

The MOVA Tool: User Manual

(version 0.4.1)

Manuel Clavel Marina Egea Viviane Torres da Silva

November 27, 2007

Contents

1	Overview	1
2	UML modeling	2
2.1	The top menu	2
2.2	The diagram container	5
2.3	The editor	7
3	SecureUML modeling	9
3.1	The top menu	10
3.2	The diagram container	13
4	UML modeling with Metrics	16

Abstract

This document is the user manual for the MOVA tool (version 0.4.1), a modeling and validation experimental tool developed at the Universidad Complutense de Madrid (Spain) by the MOVA group. This document has a descriptive purpose: it assumes familiarity with the UML [5] modeling language and the OCL [4] constraint language.

1 Overview

The MOVA tool is a Java IDE for the ITP/OCL tool [3]. Events on MOVA's worksheets and toolbars are transformed into ITP/OCL's text-input commands, which are then interpreted and executed in a Maude process running the ITP/OCL tool. The MOVA tool is freely available at <http://maude.sip.ucm.es/mova>, along with this user manual and a collection of examples.

The MOVA tool consists of three applications:

- *UML modeling* [version 0.1 or higher]: it allows the user to draw UML class and object diagrams, write and check OCL invariants, write and evaluate OCL queries, and define OCL operations to be used in invariants and queries.
- *SecureUML modeling* [versions 0.4.1 or higher]: it allows the user to draw SecureUML [2] diagrams, write and evaluate OCL security policies, and define OCL operations to be used in security policies.
- *UML modeling with metrics* [versions 0.2.1 or higher]: it allows the user to draw UML class and object diagrams, write and check OCL invariants, write and evaluate OCL queries, write and evaluate OCL metrics, and define OCL operations to be used in invariants, queries, and metrics.

These applications share a common OCL editor/checker/evaluator which includes a model-based syntax-guiding facility.

At the beginning of each MOVA session, the user is requested to choose one of the MOVA applications. By default the application selected is UML modeling.

Differences with MOVA 0.3.1 This latest version makes available the SecureUML modeling application.

2 UML modeling

This application allows a user to draw UML class and object diagrams, write and check OCL invariants, and write and evaluate OCL operations and queries.

The main window is divided in two parts: the *top menu* and the *diagram container*. The top menu includes the options of creating diagrams from scratch, saving diagrams in files, opening diagrams previously saved, and printing diagrams. It also includes the option of inserting elements in diagrams, which are given a default name.

The diagram container holds the class and object diagrams drawn by the user; object diagrams of the same class diagram are kept together. Class and object diagrams are drawn in diagram pallets which are stack in a pile. If the container is not empty, the *working* diagram is the diagram drawn in the visible pallet, i.e., the pallet at the top of the stack. Class diagrams are *closed* in the diagram container when they hold object diagrams. Closed class diagrams can not be modified.

2.1 The top menu

The top menu options are distributed in submenus: File, Edit, View, Insert, and Help. The top menu also contains icons that provide quick access to particular menu options.

The File submenu includes the following options:

- **File|New|Class Diagram.** It creates a new class diagram pallet in the diagram container, which becomes the working diagram.
- **File|New|Object Diagram.** It creates a new object diagram pallet in the diagram container, which becomes the working diagram.
- **File|Open|Class Diagram.** It creates a new class diagram pallet in the diagram container, and draw in it the class diagram contained in the file selected by the user. This file must contain the class diagram in (MOVA) XML format. The new class diagram pallet becomes the working diagram.
- **File|Open|Object Diagram.** It creates a new object diagram pallet in the diagram container, and draw in it the object diagram contained in the file selected by the user. This file must contain the object diagram in (MOVA) XML format. The new object diagram pallet becomes the working diagram.
- **File|Load|Invariants, operations.** It adds to the working class diagram the invariants and operations contained in the file selected by the user. This file must contain the invariants in (MOVA) text format.
- **File|Load|Operations.** It adds to the working class diagram the operations contained in the file selected by the user. This file must contain the operations in (MOVA) text format.
- **File|Close.** It closes the working diagram. If the working diagram is a class diagram it also closes all its object diagrams.
- **File|Page setup.** It allows the user to change the page configuration (media, orientation, and margins) of the working diagram for printing purposes.
- **File|Print.** It allows the user to print the working diagram, possible changing its current page configuration.
- **File|Export to EPS.** It saves the working diagram in EPS format in the file selected by the user.
- **File|Save as|Diagram.** It saves the working diagram in (MOVA) XML format in the file selected by the user.
- **File|Save as|Operations, Invariants.** It saves the operations and invariants inserted in the working class diagram in (MOVA) text format in the file selected by the user.
- **File|Exit.** It close the application.

