Delivery date:

04-08-2016

Authors:

Sigve Karolius

MoDeNa

Deliverable D5.9

Testing - Framework Production Release

Principal investigators:

Sigve Karolius

Collaborators:

Heinz A Preisig
Henrik Rusche

Project coordinator:

Heinz A Preisig

heinz.preisig@chemeng.ntnu.no

Abstract:

The MoDeNa software framework testing is intended to reveal potential problems that may arise when installing and using the software on different platforms and operating systems. Therefore, the testing was carried out on a laptop and two types of centralised systems, a server and HPC cluster. Versions of the dependencies, as well as the MoDeNa software API and workflow manager, was tested and verified.

Contents

1 Introduction

The purpose of the software testing is to verify that the documentation of the framework, i.e. dependencies and install procedures of the dependencies and examples is correct and that it works on a wide variety of platforms. The goal is to test the components software framework, as well as the multiscale application developed in the project. However, the important aspects to test are different for the two:

- The MoDeNa software framework
  - Dependencies
  - Install procedure
  - Application Program Interface (API)
  - Database
- Polyurethane multiscale application
  - Dependencies
  - Install procedure (individual modules)
  - Running the applications

The different areas that were investigated during the testing phase are introduced below.

Dependencies
: Dependencies are software packages that are required in order to use a particular software. The dependency chain for the MoDeNa software framework is small, but the polyurethane application have modules which relies on specific versions of other software packages. The testing will verify that the documented versions of the dependencies works.

Install procedures
: The software framework and multiscale applications are written in a number of different programming languages (Fortran, C, C++, Python). The testing will verify that the documented install/compiling procedures are correct.

API
: The MoDeNa software framework provides an API which is used in order to embed surrogate models into the modules that make up the multiscale model. The API is tested by running the example cases ensuring that backward and forward mapping surrogate models behave as expected, as well as index sets and substitute models. This also includes the computational workflow orchestrator, i.e. Fireworks Jain et al. (2015).

Database
: The software framework must be capable of using the MongoDB database whilst it is being hosted remotely or on the local laptop. The reason for this is that some computer systems, e.g. clusters, will not allow a user to host a database on the system itself.

The ultimate goal of the testing phase is to reveal potential problems, verify that the software can run on the target platforms and that key properties work. The following report summarises the results and experiences learned from the testing phase, and specifies the improvements necessary in order to fix potential problems.

2 Testing Environments

The testing has been performed on multiple Linux-based systems. The purpose was to test the software on a variety of operating systems and hardware, both as an administrator, e.g. on a personal laptop, and on a multi-user system without administrator rights. The tests environments are chosen to represent possible situations that a user would like to use the software framework. It has been tested on a variety of operating systems, as well as hardware.

2.1 Laptop

The specifications for the laptop used for the testing are the following: Dell latitude-e7450 , Intel i7-5600U CPU @ 2.60GHz, 16GB DDR3 1600 MHz RAM, Ubuntu 15.04 64bit OS. Additionally, the Linux operating systems that have been used by project partners are: Debian 7/8 and openSUSE. Earlier versions of the software framework was also tested on OS-x 10.9 (Mavericks), shown in testing reports for earlier versions **?**, but there are currently nobody in the project that uses a Macintosh computer.

2.2 Server

The specifications for the server are the following: 2 $\times$ Intel Xeon X5650 CPU @ 2.67GHz, 23GB RAM, Ubuntu 16.04 64bit OS.

Running the software on a server, where a user will typically not have administrative rights, can potentially lead to problems with installing dependencies. Moreover, system administrators are typically conservative with respect to updates or installing brand new software. Consequently, running the MoDeNa software framework, and particularly the multiscale application, on a server will most likely require the user to install dependencies in "user-space", i.e. locally for a single user.

