

Constraints

[1] A routine can not call a constructor. 
self.invokedRoutine -> select (r | r.oclIsTypeOf (Procedure))
-> forAll (p | not p.oclAsType (Procedure).isConstructor)

Procedure

Description
It declares a procedure that can be invoked by passing a fixed number of arguments.

Generalizations

- Routine

Attributes

- isConstructor: Boolean [1] It determines whether the procedure is constructor.

Associations

No additional associations.

Constraints

[1] A procedure does not have a return type. 
self.ownedParameter -> select(p | p.direction = #return)
-> isEmpty ()
[2] The constructor of a class can not be abstract. 
self.isConstructor implies not self.isDeferred

Routine

Description
It specifies the characteristics of an Eiffel routine.
Appendix A

Generalizations

- Operation (from Kernel)

Attributes

- isDeferred: Boolean [1] It specifies whether a routine is deferred, i.e., without implementation.
- isFrozen: Boolean [1] It specifies whether a routine is final, i.e., it can not be redefined in a descendant class. It redefines RedefinableElement::isLeaf.

Associations

- body: Implementation [0..1] It refers to the routine implementation.
- class: EiffelClass [1] It refers to the class declaring this routine. It redefines Operation::class.

Constraints

[1] A deferred routine does not have implementations. self.isDeferred implies self.body -> isEmpty()

A.2 PSM METAMODEL: JAVA PLATFORM

Description of Metaclasses

Next, we describe the main metaclasses of the PSM-Java

Figure 4. PSM-Java metamodel: Diagram of classes

![Diagram of classes](image-url)
AssociationEnd

**Description**
It specifies the characteristics of an Eiffel routine.

**Generalization**
- Property (from Kernel)

*Figure 5. PSM-Java metamodel: Diagram of interfaces*

*Figure 6. PSM-Java metamodel: Diagram of classes and interfaces*
Attributes
No additional attributes.

Associations

• class: JavaClass [0..1] It refers to the class of which this association-end is part. It redefines Property::class.

Constraints

[1] An association-end is a property that is member of an association. self.association -> size () = 1

Constructor

Description
It designs an operation that is used to create class instances. They can not be explicitly invoked by means of expressions of method call. Constructors do not have return type and have the same name of the class containing the declaration. Constructor declarations can not be inherited.

Generalization

• JavaOperation
Appendix A

Figure 8. PSM-Java metamodel: Diagram of types (UML, OCL and Java)

Attributes
No additional attributes.

Associations
No additional associations.

Figure 9. PSM-Java metamodel: Diagram of packages
Appendix A

Constraints

[1] Constructors do not have return type. self.type -> isEmpty ()
[2] The constructor name is equal to the name of the class containing the declaration. self.name = self.class.name

Field

Description
It specifies the attributes declared in a class or interface.

Generalization

• Property (from Kernel)

Attributes

• isFinal: Boolean [1] It specifies whether an attribute is final, i.e., constant. In this case, it must have an initial value. It redefines RedefinableElement::isLeaf.
• isTransient: Boolean [1] It specifies whether an attribute is part of the persistent state of the object.
• isVolatile: Boolean [1] It specifies whether an attribute is volatile, i.e., it is accessed asynchronously.

Associations

• class: JavaClass [0..1] It refers to the class declaring this attribute. It redefines Property::class.

Constraints

[1] An attribute is a property that is part of a class and is not member of associations. self.class -> size () = 1 and self.association -> isEmpty () and self.opposite -> isEmpty ()

Implementation

Description
It specifies the procedure of the operation.

Generalization

• Element (from Kernel)
Appendix A

Attributes

- Procedure: Expression [0..1] It refers to the procedure of the operation.

Associations

- invokedMethod: Method [*] It refers to the methods invoked in the body of an operation.
- referencedField: Field [*] It refers to the variables referred in the body of an operation.
- signature: JavaOperation [1] It specifies the operation that implements.

Constraints
No additional constraints.

JavaClass

Description
A Java class describes a set of objects sharing the same specifications of features, constraints and semantics.

Generalizations

- Class (from Kernel), Classifier (from Templates), BehavioredClassifier (from Interfaces)

Attributes

- isFinal: Boolean It specifies whether the class can have subclasses. It redefines RedefinableElement::isLeaf.
- /isGeneric: Boolean It specifies whether the class is generic. It is a derived attribute.
- isStatic: Boolean It specifies whether the class is static.

Associations

- associationEnd: AssociationEnd [*] It refers to the own association-end of the Java class. It is subset of Class::ownedAttribute
- field: Field [*] It refers to own variables of the Java class. It is a subset of Class::ownedAttribute
- /implement: JavaInterface [*] It refers to the Java interfaces implemented by this class. It is derived.
- javaOperation: JavaOperation [*] It refers the own operations of the class. It redefines Class::ownedOperation
- javaPackage: JavaPackage [0..1] It refers to the package in which it is declared. It redefines Type::package.
- nestedClass: JavaClass [*] It refers to the Java classes declared within of the body of a JavaClass (nested classes).
• nestedInterface: JavaInterface [*] It refers to the Java interfaces declared within the body of a
JavaClass (nested interfaces). It is a subset of Class::nestedClassifier.
• /parameters: TemplateParameter [*] It refers the set of parameters of the class. It is derived.
• /superClass: JavaClass [1] It refers to the superclass of a Java class. It redefines Class::superclass.
It is derived.