The Edit submenu includes the following options:

- **Edit|Zoom +.** It zooms-in in the working diagram.
- **Edit|Zoom -.** It zooms-out in the working diagram.

The View submenu includes the following options:

- View|Invariants. It allows the user to view the invariants added to the working class diagram, and select those to be checked over the object diagrams linked to the working class diagram.
- View|Class diagram. It allows the user to hide/show the roles, multiplicities, and names of the associations, and the attributes of the classes, depicted in the working class diagram.
- View|Object diagram. It allows the user to hide/show the roles and names of the links, and the attributes of the objects, depicted in the working object diagram.

The Insert submenu includes the following options:

- Insert|Class diagram|Class. It inserts a class in the working class diagram. This element is drawn as a rectangle in the location where the user clicked upon. The class is assigned a default name.
- Insert|Class diagram|Enum. It inserts an enumeration class in the working class diagram. This element is drawn as a rectangle in the location where the user clicked upon. The enumeration class is assigned a default name.
- Insert|Class diagram|Association. It inserts a binary association in the working class diagram. This element is drawn as an association-arrow between the two classes upon which the user clicks at the beginning and the end of a sequence of clickings. If the size of the sequence of clickings is greater than two, the association-arrow is drawn in segments, and the intermediate clicks determine the locations of the end-points of each of the segments. The association is assigned a default name, default roles, and default multiplicities.
- Insert|Class diagram|Generalization. It inserts a generalization in the working class diagram. This element is drawn as a generalization-arrow between the two classes upon which the user clicks at the beginning and the end of a sequence of clickings. The class clicked at the end of the sequence is the super-class. If the size of the sequence of clickings is greater than two, the generalization-arrow is drawn in segments, and the intermediate clicks determine the locations of the end-points of each of the segments.
- Insert|Class diagram|Invariant. It allows the user to add an invariant to the working class diagram. The context (if any) of the invariant is determined by the user, by clicking upon the working diagram. If the user clicks upon a class, the invariant is written in the context of this class; otherwise, the invariant is written without a context. Invariants are written using the MOVA editor as described in Section 2.3.

- **Insert|Class diagram|Operation.** It allows the user to add an operation to the working class diagram. The context (if any) of the operation is determined by the user, by clicking upon the working diagram. If the user clicks upon a class, the operation is written in the context of this class; otherwise, the operation is written without a context. The name, arguments, and resulting type of the operation are introduced by the user in the operation window; the body of the operation is written using the MOVA editor as described in Section 2.3.
- **Insert|Object diagram|Object.** It inserts an object (of the class chosen by the user) in the working object diagram. This element is drawn as a rectangle in the location where the user clicked upon. The object is assigned a default name and its attributes are assigned a default value.
- **Insert|Object diagram|Link.** It inserts a link (of the association chosen by the user) in the working object diagram. This element is drawn as a link-arrow between the two objects upon which the user clicks at the beginning and the end of a sequence of clickings. If the size of the sequence of clickings is greater than two, the link-arrow is drawn in segments, and the intermediate clicks determine the locations of the end-points of each of the segments.
- **Insert|Object diagram|Query.** It allows the user to execute a query over the working object diagram. Queries are written using the MOVA editor as described in Section 2.3.

The Help submenu includes the following options:

- **Help|About us.** It provides information about the MOVA project.

The top menu contains icons that provide quick access to particular menu options. Table 1 shows the menu option associated with each icon in the top menu.

2.2 The diagram container

The diagram container holds the diagram pallets. Diagrams pallets have a notched tab along their top. The notched tabs of the class diagrams are visible, as well as those of the object diagrams of the working class diagram. The user can change the working diagram by clicking upon its notched tab.

