TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
The University of Resources. Since 1765.

MonolithFE^2— A monolithic FE^2 implementation for Abaqus

Version 1.02a

Nils Lange, Geralf Hütter, Björn Kiefer

July 23, 2021

Contents

1 Description

MonolithFE^{2}is an open-source program for Abaqus, distributed under a CC BY-NC-SA 4.0 license. Details on the implemented theory can be found in [1]. Please refer to this publication if you use MonolithFE^2.

The main idea behind the present FE^2 implementation is to use Abaqus Standard for solving the macro FE problem and a self-written light-weight code for solving the micro problems. The micro-macro data exchange is performed through the Abaqus UMAT interface. MonolithFE^2 employs the UEL interface of Abaqus at the micro-scale. An UEL is shipped with MonolithFE^2. This UEL comprise a certain number of established element types and employs the UMAT interface for the material law at the micro-scale. Thus, previously developed UELs and UMATs can be employed directly at the micro-scale and be tested in Abaqus directly independent of MonolithFE^2. By default, an elastic-plastic MISES material routine UMAT in rate formulation is employed at the microscale. The preprocessing for the micro-scale is done in Abaqus/CAE with aid of a Python plug-in. The postprocessing of selected microscopic FE problems can be done by a re-simulation in Abaqus with the actual deformation history of a macroscopic integration point.

1.1 Features

- monolithic and staggered algorithm
- periodic boundary conditions at micro-scale
- small deformation and large deformation theory
- UMAT interface to Abaqus at macro-scale
- Intel MKL PARDISO solver on the microscale
    - for symmetric and unsymmetric (but structurally symmetric) matrices
    - parallelizable: Intel MKL PARDISO solver can be run in parallel for each macroscopic integration point (limited to a shared memory architecture and not necessarily favorable, since the FE^2-method parallizes very well)
- modular concept: UEL, UMAT and UHARD interfaces at micro-scale for easy extensibility
- implemented element types (via UELlib [2]):
    - plane stress (nested Newton algorith after Dodds, 1987), plane strain and 3D
    - different element types (quadrilateral, triangular, tetrahedral, hexagonal)
    - linear and quadratic shape functions with full or reduced integration[1]
    - (rate-independent) elastic-plastic MISES material in rate formulation as micro UMAT with tabular yield curve (as UHARD)
- Python plug-ins for preprocessing and postprocessing of micro-scale models in Abaqus/-CAE (meshing, material assignment, convergence parameters etc.)
- 3D Models at the microscale with plane strain/axisymmetric elements at macro-scale

---

[1] In contrast to Abaqus, the fully integrated elements use the same quadrature rule for all terms, i.e., no selective reduced integration.

- parallelizability: Abaqus can be run in parallel (also over multiple computer nodes)
- binaries for Linux contained

1.2 Requirements

- Abaqus/Standard version 2017 or later
- Intel Fortran version 2017 or later (moved to Intel OneAPI HPC Toolkit) with correctly set environment variables
- Intel MKL version 2017 or later (in Intel OneAPI Base Toolkit) with correctly set environment variables
- only under Windows[2]: Microsoft Visual Studio Community or Enterprise (for the linker)

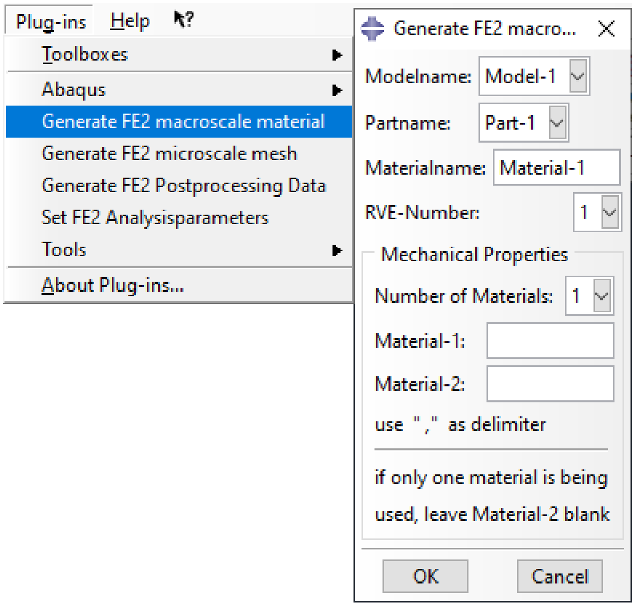
The program has been developed and tested extensively with Abaqus 2020 and Intel Fortran 2020 under Debian Linux 9 and Intel OneAPI 2021 under Windows 10.

2 Quick Start Guide

0. Install the Python plugins by copying the folder `abaqus_plugins` to either the current directory, home directory or Abaqus installation directory. In the next start of Abaqus/CAE these plugins are available.
    - only MS Windows: Copy the files `abaqus_6.env` and `win86_64.env` from the subfolder `src` to one of the working/home/installation directory to have the linker options correctly set (as described in detail in section 6.2).
1. Generate micro-scale model(s) `*.FE#` (mesh and boundary conditions) as described in section 4.1, or take existing file from `examples/` (section 3) whereby `#` is the RVE-Number (a label between 1 and 5) and * is the job name of the macro problem (→ e.g. "beam_model.FE1").
2. Generate a macro-scale model for Abaqus, e.g. with Abaqus/CAE, as usual and assign a user-defined material to the FE^2 regions. When using Abaqus/CAE this can be done using the plugin "Create_FE2_Material":

---

[2] see Intel Forum for detailed instructions on installation


Plug-ins
Help
Toolboxes
Abaqus
Generate FE2 macroscale material
Generate FE2 microscale mesh
Generate FE2 Postprocessing Data
Set FE2 Analysisparameters
Tools
About Plug-ins...
Generate FE2 macro...
OK
Cancel

Modelname: Model-1
Partname: Part-1
Materialname: Material-1
RVE-Number: 1
Mechanical Properties
Number of Materials: 1
Material-1:
Material-2:
use "," as delimiter
if only one material is being used, leave Material-2 blank

Therein, the RVE number refers to # of step 1. The user material can also be specified directly in the `.inp` file:

```
*User Material, constants=%NPROPS
 %PROP(1),  %PROP(2),  %PROP(3),  ...
 %PROP(9), %PROP(10),...
  ...
 *Depvar
 9
```