Constraints

[1] Nested classifiers of a class or interface can only be of the type JavaClass or JavaInterface. self.nestedClassifier -> forAll (c | c.oclIsTypeOf (JavaClass) or c.oclIsTypeOf (JavaInterface))
[2] The implemented interfaces are those referred through the interface realization. implement = self. interfaceRealization.contract
[3] A class having at least an abstract method must be declared abstract. self.javaOperation -> select (op | op.oclIsTypeOf (Method)) -> exists (m | m.oclAsType(Method).isAbstract) implies self.isAbstract
[4] A class that is declared abstract does not have a constructor explicitly defined. self.isAbstract implies self.javaOperation -> select (op | op.oclIsTypeOf (Constructor)) -> isEmpty ()
[5] A class that is declared final does not have subclasses, i.e., it is not superclass of any class. self.isFinal implies self.javaPackage.ownedMember -> select (m | m.oclIsTypeOf (JavaClass)) -> forAll (c | c.oclAsType (JavaClass).superClass <> self)
[6] The access level protected, private and static can be only applied to nested classes, i.e., declared
within the declaration of another class. (self.visibility = #protected or self.visibility = #private or self.isStatic) implies self.javaPackage.ownedMember -> select (m | m.oclIsTypeOf (JavaClass)) -> exists (c | c.oclAsType (JavaClass).nestedClass -> includes (self))
[7] Private methods of a class can not be declared abstract. self.javaOperation -> select (op | op.oclIsTypeOf (Method)) -> forAll (m | m.visibility = #private implies not m.oclAsType (Method).isAbstract)
[8] Static methods of a class can not be declared abstract. self.javaOperation -> select (op | op.oclIsTypeOf (Method)) -> forAll (m | m.isStatic implies not m.oclAsType (Method).isAbstract)
[9] A method that isFinal can not be declared abstract. self.javaOperation -> select (op | op.oclIsTypeOf (Method)) -> forAll (m | m.oclAsType (Method).isFinal implies not m.oclAsType (Method).isAbstract)
[10] A class is generic if has a template signature. isGeneric = (self.ownedTemplateSignature -> size () =1)
[11] parameters is derived starting from of the parameters of the template signature. /parameters = self. ownedTemplateSignature.parameter

JavaInterface

Description
It describes the characteristics of the interfaces in the Java platform.
Appendix A

Generalizations

- Interface (from Interfaces), Classifier (from Templates)

Attributes

No additional attributes.

Associations

- `associationEnd: AssociationEnd [*]` It refers to the own association-ends of a JavaInterface. It is a subset of `Interface::ownedAttribute`.
- `field: Field [*]` It refers to the own fields of a JavaInterface. It is a subset of `Interface::ownedAttribute`.
- `javaPackage: JavaPackage [0..1]` It refers to the package in which it is declared. It redefines `Type::package`.
- `method: Method [*]` It refers to the own methods of a JavaInterface. It redefines `Interface::ownedOperation`.
- `nestedClass: JavaClass [*]` It refers to the classes that are declared within the body of a JavaInterface (nested classes). It is a subset of `Interface::nestedClassifier`.
- `nestedInterface: JavaInterface [*]` It refers to the interfaces that are declared within the body of a JavaInterface (nested interfaces). It is a subset of `Interface::nestedClassifier`.
- `superInterface: JavaInterface [*]` It refers to the super-interfaces of a JavaInterface. It is derived.

Constraints

[1] The Java interfaces are implicitly abstract. self.isAbstract
[2] The own member of an interface are implicitly public. self.ownerMember -> forAll (m | m.visibility = #public)
[3] Nested classifiers of an interface can only be of the type JavaClass or JavaInterface. self.nestedClassifier -> forAll (c | c.oclIsTypeOf (JavaClass) or c.oclIsTypeOf (JavaInterface))
[4] An interface can only be declared private or protected if it is directly nested in the class declaration. (self.visibility = #protected or self.visibility = #private) implies self.package.ownedMember -> select (m | m.oclIsTypeOf (JavaClass)) -> exists (c | c.oclAsType (JavaClass).nestedInterface -> includes (self))
[5] An interface can only be declared static if it is directly nested in the class or interface declaration. self.isStatic implies self.package.ownedMember -> select (m | m.oclIsTypeOf (JavaClass)) -> exists (c | c.oclAsType (JavaClass).nestedInterface -> includes (self)) or self.package.ownedMember -> select (m | m.oclIsTypeOf (JavaInterface)) -> exists (i | i.oclAsType (JavaInterface).nestedInterface -> includes (self))
[6] Methods that are declared in an interface are abstract and hence do not have implementations. self.method ->forAll (m| m.isAbstract and m.body -> isEmpty ())
[7] Methods of an interface can not be declared static. self.method -> forAll (m| not m.isStatic)
Methods of an interface can not be synchronized. self.method -> forAll (m | not m.isSynchronized)

Fields of an interface are implicitly public, static or final. self.field -> forAll (f | f.visibility = #public and f.isStatic and f.isFinal)

superInterface is derived of the generalization. /superInterface = self.generalization.general

Parameters are derived from the parameters of the template signature. /parameters = self.ownedTemplateSignature.parameter

JavaOperation

Description
It describes the characteristics of the interfaces in the Java platform.

Generalization

• Operation (from Kernel, from Interfaces)

Attributes
No additional attributes.

Associations

• class: JavaClass [0..1] It refers to the class declaring the operation. It redefines Operation::class.
• body: Implementation [0..1] It refers to the implementation of the operation.
• javaException: JavaClass [*] It refers to the types that represent the exceptions that can appear during an invocation of this operation.

Constraints

[1] An abstract operation does not have implementation. self.isAbstract implies self.body -> isEmpty ()

JavaPackage

Description
It is used for grouping elements. Its members can be classes, interfaces or sub-packages.

Generalization

• Package (from Kernel)

Attributes
No additional attributes.
Appendix A

Associations

- javaClass: JavaClass [*] It refers to classes that are members of this package. It is a subset of Package::ownedType.
- javaInterface: JavaInterface [*] It refers to all interfaces that are members of this package. It is a subset of Package::ownedType.
- /subpackage: Package [*] It refers to the packages that are members of this package. It is derived.

Constraints
No additional constraints.

Method

Description
It declares an operation that can be invoked by passing a fixed number of the arguments.

