AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

AGH UNIVERSITY OF SCIENCE
AND TECHNOLOGY

AGH

Validation of a structure of a multiscale model with OWL and metaprogramming

Piotr Macioł
AGH University of Science and Technology
Department of Applied Computer Science and Modelling
pmaciol@agh.edu.pl

14.04.2016, Barcelona

Agenda

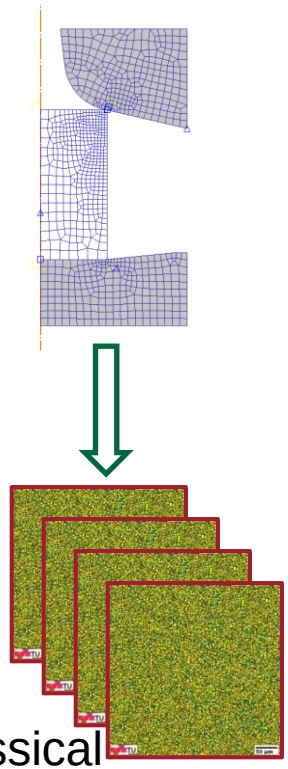
- Why validation?
- Why metaprogramming?
- Why OWL?
- Conclusions

Why validation?

» Agile Multiscale Modelling Methodology

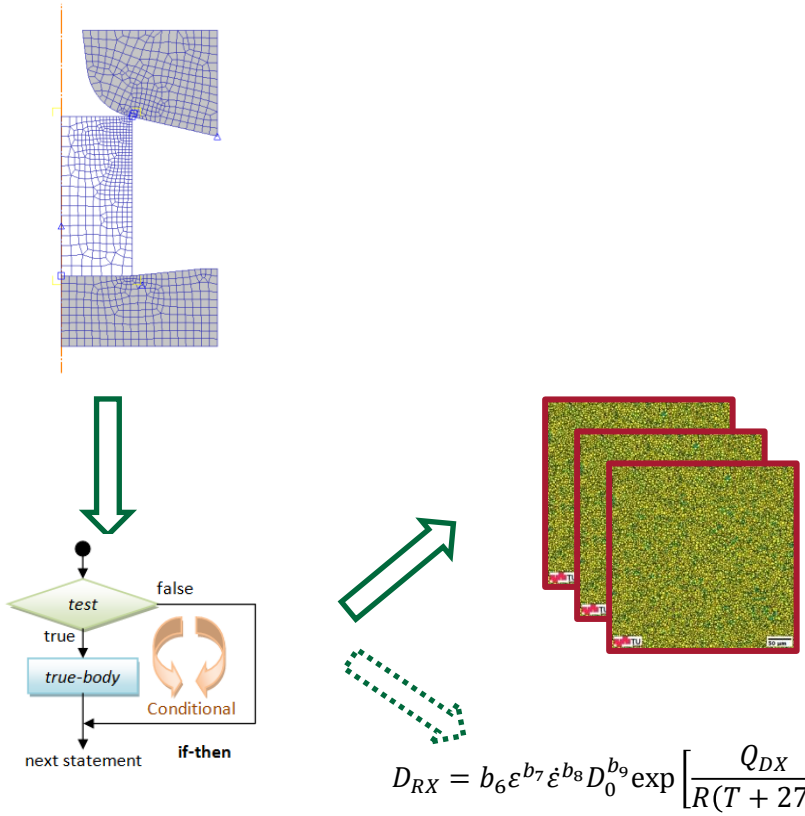
- Framework for adaptive multiscale modelling
- Fine scale models (equation, simple, complex) are selected in the run-time, basing on actual state of coarse scale model
- Knowledge Based System decides
- Better efficiency, the same reliability

Adaptive Multiscale Modelling Methodology (AM3)



Classical approach

- Design-time knowledge of interfaces
- Design-time location of fine scale models



$$D_{RX} = b_6 \varepsilon^{b_7} \dot{\varepsilon}^{b_8} D_0^{b_9} \exp\left[\frac{Q_{DX}}{R(T+273)}\right]$$

AM3 approach

- Interfaces not known on design-time
- Run-time choice of fine scale models -> better efficiency

Why validation?

» Workflow:

- The framework is developed by software developers
- The particular multiscale model is defined by modeler/researcher
- Definition of multiscale model:
  - What the coarse scale model is
  - Which fine scale models can be used
  - What are the rules for choosing fine scale models
  - Validation, are all models compatible
- Compatibility means:
  - are each input variables for fine scale model are available
  - are outputs from fine scale model(s) sufficient to solve coarse scale problem

Why validation?