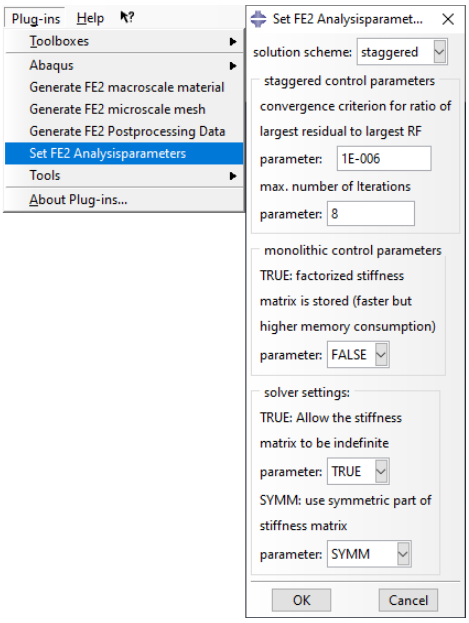
Therein, `%PROP(1)` is the number # to be used, whose constitutive parameters are defined by the following entries. At each microscopic integration point the same UMAT is called, but 2 sets of material routine parameters can be specified. Each micro element gets a label indicating whether set 1 or set 2 is used, c.f. section 4.1 for details. Then, `%PROP(2)` is the integer number of constitutive parameters of set 1 and `%PROP(3)` is the number of constitutive parameters of set 2. The subsequent `%PROP` parameters form a sequence of lists for the constitutive parameters at the microscopic scale which is passed to the microscopic material routine. `%PROP(4:4+%PROP(2))` are the constitutive parameters of set 1 and `% PROP(5+%PROP(2):%NPROPS)` are the constitutive parameters of set 2. `%NPROPS` is equal to 3+`%PROP(2)+%PROP(3)`. For instance, the definition

```
*User Material, constants=11
 1.,  6.,  2.,60., 0.3, 0.6,  0.,  2.
 1.5, 600, 0.3
*Depvar
 9
```

has the following meaning in combination with the elastic-plastic micro-material routine: The mesh number 1 is being used. Set 1 has 6 constitutive parameters, which are: $E = 60.0$ MPa, $\nu = 0.3$, $Y_1 = 0.6$ MPa, $\varepsilon_{\mathrm{pl}\,1} = 0.0$, $Y_2 = 2.0$ MPa and $\varepsilon_{\mathrm{pl}\,2} = 1.5$. Set 2 has 2 constitutive parameters, which are: $E = 600.0$ MPa and $\nu = 0.3$. If another UMAT then the default rate independent Mises plasticity is used, the constitutive parameters are interpreted by the used UMAT in the order the UMAT expects the properties to be.

Note: Specifying `*Depvar` isn't mandatory since the program only saves the deformation history (strain resp. displacement gradient) in the solution dependent variables (SDV) for postprocessing, since in large deformation simulations the displacement gradient isn't accessible in the Abaqus Viewer. When the displacement gradient is needed for postprocessing it's necessary to request SDVs in the "Field Output Request".

3. Analysis parameters can be set by the plugin "Set FE2 Analysisparameters". This plugin creates a configuration file `FE2_Analysisparameters.cfg`, to be placed in the directory from which the Abaqus job is started. If no file is supplied, the default parameters will be used: monolithic algorithm without storing the factorized stiffness matrix, symmetric stiffnessmatrix.


Plug-ins
Help
Toolboxes
Abaqus
Generate FE2 macroscale material
Generate FE2 microscale mesh
Generate FE2 Postprocessing Data
Set FE2 Analysisparameters
Tools
About Plug-ins...
Set FE2 Analysisparamet...
OK
Cancel

solution scheme: staggered

staggered control parameters

convergence criterion for ratio of largest residual to largest RF

parameter: 1E-006

max. number of Iterations

parameter: 8

monolithic control parameters

TRUE: factorized stiffness matrix is stored (faster but higher memory consumption)

parameter: FALSE

solver settings:

TRUE: Allow the stiffness matrix to be indefinite

parameter: TRUE

SYMM: use symmetric part of stiffness matrix

parameter: SYMM

4. Run FE^2 simulation by command

```
abaqus job=XXX user=PATHTODIR/bin/UMATmacro-std.o cpus=%n_cpus
```

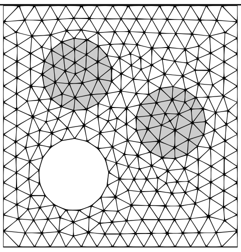
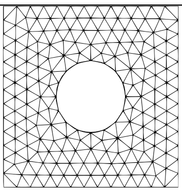
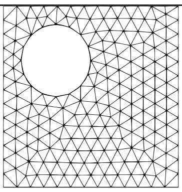
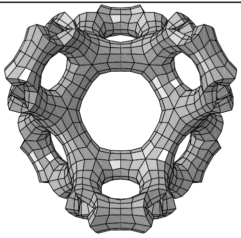
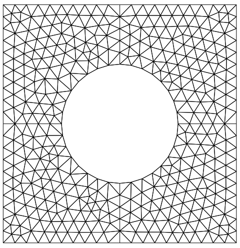
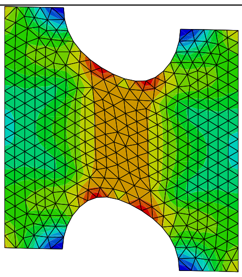
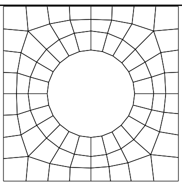
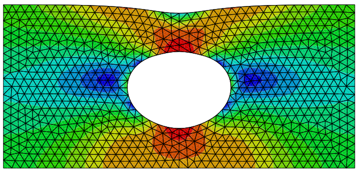
under Linux or

