MQ-2: A Tool for Prolog-based Model Querying

Vlad Acretoaie and Harald Störrle

Department of Informatics and Mathematical Modeling,
Technical University of Denmark
Richard Petersens Plads, 2800 Lyngby, Denmark
`s100988@student.dtu.dk,hsto@imm.dtu.dk`

Abstract. MQ-2 integrates a Prolog console into the MagicDraw[1] modeling environment and equips this console with features targeted specifically to the task of querying models. The vision of MQ-2 is to make Prolog-based model querying accessible to both student and expert modelers by offering powerful query features and a tight integration with the host modeling environment.

1 Motivation

MQ-2 is designed to support the model querying approach described in [1, 2] and its successor, the Visual Model Query Language (VMQL) [3]. The main impetus behind the development of MQ-2 has been the feedback gathered in follow-up interviews with participants to a paper-based usability study of VMQL [3]. A consensus has emerged among interviewees concerning the high impact of tool support on the usability of any model querying approach. MQ-2 leverages this observation and brings VMQL one step closer to its goal of becoming a fully usable model querying solution targeted at student and expert modelers.

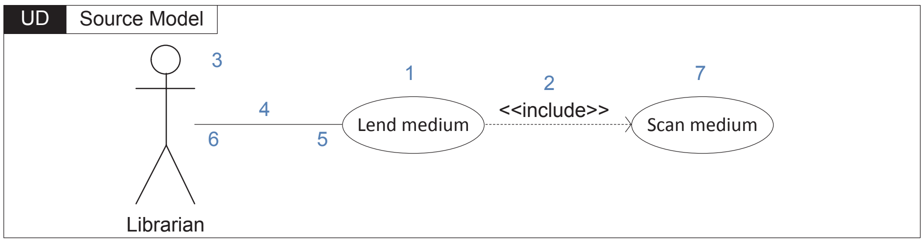
The remainder of this paper is organized as follows. Section 2 introduces the querying approach supported by MQ-2, Section 3 provides an overview of MQ-2's architecture and Section 4 proposes a demonstration plan.

2 Querying

Consider the use case diagram in Fig. 1, inspired by the Library Management System (LMS) test scenario (see Sec. 4). In order to perform queries on this diagram, it must first be transformed from its XMI representation into the Prolog fact database also shown in Fig. 1, with model element IDs highlighted in blue in both the diagram and its Prolog representation. This database consists of facts of the form `me(type-id, [tag-value], ...])`, where `type` is a model element's metaclass, `id` is an arbitrary unique identifier, `tag` is an atom representing one of the model element's properties, and `value` is the value for this property. There is a one-to-one mapping between model elements and Prolog facts.

Once a model's Prolog representation is created, it can be queried from any Prolog console. For instance, the query

[1] https://www.magicdraw.com/



```
:-module('Source Model',[me/2]).
    me(model-0,[annotation-id(1),ownedMember-ids([1,3,4,7]),name 'LMS',
                visibility-public]).
    me(useCase-1,[ownedMember-ids([2]),name-'Lend medium',
                  visibility-public]).
    me(include-2,[visibility-public,addition-id(7)]).
    me(actor-3,[name-'Librarian',visibility-public).
    me(association-4,[ownedMember-ids([5,6]),visibility-public,
                      navigableOwnedEnd-ids([6,5]),memberEnd-ids([6,5])]).
    me(property-5,[visibility-private,type-id(1),association-id(4)]).
    me(property-6,[visibility-private,type-id(3),association-id(4)]).
    me(useCase-7,[name-'Scan medium',visibility-public]).
```

Fig. 1. A use case diagram (top) and its encoding as a Prolog fact database (bottom)

```
me(useCase-Id,Attrs), member(name-'Lend medium',Attrs).
```

returns all model elements of metaclass `useCase` having the value 'Lend medium' for their `name` meta-attribute. It also binds the returned model elements' identifiers to the `Id` variable and their list of meta-attributes to the `Attrs` variable. In short, the query finds the *Lend medium* use case. Its execution is facilitated by the integration of a Prolog console into MagicDraw provided by MQ-2.

The MQ-2 console offers several model querying specific features not available in a generic Prolog console, as specified in Table 1.

Table 1. MQ-2 console features

Transforming models to Prolog fact databases.
Pre-consulted library predicates.
Showing query results sequentially or all at once.
Showing query results in the MagicDraw Search Results Tree.
Highlighting query results in diagrams where they appear.
Highlighting selected console text in relevant diagrams.

However, queries formulated using the `me` predicate directly are cumbersome to formulate. To compensate for this, MQ-2 implements several library predicates introduced in [1, 2] (see Fig. 2). Using library predicates, retrieving the *Lend medium* use case can be accomplished more intuitively via the `get_me` predicate:

```
get_me(model, name-'Lend medium', useCase-Id, _).
get_me(MODEL, TAG-VAL, METACLASS-ID, VAL)
        Matches all elements of MODEL containing the TAG-VAL meta-attribute pair.
match(SOURCE_MODEL, QUERY_MODEL, BINDINGS)
        Returns bindings between QUERY_MODEL and  SOURCE_MODEL.
match(SOURCE_MODEL, QUERY_MODEL, CONSTRAINTS, BINDINGS)
        Returns bindings between QUERY_MODEL and SOURCE_MODEL considering a list of VMQL constraints.
```

Fig. 2. Sample MQ-2 library predicates

The `match` predicate allows formulating queries using the host modeling language. It returns a list of bindings between elements of the query and source models, and optionally accepts a list of VMQL constraints. For instance, the `distinct` constraint specifies that no two query model elements may be bound to the same source model element. A complete list of VMQL constraints is available in [3]. A future goal for MQ-2 is to support the specification of constraints directly on the query model as comments endowed with the `<vmql>` stereotype.