The icons in the left-margin of the diagram container provide quick access to particular menu options. The icons shown depend on the working diagram: a different set is shown from class diagrams than from object diagrams. Tables 2 and 3 show the menu option associated with each icon in the left-margin, except the pointer-icon which is used to select elements in the working diagram. Selected elements can be edited or removed by clicking upon them with the right-button of the mouse and choosing the appropriate option in the selection menu: **Properties** for editing, and **Remove** for removing. Elements can also be edited by double-clicking upon them.











	File New Class diagram
	File New Object diagram
	File Open Class diagram
	File Open Object diagram
	File Load Invariants, operations
	File Close
	File Save as Diagram
	File Save as Invariants, operations
	File Exit
	Help About us

Table 1: The UML modeling top menu icons.





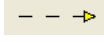



	Cursor
	Insert Class diagram Class
	Insert Class diagram Enum
	Insert Class diagram Association
	Insert Class diagram Association Class
	Insert Class diagram Generalization
	Insert Class diagram Invariant
	Insert Class diagram Operation

Table 2: The UML modeling class diagram container icons.

- *Classes*: The user can modify the name of selected class, and insert or remove its attributes. Attributes must have a predefined type (*Integer*, *Boolean*, and *String*) or the type introduce by an enumeration class.
- *Associations*: The user can modify the name of selected association, its roles and their multiplicities.
- *Objects*: The user can modify the name of a selected object and the value of its attributes.

2.3 The editor

The editor assists the user in writing OCL expressions in three different contexts: adding an invariant to a class diagram; writing a query about an object diagram; and defining the body of an operation.

To write an expression the user selects *patterns* from lists that are built at “run-time” when the buttons *Start*, *Dot*, *Arrow*, or *Space* are pressed. The actual patterns shown in the lists depend on the model under consideration, the current type of the expression, and the button that has been pressed: the *Start* button is used to start writing an expression; the *Dot* button is used to access a property of a class; the *Arrow* button is used to access a property of a collection; and the *Space* button is used to introduce a logical or an arithmetic operator. The current type of an expression is shown in the *Current Type* subwindow.










	Insert Object diagram Object
	Insert Object diagram Link
	Edit Zoom +
	Edit Zoom -
	View Invariants
	View Class diagram/Object diagram
	It opens the metrics editor
	It checks the invariant selected by the user.
	Insert Object diagram Query

Table 3: The UML modeling object diagram container icons.

Patterns can contain *holes*, which are *types* enclosed in angle brackets possibly followed by an index. For example, the *exists*-pattern contains the hole $\langle \text{Boolean} \rangle$. When a pattern with holes is selected the user has to fill its holes with expressions of the appropriate type. Each of these expressions is written in a new editor window, that is a *child* of the window in which the pattern has been selected. The type of the hole to be filled in a child window is shown in its **Target Type** subwindow, and the context in which this hole appears is shown in its **Context** subwindow.

The type of the hole to be filled in a child window is shown in its **Target Type** subwindow, and the context in which this hole appears is shown in its **Context** subwindow. A hole with type **Any** can be filled with expressions of any type. When the user presses the button **OK** in a child window, the expression written in its **Expression** subwindow replaces the corresponding hole in its **Context** subwindow; if there are no more holes to be filled, the expression in the **Context** subwindow is appended to the right of the expression written in the **Expression** subwindow of its parent.

There are two special patterns for writing set of elements:

- $\text{Set}\{\}$ denotes the empty set of elements of the type selected by the user; and
- $\text{Set}\{\langle \text{type} \rangle\} \rightarrow \text{union}(\langle \text{Set}[\langle \text{type} \rangle] \rangle)$ denotes the union of two sets of elements of the type selected by the user: the first contains just one element and the second can contain arbitrary number of elements.

There is also a special pattern for writing if-then-else statement whose then- and else- expressions are of the type selected by the user:

- $\text{if}(\langle \text{Boolean} \rangle) \text{then}(\langle \langle \text{type} \rangle : 1 \rangle) \text{else}(\langle \langle \text{type} \rangle : 2 \rangle) \text{fi}$.