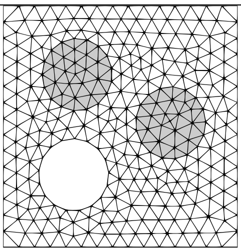
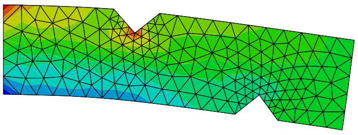
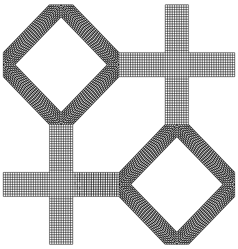
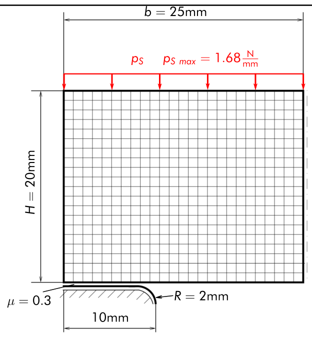
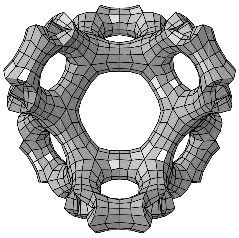
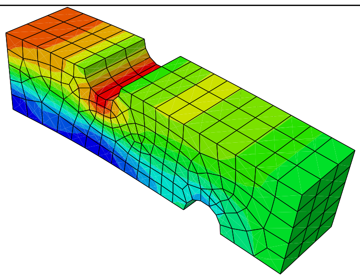
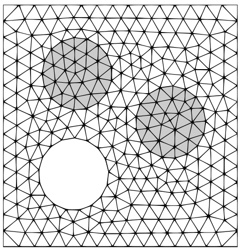
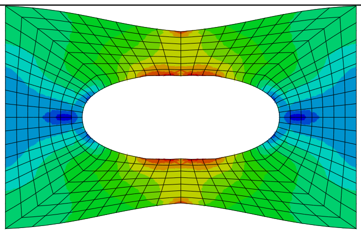
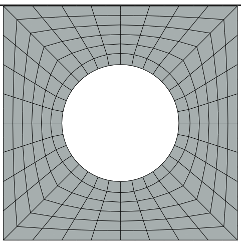
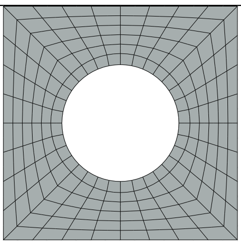
```
abaqus job=XXX user=PATHTODIR/bin/UMATmacro-std.obj cpus=%n_cpus
```

under Windows or create Job in Abaqus/CAE and choose `UMATmacro-std.o` or `UMATmacro-std.obj`, respectively, as User subroutine file. (tested with Abaqus versions 2017, 2018 and 2020)

3 Examples

Name and reference	micro	macro

homogeneous [3]	homogenous microstructure, 4 rectangular elements	pure shear, pure bending
bimaterial [3]	simple microstructure, 4 rectangular elements, two sets with different constitutive parameters,	pure shear, pure bending
Miehe_Koch_models composite [1, 3]		-
Miehe_Koch_models porous_microstructure_central [1, 3]		-
Miehe_Koch_models porous_microstructure_excentric [3]		-
open_pored_3D_foam [3]		-
notched_plate_shear		
plate_with_hole_tension [3]		

sharp_notched_beam_2D [1, 3]		
2D_foam_filter [4]		 $b = 25\text{mm}$ p_S $p_{S\,max} = 1.68\,\frac{\text{N}}{\text{mm}}$ $H = 20\text{mm}$ $\mu = 0.3$ $R = 2\text{mm}$ 10mm
3D_beam_3D_foam_power_law_hardening [1, 3]		
quater_plate_hole_tension		 quarter model at large deformations
porous		single element under uniaxial tension with strain relief at 6 points at large deformations
porous_cyclic		single element under uniaxial cyclicloading at large deformations

Selected examples can be run automatically (in Linux) by the command

```
make verify
```

Further parameters like `NCPUS=...` or `ABACALL=...` can be passed, cf. section 6.2.

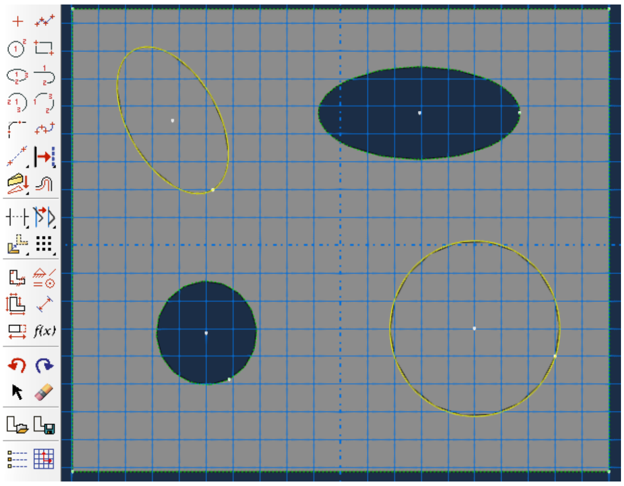
4 Creating Microscale Models

General requirements:

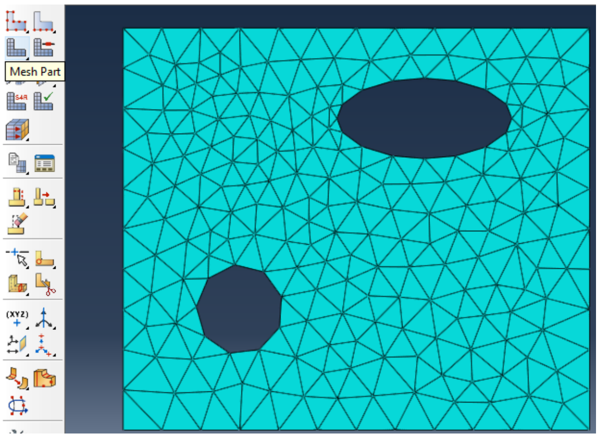
- same element type in complete micro-scale model
- congruent meshes at homologous parts of the boundary

4.1 Within Abaqus/CAE using Plug-In for 2D models

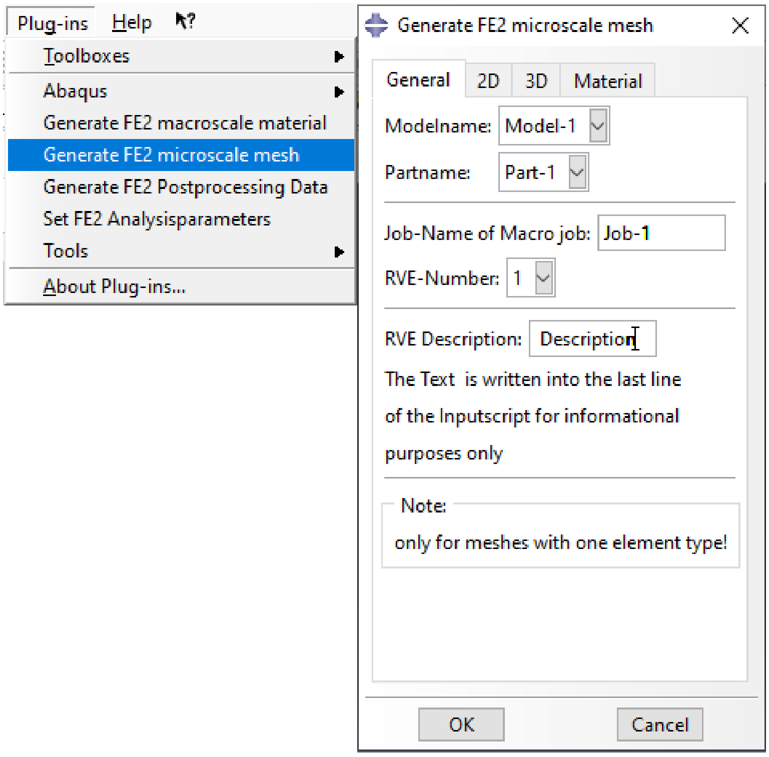
- Create a 2D Planar Part with rectangular outer shape. It can include any kind of pores and may be partioned.



