MACH 0.94 (subsonic)

Harald Störrle
Department of Applied Mathematics and Computer Science
Technical University of Denmark (DTU), Technical Report 2014-02
hsto@dtu.dk

March 13, 2014

1 Introduction

The Model Analysis and Checker (MACH) tool is an experimental tool to deploy and test advanced algorithms for analyzing UML models. MACH is a flexible and lightweight framework for the loose integration of independent tools, and a set of tools integrated this way under a common UI. All parts of MACH are implemented using SWI-Prolog [5]. Many of its more advanced parts have been subject to previous research publications.

MACH is targeted at modelers that already have a fairly good understanding of UML as a modeling language. Given its purpose and intended audience, MACH puts little focus on UI, making it somewhat challenging to use. MACH is more of a tool demonstrator rather than a high-productivity CASE tool.

2 Getting started

2.1 Obtaining and installing MACH

MACH requires SWI Prolog to be installed. To the best of our knowledge, MACH works with any recent version of SWI-Prolog. SWI Prolog can be downloaded from `www.swi-prolog.org` and is available for many platforms. The installation is easy, but takes a a few minutes. MACH can be downloaded for free from `www.compute.dtu.dk/~hsto/tools`. Installing MACH amounts to unpacking the zip and placing the executable in the directory that also contains the `swipl.dll` (or equivalent thereof on other platforms). Start

MACH by running the executable (i.e., double-clicking or similar). MACH will open a console window. In the console window, it's probably best to first set the path to the directory with samples that comes with MACH. It is placed inside MACH, so you just have to say "`cd "samples"`". See below for a sample session.

2.2 First steps

Conventions Commands are entered at the prompt and terminated by the ⟨CR⟩ character. Names containing special symbols like spaces, slashes, hash, or dollar signs must be enclosed in double quotes. There are some problems with long file names, and it is thus recommended to use short and simple names when working with MACH. The MACH command line has a history function, that is, the cursor keys ⟨UP⟩ and ⟨DOWN⟩ recall the previous commands. MACH is exited by typing "quit". There is currently only a rudimentary help function, help for specific commands is not implemented. Similarly, the commands "magic" and "pray" are not yet implemented. Observe that MACH is case sensitive.

Issues Every now and then, a command fails and MACH terminates unexpectedly. For instance, this happens when model names that are supposed to be enclosed by double quotes are not properly closed, i.e., the trailing quotes are omitted. While inconvenient, the user can often recover from this fault quite easily by restarting MACH (saying "mach." at the PROLOG command line). This allows the user to carry on exactly where MACH exited: the environment (such as the current working directory and the opened models) are maintained. At this point, MACH has only been tested with the XMI produced by MD 16.9 to 17.0.3. Since commercial and academic UML modeling tools tend to deviate from the XMI standard, using other file formats might result in incompatibilities.

Getting help The command "help" reminds you of some basic commands and conventions. The command "help commands" lists the available commands alphabetically and provides a short description. The command "help X" will provide a more detailed description for the command X. Any characters after X are ignored so that if a command fails, one can simply recall the previous command from the command history by CURSOR UP, go to its beginning by POS1, and insert "help" to obtain help on the respective command.

```
Welcome to MACH 0.93 ("Subsonic") (C) 2013-2014, H. Strrle
no warranties whatsoever
MACH is case sensitive - exit by saying "quit"
>
> pwd
 h:/_arbeitsbereich/projekte/momat/
> cd "mach/samples"
> pwd
 h:/_arbeitsbereich/projekte/momat/mach/samples/
> ls -m
        LMS_2011_2.mdxml         m2.mdxml       IE1.mdxml IE2.mdxml
> open "LMS_2011_2"
%  LMS_2011_2.pl compiled into LMS_2011_2 0.22 sec, 2,586 clauses
> size "LMS_2011_2"
Model LMS_2011_2 has 2584 elements with 7568 attributes.
This is magnitude 4, a medium model.
> frequency "LMS_2011_2" width 60 min 25 sort alphabetical
                      activity  33 ++
              activityFinalNode  26 ++
              activityPartition  69 +++++
                    association 120 ++++++++
            callBehaviorAction 231 ++++++++++++++++
              centralBufferNode  26 ++
                          class  30 ++
          componentRealization  26 ++
                  connectorEnd  32 ++
                   controlFlow 369 +++++++++++++++++++++++++
                dataStoreNode  26 ++
                 decisionNode  58 ++++
            enumerationLiteral  37 ++
                      forkNode  27 ++
                       include  60 ++++
                   initialNode  31 ++
                      inputPin  39 +++
                literalInteger  38 +++
messageOccurrenceSpecification  39 +++
                    objectFlow 105 +++++++
                     operation  58 ++++
                     outputPin  35 ++
                       package  37 ++
                     parameter  78 +++++
                      property 377 +++++++++++++++++++++++++
         receiveOperationEvent  43 +++
                         state  38 +++
                    transition  62 ++++
                       trigger  45 +++
                       useCase  32 ++
>
```

3 Working with directories

MACH offers some commands to navigate directory trees. The syntax is vaguely reminiscent of a UNIX shell, but offers only a very small number of the most basic options. Recall that using some special characters such as white spaces will require that you enclose your path expression in double quotes.

