dune-testtools: Modelling dynamic and static variations of system tests through ini files

Dominic Kempf, Timo Koch

Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, Heidelberg,
Institut für Wasser- und Umweltsystemmodellierung, Stuttgart

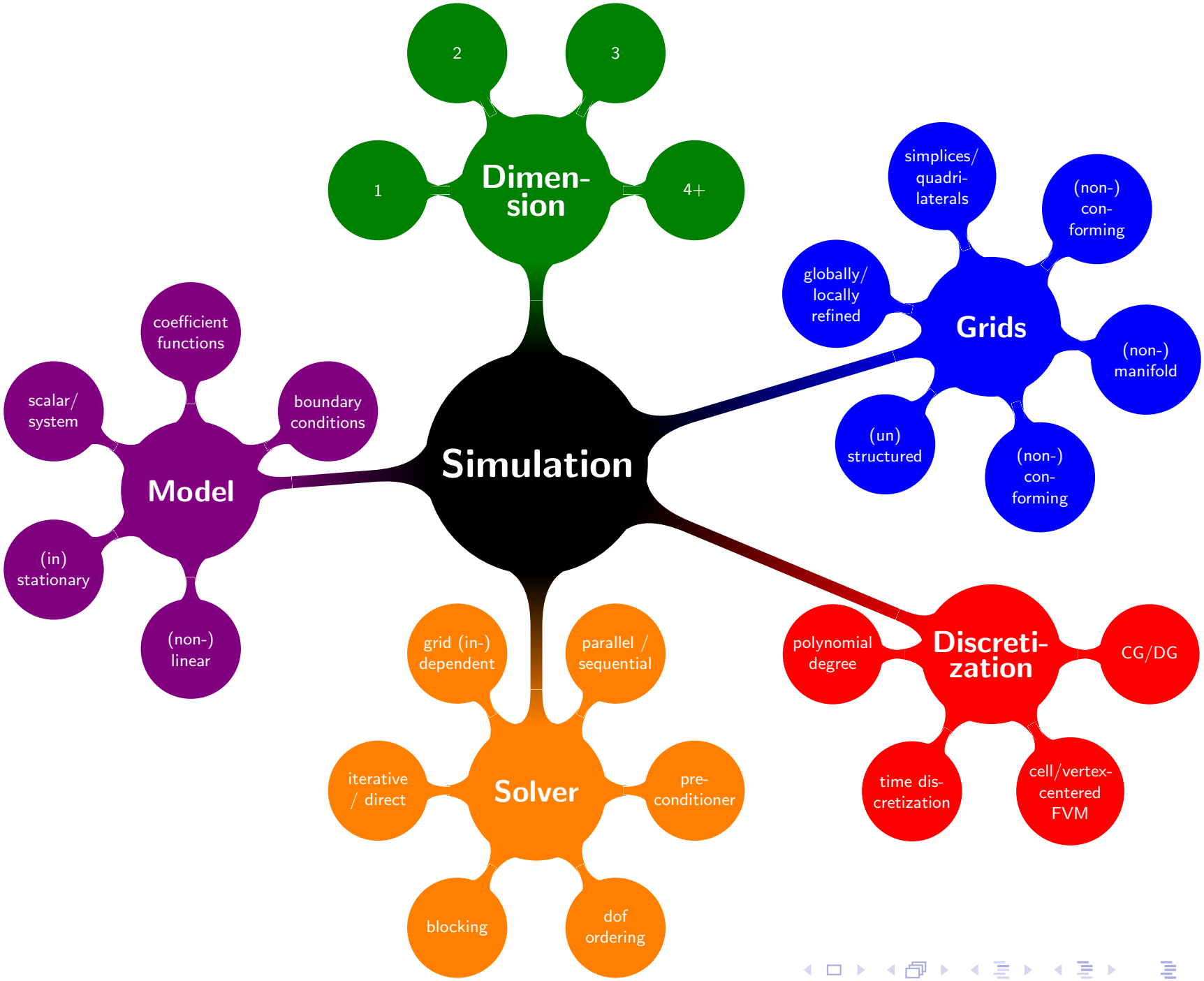
28. September 2015

The Dune testing suite is not extensive enough and currently limited to unit testing.

The `dune-pdelab` and `DuMux` teams are working on a joint project to add system testing tools. The module `dune-testtools` provides the "local" infrastructure for system testing.

Goals of `dune-testtools`:

- Provide tools to describe system tests.
- Integrate those tools into the Dune buildsystem.
- Develop testing tools specific to numerical software.
- Staying as close as possible to the user workflow.