- Mesh the Part (not its Instance!) with either quadrilaterail or triangular elements (Only meshes with exactly one element type are allowed!). Ensure that the nodes at opposite boundary faces are congruent.



- If different material properties shall be assigned to a certain region of the micro-structure, create a respective Set "2".
- Open the Plugin „generate FE2 microscale mesh“. Specify the created Model and Part. „Job-Name of Macro job“ is the Job name of the macro problem to be solved. A short description may be added.


Plug-ins
Help
Toolboxes
Abaqus
Generate FE2 macroscale material
Generate FE2 microscale mesh
Generate FE2 Postprocessing Data
Set FE2 Analysisparameters
Tools
About Plug-ins...
Generate FE2 microscale mesh
General
2D
3D
Material
OK
Cancel

Modelname: Model-1

Partname: Part-1

Job-Name of Macro job: Job-1

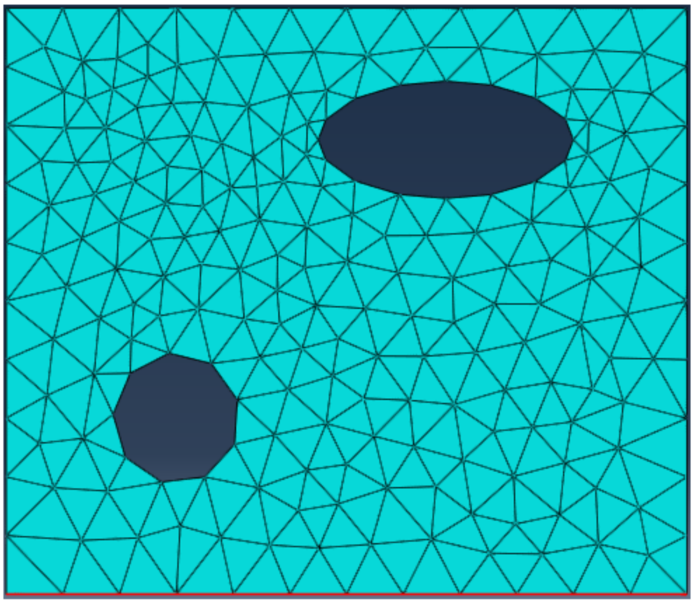
RVE-Number: 1

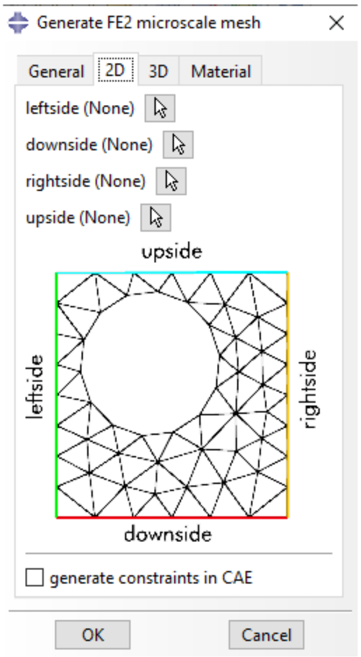
RVE Description: Description

The Text is written into the last line of the Inputscript for informational purposes only

Note: only for meshes with one element type!

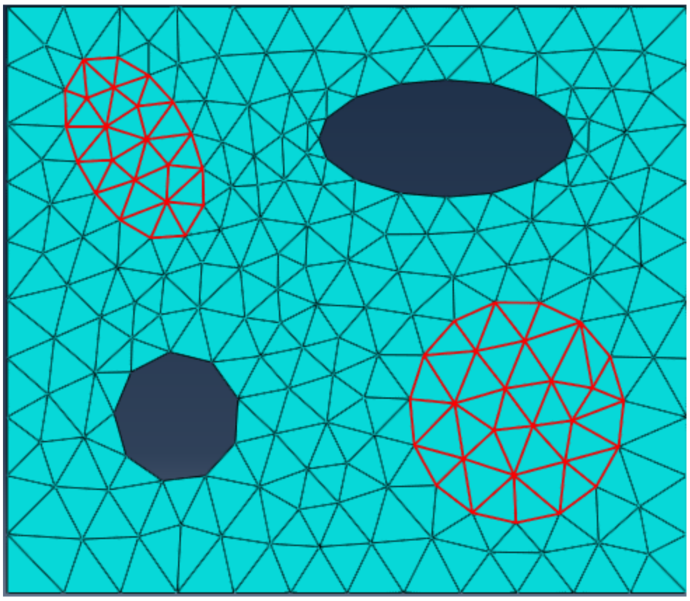
- In the tab „2D“ Tab first click on the mouse courser button and then select the respective nodes of the edges of the created RVE (use rectangular box selection by holding the left mouse button pressed, to not miss a node!). Confirm each time with „Done“.

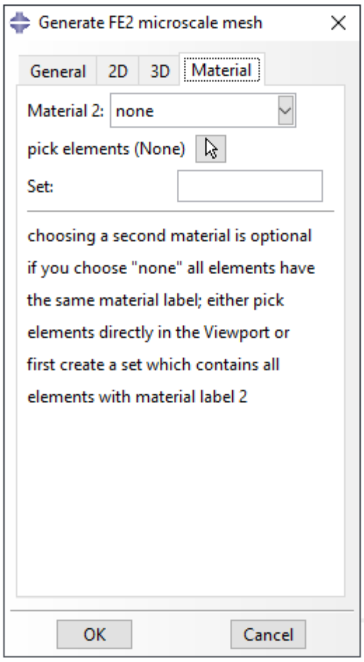




- If some micro problems are to be postprocessed after the FE^2 simulation, tick the box „generate constraints in CAE“.
- If 2 different materials shall be used, continue with the dialogue „Material“. Either select

all elements with Material label 2 ("pick elements") or specify a Set that contains these elements. All other elements get material label 1.





