MDELite Manual

August 2015

1. Introduction

MDELite is an approach for teaching and exploring concepts in *Model Driven Engineering* (MDE) as an alternative to Eclipse tools. Instead of storing models and metamodels as obscure and unreadable XML documents, it encodes them as readable relational databases expressed as Prolog facts. Instead of using the convoluted *Object Constraint Language* (OCL) to express constraints, it uses Prolog, a fundamental language in Computer Science. And rather than using the *Atlas Transformation Language* (ATL), which is an outgrowth of OCL, to write *model-to-model* (M2M) transformations, MDELite again relies on Prolog. For its *model-to-text* (M2T) tool, MDELite uses Apache Velocity, an off the shelf tool used in industry. The benefits and broad explanation of MDELite are explained in a 2013 MODELS paper.

This is a manual for `MDELite`, a Java framework that implements the MDELite approach. Here is the table of contents:

- Installation of `MDELite`
- `MDELite`Utilities
- A Tutorial on Prolog-Relational Schemas
- Big Picture on MDELite

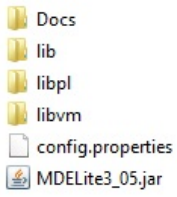
Warning: As hard as I have tried, I know there are bugs in `MDELite`. Please let me know when you find them. I will do my best to fix them.

2. Installation

You can download `MDELite` from this link:

```
www.cs.utexas.edu/users/schwartz/MDELite/index.html
```

The `MDELite` directory and executable contains:



- **Docs** – a directory with this documentation,
- **lib** – a library of jar files needed by `MDELite`,
- **libpl** – a library of Prolog files,
- **libvm** – an empty library of velocity templates, and
- **config.properties** – a Java properties file that contains only one setting the absolute path to the SWI Prolog command line (not Windows GUI) executable.

To install `MDELite`, you must:

1. Set the value of `SWI_PROLOG_LOCATION` in `config.properties`; this is the absolute path to `swipl.exe`, the `SWI_PROLOG` executable,
2. Place the `MDELite` directory in a global directory where MDELite applications reside, and
3. Add `MDELite.jar` to your `CLASSPATH`.

Here's how you can check to see if you did the above tasks correctly: Run `Utils.VerifyInstallation`. If you did the above steps correctly, you'll get:

```
> java Utils.VerifyInstallation
HomePath: C:/.../MDELite3.05/...
swipl is working!
Violet should be running now.  If not, something is wrong.
In any case, please close Violet...
MDELite Ready to Use!
```

Here's what the above output means:

- `HomePath` tells you where `MDELite.jar` resides,
- Runs `swipl.exe` to verify that your setting in `config.properties` is correct, and
- Invokes the Violet tool – you have to exit Violet for the last sentence to be reported.

3. MDELite Utilities

`MDELite` has a `Utils` directory currently with three programs:

- `Utils.VerifyInstallation` – we already discussed this. The program verifies that you have correctly installed MDELite by running sanity checks:

  ```
  > java Utils.VerifyInstallation
  ```

- `Utils.Violet` – this program invokes the Violet tool. You can invoke Violet directly through its jar file, but calling it from a command line is painful; `Utils.Violet` makes it easy. Further, you have an option: if you give `Utils.Violet` *any* argument, it will wait until Violet is closed. Otherwise, `Utils.Violet` spawns Violet and immediately returns allowing you to invoke other programs on a command line:

  ```
  > java Utils.Violet
      // spawns Violet and returns immediately
  ```

```
> java Utils.Violet <anything>
     // spawns Violet and waits for Violet to close
```

- `Utils.ElaborateSchema` – `MDELite` uses a particular format to define relational database schemas. You can define a short version of a schema in a Prolog file, say `myschema.pl`, and use `Utils.ElaborateSchema` to elaborate your short definition into a version that `MDELite` can use. More on this in the next section.

```
> java Utils.ElaborateSchema myschema.pl
myschema.schema.pl produced
```

4. Tutorial on MDELite Relational-Prolog Schemas

In MDE-speak, a model conforms to a metamodel. In MDELite-speak, a meta model is a relational schema; a relational database is a model that conforms to its schema.