When the editor is used for adding an invariant to a class diagram, the type shown in the **Target Type** subwindow of the initial window is **Boolean**. When the editor is used for writing a query about an object diagram, the type shown in the **Target Type** subwindow of the initial window is **Any**. Finally, when the editor is used for writing the body of an operation, the type shown in the **Target Type** subwindow of the initial window is its resulting type.

3 SecureUML modeling

This application supports the automated analysis of security-design models [2], following the metamodel-based approach proposed [1]. In particular, it allows a user to draw security-design models and scenarios, and to write and evaluate OCL queries on these models to automatically analyse their security policies.

The main window is divided in two parts: the *top menu* (Section 3.1) and the *diagram container* (Section 3.2). The top menu includes the options of creating diagrams from scratch, saving diagrams in files, opening diagrams previously

saved, and printing diagrams. It also includes the option of inserting elements in diagrams.

The diagram container holds the security-design diagrams and scenarios drawn by the user; scenarios of the same security-design diagram are kept together. Security-design diagrams and scenarios are drawn in diagram pallets which are stack in a pile. If the container is not empty, the *working* diagram is the diagram drawn in the visible pallet, i.e., the pallet at the top of the stack. Security-design diagrams are *closed* in the diagram container when they hold scenarios. Closed security-design diagrams cannot be modified.

This application shares with the other MOVA applications the OCL editor/checker/evaluator (see Section 2.3), which can be used in this case to write authorization constraints, and to write and execute queries on security-design and scenarios diagrams. Notice that the current version of the tool includes as predefined OCL operations. all the analysis operations defined [1].

3.1 The top menu

The top menu options are distributed in submenus: File, Edit, Insert, and Help. The top menu also contains icons that provide quick access to particular menu options.

The File submenu includes the following options:

- File|New|Security Diagram. It creates a new security-design diagram pallet in the diagram container, which becomes the working diagram.
- File|New|Object Security Diagram. It creates a new scenario diagram pallet in the diagram container, which becomes the working diagram.
- File|Open|Security Diagram. It creates a new security-design diagram pallet in the diagram container, and draw in it the security-design diagram contained in the file selected by the user. This file must contain the security diagram in (MOVA) XML format. The new security-design diagram pallet becomes the working diagram.
- File|Open|Object Security Diagram. It creates a new scenario diagram pallet in the diagram container, and draw in it the scenario diagram contained in the file selected by the user. This file must contain the scenario diagram in (MOVA) XML format. The new object security diagram pallet becomes the working diagram.
- File|Close. It closes the working diagram. If the working diagram is a security-design diagram it also closes all its scenario diagrams.
- File|Page setup. It allows the user to change the page configuration (media, orientation, and margins) of the working diagram for printing purposes.
- File|Print. It allows the user to print the working diagram, possible changing its current page configuration.

- File|Export to EPS. It saves the working diagram in EPS format in the file selected by the user.
- File|Save as|Diagram. It saves the working diagram in (MOVA) XML format in the file selected by the user.
- File|Exit. It closes the application.

The Edit submenu includes the following options:

- Edit|Zoom +. It zooms-in in the working diagram.
- Edit|Zoom -. It zooms-out in the working diagram.

The Insert submenu includes the following options:

- Insert|Security diagram|Role. It inserts a role in the working security-design diagram. This element is drawn as a rectangle labeled $\langle\langle Role \rangle\rangle$ in the location where the user clicked upon. The tool asks the user to insert a name for the new role.
- Insert|Security diagram|Entity. It inserts an entity in the working security-design diagram. This element is drawn as a rectangle labeled $\langle\langle Entity \rangle\rangle$ in the location where the user clicked upon. The tool asks the user to insert a name for the new entity.
- Insert|Security diagram|Permission. It inserts a permission in the working security-design diagram. This element is drawn as a rectangle labeled $\langle\langle Permission \rangle\rangle$ in the location where the user clicked upon. The tool asks the user to insert a name for the new permission.
- Insert|Security diagram|Role hierarchy. It inserts a hierarchy between two roles in the working security-design diagram. This element is drawn as a generalization-arrow between the two roles upon which the user clicks at the beginning and the end of a sequence of clickings. The role clicked at the end of the sequence is the super-role. If the size of the sequence of clickings is greater than two, the generalization-arrow is drawn in segments, and the intermediate clicks determine the locations of the end-points of each of the segments.
- Insert|Security diagram|Association. It inserts an association relation between two entities in the working security diagram. This element is drawn as a bidirectional arrow between the two entities upon which the user clicks at the beginning and the end of a sequence of clickings. If the size of the sequence of clickings is greater than two, the arrow is drawn in segments, and the intermediate clicks determine the locations of the end-points of each of the segments. At the end of the sequence of clickings, a window is opened up and the user is asked for inserting a name for the association, and a name for the origin and target association-ends along with the multiplicity of each of these ends.

