# MQ-2 User Guide

Vlad Acretoaie
Department of Informatics and Applied Mathematics
Technical University of Denmark
rvac@dtu.dk

22.03.2013

## 1 Introduction

This document describes the usage of the MQ-2 plug-in for MagicDraw. The plug-in implements the Visual Model Query Language (VMQL), a novel query by-example approach for UML (and other) models. For a detailed description of VMQL see [1], and for an overview of the plug-in's implementation see [2].

## 2 Using the MQ-2 Prolog console

The Prolog console can be opened by selecting the *MQ-2* entry from the *Tools* menu in the MagicDraw menu bar. This action will cause the MQ-2 Prolog Console window to appear at the bottom of the MagicDraw application window (see Figure 1). The console may only be opened if a model is already open in MagicDraw. The console's initial behavior is that of a regular Prolog console: it allows executing queries supported by SWI-Prolog, and prints query results and error messages. The following keys must be used to execute queries in the MQ-2 Prolog console:

- `Return`: Prints the next query result. The behavior of the `Return` key may be modified using the Quick Actions drop-down on the Console Tool Bar (see Section 2.2).
- `Up`: Displays the previous query.
- `Down` Displays the next query.

Figure 1: MagicDraw window featuring the MQ-2 Prolog console. Model elements returned by the last console query are highlighted in green.

The MQ-2 Prolog console offers a number of features aimed specifically at the task of model querying. These features are accessible through the Console Tool Bar, placed above the console text area, and are detailed in what follows.

## 2.1 Consulting models

Before the model query facilities provided by the MQ-2 Prolog console can be used, a model must be consulted. The Console Tool Bar features the following buttons enabling model consulting:

- `Consult the currently active model:` Consults the model currently open in MagicDraw so that it is available for querying.
- `Select a MagicDraw model to consult`: Opens a file browser allowing users to select a MagicDraw project file to consult so that it is available for querying. Note that only projects stored in the MDXML file format may be selected.
- `Re-consult the MagicDraw model`: Re-consults the last consulted model, regardless if it is open in MagicDraw or not.
- `Re-start the Prolog console`: Clears the console contents and un-consults all previously consulted modules.

The `Explicit header` check-box allows users to select whether or not the `me/2` predicate should be included in the header of the Prolog module generated when a model is consulted through one of the methods above. In case it is checked, the `me/2` predicate will be included in the generated module header, and model elements may be accessed from the console using predicates of the form `me(Type,Id-Attributes)`. In case it is left un-checked, the `me/2` predicate will be omitted from the generated module header, and model elements may be accessed from the console using predicates of the form `'ModelName':me(Type,Id-Attributes)`, where 'ModelName' is the name of the consulted MagicDraw model. By default, the `Explicit header` check-box is un-checked.

**Warning:** *When using MQ-2 with versions of SWI-Prolog older than 6.1.9, consulting two Prolog modules that expose the same predicate in their headers causes MagicDraw to crash. For this reason, it is recommended to leave the* `Explicit header` *check-box un-checked.*

## 2.2 Querying models

Once a model has been consulted, it is possible to execute queries on it. The most direct way to query a consulted model is through the `me(Type-Id,Attrs)` predicate, where `Type` is the meta-type of a model element, `Id` is its unique generated identifier, and `Attrs` is a list of the element's meta-attributes and their values. For instance, a query retrieving a class named Reader has the following form:

```
me(Type-Id,Attrs), member(name-'Reader',Attrs).
```

Additional MQ-2 library predicates that can be used to query model elements are presented in Section 2.3. While all queries can be executed by simply typing them into the console and pressing the `Return` key, the Console Tool Bar features the Quick Actions drop-down that can be used to add custom behavior to the `Return` key. The Quick Actions drop-down contains the following entries:

- `Print next solution:` When this option is selected, pressing the `Return` key causes the next query solution to be printed on the console. If no other solutions exist, a new prompt is printed. This is the standard Prolog console behavior.
- `Print all solutions:` When this option is selected, pressing the `Return` key causes all query solutions to be printed, followed by a new prompt.
- `Show all solutions in tree:` When this option is selected, pressing the `Return` key causes all query solutions to be printed and all model elements included in the query solution to be shown in MagicDraw's Search Results Tree. A new prompt is printed on the console.
- `Highlight all solutions:` When this option is selected, pressing the `Return` key causes all query solutions to be printed and all model elements included in the query solution to be highlighted in the diagrams in which they appear. The highlight color may be selected using the `Select highlight color` button on the Console Tool Bar. A new prompt is printed on the console.
- `Show selection in tree:` When this option is selected, pressing the `Return` key causes all model elements which can be identified by the selected console text to be displayed in MagicDraw's Search Results Tree.