Simulation
Dimen-
sion
1
2
3
4+
Model
coefficient
functions
scalar/
system
boundary
conditions
(in)
stationary
(non-)
linear
Grids
simplices/
quadri-
laterals
(non-)
con-
forming
globally/
locally
refined
(non-)
manifold
(un)
structured
(non-)
con-
forming
Solver
grid (in-)
dependent
parallel /
sequential
iterative
/ direct
pre-
conditioner
blocking
dof
ordering
Discreti-
zation
polynomial
degree
CG/DG
time dis-
cretization
cell/vertex-
centered
FVM

What does a system test consist of?

A system test application consists of:

- One single `.cc` source file
- One `.ini` file
- Possibly some header files
- A single line of `CMake` code
- Maybe some reference data (see later)

What does a system test consist of?

A system test application consists of:

- One single `.cc` source file
- One `.ini` file
- Possibly some header files
- A single line of `CMake` code
- Maybe some reference data (see later)

The `.ini` file is used to model the variants in the system test.

What does a system test consist of?

A system test application consists of:

- One single `.cc` source file
- One `.ini` file
- Possibly some header files
- A single line of `CMake` code
- Maybe some reference data (see later)

The `.ini` file is used to model the variants in the system test.

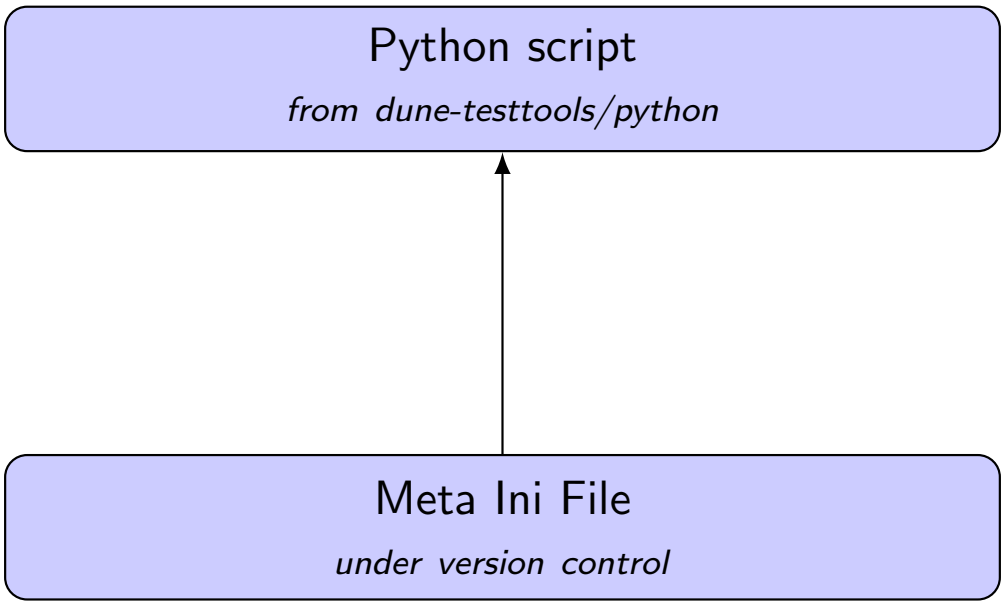
How is this done?