2.3 Cluster

The software framework was tested on one of the "heavy computing" facilities at the university: Kongull (now shut down). The specifications for the compute nodes are: 2 $\times$ Intel Xeon CPU E5-2670 @ 2.60GHz 8-core (Sandy Bridge) processors, 32 GB DDR3 1600MHz RAM, CentOS 5.3 64bit OS. The major difference from running on a laptop or server is that the heavy computing resource requires computational jobs to be dispatched through a "batch" queuing system, meaning that the computational jobs are dispatched to a software which allocates the desired resources automatically when hardware is available.

The queuing system used on "Kongull" is the so-called "portable batch system" (PBS). A simple script for the PBS queuing system looks like this:

```
#!/bin/bash
#PBS -N JOB_NAME
#PBS -A ACCOUNT_NAME
#PBS -l nodes=1
#PBS -l walltime=00:30:00

python MyProgram.py
```

The reason why this is of particular interest for MoDeNa is that the software framework works by dispatching many jobs to the queue, something which system administrators may not be happy with.

3 Dependencies

Software dependencies are often introduced in order to avoid "reinventing the wheel". Therefore software systems take advantage of other software components that are already available; thus, saving time and avoiding bugs in the development process. The downside is that one dependency may introduce several dependencies; hence, referring to the resulting list of dependencies as a "dependency chain". The second problem is that the dependencies may be version dependent, i.e. works with version "3.4", but not "3.3".

Therefore, the testing phase must verify which versions of the various dependencies that are necessary. Please note that the install procedures tested assumes that the user has "administrator privileges". This will most likely not be the case on centralised systems, such as severs and cluster. Here one will have the option of asking the system administrator to install the required programs or manually install them in "user space". The later was used in the test case, but the procedures were considered too convoluted to be documented in this report. Therefore, anyone that have software which is out of date will be encouraged to talk to their system administrator.

3.1 Framework Dependencies

The dependency chain for the MoDeNa software framework is fairly lightweight, as many Linux distributions have Python and R preinstalled. However, during the testing phase it was found that it may be a problem on server/cluster systems that the versions are old and requires an update.

The dependencies for the MoDeNa project are listed in Table 1. Even though some of the dependencies will introduce dependency chains, especially the python and R packages, the most important factor is to ensure that the version is correct. The installation procedures for the MoDeNa framework have also

Dependency	Type	Version
R	Language and library	$\geq$ 3.1.0
Python	Language and library	$\geq$ 2.7.0
cmake	Tool	$\geq$ 3.0.0
libtool	Tool	$\geq$ 2.4.2
automake	Tool	$\geq$ 1.14.1
mongodb	Database system	$\geq$ 3.0.11
blessings	Python package	$\geq$ 1.5
rpy2	Python package	$\geq$ 2.5.6
pymongo	Python package	$\geq$ 2.5.2
mongoengine	Python package	$\geq$ 0.10.5
FireWorks	Python package	$\geq$ 1.2.5
nlmrt	R package	$\geq$ 2013.9.25
lhs	R package	$\geq$ 0.10

Table 1: Dependencies for the MoDeNa software framework

been tested, however, note that "administrator privileges" are required. Python (pip) and R (CRAN) packages can easily be installed in "user space", but aptitude packages will have to be compiled from source. During the test phase this had to be done on both the server and cluster system, but the procedure is not documented in this report.

3.1.1 Aptitude Packages

The aptitude package manager is a common platform on Linux systems. The software packages that are available through aptitude are tried and tested, and does not lie on the cutting-edge of the development. The positive side effect is that the version is more likely to be stable, but the downside is that bugs take a long time to get fixed.

The MoDeNa software framework recommends installing as much software as possible using a package manager. The names of the software depends on the package manager used, but for aptitude the following packages can be installed:

```
user@machine> sudo apt-get install cmake, automake libltdl-dev libltdl7 mongodb \
    python-rpy2 python-pip python-scipy python-rpy2 python-blessings \
    r-base r-base-dev
```

The MoDeNa software framework also requires Python and R packages that are not available through aptitude, for these the package manager for the respective programming languages must be used.