- **Insert|Security diagram|Permission assignment.** It inserts an assignment relation among a role, a permission, and an entity in the working security-design diagram. This element is drawn with a solid line between the role and the entity, and a dashed line between the permission and this solid line. The role, the permission, and the entity are those upon which the user clicks at the beginning, in the middle, and the end of a sequence of clickings.
- **Insert|Object security diagram|Entity instance.** It inserts an instance of the entity chosen by the user in the working scenario diagram. This element is drawn as a rectangle in the location where the user clicked upon. The entity instance is assigned a default name and its attributes are assigned a default value. It is also shown whether the instance can be a *user* in the working scenario diagram (and therefore can be assigned a role) which depends on whether the entity has been declared or not as a *user-provider* entity (more on Section 3.2).
- **Insert|Object security diagram|Link.** It inserts a link of the association chosen by the user in the working scenario diagram. This element is drawn as a link-arrow between the two entity instances upon which the user clicks at the beginning and the end of a sequence of clickings. If the size of the sequence of clickings is greater than two, the link-arrow is drawn in segments, and the intermediate clicks determine the locations of the end-points of each of the segments.
- **Insert|Object security diagram|Role.** It inserts a role in the working scenario diagram. The role has to be chosen from the collection of roles defined in the security-design diagram which had not been yet introduced in the scenario. This element is drawn like in the security-design diagram, and it is placed in the location where the user click upon.
- **Insert|Object security diagram|Role assignment.** It inserts a relation between a user, that is, an instance of a user-provider entity, and a role in the working scenario diagram. This element is drawn as a bidirectional arrow between the user and the role upon which the user clicks at the beginning and the end of a sequence of clickings. If the size of the sequence of clickings is greater than two, the bidirectional arrow is drawn in segments, and the intermediate clicks determine the locations of the end-points of each of the segments.

The **Help** submenu includes the following options:

- **Help|About us.** It provides information about the MOVA project.

The top menu contain also icons that provide quick access to particular menu options. Table 4 shows the menu option associated with each icon in the top menu.












	File New Security diagram
	File New Object security diagram
	File Open Security diagram
	File Open Object security diagram
	File Close
	File Save as Diagram
	File Exit
	Help About us

Table 4: The SecureUML modeling top menu icons.



3.2 The diagram container

The diagram container holds the diagram pallets. Diagrams pallets have a notched tab along their top. The notched tabs of the security-design diagrams are visible, as well as those of the scenario diagrams of the working security-design diagram. The user can change the working diagram by clicking upon its notched tab.

The icons in the left-margin of the diagram container provide quick access to particular menu options and also to additional functionalities of this application. The icons shown depend on the working diagram: a different set is shown from security-design diagrams than from scenario diagrams. When the working diagram is a security-design diagram, the icons in the left-margin which are linked to menu options are shown in Table 5; the rest of the icons are described below.

	It is used to select an element in the working security-design diagram.
	It is used to associate an authorization constraint to a particular permission in the working security-design diagram. The authorization constraint is linked to the permission's dash-arrow which the user clicks upon.
	It is used to execute a query on the working security-design diagram.

When the working diagram is a scenario diagram, the icons in the left-margin which are linked to particular menu options are shown in Table 6; the rest of the icons are described below.

	It is used to select an element in the working scenario diagram.
	It is used to execute a query on the working scenario diagram.