Material 2: none

pick elements (None)

Set:

choosing a second material is optional if you choose "none" all elements have the same material label; either pick elements directly in the Viewport or first create a set which contains all elements with material label 2

- Up to 5 different RVE meshes can be used in one FE^2 Analysis. When creating more then one RVE input file for a macro FE Analysis, make sure to choose different RVE numbers in the General Settings of the plugin. Clicking "OK" generates a file `*.FE#` in the current working directory, where # is the RVE-Number and $*$ is the macro job name.

4.2 For 3D models with existing equations for periodic boundary conditions

4.2.1 Micromechanics Plugin

Following steps are necessary for the usage of the "Micromechanics Plugin" [5]:

1. Modelling the microstructure and generating a mesh. Note that the "Micromechanics Plugin" requires the Part to be in the Assembly and the Regions to have a Material assigned via a Section. Alternatively a model can be generated using the "Micromechanics Plugin" (FE-RVE →Library). Ensure that the nodes at opposite boundary faces are congruent. Note that this is not necessarily true for all examples of the "Micromechanics Plugin"!

2. Create a Job and give it exactly(!) the same name of the model created before.

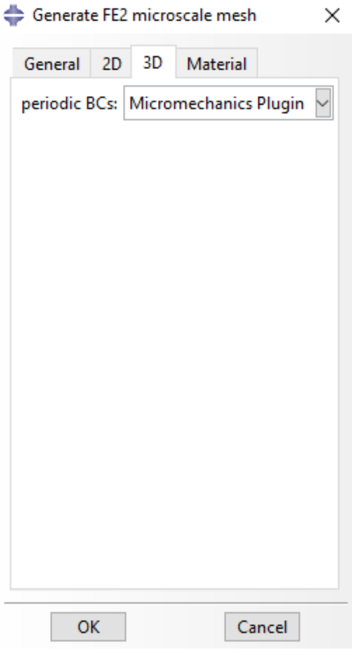
3. Create the periodic boundary conditions in the "Micromechanics Plugin" under FE-RVE →Loading by choosing the correct model and job and confirm with "OK".

4. Create the Abaqus-Inputscript by right-clicking on the Job and then on "Write Input".

5. Now start the "Generate-FE^2-Inputfile" Plugin and fill out the "General" dialogue.

6. In the "3D" dialogue choose "Micromechanics Plugin".

7. Optionally fill out the "Material" dialogue, then confirm with "OK".



periodic BCs: Micromechanics Plugin

4.2.2 FoamGUI

When the periodic boundary conditions are created for 3D foams using the "FoamGUI" [6] code, then they are included in the Abaqus-Inputscript as:

```
 *Equation
 ...
 1
 master-label|1|-1|slave-label|1|1|reference-node|1|-1
 1
 master-label|2|-1|slave-label|2|1|reference-node|2|-1
...
```

Following steps are necessary to create a Inputfile for MonolithFEsqr:

1. Import the Inputfile by right-clicking on Models and choosing "Import...". The equations are now imported as constraints and have the names 'Eqn-1'-'Eqn-%n'
2. Now open the "Generate-FE^2-Inputfile" Plugin and fill out the "General" dialogue.
3. In the "3D" dialogue choose "FoamGUI".
4. Optionally fill out the "Material" dialogue, then confirm with "OK".

5 Postprocessing

The postprocessing for the macro problem can be done as usual in Abaqus/Viewer. For the micro problems no conventional postprocessing is available at the moment. Instead the load history of selected macroscopic integration points can be extracted and then resimulated directly in Abaqus. The process is described only for 2D models. Generally this procedure is also applicable for 3D models, but the created amplitudes must be assigned manually.

0. Before starting the FE^2 simulation select SDV as field output in the macro model. There the displacement gradient (resp. strain) will be saved. Without having the SDVs as field output no postprocessing can be done in large displacement analysis.

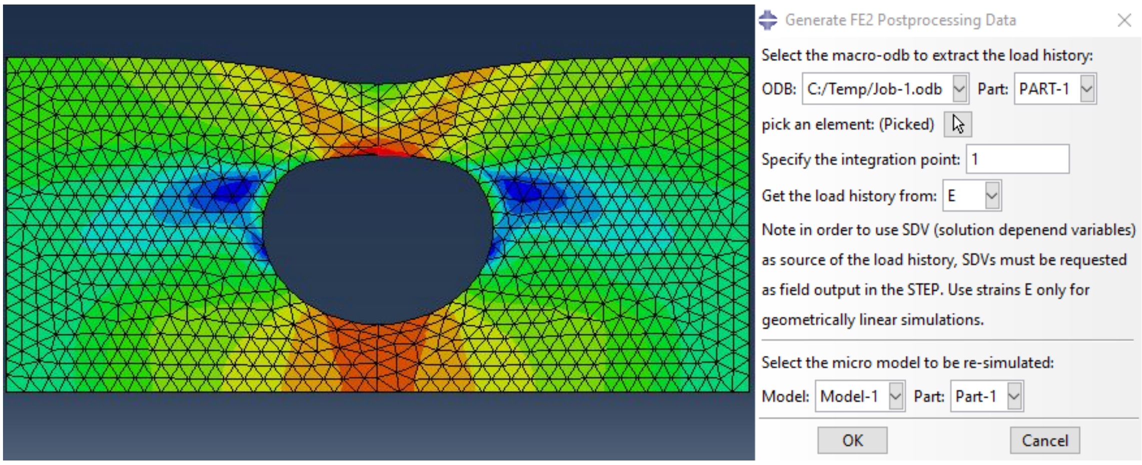
1. Open the created micro model. If the box „generate constraints in CAE“ in the „Generate FE2 microscale mesh“ Plugin as described in chapter 4.1 wasn't ticked in the first call, redo the process, now with ticked box. This creates constraints between nodes on opposite boundary's, boundary conditions, amplitudes and a step.

2. Modify the steps settings as needed.

3. Create a Material definition (or 2 depending on the model) (either a User Material [using the same micro UMAT as used in the FE^2 simulation], or a Abaqus intern Material model). Create a section (or 2) and assign them to the respective regions.