`pwd`

Shows the current working directory ("print working directory").

`cd PATH`

Move to the directory indicated by the parameter ("change directory"). Accepts a relative path separated by "/", and using ".." for the upper level directory. E.g. `cd subdir` changes the working directory to the `subdir` subdirectory of the current working directory. Path names have to be enclosed by double quotes.

`ls OPTION`

Shows the contents of the current working directory. Accepts the following values for `OPTION`:

- `m` : list only models
- `a` : list all files
- `p` : list all Prolog files

4 Working with Models

The following commands are used to access existing models in MACH. Models are assumed to be stored as files conforming to the MagicDraw MDXML format (based on the standard XMI model interchange format), having the *.mdxml* file extension.

`open FILE`

Accesses `FILE` and tries to interpret it as a model. The *.mdxml* file extension should be omitted when specifying `FILE`. E.g.: The command `open testmodel` opens the model stored in the file *testmodel.mdxml*.

`open FILE as $ALIAS`

Accesses `FILE` and tries to interpret it as a model. Assigns the name `$ALIAS` as a shorthand for the file. This alias may be used instead of a model file name in all subsequent commands. When using an alias, it must be preceded by the dollar sign. The *.mdxml* file extension should be omitted when specifying `FILE`. E.g.: The command `open testmodel as $x` opens the model stored in the file *testmodel.mdxml* and assigns `$x` as a shorthand for it.

`show all aliases`

List all currently defined aliases.

`show alias $ALIAS`

Show details of the specified alias.

`clear all aliases`

Remove all alias declarations.

`clear alias $ALIAS`

Remove the specified alias declaration.

`models`

`show all opened models`

List all the models currently opened.

5 Querying and inspecting models

The following commands are used to search for specific patterns or elements in models, and to ispect parts of models in detail.

`dump all of MODEL`

> Show a tabular overview of all elements in `MODEL`. E.g.: The command `dump all of $x` shows a tabular overview of all elements in the model previously associated with the alias `$x`, while the command `dump all of testmodel` shows a tabular overview of all elements in the model stored in the file *testmodel.mdxml*.

`dump #ID of MODEL in (detail|summary|overview)`

> Show a tabular overview of the element with identifier `ID` in `MODEL`, providing several levels of details. E.g.: The command `dump #mdx123 of $x in detail` shows a detailed tabular overview of the element with `ID = mdx123` of the model previously associated with the alias `$x`, while the command `dump #mdx123 of testmodel in summary` shows a summary tabular overview of the element with `ID = mdx123` of the model stored in the file `testmodel.mdxml`.

`find exactly QUERY in MODEL`

> Looks for the string QUERY in MODEL, as an exact match.

`find QUERY in MODEL`

> Looks for the string QUERY in MODEL, with a degree of vagueness in the matching (e.g., to correct for typos and the like).

6 Model differences and versions

`difference between MODEL1 and MODEL2`

Compute the difference between `MODEL1` and `MODEL2` and list the result in a table. The algorithms applied in this command and its options explained below are published as [1, 2, 3].

`difference between E1 of MODEL1 and E2 of MODEL2`

Compute the difference between the sub-model below the element `E1` in `MODEL1` and element `E2` in `MODEL2`. The result is presented in a table by default.

`difference between MODEL1 and MODEL2 aggregated`

After computing the difference between the two models, this option instructs MACH to try and aggregate the (large) number of low-level changes into a smaller number of high-level changes that make more sense to modelers (cf. [3]). By default, the result is presented in a table. Aggregation only works for class models.

`diff MODEL1 and MODEL2 as text`

Present the difference as prose rather than in a table. This feature works only for class models, and it is not very reliable.

7 Model similarity

`similarity of MODEL1 and MODEL2 by identifiers`

Compute the overlap of the identifiers between the two models. This metric only makes sense when applied to successive versions of a models. If the modeling tool used creates identifiers in a deterministic way, high similarity scores may be indicated for dissimilar models.