Generalizations

- JavaOperation

Attributes

- isAbstract: Boolean [1] It specifies whether a method is abstract, i.e., it does not have implementation.
- isFinal: Boolean It specifies whether a method is final. In this case, it cannot be overwritten in a derived class. It redefines RedefinableElement::isLeaf.
- isSynchronized: Boolean [1] It specifies whether a method is synchronized. It is true if acquires a lock before execution.

Associations

- interface: JavaInterface [0..1] It declares the interface that declares this method. It redefines Operation::interface.

Constraints
No additional restrictions.
A.3 ISM METAMODEL: EIFFEL LANGUAGE

Description of Metaclasses

Attribute

Description
It represents the attributes that are declared in a class, according to the specification of the Eiffel language.

Generalization

- Property (from Kernel).

Attributes

- isConstant: Boolean [1] It specifies whether an attribute is constant. In this case it must have a compulsory initial value.

Associations

- class: EiffelClass [1] It refers to the class that declares this attribute. It redefines Property::class.
- type: EiffelClass [1] It refers to the type of this attribute. It redefines TypedElement::type.

Constraints

[1] An attribute is a property that is part of a class and is not member of associations. self.class -> size () = 1 and self.association -> isEmpty () and self.opposite -> isEmpty ()

Assertion

Description
It describes assertions, according to the specification of the Eiffel language.

Generalization

- Constraint (from Kernel)

Attributes

- Tag_Mark: Identifier [0..1] It refers to the identifier of the assertion. It redefines NamedElement::name.
Associations

- class [0..1]: EiffelClass. It refers to the class that is the context in which this restriction is evaluated. It is a subset of Constraint::context.
- routine [0..1]: Routine. It refers to the routine that is the context in which this restriction is evaluated. It is a subset of Constraint::context.

Constraints

No additional restrictions.
Appendix A

Figure 11. ISM-Eiffel metamodel: Diagram of operations

Argument

Description
It describes the arguments of a routine.
Appendix A

Figure 12. ISM-Eiffel metamodel: Diagram of types

Figure 13. ISM-Eiffel metamodel: Diagram of clusters
Appendix A

Generalization

• Parameter (from Kernel)

Attributes

No additional attributes.

Associations

• type: EiffelClass [1] It refers to the type of this argument. It redefines TypedElement::type.

Constraints

No additional constraints.

Cluster

Description

It is used to group and organize classes in Eiffel.

Generalization

• Package (from Kernel)

Attributes

No additional attributes.

Associations

• ownedClass: EiffelClass [ ] It refers to Eiffel classes that are members of this cluster. It redefines Package::ownedType.

Constraints

No additional constraints.

Compound

Description

It describes a set of instructions, according to the specification of the Eiffel language.

Generalization

• Element (from Kernel)
Appendix A

Attributes
No additional attributes.

Associations
• instruction: Instruction [*] It specifies the set of instructions that forms the compound. It is ordered.

Constraints
No additional constraints.

EntityDeclaration

Description
It describes a local entity of a routine, according to the specification of the Eiffel language.

Generalization
• TypedElement (from Kernel)

Attributes
No additional attributes.

Associations
• type [1]: EiffelClass It specifies the type of the entity. It redefines TypedElement::type.

Constraints
No additional constraints.

EiffelClass

Description
An Eiffel class describes a set of objects sharing the same feature specifications, restrictions and semantics.

Generalizations
• Class (from Kernel), Classifier (from Templates)

Attributes
• isDeferred: Boolean [1] It specifies whether a class is deferred, i.e., it includes one or more features that are specified but no implemented. It redefines Classifier::isAbstract.
Appendix A

- isExpanded: Boolean [1] It specifies whether the class is flattened, i.e. its instances are objects but no references to objects.
- isObsolete: Boolean [1] It specifies whether the class is obsolete.

Associations

- attribute: Attribute [*] It refers to the own attributes of the Eiffel class. It redefines `Class::ownedAttribute`.
- eiffelFeatures: EiffelFeature [*] It refers the features of which this class is client.
- generalization: Generalization [*] It specifies the generalization for this class.
- invariant: Assertion [*] It refers to invariants of the class. It redefines `NameSpace::ownedRule`.
- ownedRoutine: Routine [*] It refers to the own routines of the class. It redefines `Class::ownedOperation`.
- /parameters: EiffelParameter [*] It refers to the set of parameters of the class. It is derived.
- /parent: EiffelClass [*] It refers to the parent class of an Eiffel class. It redefines `Class::superClass`.
- It is derived.

Constraints

1. A class having a deferred routine must be declared deferred. `self.ownedRoutine -> exists (r | r.isDeferred) implies self. isDeferred`
2. Secret routines can not be declared deferred. `self.ownedRoutine -> forAll (r | r.availability = #secret implies not r.isDeferred)`
3. Frozen routines of a class can not be declared deferred. `self.ownedRoutine -> forAll (r | r.isFrozen implies not r.isDeferred)`
4. An Eiffel class has not nested classes. `self.nestedClassifier -> isEmpty ()`
5. `ancestors` is derived of the generalization. `ancestors = self.generalization.parent`
6. `parameters` is derived from the parameters of the template signature that can be redefined. `parameters = ownedSignature.parameter`
7. Parameters of a class are of the type Eiffel class. `self.parameters.parameteredElement -> forAll (p | p.oclIsTypeOf (EiffelClass))`
8. A deferred class has not creation procedure. `self.class.isDeferred implies self.ownedRoutine -> select (p | p.oclIsTypOf (Procedure) and p.isCreator) -> isEmpty ()`
9. A flattened class has only a creation procedure without arguments. `self.class.isExpanded implies self.ownedRoutine -> select (p | p.oclIsTypeOf (Procedure) and p.isCreator) -> size () = 1 and self.ownedRoutine -> select (p | p.isCreator and p.argument -> isEmpty ()) -> size () = 1`
10. A flattened class does not have parameters. `self.class.isExpanded implies self.parameter -> isEmpty ()`

EiffelParameter

Description
It specifies the parameters of a class, according to the specification of the Eiffel language.
Appendix A

Generalizations

• TemplateParameter (from Templates)

Attributes
No additional attributes.