Alternatively, all actions included in the Quick Actions drop-down can be executed through corresponding buttons on the Console Tool Bar. The buttons do not alter the behavior of the `Return` key, but rather replace its role. The Console Tool Bar contains two additional buttons addressing Prolog query execution: the `Clear highlights` button, which clears all highlights from all diagrams (including highlights generated by VMQL queries, as discussed in Section 3.2), and the `Abort query` button, which stops the execution of the current query.

## 2.3 Library predicates

Querying models using the integrated Prolog console is facilitated by the pre-consulted MQ-2 library predicates:

- `get_me(Attr-Val,Type-Id,Attrs)`: Returns the attribute values of all model elements of type `Type` having the value `Val` for the attribute `Attr`. Example usage (finding the attributes of the class named 'Reader'):

  get_me(name-'Reader', class-Id, Attrs).

- `part_of(Kind,SuperId,SubId)`: Returns the ID of a model element representing a part of type `Kind` of the model element with ID `SuperID` in the variable `SubId`. Example usage (finding all owned ends of the model element with ID 1):

  part_of(ownedEnd, 1, SubId).

- `highlight(Elements,Color)`: Highlights the model elements identified by the `Elements` parameter in the specified `Color`. The `Elements` variable can either contain a list of `me/2` predicates, a list of model element IDs, or a list of model element names. Example usage (highlighting the class named 'Reader' in green):

  get_me(name-'Reader', class-Id, Attrs), highlight(Id, green).

In addition to the MQ-2 library predicates, users can consult their own custom defined library predicates at run time by pressing the `Consult user defined Prolog modules` button on the console tool bar. Files containing user-defined predicates must be placed in the `<MagicDraw home>/plugins/mq2/user/` directory prior to being consulted. Files containing helper predicates required by the user defined library predicates must be placed separately in the `<MagicDraw home>/plugins/mq2/user/helpers/` directory, so that they are not directly consulted in the MQ-2 Prolog console.

## 2.4 Limitations

The MQ-2 Prolog console does not support executing queries in debug mode. Calling the `debug/0` predicate must be avoided, as it will cause MagicDraw to crash. All errors that cause Prolog to enter debug mode will also cause MagicDraw to crash. This behavior occurs due to the fact that MagicDraw interprets the Prolog debug mode as a Java Virtual Machine crash.

# 3 Executing VMQL queries

Besides providing support for executing Prolog queries on models, the MQ-2 Prolog console also supports executing VMQL queries. The VMQL query execution interface, shown in Figure 2 on the bottom right corner of the MagicDraw main window, can be activated or de-activated from the `VMQL` toggle button on the Console Tool Bar.

## 3.1 Query execution

Executing a VMQL query requires a source model and a query model to be selected. The MQ-2 VMQL query execution interface assumes that the source model is the currently open MagicDraw model. Therefore, the first step in executing a VMQL query is consulting this model in the MQ-2 Prolog console, as described in Section 2.1. Selecting a query model and executing it as a VMQL query against the source model is facilitated by the following buttons on the VMQL query execution interface:

- `Select a MagicDraw project to be used as query model:` Opens a file browser allowing users to select a MagicDraw project file to be used as a VMQL query model. Note that only projects stored in the MDXML file format may be selected.
- `Re-consult the current query model:` Re-consults the last selected VMQL query model.
- `Execute the selected VMQL query:` Triggers the execution of the VMQL matching algorithm between the selected source and query models.

Query execution results are displayed in a tabular format bellow these buttons in the Bindings Table. Each binding is displayed as a row in the Bindings Table, while the first column of each row identifies the index of the binding. In case the query model includes VMQL variables, subsequent columns correspond to the values taken by these variables in each binding.