`similarity of MODEL1 and MODEL2 by edits`

Compute the similarity between `MODEL1` and `MODEL2` as correlated to the inverse of the edit distance. The values range from 1 to 0, where larger values represent models of greater similarity. Identical models have an edit distance of 0, which we interpret as a similarity of 1. The edit distance yields a more sensitive similarity metric than spectral comparison, but is limited to comparing models with a (close) common ancestor, such as one of the two models to be compared. Comparing models without a (close) common ancestor will yield edit distances larger than the target model size. This metric only makes sense when applied to successive versions of a models.

`similarity of MODEL1 and MODEL2 by length`

Compute the similarity of the two models purely based on their length. This metric may be applied to models that are not successive versions of each other, but is a very coarse approximation of true similarity.

`similarity of MODEL1 and MODEL2 by spectrum`

Compute the (scaled) cosine-similarity of the meta-class spectra of `MODEL1` and `MODEL2`. The values range from 1 to 0, where larger values represent models of greater similarity. Identical models receive a value of 1. This metric may be applied to models that are not successive versions of each other. It is a better approximation to "similarity by length".

`similarity of MODEL1 and MODEL2 by plgind`

Compute the likelihood that the two models are successive versions of each other. If the two models score high on this similarity without being proper versions of each other, this is an indication of plagiarism.

8 Model size and quality

The following commands are used to perform model manipulation tasks.

`size of MODEL`

> Provide a rough measure of the size of `MODEL`. E.g.: The command `size $x` computes the size of the model previously associated with the alias `$x`, while the command `size testmodel` computes the size of the model stored in the file *testmodel.mdxml*.

`frequency of MODEL`

> Show a histogram of the frequency distribution of meta classes in `MODEL`. E.g.: The command `frequency $x` shows a histogram of the frequency distribution of meta classes in the model previously associated with the alias `$x`, while the command `frequency testmodel` shows a histogram of the frequency distribution of meta classes in the model stored in the file *testmodel.mdxml*.

`frequency difference of MODEL_1 MODEL_2`

> Show a histogram of the frequency distribution of meta classes in `MODEL`. Both this and the previous command offer a free combination of the following options attached to their end.

`...sort DIRECTION`

> Sort the entries by increasing or decreasing size, or alphabetically by the name of the meta-class.

`...width INT`

> Scale the output to fit on INT columns.

`...scale FACTOR`

> Scale the output by the given factor (may result in visually empty bars).

`...min VALUE`

> Clip all entries smaller than the given value.

`...max VALUE`

> Clip all entries greater than the given value.

`clones of MODEL`

> Computes and lists clone candidates in `MODEL`. E.g.: The command `clones $x` shows clone candidates in the model previously associated with the alias `$x`, while the command `clones testmodel` shows clone candidates in the model stored in the file `testmodel.mdxml`. The algorithms applied in this command are published as [4].

A Cheat Sheet

Notation inspired by EBNF.

```
pwd
cd PATH
ls [-m|-a|-p]
open FILE [as $ALIAS]
show all aliases
show alias $ALIAS
clear all aliases
clear alias $ALIAS
dump all of MODEL
dump #ID of MODEL in (detail|summary|overview)
find [exactly] QUERY in MODEL
similarity of MODEL1 and MODEL2
                        [by (spectrum|edits|identifiers|length|plgind)]
diff [E1 of] MODEL1 and [E2 of] MODEL2 [ aggregated |as text]
size MODEL
frequency MODEL
frequency difference A B
clones MODEL
```

References

[1] Harald Störrle. A formal approach to the cross-language version management of models. In Ludwik Kuzniarz, Miroslaw Staron, Tarja Systä, and Mia Persson, editors, *Proc. 5th Nordic Ws. Model Driven Engineering (NW-MODE'07)*, pages 83–97. Blekkinge Tekniska Hgskolan, August 2007.

[2] Harald Störrle. Making Sense of UML Class Model Changes by Textual Difference Presentation. In Dalila Tamzalit, Bernhard Schtz, Jonathan Sprinkle, and Alfonso Pierantonio, editors, *Proc. Ws. Models and Evolution (ME)*, pages 1–6. ACM DL, 2012.

[3] Harald Störrle. Making Sense to Modelers - Presenting UML Class Model Differences in Prose. In Joaquim Filipe, Rui Csar das Neves, Slimane Hammoudi, and Lus Ferreira Pires, editors, *Proc. 1st Intl. Conf. Model-Driven Engineering and Software Development*, pages 39–48. SCITEPRESS, 2013.

[4] Harald Störrle. Towards Clone Detection in UML Domain Models. *J. Softw. Syst. Model.*, 12(2), 2013. (accepted in 2011).

[5] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.