Associations
No additional associations.

Constraints

[1] The type of the parameters of a class is EiffelClass. self.parameteredElement -> forAll (p | p.oclIsTypeOf (EiffelClass))

EiffelFeature

Description
It declares a feature, according to the specification of the Eiffel language.

Generalizations

• NamedElement

Attributes

• availability: FeatureAvailability [1] It refers to the availability of the feature. It redefines NamedElement::visibility.

Associations

• clients: EiffelClass[*] It refers to the classes for which this feature is available.

Constraints

[1] If the feature is selectively available, then it must be associated to a list of clients, else the list of clients is empty. if self.availability = #selectively_available then self.client -> size () > 0 else self.client -> isEmpty() endif

FeatureAvailability

Description
FeatureAvailability is an enumeration of the following values:
• available
• secret
• selectively available

which determine whether a feature is available in all classes, some classes or no classes.

Generalizations
• None

Function

Description
It declares a function, according the specification of the Eiffel language.

Generalizations
• Routine

Attributes
No additional attributes.

Associations
• /type: EiffelClass[1] It refers to the return type of the function. It redefines TypedElement::type.

Constraints
No additional restrictions.

Instruction

Description
It describes an instruction, according to the specification of the Eiffel language.

Generalizations
• NamedElement (from Kernel)

Attributes
No additional attributes.

Associations
• routineBody: RoutineBody It refers to the body of the routine of which this instruction forms a part.
Appendix A

- routine: Routine It refers to the routine that declares the clause rescue of which this instruction is a part.

Constraints
No additional constraints.

Routinebody

Description
It specifies the body of the routine, according to the specification of the Eiffel language.

Generalization
- Element (from Kernel)

Attributes
- is Deferred: Boolean It specifies whether the body is deferred, i.e., it is not implemented.

Associations
- signature: Routine [1] It refers to the routine to which corresponds this implementation.
- instruction: Instruction [0..1] It refers to the instruction that composes the body of the routine.

Constraints

[1] If the body if the routine is deferred, then the routine declaring it is also deferred. self.isDeferred implies self.signature.isDeferred

Procedure

Description
It declares a procedure, according to the specification of the Eiffel language.

Generalizations
- Routine

Attributes
- isCreator: Boolean [1] It determines whether the procedure is the creation.

Associations
No additional associations.
Appendix A

Constraints

[1] A procedure does not have a return type. self.ownedParameter -> select (p | p.direction = #return)
-> isEmpty ()
[2] If a procedure is a creation procedure then it can not be deferred. self.isCreator implies not self.
isDeferred

Routine

Description
It specifies the features of a routine Eiffel.

Generalizations

• Operation (from Kernel), Feature

Attributes

• isDeferred: Boolean [1] It specifies whether a routine is deferred, i.e., it does not have implementation.
• isFrozen: Boolean [1] It specifies whether a routine is final, i.e., it can not be redefined in a descendent class. It redefines RedefinableElement::isLeaf
• isOnce: Boolean [1] It specifies whether the routine is executed only a time.
• isObsolete: Boolean [1] It specifies whether the routine is obsolete.

Associations

• argument: Argument [*] It refers to the formal arguments of the routine. It redefines Operation::OwnedParameter.
• body: RoutineBody [1] It refers to the implementation of the routine.
• class: EiffelClass [1] It refers to a class that declares this routine. It redefines Operation::class.
• ensure: Assertion [*] It specifies the postconditions of the routine. It redefines Operation::postcondition.
• localEntity: EntityDeclaration [*] It specifies the local entities of the routine.
• require: Assertion [*] It specifies the preconditions of the routine. It redefines Operation::precondition.
• rescue: Instruction [0..1] It specifies that the answer to an exception occurs during the execution of the routine.

Constraints

[1] If a routine is deferred then it does not have implementation. self.isDeferred implies self.body-
>isEmpty ()
[2] If a routine is frozen, then it can not be deferred. self.isFrozen implies self.isDeferred
Appendix A

Simple Instruction

Description
It describes a simple instruction, according to the specification of the Eiffel language.

Generalizations

• NamedElement (from Kernel)

Attributes
No additional attributes.

Associations
No additional associations.

Constraint
No additional restrictions.

A.4 ISM METAMODEL: JAVA LANGUAGE

Description of Metaclasses

Block

Description
It specifies the code block implementing an operation, according to the specification of the Java language.

Generalizations

• Action (from Action)

Attributes
No additional attributes

Associations

• blockStatement: blockStatement [0..1] It refers to the statements of which this block is part.
• implementation: Implementation [1] It refers to the implementation of which this block is part.
Figure 14. ISM-Java metamodel: Diagram of classes

Figure 15. ISM-Java metamodel: Diagram of interfaces
Appendix A

Figure 16. ISM-Java metamodel: Diagram of classes and interfaces

Constraints
No additional constraints.

Constructor

Description
It is a constructor, according to the definition in the Java language.

Generalization

• JavaOperation

Attributes
No additional attributes.

Associations
No additional associations.

Constraints

[1] A constructor does not have a return type. self.returnType -> isEmpty ()
[2] The constructor name is equal to the class name including the declaration. self.name = self.class. name

Field

Description
It represents an attribute, as is defined in the Java language.
Generalization

- Property (from Kernel)

Attributes

- isFinal: Boolean [1] It specifies whether an attribute is final, i.e., constant. If an attribute is final, then it must have an initial value.
- isTransient: Boolean [1] It specifies whether an attribute is part of the persistent state of the object.
- isVolatile: Boolean [1] It specifies whether an attribute is volatile, i.e., it is accessed non-synchronously.
Appendix A

Figure 18. ISM-Java metamodel: Diagram of implementations

Associations

- `class: JavaClass [0..1]` It refers to the class declaring this attribute. It redefines `Property::class`.
- `javaType: JavaType [1]` It refers to the attribute type. It redefines `TypedElement::type`.