Finally, elements can be edited or removed by double clicking upon them. In particular,

- *Entities*: By double clicking upon an entity, the user can declare it as a *user-provider* entity, modify its name, and insert or remove its attributes and methods (both query and non-query). He/she can also delete the entity.
- *Association*: By double clicking upon an association, the user can modify its name, and the names and multiplicities of its association-ends. He/she can also delete the association.
- *Role*: By double clicking upon a role, the user can modify its name and delete it.
- *Permission*: By double clicking upon a permission, the user can modify its name and delete it.
- *Role hierarchy*: By double clicking upon a role hierarchy, the user can delete it.
- *Action assignment*: By double clicking upon a permission's dash-line, the user can assign or remove (atomic or composite) actions on the associated entity to the permission.




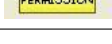




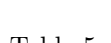
	Cursor
	Insert Security diagram Role
	Insert Security diagram Entity
	Insert Security diagram Permission
	Insert Security diagram User
	Insert Security diagram Role hierarchy
	Insert Security diagram Association
	Insert Security diagram Permission assignment
	Insert Security diagram Query

Table 5: The SecureUML modeling diagram container icons.



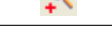


	Insert Object security diagram Entity instance
	Insert Object security diagram Link
	Edit Zoom +
	Edit Zoom -
	Insert Object security diagram Query

Table 6: The SecureUML modeling diagram container icons.

4 UML modeling with Metrics

UML Modeling with Metrics adds to *UML modeling* the possibility of evaluating metrics over the working class diagram. Both applications have similar interfaces, the only difference being that the Insert|Class Diagram|Metric option is active in *UML Modeling with Metrics*.

In MOVA, metrics are OCL queries over the instance of the MOVA metamodel corresponding to the working class diagram. The editor can assist the user in writing these metrics: in this case the patterns offered by the editor, and the result of its evaluation, correspond to:

- the classes, attributes, roles, generalizations, and associations included in the MOVA metamodel.
- the objects and links included in the instance of the MOVA metamodel that corresponds to the working class diagram; this instance is automatically generated by the application.

To ease the task of writing queries, the editor offers additional patterns that correspond to metrics/queries which belong to the MOVA metrics library. The complete list of metrics/queries included in this library is shown below: Each metric/query is first informally explained and then formally defined/implemented as an OCL query over the MOVA/UML metamodel

Class Metrics/Meta Queries
<i>The number of proper attributes of the class c.</i>
numAttrClass(c:MovaClass)::Integer
c.ownedAttribute→size()
<i>The set of the immediate sucesors of the class c.</i>
childClass(c:MovaClass):Set[Classifier]
c.nonNavigable→collect(g:Generalization g.specific)
<i>The number of the immediate sucesors of the class c.</i>
numChildClass(c:MovaClass)::Integer
childClass(c)→size()
<i>The number of proper connectors of the class c.</i>
numConnClass(c:MovaClass)::Integer
c.inAssociation→size() + c.generalization→size() + c.nonNavigable→size()
<i>The set of the immediate ancestors of the class c.</i>
parentClass(c:MovaClass)::Set[MovaClass]
c.generalization→collect(g g.general.oclAsType(MovaClass))
<i>The set of all the ancestors of the class c.</i>
parentClassAll(c:MovaClass)::Set[MovaClass]
parentClass(c)→union(parentClass(c)→collect(p parentClassAll(p)))

Class Metrics/Meta Queries
<i>The set of the immediate inherited attributes of the class c.</i>
<code>inherAttrClass(c:MovaClass)::Set[Property]</code>
<code>parentClass(c)→collect(c1 c1.ownedAttribute)</code>
<i>The set of all inherited attributes of the class c.</i>
<code>inherAttrClassAll(c:MovaClass)::Set[Property]</code>
<code>parentClassAll(c)→collect(c1 c1.ownedAttribute)</code>
<i>The number of all inherited attributes of the class c.</i>
<code>numInherAttrClassAll(c:MovaClass)::Integer</code>
<code>inherAttrClassAll(c)→size</code>
<i>The set of all (proper and inherited) attributes of the class c.</i>
<code>attrClassAll(c:MovaClass)::Set[Property]</code>
<code>inherAttrClassAll(c)→union(c.ownedAttribute)</code>
<i>The set of data types of all the (proper and inherited) attributes of the class c.</i>
<code>typeAttrClassAll(c:MovaClass)::Set[Type]</code>
<code>attrClassAll(c)→collect(p p.dataType)</code>
<i>The value of checking whether the class c has an attribute of end type t.</i>
<code>hasAttrClass(c:MovaClass, t:Type)::Boolean</code>
<code>typeAttrClassAll(c)→includes(t)</code>

