The MOVA Tool: User Manual (version 0.2)

Manuel Clavel Marina Egea Viviane Torres da Silva

February 12, 2007

Contents

Abstract

This document is the user manual for the MOVA tool (version 0.2), 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 [2] modeling language and the OCL [2] constraint language.

1 Overview

The MOVA tool consists of three application:

- *UML modeling*: 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*: it allows the user to draw SecureUML [1] diagrams, write and evaluate OCL security policies, and define OCL operations to be used in security policies.
- *UML modeling with metrics*: 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.

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

Differences with MOVA 0.1 The latest version integrates an OCL editor to assist the user in writing invariants, queries, and operations. The editor is described in Section 2.3.

2 MOVA 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

C	File|New|Class diagram
O	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
M	Help|About us

Table 1: The UML modeling top menu icons.

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. Table 2 shows 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-bottom 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.

- *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. The editor main window is shown in Figure 1.

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

Patterns can contain *holes*, which are *types* enclosed in angle brackets possibly followed by an index. 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 have been selected. Holes with types integer, boolean, string, and type are special: they are filled in 'ad hoc' windows, writing an integer number, a boolean value, a string, or a class or collection type, respectively. 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

CL	Insert|Class diagram|Class
ENUM	Insert|Class diagram|Enum
↔	Insert|Class diagram|Association
- - ▷	Insert|Class diagram|Association Class
—▷	Insert|Class diagram|Generalization
I	Insert|Class diagram|Invariant
O	Insert|Class diagram | Operation
OBJ	Insert|Object diagram|Object
LINK	Insert|Object diagram|Link
🔍+	Edit|Zoom +
🔍-	Edit|Zoom -
I	View|Invariants
☰	View|Class diagram/Object diagram
M	*This option not available in version 0.2.*
✓	It checks the invariant selected by the user.
Q	Insert|Object diagram|Query

Table 2: The UML modeling diagram container icons.

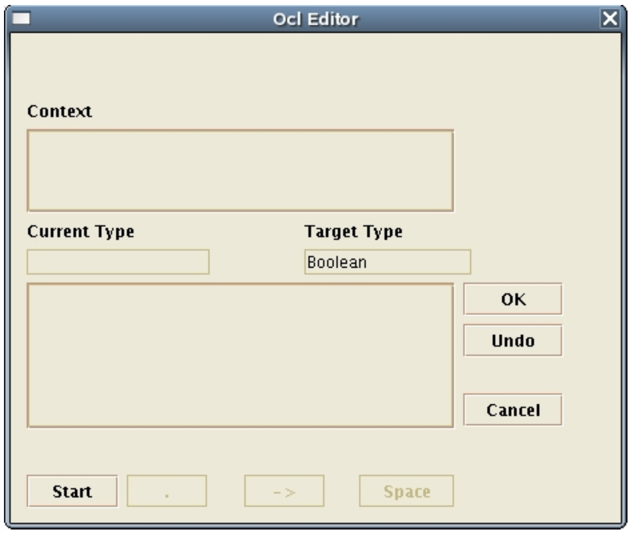

Figure 1: The editor window.

user presses the bottom 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:

- Set{} denotes the empty set of elements of the type selected by the user; and
- Set{⟨⟨type⟩⟩}→union(⟨Set[⟨type⟩]⟩) 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 statements whose conditional parts are of the type selected by the user:

- if(⟨Boolean⟩)then(⟨⟨type⟩:1⟩)else(⟨⟨type⟩:2⟩)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.

References

[1] 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.

[2] Object Management Group. Unified Modeling Language specification, 2004. `http://www.uml.org`.