4. Now load the odb-file resulting from the FE^2 simulation.

5. Open the Plugin ”Generate FE2 Postprocessing Data“. Fill out the dialogue. The therein mentioned micro model is the model from point 1 and should still be open. Clicking on ”OK“ creates amplitudes for the strain resp. displacement gradient in the micro CAE model.



Generate FE2 Postprocessing Data

Select the macro-odb to extract the load history:

ODB: C:/Temp/Job-1.odb Part: PART-1

pick an element: (Picked)

Specify the integration point: 1

Get the load history from: E

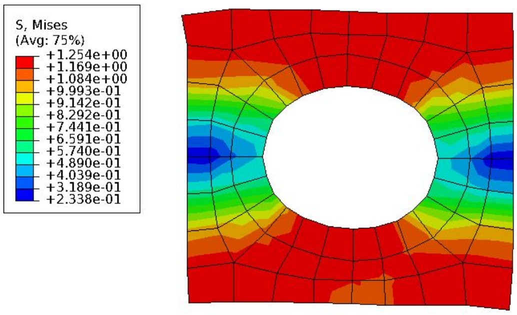
Note in order to use SDV (solution dependend variables) as source of the load history, SDVs must be requested as field output in the STEP. Use strains E only for geometrically linear simulations.

Select the micro model to be re-simulated:

Model: Model-1 Part: Part-1

6. Now create a job for the micro model and submit it.

7. Do the postprocessing as usually with the Abaqus/Viewer.



8. Repeat steps 5 to 7 for all macro integration points to be postprocessed.

6 Source Code and Compilation

6.1 Installation of the Abaqus Plugins

The folder `abaqus_plugins` has to be copied either to the current directory, home directory or Abaqus installation directory. In the next start of Abaqus/CAE these Plugins are available.

6.2 Compilation

The code is compiled under Linux by command

```
make ABACALL=abqXXX MATERIAL=...
```

or by

```
compile.bat
```

under Winodws. These commands create the object files `UMATmacro-std.o` and `UMATmacro-std.obj` which can be used as described in section 2.

Required additional packages:

- UELlib [2]
- UEL large deformation

The location of these packages is given in the `Makefile` and `compile.bat` and can be set by `make UELlibDIR=...` etc or modifying `compile.bat`. In the same way, the command to call Abaqus can be set by `make ABACALL=...`. Alternatively, the files can be made accessible either directly in folder `src/` (linked or copied) or the respective directories are added to the include path with option `-I` of line `compile_fortran` in the environment file (`lnx86_64.env`/`win86_64.env` loaded from `abaqus_v6.env`). With `MATERIAL=directory` a user defined material behavior can be specified. The `directory` coincides with the name of the folder containing the necessary source files to be placed in the `materialroutines` folder of uel-large-deformation. For further details look into chapter 7. When `MATERIAL` is omitted the standard value is `Mises`.

Further requirements:

- Linux: Compiler options in the environment file `lnx86_64.env` loaded from `abaqus_v6.env` (as set in present package) in line `compile_fortran`
  - `-mkl=cluster`: include MKL
  - `-heap-arrays`: Create large arrays in heap instead of stack to avoid stack overflow
  - `-nostandard-realloc-lhs`: Turned out to be necessary to avoid certain runtime errors.
  - The Fortran preprocessor `FPP` must be switched off by deleting the option `-fpp`. The reason is that the present code is written in free-form Fortran whereas other included files may be written in fixed form. The present implementation employs the compiler directives `!DEC$ FREEFORM` and `!DEC$ NOFREEFORM` to switch between both settings. However, this compiler directive is incompatible with `FPP`.
- Windows: Changes in the environment file `win86_64.env` loaded from `abaqus_v6.env` (as set in present package)

  - additional options to `compile_fortran`: `\heap-arrays`, `\nostandard-realloc-lhs`, `\names:lowercase` (problem of Abaqus2020 with OneAPI), `\I"%MKLROOT%\include"`
  - additional option to `link_sl`: `mkl_rt.lib`

- The program adapts free-form versions of the required interface definitions from `SMAAspUser*.hdr`. It may be necessary to check whether these definitions have been changed in other version of Abaqus.

- The UEL has to provide an additional subroutine `GET_n_STATEV_elem`, which returns the number of state variables which have to be reserved by MonolithFE^{2}for each element (`NSVARS` in the head of UEL), cf. attached file `UEL.f`.

6.3 Source Code

The structure of the components of MonolithFE^{2}is illustrated in Fig. 1. It may be added that the file `*.FE*`, created by the Python plug-in, represents the node-to-element connectivity in form of lists and the connection between the entries of the stiffness matrices of the micro-UELs and the global stiffness matrix in Compressed Sparse Row (CSR) for the solver. The following table explains selected Fortran routines.

filename	routine	description
`UMATmacro.f`	UMAT	actual macro-UMAT interface to be called from Abaqus at each macro-GP; includes all needed source files; gets all pointers to the needed data; calls the main program (staggered/monolithic) to get macro STRESS and macro DDSDDE; extrapolates micro displacements in staggered case
`FE2MODULE.f`	main_program_staggered	actual main program for the staggered algorithm, which calls all needed routines (assemble, solve...) in the right order
`FE2MODULE.f`	main_program_monolithic	main program for monolithic algorithm
`FE2MODULE.f`	enforce_constraint	enforces constraints for the periodic boundary conditions
`FE2MODULE.f`	update_displacements	when the increment of the displacements in the Newton-Rhapson algorithm was computed, this routine updates the displacements
`FE2MODULE.f`	assemble	this routine calls the micro element routine for each micro element and assembles the stiffness matrices, the residual and the macro STRESS
`FE2MODULE.f`	static_condensation	this routine computes the macro stiffness matrix DDSDDE and macro stress STRESS
`Solver.f`	initialize	Create memory for solving sparse systems of equations with the MKL PARDISO solver and get a pointer to that memory
`Solver.f`	factor	factor the matrix
`Solver.f`	solve	solve system for given right hand sides
`Solver.f`	finish	free solver storage
`Solver.f`	get_permutation_matrix	get a permutation vector for a given matrix structure
`Solver.f`	interpret_error	interpret a possible PARDISO error
`SMAAspUserArraysFortran.f`	SMA*Array*CreateFortan* SMA*Array*AccessFortan* SMA*Array*DeleteFortan*	Abaqus provides routines called "allocatable arrays" to create threadsafe data and access them via pointers. These routines output cray pointers, which aren't FORTRAN Standard and can't be used in user derived types. Therefore these routines act as a interface to the Abaqus allocatable arrays and output FORTRAN Standard conform pointers.
`manage_data.f`	UEXTERNALDB	Abaqus interface UEXTERNALDB is called from Abaqus at the start/end of a analysis and start/end of a timestep. It's used to manage the mesh and state variable data at the correct point of the analysis
`manage_data.f`	readdata	at the analysis begin the micro mesh & analysis parameters are read and stored