The downside of using aptitude is that, unlike Macintosh's package manager "macports", it can not install packages in "user space", i.e. in the directory tree belonging to a specific user. This is particularly useful on centralised systems, where a regular user does not have administrative (sudo) privileges. Consequently, if the software version on the cluster or server is not up-to-date, the system administrator should be contacted to avoid having to compile the libraries manually.

3.1.2 Python Package Manager

The Python Package Index (PyPI) is a package manager for python modules. It is not included in the standard python distribution, and hence pip must be installed either using the standard python package manager "easy_install" or the os-system "aptitude". The reason for using `pip` is that it is recommended by the python foundation, as the features goes beyond that of `easy_install`.

The required python packages are installed as follows:

```
user@machine> sudo pip install FireWorks pymongo mongoengine
```

The packages on PyPI are generally more up-to date than the equivalent aptitude versions. However, this may affect the stability of the packages. The MoDeNa framework recommends installing packages from aptitude, but some packages are not yet available.

The advantage of using `pip` is that packages can easily be installed in user space.

```
user@machine> pip install --user <PACKAGE>
```

During the testing phase this was necessary in order to install all dependencies in the server environment.

3.1.3 R Package Manager

The Comprehensive R Archive Network (CRAN) is a network of servers containing identical and up-to-date versions of libraries of code.

Packages can be installed directly from inside R, and is automatically installed in user space. The following commands install the dependencies necessary for the MoDeNa framework.

```
user@machine > R
install.packages('lhs',    repos='http://cran.cnr.Berkeley.edu/',
                   lib=.libPaths()[1], dependencies=TRUE)
install.packages('nlmrt', repos='http://cran.cnr.Berkeley.edu/',
                  lib=.libPaths()[1], dependencies=TRUE)
q()
```

The repository that is used is hosted at UC Berkley, there is no particular reason for choosing Berkley, other than that by specifying the repository when running the install command one is not prompted to choose a repository, which may be confusing to some users.

3.2 Polyurethane Application Dependencies

The dependencies related to the Polyurethane multiscale application are listed in Table 2. These dependencies are heavily version dependent and most of them are very large. Moreover, the dependency chain related to the applications may be extensive. Installing the individual dependencies are documented in the applications themselves. Some can be installed directly using a package manager, while other dependencies require a more manual approach.

Dependency	Type	Version
LAPACK/BLAS	Library	$\geq 3.4.2$
Boost	Library	$\geq 1.58.0$
gmsh	Tool	$= 2.8.5$
PETSc	Tool	$= 3.4.5$
openFOAM	Tool	$= 2.3$

Table 2: Dependencies for the Polyurethane multiscale application.

The procedures are not documented here because the problems that are encountered will depend heavily on operating system and version, as well as the versions of the libraries that are installed.

4 Database

The MongoDB database is used to store the surrogate models, as described in Karolius et al. (8 31). The database can in principle be hasted anywhere in the world, and the environment variable `MODENA_URI` can be set in order to specify the location of the database.

The environment variable can either be set in the shell, before running the application, or it can be set in the initialisation script which initialises the models. This provides a flexible way of switching between different databases and locations.

4.1 Locally hosted

The default behaviour is to look for the database **test** at port 27017 (default MongoDB demon port) at location `localhost`, which is a synonym for the IP address 127.0.0.1. The equivalent `Modena_URI` is: `mongodb://localhost:27017/test`, however, this need not be specified as it is the default MoDeNa location.

Hosting the database locally results in very quick communication, however, one of the drawbacks is maintenance. The following problems where experienced, but in all cases the errors were traced back to hardware limits or faulty installation, not faults in the MongoDB software.