Class Metrics/Meta Queries
<i>The set of the immediate inherited associations of the class c.</i>
<code>inherAssocClass(c:MovaClass)::Set[Association]</code>
<code>parentClass(c)→collect(c1 c1.inAssociation)</code>
<code>inherAssocClassAll(c:MovaClass)::Set[Association]</code>
<i>The set of all inherited associations of the class c.</i>
<code>parentClassAll(c)→collect(c1 c1.inAssociation)</code>
<i>The number of all inherited associations of the class c.</i>
<code>numInherAssocClassAll(c:MovaClass)::Integer</code>
<code>inherAssocClassAll(c)→size</code>
<i>The set of all (proper and inherited) associations of the class c.</i>
<code>assocClassAll(c:MovaClass)::Set[Association]</code>
<code>inherAssocClassAll(c)→union(c.inAssociation)</code>
<i>The set of data types of all the (proper and inherited) attributes of the class c.</i>
<code>typeAssocClassAll(c:MovaClass)::Set[Type]</code>
<code>assocClassAll(c)→collect(p p.endType)</code>
<i>The value of checking whether the class c has an association with end type t.</i>
<code>hasAssocClass(c:MovaClass, t:Type)::Boolean</code>
<code>typeAssocClassAll(c)→includes(t)</code>

Class Metrics/Meta Queries
avgCoupledTo(c1:MovaClass, c2:MovaClass)::Boolean
<i>The value of checking whether the class c1 is coupled with the class c2.</i>
hasAttrClass(c1, c2) or hasAssocClass(c1, c2)
<i>The set of classes that are coupled with the class c.</i>
avgCouplings(c:MovaClass)::Set[MovaClass]
MovaClass.allInstances→excluding(c) →select(c1 avgCoupledTo(c1,c) or avgCoupledTo(c,c1))
<i>The number of classes that are coupled with the class c.</i>
numAvgCouplings(c:MovaClass)::Integer
avgCouplings(c)→size()

Diagram Metrics
<i>The number of classes in the diagram.</i>
numClass::Integer
MovaClass.allInstances→size()
<i>The number of associations in the diagram.</i>
numAssoc::Integer
Association.allInstances→size()
<i>The number of generalizations in the diagram.</i>
numGenr::Integer
Generalization.allInstances→size()
<i>The number of classifiers in the diagram.</i>
numClasf::Integer
Classifier.allInstances→size()
<i>The number of connectors in the diagram.</i>
numConn::Integer
Relationship.allInstances→size()
<i>The number of attributes in the diagram.</i>
numAttr::Integer
(MovaClass.allInstances→collect(c attrClassAll(c)))→asSet()→size()
<i>The number of all inherited attributes in the diagram.</i>
numInherAttr::Integer
(MovaClass.allInstances→collect(c inherAttrClassAll(c)))→asSet()→size()

References

- [1] David Basin, Manuel Clavel, Jürgen Doser, and Marina Egea. Automated analysis of security-design models. Submitted for publication. <http://maude.sip.ucm.es/~clavel/pubs>.

- [2] David A. Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, 2006.
- [3] Manuel Clavel and Marina Egea. ITP/OCL: A rewriting-based validation tool for UML+OCL static class diagrams. In Michael Johnson and Varmo Vene, editors, *AMAST*, volume 4019 of *Lecture Notes in Computer Science*, pages 368–373. Springer, 2006.
- [4] Object Management Group. Object Constraint Language specification, 2004. <http://www.omg.org>.
- [5] Object Management Group. Unified Modeling Language specification, 2004. <http://www.uml.org>.