Constraints

[1] An attribute is a property that is a part of a class and is not member of associations. `self.class -> size () = 1 and self.association -> isEmpty () and self.opposite -> isEmpty ()`

Implementation

Description
It specifies a procedure that obtains the result of the operation.

Generalization

- Element (from Kernel)

Attributes
No additional attributes.
Appendix A

Figure 19. ISM-Java metamodel: Diagram of types

Associations

- block: Block [1] It specifies the code block of the implementation.
- invokedMethod: Method [*] It refers to the methods invoked in the body of an operation.
- referencedField: Field [*] It refers to the variables referred in the body of an operation.
- signature: JavaOperation [1] It specifies the operation signature

Constraints

No additional constraints.

JavaClass

Description

A Java class as is defined in the Java language.

Generalizations

- Class (from Kernel), Classifier (from Templates), BehavioredClassifier (from Interfaces)

Attributes

- isFinal: Boolean It specifies whether the class can have subclasses. It redefines RedefinableElement::isLeaf.
- /isGeneric: Boolean It specifies whether the class is generic. It is a derived attribute.
- isStatic: Boolean It specifies whether the class is static.
Appendix A

Figure 20. ISM - Java Metamodel: Diagram of packages

Associations

- field: Field [*] It refers to the own variables of the Java class. It redefines Class::ownedAttribute.
- /implement: It refers to the Java interfaces that are implemented by this class. It is derived.
- javaOperation: JavaOperation [*] It refers the own operations of the class. It redefines Class::ownedOperation.
- javaPackage: JavaPackage [0..1] It refers to the package in which is declared. It redefines Type::package.
- nestedClass: JavaClass [*] It refers to the Java classes that are declared within the body of a Java class (nested classes). It is a subset of Class::nestedClassifier.
- nestedInterface: JavaInterface [*] It refers to the Java interfaces that are declared within the body of a JavaClass (nested interfaces). It is a subset of Class::nestedClassifier.
- /parameters: JavaParametes [*] It refers to the set of parameters of a class. It is derived.
- /superClass: JavaClass [1] It refers to a superclass of a Java class. It redefines Class::superClass. It is derived.

Constraints

[1] Nested classifiers belonging to a class or interface can only be of type JavaClass or JavaInterface. self. nestedClassifier -> forAll (c | c.oclIsTypeOf (JavaClass) or c.oclIsTypeOf (JavaInterface))

[2] The implemented interfaces are those referred through the interface realization. implement = self. interfaceRealization.contract
Appendix A

[3] A class that has at least one abstract method must be declared abstract. self.javaOperation -> select (op | op.ocIsTypeOf (Method)) -> exists (m | m.oclAsType (Method).isAbstract) implies self.isAbstract.

[4] An abstract class does not have a constructor defined explicitly. self.isAbstract implies self.javaOperation -> select (op | op.ocIsTypeOf (Constructor)) -> isEmpty ()

[5] A class that is declared final cannot have subclasses, i.e., it is not superclass of any class in the package. self.isFinal implies self.javaPackage.ownedMember -> select (m | m.ocIsTypeOf (JavaClass)) -> forAll (c | c.oclAsType (JavaClass).superClass <> self)

[6] The access level protected, private or static can only be applied to nested classes, i.e., that are declared within the declaration of another class. (self.visibility = #protected or self.visibility = #private or self.isStatic) implies self.javaPackage.ownedMember -> select (m | m.ocIsTypeOf (JavaClass)) -> exists (c | c.oclAsType (JavaClass).nestedClass -> includes(self))

[7] Private methods of a class can not be declared abstract. self.javaOperation -> select (op | op.ocIsTypeOf (Method)) -> forAll (m | m.visibility = #private implies not m.ocIsTypeOf (Method).isAbstract)

[8] Static methods of a class can not be declared abstract. self.javaOperation -> select (op | op.ocIsTypeOf (Method)) -> forAll (m | m.isStatic implies not m.ocIsTypeOf (Method).isAbstract)

[9] Final methods of a class can not be declared abstract. self.javaOperation -> select (op | op.ocIsTypeOf (Method)) -> forAll (m | m.ocIsTypeOf (Method)).isFinal implies not m.ocIsTypeOf (Method).isAbstract)

[10] A class is generic if it has a signature template. isGeneric = (self.ownedTemplateSignature -> size () =1)

[11] Parameters are derived through the parameters of the signature template. /parameters= self.ownedTemplateSignature.parameter

[12] A class is concrete, if its methods have associated an implementation. not self.isAbstract implies self.allMethod () -> forAll (m | self.allBody () -> exist (b | b.signature = m))

[13] Elements, that can be actual parameters of a formal parameter, are of type Java types. self.parameters.parameteredElement -> forAll (p | p.ocIsTypeOf (JavaType))

Additional Operations

[1] allMethod is the set of all methods, i.e., the methods that are own, inherited and the methods of the interfaces implemented. allMethod () : Set(Method) allMethod () = self.allClassMethod() -> union(self.implement.allInterfaceMethod()) allClassMethod(): Set(Method) allClassMethod () = self.javaOperation -> select (o | o.ocIsTypeOf (Method)) -> union (self.superClass.allClassMethod () allInterfaceMethod () : Set (Method) allInterfaceMethod () = self.method -> union(self.super-Interface.allInterfaceMethod())

[2] allBody is the set of all method implementations of a class, i.e., both own and inherited. allBody () : Set (Implementation) allBody = self.allMethod ().body

JavaInterface

Description
It describes the characteristics of an interface according to the Java language.
Appendix A

Generalizations

- Interface (from Interfaces), Classifier (from Templates).

Attributes
No additional attributes.