Insufficient free space
: The MongoDB journal files require $3GB$ of free hard disk space, which may be a problem when trying to run from a USB stick. This can be solved by setting **smallfiles** = **true** in the file: `/etc/mongodb.conf`

couldn't connect to server
: An ambiguous error message whose root cause can be a number of things. The first step is commonly to remove the file: `/var/lib/mongod/mongod.lock`, this file is suposed to be deleted when mongodb stops. An impropper shutdown consequently leads to this file blocking access to the database. Afterwards the service can be started `sudo service mongodb start`, which should fix the problem.

4.2 Remotely hosted

Being capable of using a database that is hosted remotely is essential for employing the MoDeNa framework on servers or HPC clusters. The service **mlab** was used in the testing phase as the remote location for the database.

The service provides a `URI` to connect, e.g.
`mongodb://<dbuser>:<dbpassword>@ds011840.mlab.com:11840/<dbname>`.
Here one would simply put information such as: **username** (`<dbuser>`), **password** (`<dbpassword>`) and **database name** (`<dbname>`) directly into the `Modena_URI` string, and the MoDeNa software framework can use it directly.

The testing did not experience any problems with the remotely hosted database, other than it being significantly slower than the locally hosted.

5 Software framework

This section summarises the experiences learned through running the MoDeNa software framework on multiple platforms. The cases that where used in this part of the testing procedure where the standard example cases, which involves minimal code and no special dependencies other than the MoDeNa framework itself.

This ensures that the problems encountered are likely to arise due to the MoDeNa software framework and its dependencies, not third-party software.

5.1 Install procedure

The documented install procedure for the MoDeNa software framework is the following:

```
user@machine> cmake -DCMAKE_INSTALL_PREFIX:PATH=$HOME .
user@machine> make
user@machine> make install
```

The flag `-DCMAKE_INSTALL_PREFIX:PATH=$HOME` is specifying that the framework is to be installed in user space. However, this requires the dependencies of the software framework to be installed, during the testing phase the only problems encountered during the installation of the MoDeNa framework arose due to dependencies being out of date.

5.1.1 Compiling MoDeNa models

The MoDeNa models used in the examples are all built and compiled using cmake/make as follows:

```
user@machine> cmake .
user@machine> make
```

However, some older versions of cmake where having problems locating `FindMODENA.cmake`, which can be fixed in a number ow ways. The easiest way is to set the environment variable `MODENA_DIR=$HOME/lib/cmake`.

5.2 Application Program Interface

The application program interface (API) is used in the recipes that embeds surrogate models into applications, as described in deliverable Rusche and Karolius (5 13). One of the primary functionality of the API is that it must be capable of handling over control of the multiscale simulation when a surrogate model is out of bounds or uninitialised and the error check that has been implemented by the developer of the model is triggered.

As a consequence, testing the API will also reveal faults in the high-level python library that interfaces the MoDeNa framework to both rhe MongoDB database, as well as the workflow orchestrator (FireWorks).

The testing found that there is a problem in the substitution of surrogate models, specifically in the error handling of uninitialised substituted models. The error caused the framework to crash if the surrogate model did not already exist in the database. This has now been fixed, and the substituted models are now instantiated iteratively.

5.3 Workflow Management

Since the MoDeNa framework is necessary to set the "pythonpath", i.e. the environment variable `PYTHONPATH=$HOME/lib/python2.7/site_packages`. This is to ensure that Python can locate the MoDeNa module in the file tree.

By testing the workflow management, i.e. the orchestrator, we are testing that the parameter estimation framework is working. The parameter estimation framework should be capable of handling the following states of a surrogate model:

1. Initialise
2. Expand validated domain
3. Improve parameter accuracy

The parameter estimation framework has been implemented in python, and uses FireWorks in order to stage computational jobs and orchestrate the workflow. The testing verifies that the MoDeNa framework is capable of handling:

1. Automatic initialisation of surrogate models
2. On-line parameter estimation expanding the validated domain of a surrogate modelk
3. Parmeter validation and improvment

5.3.1 Initialisation

One of the new additions to the MoDeNa framework is that surrogate models does not have to be instantiated in a separate step. Previously, the multiscale simulation was a two-step procedure, first instantiating the surrogate models and then running the multiscale application. Now, however, the MoDeNa framework will now atomatically locate and initialise the uninitialised surrogate models, provided that the module is located in the directory `MoDeNaModels`.