`manage_data.f`	manage_STATEV_data	at the end of a time step the state variables of all macro GPs are written to: t→t-1, t+1→t; at the start of a time step the data is initialized; at the end of the analysis the allocated disk space is freed
`type_macro_GP_DATA.f`	allocate_data get_pointer deallocate_data	in this module a user derived FORTRAN type is declared which encapsulates all the macro GP state date (micro displacements, state variables of the micro GPs, ...)
`type_meshparameters.f`	read_data get_pointer deallocate_data	In this module a user derived FORTRAN type is declared which encapsulates all the data for defining the microscopic mesh (coordinates, element to node connection, etc.) plus the program-flow/convergence parameters. The create* routine reads the data from disk, the get_pointer* routine accesses the data and the deallocate* routine frees the allocated memory
`type_analysisparameters.f`	read_data get_pointer deallocate_data	In this module a user derived FORTRAN type is declared which encapsulates all the data for defining analysis parameters, for example if the monolithic or staggered algorithm shall be used, if symmetric matrix storage shall be used etc.
`utility_transform_stress_stiffness.f`	Get_abaqus_stress_stiffness	return STRESS and DDSDDE in the format Abaqus expects it.
`utility_transform_stress_stiffness.f`	nomina2	The 1st Piola Kirchhoff stress is transformed to Cauchy stress (also the material tangent: nominal tangent → DDSDDE).

7 Defining a different Material behavior

Defining the material behavior is similar to Abaqus. A material routine has to be written using the UMAT interface as declared in the Abaqus Manual [7]. The UMAT has to be named precisely "UMAT.f". The routine has to be written for 3D, plane strain or axisymmetric deformation states. The plane stress case is handled internally. The file UMAT.f can also contain various `SUBROUTINES` or `FUNTIONS` but no `MODULES`. Preferred is FORTRAN 90 free form code. When the UMAT is written in FORTRAN 77 fixed form style compiler directives have to be included before and after the subroutine as shown below:

```
!DEC$ NOFREEFORM

SUBROUTINE UMAT(...)

...user coding to define UMAT...

...all variables are defined just as in the Abaqus/Standard UMAT interface
and have the same type and dimension...

END SUBROUTINE UMAT

!DEC$ FREEFORM
```

The number of state variables required by the UMAT has to be specified in the Routine GetNSTATV. Name this routine precisely "GetNSTATV.f".

```
FUNCTION GetNSTATV(NTENS,NDI,PROPS)

INTEGER(KIND=AbqIK)::  GetNSTATV
INTEGER(KIND=AbqIK), INTENT(IN)::  NTENS,NDI
REAL(KIND=AbqRK),INTENT(IN)::  PROPS(:)

...user coding to define GetNSTATV. If NTENS==3 (plane stress) add +1 to that
value since the plane stress algorithm needs one internal state variable too...

END FUNCTION
```

Write a routine to check if the material parameters are reasonable and name it precisely "CheckMaterialParameters.f". Use the STDB_ABQERR function to stop the simulation and report errors to the inputfile [7]. If you do not want to check the parameters just return using the FORTRAN statement `RETURN`.

```
SUBROUTINE CheckMaterialParameters(PROPS)

REAL(KIND=AbqRK),INTENT(IN)::  PROPS(:)
```

```
...
!  Check the properties

if ( ...  )  then
CALL STDB_ABQERR(-3, "CheckMaterialParameters reports:  ...  ", 0, 0.0, " ")
else if ( ...  )  then
...
end if

END SUBROUTINE CheckMaterialParameters
```

Put all 3 files into a directory with a reasonable name inside of the `materialroutines` folder of uel-large-deformation. When the program is compiled as described in section 6.2 specify `MATERIAL=` with the name of the just created directory.

8 Limitations, Problems and Future Developments

8.1 Limitations

- maximum 2 sets of constitutive parameters per micro mesh definition
- maximum 5 different micro mesh definitions per macorscopic simulation
- single element type per micro mesh definition
- single micro-material routine (UMAT) per simulation (`PROPS` may be used to switch between different encapsulated material routines); MonolithFE^2 has been successfully tested with the UMAT_creep material routine (requires to make the UMAT accessible for compilation, as shown in chapter 7)
- Python plug-in „generate FE2 Inputfile“supports 3D micromeshes only if constraints already generated in Abaqus ”equations“ (Micromechanics Plugin or FoamGUI)
- purely mechanical problems (no "multiphysics")
- distributed-memory computations ("multi-node") may not work with the monolithic-stored stiffness matrix factorization algorithm in some cases (first tests yield no problem yet)

8.2 Known Issues

- sometimes convergence problems, specifically:
  - large deformation analysis
  - plane stress (especially in large deformation analysis)

8.3 Software environment

- The file `ABA_PARAM.INC` is not found by Abaqus by default under Linux, but put "artificially" to folder `src/`. Its contained must be checked under different OS and computer architectures.
- Integer kind specifications in `SMAAspUserArraysFortran.f` are fixed (adopted from `SMAAspUserSubroutines.hdr` of Abaqus) and may be specific to OS and computer architecture as well.

9 Version history

date	description
2020-07-29	final version of diploma thesis of N. Lange [3]
2020-12-17	Material in rate formulation; large deformations; all elements of UELlib added; in Plugin "Generate FE^2 Inputfile" plane strain/plane stress/3D stress selectable; general applicable plane stress algorithm in `UXMATMises.f` added; stiffnessmatrix factorization indefinite/positiv definite selectable
2021-02-03	switched from MKL DSS solver to MKL PARDISO solver due to an existing memory leak when using the DSS solver, added a postprocessing for micro problems
2021-04-20	version 1.0 published
2021-05-06	v1.01 with binaries for Windows
2021-07-20	v1.02, Inputfile adopts *Keyword style of Abaqus, number of node dof's named generally to be compatible with generalized continua, material routines can now be integrated more easily without changing the UEL.f file
2021-07-23	v1.02a, bug removed; added: automatic verification run auf examples within Makefile

References

[1] N. Lange, G. Hütter, B. Kiefer: An efficient monolithic solution scheme for FE^2 problems, Computer Methods in Applied Mechanics and Engineering 382 (2021), 113886, Preprint: arxiv.org/2101.01802.