Associations

- field: Field [*] It refers to the own fields of a JavaInterface. It redefines Interface::ownedAttribute.
- javaPackage: JavaPackage [0..1] It refers to the package in which is declared. It is subset of Type::package.
- method: Method [*] It refers to the own methods of a JavaInterface. It redefines Interface::ownedOperation.
- nestedClass: JavaClass [*] It refers to the classes that are declared within the body of a JavaInterface (nested classes). It is a subset of Interface::nestedClassifier.
- nestedInterface: JavaInterface [*] It refers all interfaces that are declared within the body of a JavaInterface (nested interfaces). It is a subset of Interface::nestedClassifier.
- /parameter: JavaParameter [1] It refers to the set of parameters of an interface. It is derived.
- /superInterface: JavaInterface [*] It refers to the superinterfaces of a JavaInterface. It is derived. It redefines Classifier::general

Constraints

[1] Interfaces are implicitly abstract. self.isAbstract
[2] The own members of an interface are implicitly public. self.ownerMember -> forAll (m | m.visibility = #public)
[3] Nested classifiers of an interface can only be of the type JavaClass or JavaInterface. self.nestedClassifier -> forAll (c | c.oclIsTypeOf (JavaClass) or c.oclIsTypeOf (JavaInterface))
[4] An interface that is directly nested in the class declaration can only be declared private or protected. (self.visibility = #protected or self.visibility = #private) implies self.package.ownedMember -> select (m | m.oclIsTypeOf (JavaClass)) -> exists (c | c.oclAsType (JavaClass).nestedInterface -> includes (self))
[5] An interface that in the class or interface declaration is nested can only be declared static. self.isStatic implies self.package.ownedMember -> select (m | m.oclIsTypeOf (JavaClass)) -> exists (c | c.oclAsType (JavaClass).nestedInterface -> includes (self)) or self.package.ownedMember -> select (m | m.oclIsTypeOf (JavaInterface)) -> exists (I | i.oclAsType (JavaInterface).nestedInterface -> includes (self))
[6] Methods that are declared in an interface are abstract and hence do not have implementation. self.method -> forAll (m | m.isAbstract and m.body -> isEmpty ())
[7] Methods of an interface can not be declared static. self.method -> forAll (m | not m.isStatic)
[8] Methods of an interface can not be synchronized. self.method -> forAll (m | not m.isSynchronized)
Fields of an interface are implicitly public, static or final. self.field -> forAll (f | f.visibility = #public and f.isStatic and f.siFinal)

superInterface is derived of the generalization. /superInterface = self.generalization.general

parameters are derived through the parameters of the signature template. /parameters = self.ownedTemplateSignature.parameter

Elements that can be actual parameters of a formal parameter are types of Java. self.parameters. parameteredElement -> forAll (p | p.oclIsTypeOf (JavaType))

JavaOperation

Description
It describes a method according to the specification of the Java language.

Generalizations

• Operation (from Kernel)

Associations

• body: Implementation [0..1] It refers to the implementation of the operation.
• class: JavaClass [0..1] It refers to the class that implements this operation. It redefines Operation::class.
• javaExceptions: JavaClass [*] It refers to the types representing the exceptions that can occur during an invocation of this operation. It redefines Operation::raisedException.
• parameter: OperationParameter [*] It specifies the parameter of the operation. It redefines Operation::ownedParameter
• /returnType: JavaType [0..1] It specifies the return type of the operation. It redefines Operation::type.

It is derived.

Constraints

[1] An abstract operation does not have implementation. self.isAbstract implies self.body -> isEmpty ()

JavaPackage

Description
It is a package as is defined in the Java language.

Generalizations

• Package (from Kernel)
Appendix A

Attributes
No additional attributes.

Associations

- javaClass: JavaClass [*] It refers to the classes that are members of this package. It is a subset of Package::ownedType.
- javaInterface: JavaInterface [*] It refers to the interfaces that are members of this package. It is a subset of Package::ownedType.
- /subpackage: JavaPackage [*] It refers to the packages that are members of this package. It redefines Package::nestedPackage. It is derived.

Constraints

[1] Members of a package can only be classes, interfaces or sub-packages. self.ownedMember -> forAll (m | m.oclIsTypeOf (JavaInterface) or m.oclIsTypeOf (JavaClass) or m.oclIsTypeOf (JavaPackage))

Method

Description
It describes a method according to its definition in Java language.

Generalizations

- JavaOperation

Attributes

- isAbstract: Boolean [1] It specifies whether a method is abstract, i.e., without implementation.
- isFinal: Boolean [1] It specifies whether a method is final. In this case, it can not be overwritten in a derived class. It redefines RedefinableElement::isLeaf.
- isNative: Boolean [1] It specifies whether a method is native.
- isSynchronized: Boolean [1] It specifies whether a method is synchronized. It is true if acquires a lock before execution.

Associations

- interface: JavaInterface [0..1] It declares the interface declaring this method. It redefines Operation::interface.

Constraints

[1] A native method can not be abstract. self.isNative implies not self.isAbstract
If a method has a return type then it must have a return statement. \( \text{self.type -> size() = 1} \) \textbf{implies} \( \text{self.body.block.oclIsTypeOf(Return)} \) \textbf{or} \( \text{self.body.block.oclIsKindOf(BlockStatement)} \) \textbf{and} \( \text{self.body.block.allStatement() -> exists (sent | sent.oclIsTypeOf(Return))} \)

**Additional operations**

allStatement is the set of all statements that conforms the body of the method. \( \text{allStatement()}: \text{Set(Statement)} \) \( \text{allStatement()} = \text{self.subBlock -> union (self.subBlock.allStatement())} \)

**OperationParameter**

**Description**
It specifies the parameters of an operation according to the specification of the Java language.

**Generalization**

- Parameter (from Kernel)

**Attributes**
No additional attributes.

**Associations**

- type: JavaType [1] It refers to the type of the parameter. It redefines \textit{TypedElement::type}.

**Constraints**
No additional constraints.

**A.5 ISM METAMODEL: C++ LANGUAGE**

**Descriptions of Metaclasses**

**C++ Class**

**Description**
A C++ class describes a set of objects that share the same specifications of features, restrictions and semantics.