Workflow of the expansion process

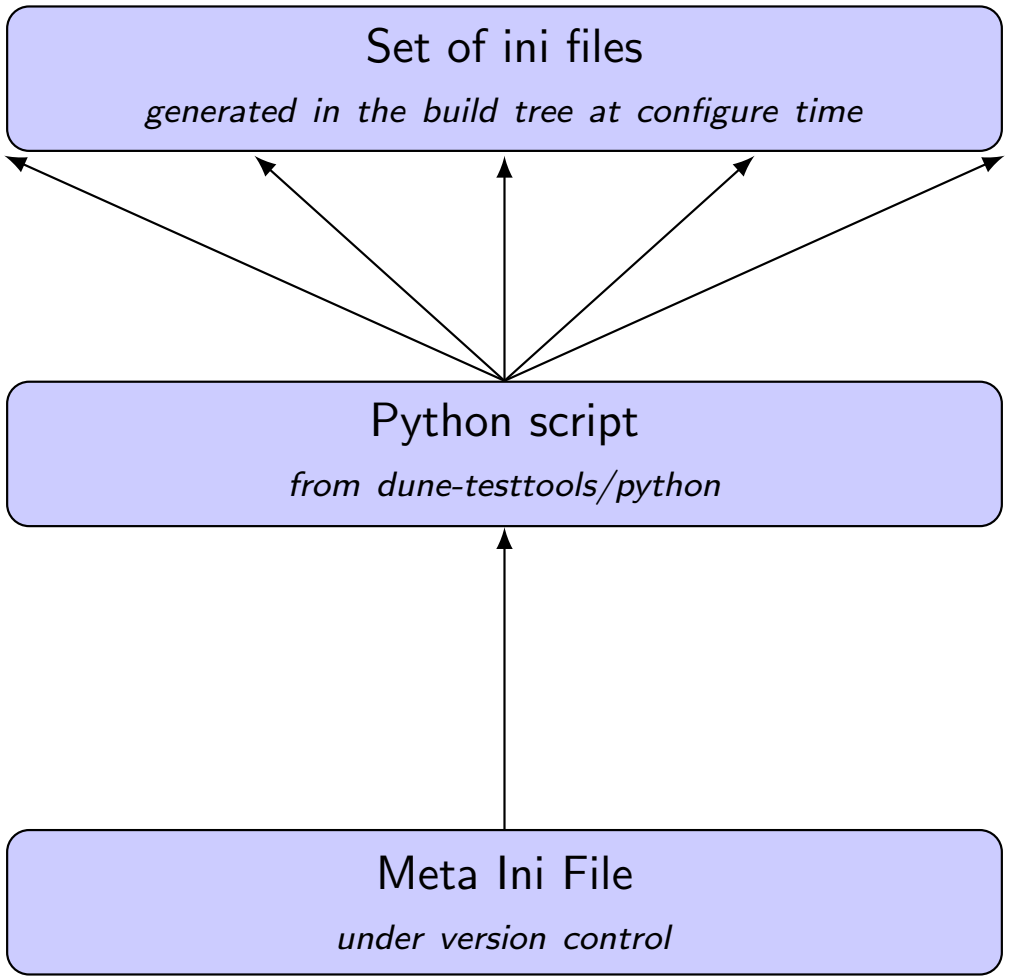
Meta Ini File

under version control

Workflow of the expansion process



Workflow of the expansion process



Introducing command syntax

$$< key > \equiv < value > \{ | < command > \{ < args > \}^* \}^*$$

The most important command is the `expand` command:

```
timestep = 0.5, 1 | expand
level = 1, 2, 3 | expand
```

Introducing command syntax

$$< key > \equiv < value > \{ | < command > \{ < args > \}^* \}^*$$

The most important command is the `expand` command:

```
timestep = 0.5, 1 | expand
level = 1, 2, 3 | expand
```

It is also possible to connect expansion processes:

```
timestep = 0.5, 1 | expand refinement
level = 1, 2 | expand refinement
```

Key-dependent values

```
name = file1 , file2 | expand
file = {name}.ext
```

Key-dependent values

```
name = file1 , file2 | expand
file = {name}.ext
```

```
lin = linear, nonlinear | expand
stat = stationary, instationary | expand

parameter = {{lin}_{stat}}

linear_stationary = 1.0
nonlinear_stationary = 2.0
linear_instationary = 3.0
nonlinear_instationary = 1.0
```

Other commands in meta ini files

There are some more commands:

- `unique` makes keys unique by appending consecutive numbering.
- `include` or `import` allow combining ini files.
- `tolower`, `toupper`, `eval` allow simple value modification
- Some testtools are controlled through specific commands

Additionally, there are some special keys that are recognized by the expansion process or testtools. All those start with two underscores.

- `__name` defines the filename. The `unique` command is automatically applied to this key.

Excluding some variants from the product set

Often, the set of possible configurations cannot be expressed as a simple product set, but some more sophisticated constraints are to be applied. This can be done in the same

```
grid = yasp, ug | expand
degree = 1, 2 | expand

{grid} == ug and {degree} == 2 | exclude
```

- The exclude command does not operate on a key/value pair!
- After resolving curly brackets, the value is interpreted as a boolean python expression
- All configurations, where it eavulates to `True` are discarded.

Build system integration / Static variations

The `__static` section in a meta ini file is a special section dedicated to static variations

Build system integration / Static variations

The `__static` section in a meta ini file is a special section dedicated to static variations

- A meta ini file generating static variations

```
yasp = Dune::YaspGrid<2>
foam = Dune::FoamGrid<2,3>
[__static]
GRIDTYPE = {yasp}, {foam} | expand
```

Build system integration / Static variations

The `__static` section in a meta ini file is a special section dedicated to static variations

- A meta ini file generating static variations

```
yasp = Dune::YaspGrid<2>
foam = Dune::FoamGrid<2,3>
[__static]
GRIDTYPE = {yasp}, {foam} | expand
```

- The C++ code

```
typedef GRIDTYPE Grid;
//...
```

Build system integration / Static variations

The `__static` section in a meta ini file is a special section dedicated to static variations

- A meta ini file generating static variations

```
yasp = Dune::YaspGrid<2>
foam = Dune::FoamGrid<2,3>
[__static]
GRIDTYPE = {yasp}, {foam} | expand
```

- The C++ code

```
typedef GRIDTYPE Grid;
//...
```

- How does the CMakeLists.txt look like?