[2] G. Hütter, S. Roth, R. Skrypnyk: UELlib – A library for user-defined elements in Abaqus, Technical Report, TU Bergakademie Freiberg, Institute of Mechanics and Fluid Dynamics.

[3] N. Lange: Implementation of a monolithic FE^2 program (in German), diploma thesis, TU Bergakademie Freiberg, 2020.

[4] G. Hütter, C. Settgast, N. Lange, M. Abendroth, B. Kiefer: A hybrid approach for the multi-scale simulation of irreversible material behavior incorporating neural networks, Proc. Appl. Math. Mech. 20 (2020), e202000248.

[5] R. McLendon: Micromechanics Plugin for Abaqus, `https://www.linkedin.com/pulse/micromechanics-plugin-abaqus-ross-mclendon`, 2017.

[6] M. Abendroth, E. Werzne, C. Settgast, S. Ray: An Approach Toward Numerical Investigation of the Mechanical Behavior of Ceramic Foams during Metal Melt Filtration Processes, Adv. Eng. Mater. 19 (2017), 1700080. DOI:10.1002/adem.201700080

[7] ABAQUS/Standard User's Manual, Version 6.9, Michael Smith, Dassault Systèmes Simulia Corp, 2009

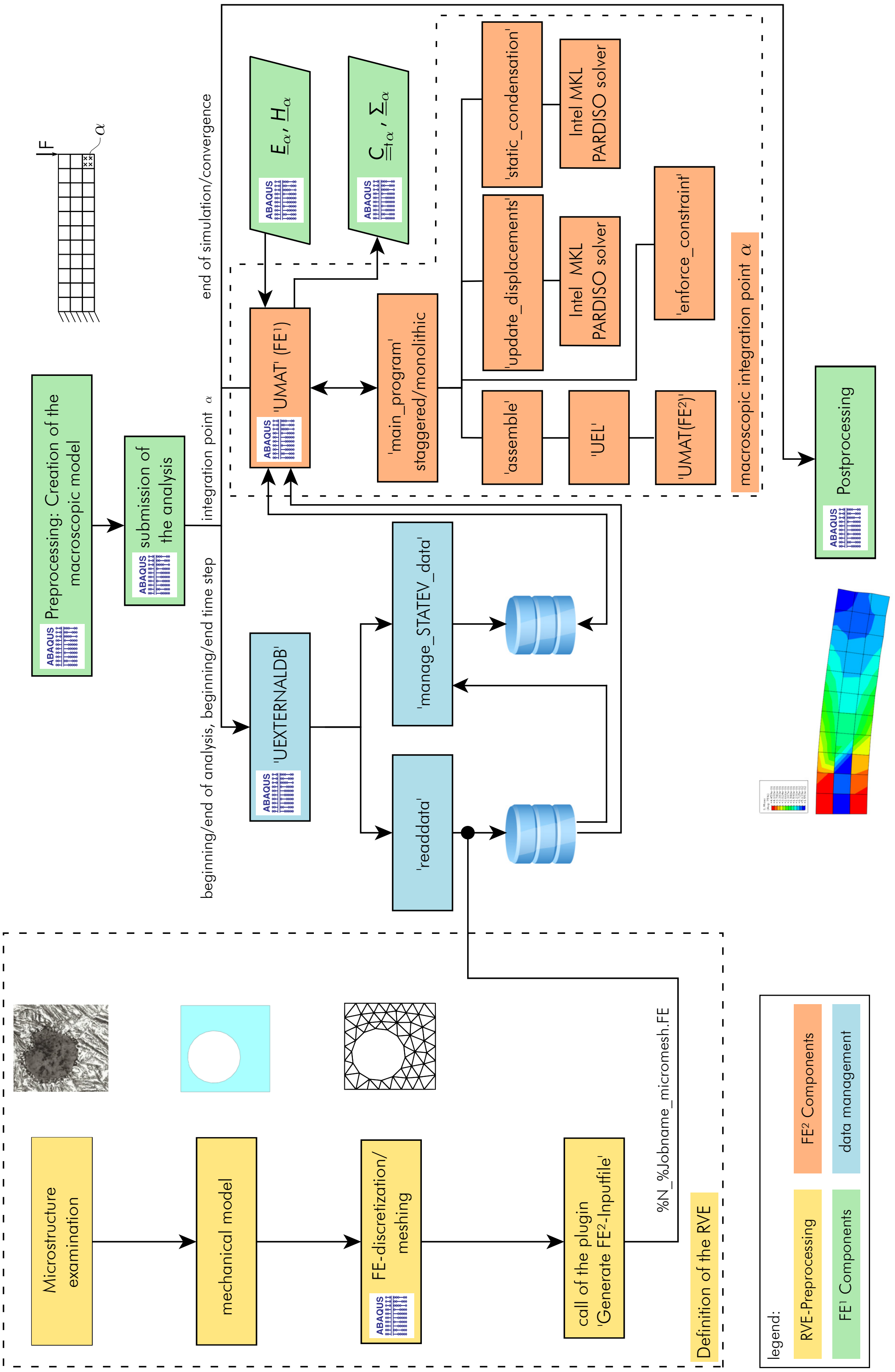

F
α
ABAQUS
Preprocessing: Creation of the macroscopic model
submission of the analysis
integration point α
end of simulation/convergence
beginning/end of analysis, beginning/end time step
'UEXTERNALDB'
'readdata'
'manage_STATEV_data'
'UMAT' (FE¹)
E_α, H_α
$\underline{C}_{t\alpha}$, Σ_α
'main_program' staggered/monolithic
'assemble'
'UEL'
'UMAT(FE²)'
'update_displacements'
Intel MKL PARDISO solver
'static_condensation'
Intel MKL PARDISO solver
'enforce_constraint'
macroscopic integration point α
Postprocessing
Microstructure examination
mechanical model
FE-discretization/ meshing
call of the plugin 'Generate FE²-Inputfile'
%N_%Jobname_micromesh.FE
Definition of the RVE
legend:
RVE-Preprocessing
FE¹ Components
FE² Components
data management

Fig. 1: Structure of MonolithFE^2