The testing procedure found that the automatic initialisation works as intended, also for so-called "substituted models". However, it is relying on the `_id` of the surrogate model being the same as the name of the python module, i.e. the directory in which the surrogate model is defined.

However, the automatic initialisation procedure will be edited in future versions of the software framework, removing the somewhat illogical requirement that the module name and model "id" must be the same.

As mentioned in the API section there was a problem instantiating substituted surrogate models that where not already defined in the database, but this has now been fixed. One conclusion from the testing phase is that the automatic initialisation procedure should be improved for the purpose of removing the two-stage strategy for performing the multiscale simulation. This would simplify the user experience to two rules:

1. The python module should be located in the MoDeNaModels directory
2. The `_id` of the surrogate model, defined in the Python recipe, is used in the API to fetch a surrogate model.

5.3.2 On-line parameter estimation

The on-line parameter estimation involves two components:

1. Out of bounds
2. Parameters not valid

The background for the automatic parameter estimation can be found in Franceschini and Macchietto (2008), and the parameter estimation framework itself is reported in Karolius (8 31) and Preisig et al. (2 31).
The testing found that the parameter estimation framework works as expected, albeit with a couple ov caviats.

1. Remotely hosted databases slows down the parameter estimation framework, especially the execution of individual detailed simulations
2. Heavy Computer systems sometimes have limits to how many jobs that one user can submit into the queue

It was expected that the communication would slow down the execution of individual simulations, however, when a MD simulation requires one week to run a few extra seconds due to database hosting is neglible. However, the queue limitation is a annoyance, and the software framework should be modified such that it runs on a laptop and only performs individual detailed simulations on HPC clusters. This will require a new ModenaFireTask with some more infrastructure around it.

6 Polyurethane

The polyurethane multiscale model contains several applications that models different multiscale phenomena. Both of the applications that where tested have simple shell scripts for compiling the detailed applications that are used. Moreover, the procedures for performing the multiscale simulation may be somewhat convoluted, involving several steps of initialising surrogate models and running workflows before starting the desired application. Both of these aspects can be addressed in the software framework.

The following sections summarises the experiences learned by running the two multiscale applications. However, the section assumes that the dependencies for all of the applications have already been installed.

6.1 Foam Expansion

The foam expansion application models the evolution of the polyurethane foam in a geometry. The polymer monomers and blowing agent is mixed at the start of the simulation and the polymerization reaction generates bubbles that expands the volume of the system. The following models where used in the application:

- 0D CFD, Karimi and Marchisio (2015)
- Kinetics, Daiss et al. (2 25)
- Bubble growth, Kosek et al. (3 31)
- Rheology, Anderson et al. (4 11)
- Surface Tension, Mairhofer and Gross (8 31)
- Polymer Viscosity

The testing found that the application runs as expected, but that the initialisation procedure can be difficult to follow for someone who are not familiar with the models. The procedure works, but it may be addressed from a software perspective performing a system identification of the bubble growth model. This has not yet been addressed and will require a require a new surrogate model to be defined.

6.2 Foam Ageing

The foam aging application models how the thermal properties of the finished foam degrades over time. The models that are used in the foam aging application are the following:

- Foam Aging
- Foam Conductivity
- Solubility

The testing revealed no problems with the foam aging application, it works as expected on all platforms.

7 Discussion

The testing phase has verified that the software framework indeed works on different platforms an operating systems. In this context it was found that the software versions that are available centralised systems seems to be lagging. However, with the release of the next long time supported (LTS) version of Ubuntu Linux (16.04) it is expected that this problem will disappear as system administrators migrate to the newer operating systems.