Build system integration / The CMake macros

- One macro to rule them all: static & dynamic variations

```
add_dune_system_test(SOURCE src1
                     BASENAME base
                     INIFILE ini
                     TARGETS output
                     [SCRIPT script])
```

Build system integration / The CMake macros

- One macro to rule them all: static & dynamic variations

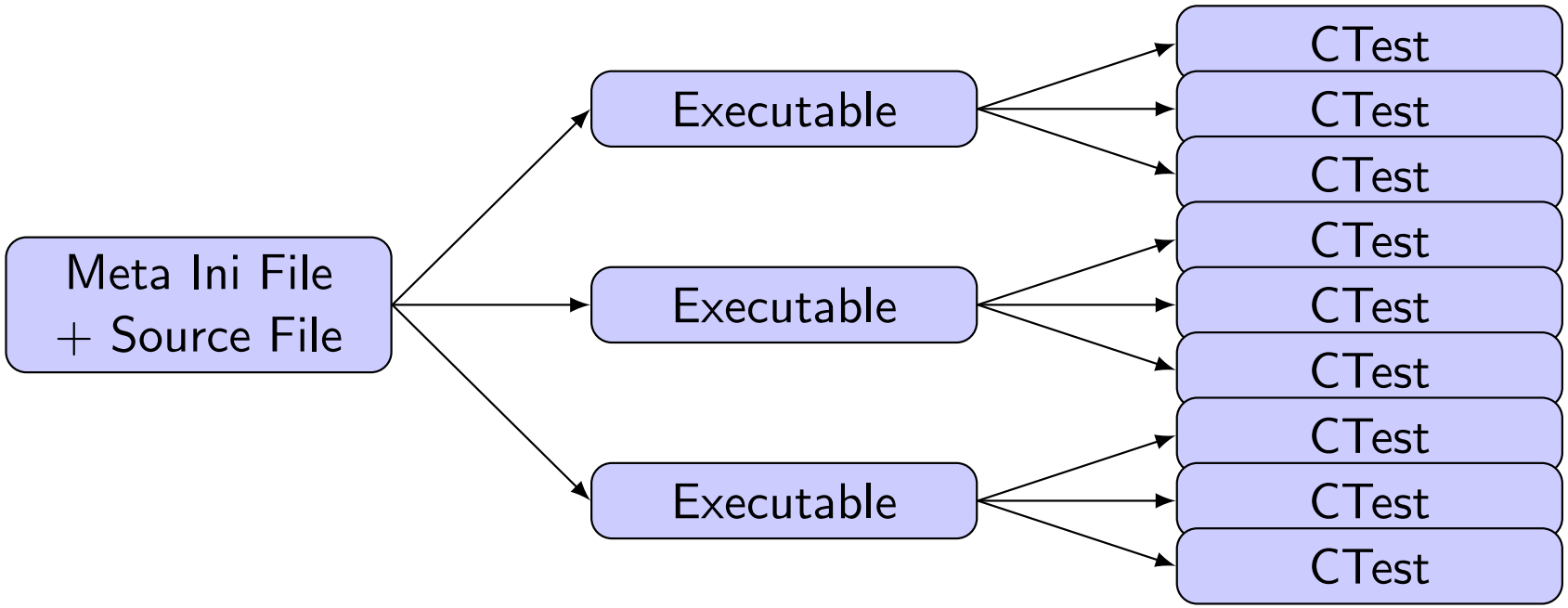
```
add_dune_system_test(SOURCE src1
                     BASENAME base
                     INIFILE ini
                     TARGETS output
                     [SCRIPT script])
```

Meta Ini File
+ Source File

Build system integration / The CMake macros

- One macro to rule them all: static & dynamic variations

```
add_dune_system_test(SOURCE src1
                     BASENAME base
                     INIFILE ini
                     TARGETS output
                     [SCRIPT script])
```



`dune-testtools` is available on:

`http://conan2.iwr.uni-heidelberg.de/git/quality/dune-testtools` It depends on the build system extension module `dune-python` available from

`http://conan2.iwr.uni-heidelberg.de/git/quality/dune-python`
Documentation is also available!

If you have any questions, just contact:

- dominic.kempf@iwr.uni-heidelberg.de
- timo.koch@iws.uni-stuttgart.de