**Generalizations**

- Class (from Kernel), Classifier (from Templates)

**Attributes**
Appendix A

Figure 21. ISM–C++ metamodel: Diagram of classes

- `class-key`: Class-Key [1] It specifies the type of the class, i.e., if it is a class, structure or union.
- `isFinal`: Boolean [1] It specifies if the class has subclasses. It redefines `RedefinableElement::isLeaf`.
- `/isGeneric`: Boolean It specifies if the class is generic. It is a derived attribute.

**Associations**

- `variable`: Variable [*] It refers to the own variables of the C++ class. It redefines `Class::ownedAttribute`.
- `nestedClass`: C++Class [*] It refers to the C++ classes that are declared within the body of a C++ class (nested classes). It is a subset of `Class::nestedClassifier`.
- `/superClass`: C++Class [*] It refers to the superclasses of a C++ class. It redefines `Class::superClass`. It is derived.
- `function`: C++MemberFunction [*] It refers to the own functions of the class. It redefines `Class::ownedOperation`.
- `generalization`: C++Generalization [*] It refers to the generalizations of the class. It redefines `Class::Generalization`.
- `friendClass`: C++Class [*] It refers to the friend classes of the class.
Appendix A

Figure 22. ISM-C++ metamodel: Diagram of functions

- **friendFunction**: C++Function [*] It refers to the friend functions of the class.
- **/parameters**: C++Parameter [*] It refers to the set of parameters of the class. It is derived.

**Constraints**

1. A class that has pure virtual functions must be declared abstract. 
   
   `self.function -> select (oclIsTypeOf (Method)) -> exists (m | m.oclAsType (Method).isPureVirtual) implies self.isAbstract`

2. A class declared final does not have subclasses, i.e., it is not superclass of any class belonging to the package.
   
   `self.isFinal implies self.package.ownedMember -> select (oclIsTypeOf (C++Class)) -> forAll (c | c.oclAsType (C++Class).superClass <> self)`
Appendix A

Figure 23. ISM–C++ metamodel: Diagram of implementations

[3] Private functions of a class can not be declared abstract. self.function -> select (oclIsTypeOf(Method)) -> forAll (m | m.visibility = #private implies not m.oclAsType (Method).isPureVirtual)

[4] Final methods of a class can not be declared abstracts. self.function -> select (oclIsTypeOf(Method)) -> forAll (m | m.oclAsType (Method).isFinal implies not m.oclAsType (Method).isVirtual)

[5] A class is generic if it has a signature template. isGeneric = (self.ownedTemplateSignature -> size () =1)

[6] Parameters are derived from parameters of the signature template that is redefinable. /parameters = self.ownedTemplateSignature.parameter

[7] Friend functions are C++ functions but no member functions of a class. self.friendFunction -> forAll (f | f.isTypeOf (C++Function))

[8] A class only has a destructor. self.function -> select (oclIsTypeOf (Destructor)) ->size() <=1

C++File

Description
It represents a C++ file.

Generalizations

• Namespace (from Kernel)

Attributes

• extension: FileExtension [1] It specifies the extension of the file, i.e., if the file is header then “h”, else, if it is an implementation “c” or “cpp”. 
Associations

- `c++Project`: C++Project [1] It refers to the project of which the file is part.
- `/includedFile`: C++File [*] It refers to the set of files that are included. It is derived.
- `globalVariableandConstantDeclaration`: GlobalVariableandConstantDeclaration[*] It refers to the set of global variables and constants that are declared.
- `precompilerDirectives`: PrecompilerDirectives [*] It refers to the set of precompiler directives.
- `classifierDeclaration`: ClassifierDeclaration [*] It refers to the set of classifier declarations.
- `classifierDefinition`: ClassifierDefinition [*] It refers to the set of classifier definitions.
- `functionDeclaration`: FunctionDeclaration [*] It refers to the set of function declarations.
- `functionDefinition`: FunctionDefinition [*] It refers to the set of function definitions.

Constraints

[1] The included files are derived through the files that are included by the directive `#include`. `/includedFile = self.precompilerDirective -> collect (oclIsTypeOf (Include).headerFile)`

C++Function

Description
It is a C++ function.

Generalizations

- Operation (from Kernel)
Appendix A

Figure 25. ISM–C++ metamodel: Diagram of projects

Attributes

- isVarArg: Boolean It specifies whether the function can have variable arguments.
- linkage: Linkage-Specifier It specifies whether the function is extern, indicating to compiler that the definition of the function is in another file, or is static, i.e., its name is invisible on the outside of the file declaring it.
- isInLine: Boolean True means that the compiler will replace the function call by the function code.

Associations

- parameter: FunctionParameter [*] It specifies the function parameters. It redefines Operation::ownedParameter.
- pre: C++Assertion [*] It refers to the preconditions of the function. It redefines Operation::precondition.
- post: C++Assertion [*] It refers to the postconditions of the function. It redefines Operation::postcondition.
- returnType: C++Classifier [0..1] It specifies the return type of the function. It redefines Operation::type. It is derived.
Appendix A

Figure 26.

- throws: C++Class [*] It refers to the types that represent exceptions that can appear during an invocation of this operation. It redefines Operation::raisedException.
- body: Implementation [0..1] It refers to the implementation of the function.

Constraints

[1] A pure virtual function does not have implementation. self.isPureVirtual implies self.body -> isEmpty()

C++ Generalization

Description
It represents a generalization in C++.

Generalizations
- Generalization (from Kernel)

Attributes
- access-specifier: AccessSpecifier It specifies what the access type of the members of the base class.
- isVirtual: Boolean It specifies whether the inheritance is virtual.
Appendix A

Associations

- general: C++Class [1] It refers to the more general class in the generalization. It redefines Generalization::general.
- specific: C++Class [1] It refers to the more specific class in the generalization. It redefines Generalization::specific.

Constraints
No additional constraints.

C++MemberFunction

It is a function that is member of a C++ class.

Generalizations

- C++Function

Attributes
No additional attributes.

Associations

- class: C++Class [1] It refers to the class to which the function belongs. It redefines Operation::class.