MDELite allows you to outline a relational schema in a prolog file. Here is a typical "short" declaration in `school.pl`:

```
dbase(school,[person,professor,department,student]).

table(person,[id,"name"]).
table(professor,[deptid]).
table(department,[id,"name","building"]).
table(student,[utid]).

subtable(person,[professor,student]).
```

Here's what the above outline means:

- The name of this schema is `school`. It contains a list of 4 tables: `person`, `professor`, `department`, `student`.
- Every table has a name and a list of columns/attributes. The `person` table has two attributes: `id` (identifier) and "`name`".

The following lines define attributes that are specific to the `professor`, `department`, and `student` tables. There are three important conventions used in MDELite tables:

1. The first attribute of every MDELite table is a manufactured `id` (identifier) field.
2. There are two kinds of fields in MDELite tables: those with unquoted values and those with single-quoted values.
3. An n-tuple of a table `t` is written as a prolog fact: $\texttt{t}(\texttt{v}_1 \ldots \texttt{v}_\texttt{n})$. Some `person` tuples might be:

   ```
   person(p1,'Don').
   person(p2,'Hanna Elizabeth').
   ```

   Values of a tuple are listed in the order that their column/attributes are listed in their table definition.

   > **Note:** `id` values are unquoted as the `id` attribute name is unquoted in the table declaration. `name` values are single-quoted as the `name` attribute name is double-quoted in the table declaration.

4. Tables can be arranged in an inheritance hierarchy, which are specified by subtable declarations:

   ```
   subtable(person,[professor,student]).
   ```

   This declaration means that the subtables of `person` are `professor` and `student`. Stated differently, every `professor` and `student` is a `person`.

As mentioned earlier MDELite uses a more elaborate definition of a schema. You can produced this schema by running:

```
> java Utils.ElaborateSchema school.pl
School.schema.pl produced!
```

Besides checking the sanity of your schema outline, it produces the file `school.schema.pl` below:

```
dbase(school,[person,professor,department,student]).

table(person,[id,"name"]).
table(professor,[id,"name",deptid]).
table(department,[id,"name","building"]).
table(student,[id,"name",utid]).

subtable(person,[professor,student]).

tuple(person,L):-person(A1,A2),L=[A1,A2].
tuple(professor,L):-professor(A1,A2,A3),L=[A1,A2,A3].
tuple(department,L):-department(A1,A2,A3),L=[A1,A2,A3].
tuple(student,L):-student(A1,A2,A3),L=[A1,A2,A3].

personALL(A1,A2):-person(A1,A2).
personALL(A1,A2):-professorALL(A1,A2,_).
personALL(A1,A2):-studentALL(A1,A2,_).
professorALL(A1,A2,A3):-professor(A1,A2,A3).
departmentALL(A1,A2,A3):-department(A1,A2,A3).
studentALL(A1,A2,A3):-student(A1,A2,A3).
```

Here are the differences and similarities between the "short" and "long" versions:

- The `dbase` declaration is the same.
- Using `subtable` declarations (*ie,* table inheritance hierarchy information), attributes of super-tables are propagated to sub-tables. Above, every `professor` tuple and every `student` tuple will have `person` attributes.
- Tuple declarations are used by MDELite to print database. In essence, it converts each row $\texttt{t}(\texttt{v}_1 \ldots \texttt{v}_\texttt{n})$ of table `t` into a prolog fact $\texttt{tuple}(\texttt{t}, [\texttt{v}_1 \ldots \texttt{v}_\texttt{n}])$, which MDELite can then print.
- The prolog `ALL` rules are used to retrieve all tuples of a table within a table inheritance hierarchy. Above, a `personALL` tuple is either a `person` tuple, or a `professor` tuple, or a `student` tuple.

5. Big Picture on MDELite

You now have an OO framework for building an MDE application using `MDELite`. This manual isn't enough for you to proceed. You first need to read and re-read the 2013 MODELS paper on which MDELite is based. It will tell you how to use this framework even though the paper discussed the 1^{st} generation of MDELite. The ideas are the same. There are now two MDELite applications that use MDELite. They are:

- `CDExchange` – an Avalon version of the application described in the 2013 MODELS paper.
- `Avalon` – the MDELite application that is used for bootstrapping: it can build `CDExchange` and itself (`Avalon`).

The hard part of building any MDE application is implementing the arrows. `MDELite` doesn't help you with this. I suggest that you download the source code of these applications, read their manuals, and study them.