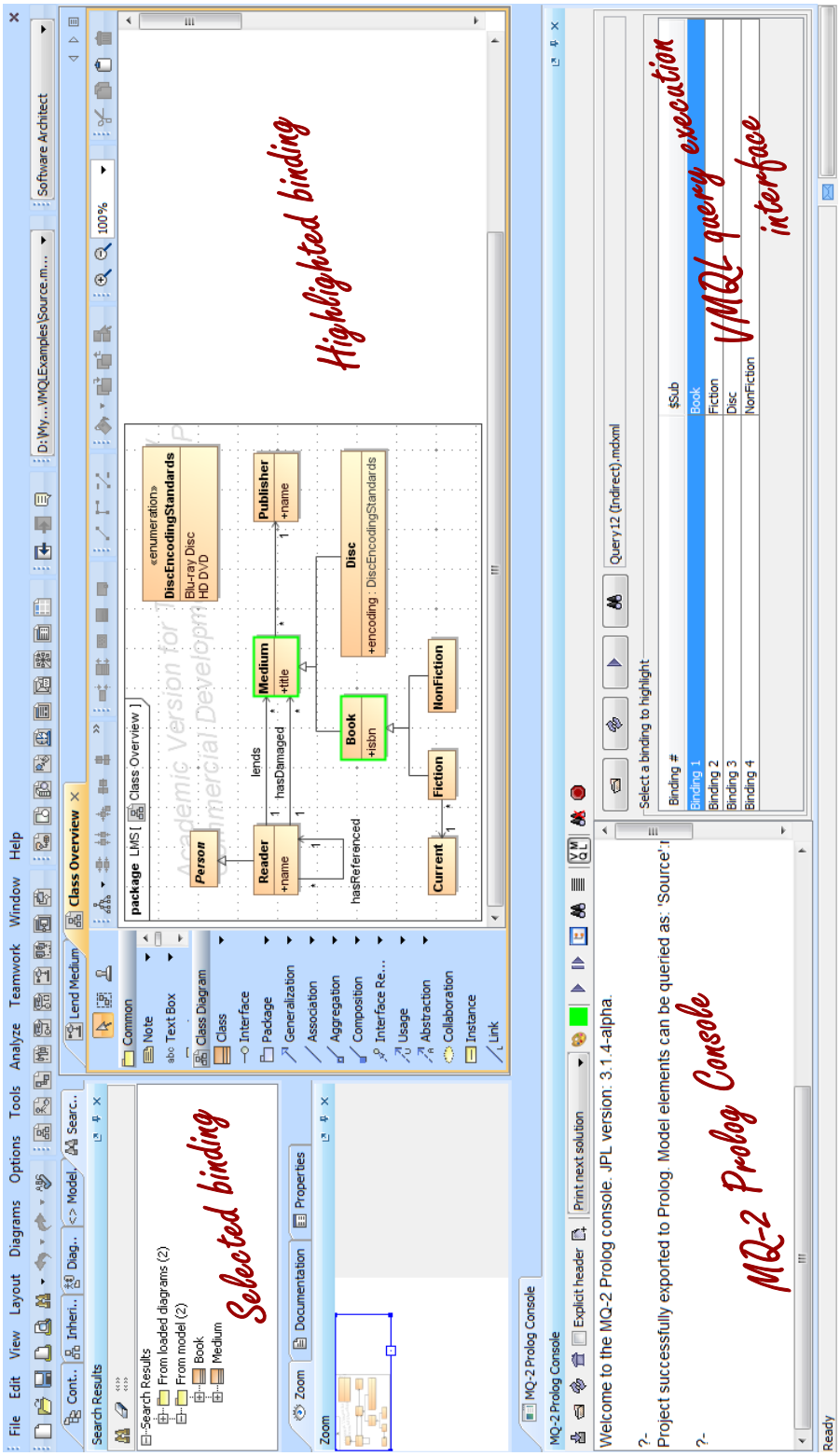


Figure 2: MagicDraw window featuring the MQ-2 Prolog console and VMQL query execution interface. The selected VMQL binding is highlighted on the model in green.

### 3.2 Result highlighting

Selecting a binding from the Bindings Table by clicking on it leads to the source model elements included in this binding being displayed in the MagicDraw Search Results Tree. The selected binding can also be highlighted on the source model's diagrams through the `Highlight the selected binding` button. Just as in the case of highlighting Prolog query results, the highlight color can be selected via the `Select highlight color` button on the Console Tool Bar, and highlights can be cleared via the `Clear highlights` button on the Console Tool Bar.

## References


[1] Störrle, H.: VMQL: A Visual Language for Ad-Hoc Model Querying. J. Visual Languages and Computing 22(1), 3-29 (2011).

[2] Acretoaie, V., Störrle, H.: MQ-2: A Tool for Prolog-based Model Querying. In: Joint Proc. co-located Events at the 8th European Conference on Modelling Foundations and Applications (ECMFA 2012), pp. 328-331. Technical University of Denmark, Kgs. Lyngby (2012).