Constraints

[1] A function that is member of a class can not be declared extern. self.linkage <> “extern”

C++Project

Description
It represents a C++ project.

Generalizations

- Namespace (from Kernel)

Attributes
No additional attributes.
Appendix A

Associations

• c++File: C++File [1.. *] It refers to the set of C++ files that belong to the project.

Constraints
No additional constraints.

ClassifierDeclaration

Description
It represents classifier declarations.

Generalizations

• Element (from Kernel)

Attributes
No additional attributes.

Associations

• C++Classifier: C++Classifier [1] It refers to the declared classifier.

Constraints
No additional constraints.

ClassifierDefinition

Description
It denotes classifier definitions.

Generalizations

• Element (from Kernel)

Attributes
No additional attributes.

Associations

• c++Class: C++Class [1] It refers to the class that is defined.

Constraints
No additional constraints.
Appendix A

Constructor

Description
It designs a function that is used to create class instances. It cannot be called explicitly by means of
invocation expression. It does not have return type. Its name is the same of the class including it. Its
declaration is no inherited.

Generalizations

• C++MemberFunction

Attributes
No additional attributes.

Associations
No additional associations.

Constraints

[1] Constructors do not have return type. self.type -> isEmpty()
[2] The name of the constructor is the same as the class name including its declaration. self.name =
    self.class.name

Destructor

Description
A destructor is a function member with the same name as the class prefixed by a “~”. Classes have
only a function destructor that does not have arguments nor return type. Destructors are usually used
to de-allocate memory and do other cleanup for a class object and its class members when the object
is destroyed.

Generalizations

• C++MemberFunction

Attributes

• isVirtual: Boolean [1] It specifies whether the destructor is virtual, i.e., if it can be redefined in the
  subclasses.

Associations
No additional associations.
Constraints

[1] A destructor does not have arguments nor return type. self.ownedParameter -> isEmpty () and self.type -> isEmpty ()
[2] The destructor name is the same as the name of the class containing it prefixed by a ~. self.name = “~”.concat (self.class.name)

FunctionDeclaration

Description
It denotes function declarations.

Generalizations

• Element (from Kernel)

Attributes
No additional attributes.

Associations

• c++Function: C++Function [1] It denotes the function that declares.

Constraints
No additional restrictions.

FunctionDefinition

Description
It represents function definitions.

Generalizations

• Element (from Kernel)

Attributes
No additional attributes.

Associations

• c++Function: C++Function [1] It denotes the function that defines.

Constraints
No additional constraints.
Appendix A

GlobalVariableandConstantDeclaration

Description
It denotes global variables and constants that are declared in a C++ file.

Generalizations

• Element (from Kernel)

Attributes
No additional attributes.

Associations
No additional associations.

Constraints
No additional constraints.

HeaderFile

Description
It represents a C++ header file.

Generalizations

• C++File

Attributes
No additional attributes.

Associations
No additional associations.

Constraints
No additional constraints.

Implementation

Description
It specifies a procedure that carries out the function result.

Generalization

• Element (de Kernel)
Attributes

- Procedure: Expression [0..1] It refers to the procedure of the function.

Associations

- function: C++Function [1] It refers to the functions to which it belongs.
- invokedMethod: Method [*] It refers to the methods called in the body of a function.
- referencedVariable: Variable [*] It refers to the variables referred in the body of a function.
- statement: Statement [0..1] It refers to the statement block of the body of the function.

Constraints
No additional constraints.

ImplementationFile

Description
It denotes a C++ implementation file.

Generalizations

- C++File

Attributes
No additional attributes.

Associations
No additional associations.

Constraints
No additional constraints.

Include

Description
It denotes precompiler directives of type #include.

Generalizations

- PrecompilerDirective

Attributes
No additional attributes.
Appendix A

Associations

- headerFile: HeaderFile [1] It denotes the header file that is included through a directive.

Constraints
No additional constraints.

Method

Description
It declares a member function of a class that can be called by passing a fixed number of arguments.

Generalizations

- C++MemberFunction

Attributes

- isConst: Boolean [1] It specifies whether a method is constant. It redefines Operation::isQuery.
- isFinal: Boolean [1] It specifies whether a method is final, i.e., if it can not be redefined in the subclasses. It redefines RedefinableElement::isLeaf.
- isVirtual: Boolean [1] It specifies whether a method is virtual, i.e., if it can be redefined in the subclasses.
- isPureVirtual: Boolean [1] It specifies whether a method is pure, i.e., if it does not have implementation.

Associations
No additional associations.

Constraints
No additional restrictions.

PrecompilerDirective

Description
It denotes precompiler directives.

Generalizations

- Element (from Kernel)

Attributes
No additional attributes.
Appendix A

Associations
No additional associations.

Restrictions
No additional restrictions.

Statement

Description
It denotes the code block that implements a function.

Generalizations

• Action (from Action)

Attributes
No additional attributes.

Associations

• implementation: Implementation [1] It refers to the implementation of which the implementation is part.
• compoundStatement: CompoundStatement [0..1] It refers to the statement block of which this is part.

Constraints
No additional constraints.

Variable

Description
It denotes the variable that is declared in the class.

Generalizations

• Property (de Kernel)

Attributes

• isConst: Boolean [1] It specifies whether a variable is constant. If it is final, must have an initial value compulsory. It redefines RedefinableElement::isLeaf.
Appendix A

- isVolatile: Boolean [1] It specifies whether a variable is volatile, i.e., if it is not accessed synchronically.
- storage: Storage-Specifier [1] It specifies the type of the variable allocation.
- var-type: Var-Type [1] It specifies the type of the variable, i.e., if the variable is object, reference or pointer.

Associations

- class: C++Class [1] It refers to a class declaring this variable. It redefines Property::class.

Constraints

[1] A variable is a property that is part of a class and is not member of associations. self.class -> size () = 1 and self.association -> isEmpty () and self.opposite -> isEmpty ()