3 Architecture

The proposed framework for Prolog-based model querying consists of a host modeling tool (currently MagicDraw) including the MQ-2 plug-in, an SWI-Prolog[2] installation, and the Java Prolog Bridge (JPL)[3] library (see Fig. 3). The host modeling tool acts as a model repository and a front-end for interacting with MQ-2, while SWI-Prolog acts as a query execution engine. MQ-2 itself is a plug-in extending the UI of the host modeling tool and providing built-in Prolog modules that implement functionality such as model matching. This architecture enables MQ-2 to remain easily portable to other host modeling tools.

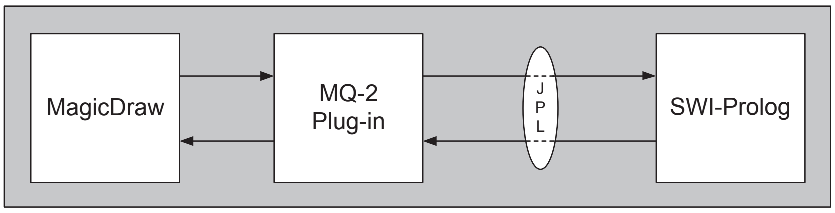
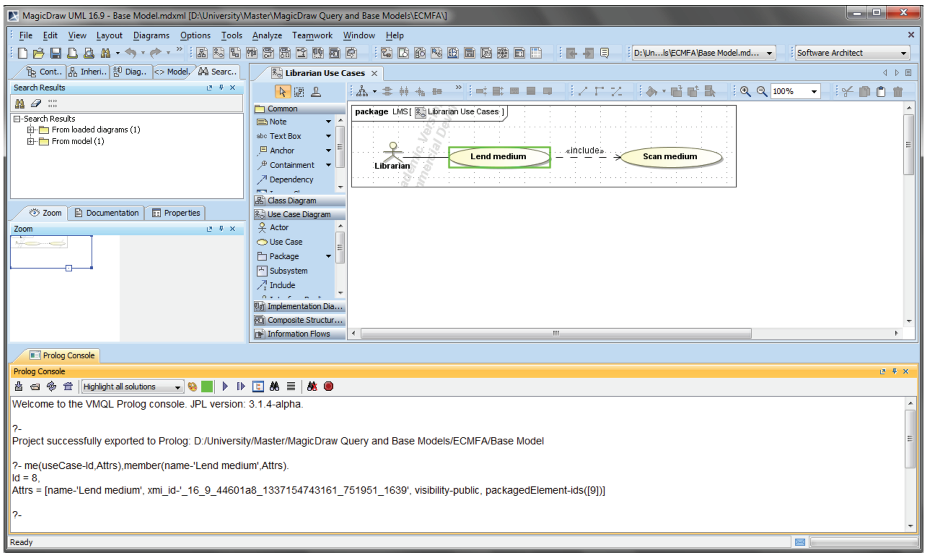


Fig. 3. MQ-2 deployment with MagicDraw as host modeling tool

A screenshot of MQ-2 is presented in Fig. 4. It features the MQ-2 Prolog console and toolbar on the bottom of the screen. The diagram pane shows the diagram introduced in Sec. 2, and the console query retrieves the *Lend medium* use case. As a result, this use case is highlighted in green on the diagram pane.

[2] http://www.swi-prolog.org/

[3] http://www.swi-prolog.org/packages/jpl/


MagicDraw UML 16.9 - Base Model.mdxml [D:\University\Master\MagicDraw Query and Base Models\ECMFA\]
File Edit View Layout Diagrams Options Tools Analyze Teamwork Window Help
Librarian Use Cases
package LMS[Librarian Use Cases]
Librarian
Lend medium
«include»
Scan medium
Search Results
Use Case Diagram
Highlight all solutions
Prolog Console
Ready

```
Welcome to the VMQL Prolog console. JPL version: 3.1.4-alpha.

?-
Project successfully exported to Prolog: D:/University/Master/MagicDraw Query and Base Models/ECMFA/Base Model

?- me(useCase-Id,Attrs),member(name-'Lend medium',Attrs).
Id = 8,
Attrs = [name-'Lend medium', xmi_id-'_16_9_44601a8_1337154743161_751951_1639', visibility-public, packagedElement-ids([9])]

?-
```

Fig. 4. Screenshot of MagicDraw featuring the MQ-2 plug-in

4 Demonstration Plan

MQ-2 will be showcased on a UML design model created by a group of students in the context of the Requirements Engineering course taught at a Master's level at the Technical University of Denmark. The model specifies the requirements for an LMS used by a local library for the purpose of managing its book inventory, loans, librarians, and readers. This usage scenario has been selected in view of the fact that MQ-2 is envisioned to be used by students taking the same course in the next academic year, providing arguably the best feedback as to whether MQ-2 meets its design goal of acting as a usable model querying tool.

The tool demonstration will include transforming the source model into a Prolog fact database, querying individual model elements, and querying model fragments containing VMQL constraints. The various query result display methods provided by MQ-2 will also be highlighted.

References

1. Störrle, H.: A logical model query interface. In: Intl. Ws. Visual Languages and Logic (VLL'09), pp.18–36. CEUR (2009).
2. Störrle, H.: A PROLOG-based Approach to Representing and Querying UML Models. In: Intl. Ws. Visual Languages and Logic (VLL'07), pp.71–84. CEUR (2007).
3. Störrle, H.: VMQL: A Visual Language for Ad-Hoc Model Querying. J. Visual Languages and Computing 22(1), 3–29 (2011).