Generally, the most problems during the test phase was related to installing the correct version of the dependencies, but there was some environment variables that should be summarised.

PYTHONPATH
: This is a Python-specific environment variable telling python where to look for modules. It is necessary because the install procedure for the MoDeNa framework currently explains how to install the software framework in "user space".

MODENA_DIR
: The environment variable is used by cmake to fine the file `FindMODENA.cmake`. This was only required to be set when using older versions of cmake (≤ 2.7)

Modena_URI
: The Modena_URI environment variable is used in order to specify where the MongoDB database is hosted. The framework will automatically look for a mongodb daemon running on localhost, therefore this environment variable is only necessary to specify when the user wants to use a externally hosted database.

The environment variables and dependency versions is the most useful information to convey to users after the testing procedure. The results from running the software framework itself, as well as the multiscale application did not reveal any large problem, but some possible improvements that should be considered.

The application program interface (API) testing revealed a problem with the automatic instantiation of substituted surrogate models. Specifically, the surrogate model had to be present in the database for the connection to work.

Another conclusion from the testing is that the two-stage process (initialisation and execution) that is required in order to run a multiscale application should be reduced to one. This will mean to ensure that any model that is stored in the `MoDeNaModels` directory can be used without running a initialisation procedure. This will require removing the current rule that the surrogate model `_id`, which is specified in the model definition, must be the same as the name of the python modules.

Ultimately, the testing did not reveal any serious problems and the software framework is ready to be deployed on Linux distributions. Extending the support to OS-x will most likely require some changes to the low-level library, as the compiler used in the testing is `gcc`, and not `clang`. However, earlier versions have been tested on OS-x.

The conclusion from the testing is that the software framework and its capabilities works as designed. Potential pitfalls that may be encountered when installing the software and its dependencies has been identified, the abilities have been tested and the multiscale application has been verified to be able to interlink multiple scales into a large multiscale application.

References

Anderson, P., Hulsen, M., and Mitrias, C. (2015-04-11). Modena - deliverable d3.2: Simulations for foams, dispersion and mixing and developed sw. Technical report, FP7 MoDeNa. 6.1

Daiss, A., Deglmann, P., and Settels, V. (2015-02-25). Modena - deliverable d1.5: Comprehensive kinetics model. Technical report, FP7 MoDeNa. 6.1

Franceschini, G. and Macchietto, S. (2008). Model-based design of experiments for parameter precision: State of the art. *Chemical Engineering Science*, 63(19):4846 – 4872. Model-Based Experimental Analysis. 5.3.2

Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.-M., Hautier, G., Gunter, D., and Persson, K. A. (2015). Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a. 1

Karimi, M. and Marchisio, D. L. (2015). A baseline model for the simulation of polyurethane foams via the population balance equation. *Macromolecular Theory and Simulations*, 24(4):291–300. 6.1

Karolius, S. (2015-08-31). Modena - deliverable d4.3: Backward mapping tool. Technical report, FP7 MoDeNa. 5.3.2

Karolius, S., Birgen, C., Preisig, H. A., and Rusche, H. (2015-08-31). Modena - deliverable d4.2: Model and data containers. Technical report, FP7 MoDeNa. 4

Kosek, J., Ferkl, P., Anderson, P., and Mitrias, C. (2014-03-31). Modena - deliverable d2.2: Dispersion: Sw tool for the initial dispersion of reactants and bubble initiation. Technical report, FP7 MoDeNa. 6.1

Mairhofer, J. and Gross, J. (2015-08-31). Modena - deliverable d1.3: Thermodynamic models. Technical report, FP7 MoDeNa. 6.1

Preisig, H. A., Thombre, M., and Karolius, S. (2014-12-31). Modena - deliverable d5.3: Parameter estimation framework. Technical report, FP7 MoDeNa. 5.3.2

Rusche, H. and Karolius, S. (2015-05-13). Modena - deliverable d5.8: Production release - recipes and adapters. Technical report, FP7 MoDeNa. 5.2