- » ’Open’ configuration of multiscale model
- » Syntactic ambiguity
- » Semantic ambiguity
- » Automatic, run-time choice of fine scale models

↓

- » No user interaction when the fine scale model is selected

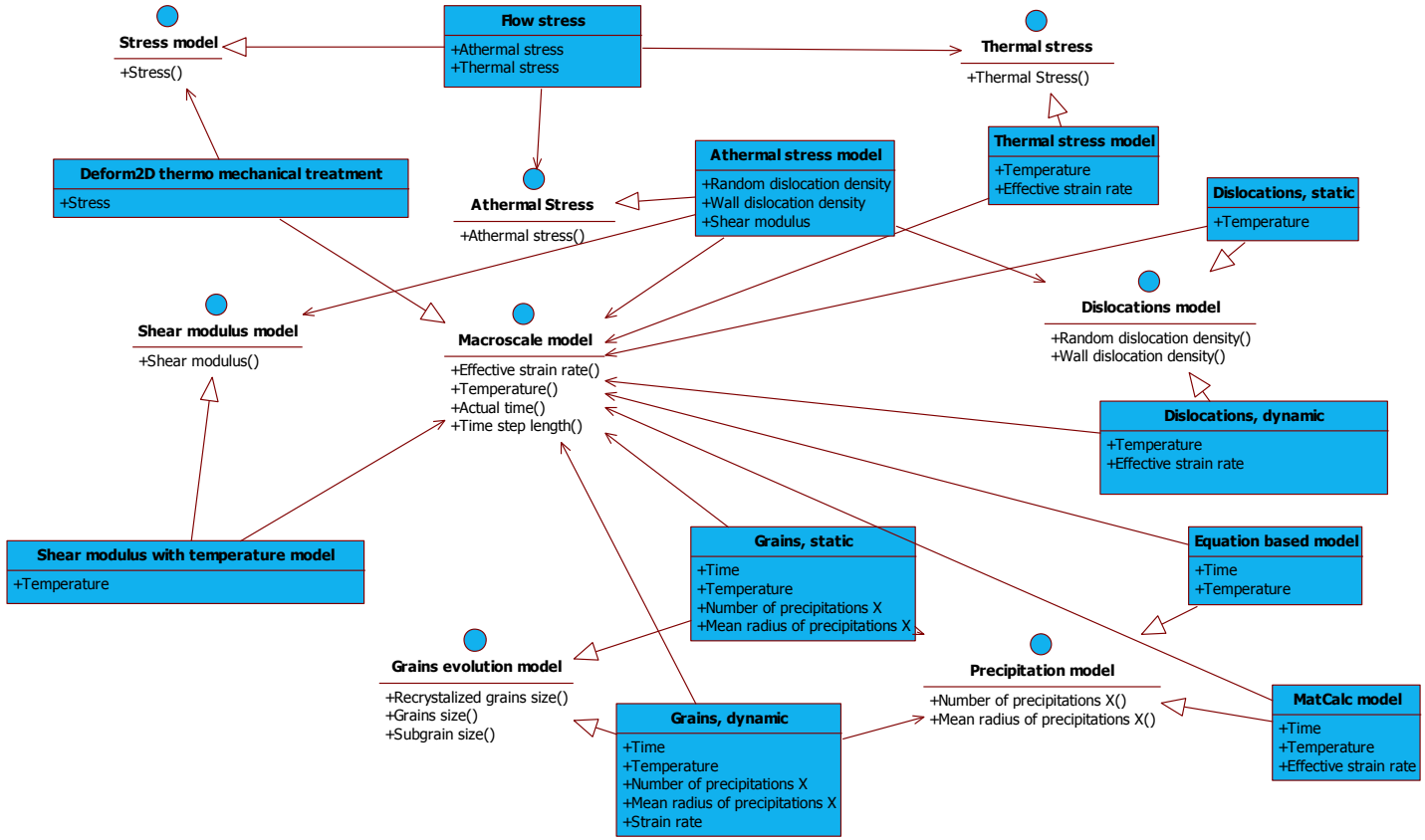
↓

- » Interfaces (passed variables) must be precisely defined

Metaprogramming validation

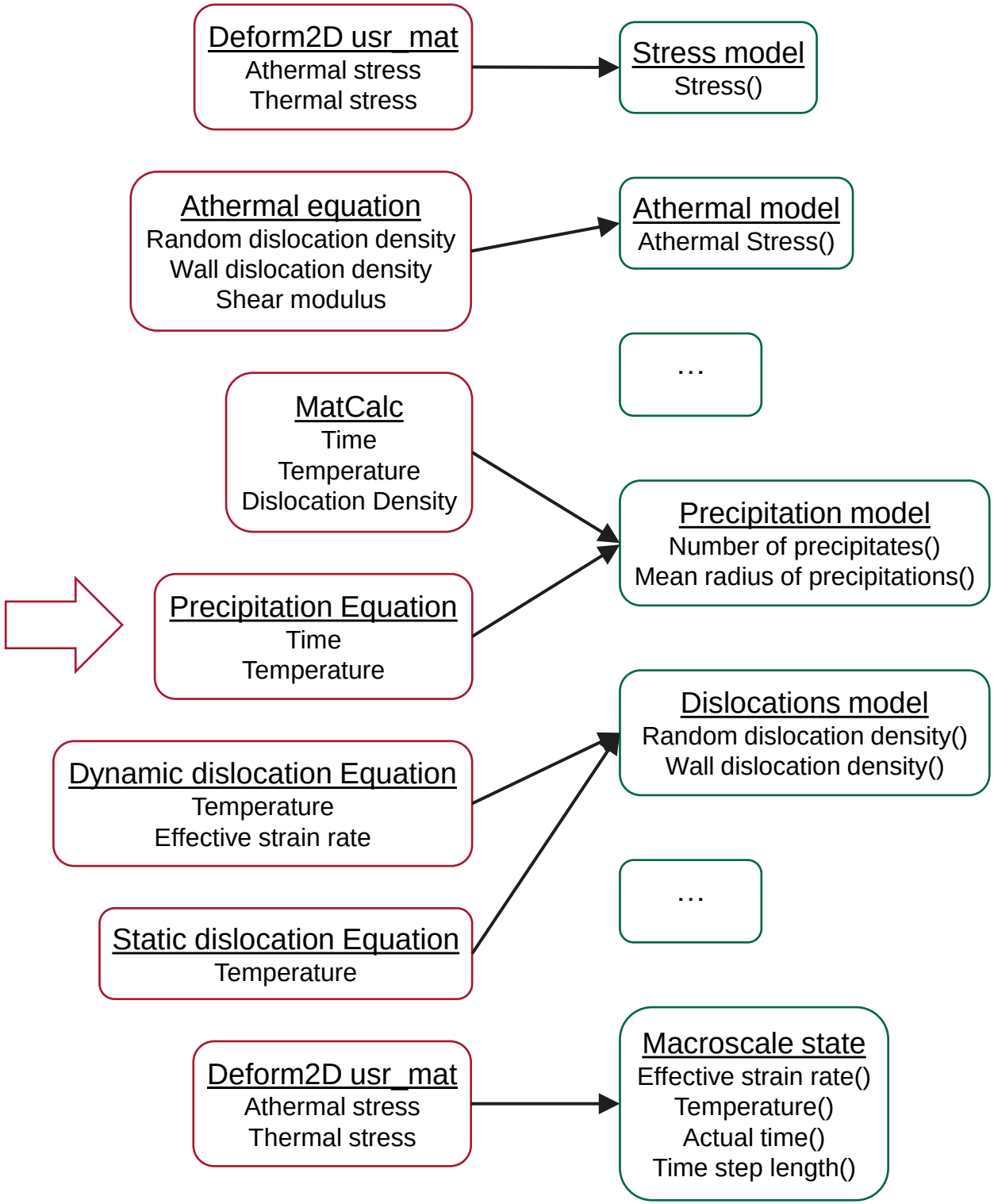
- AM3 is written in C++
- C++ is statically typed; compiler verifies compatibility of types during compilation → why not use it for validation?
- Metaprogramming → (meta)programs make programs
- In C++, metaprogramming is native with *templates* and *generic programming*
- The goals
  - use the compiler to validate are models compatible
  - Keep multiscale model definition simple

Representation of the structure of the model



- » Object-oriented representation of multiscale model
- » Thermomechanical treatment of aluminum alloy with recrystallization, grain growth and precipitation
- » Not user-friendly!

- » Submodels must be organized in hierarchy → outputs of some submodels are inputs for others
- » Wrong hierarchies (not all inputs provided) should be forbidden
- » The hierarchy of submodels in multiscale model of thermomechanical treatment of aluminum alloy
- » Human-readable, but not useful for automatic verification



Metamodelling solution

- Based on template lists approach (C++ language)
- All submodels are organized as a list
- Each submodel includes list of necessary inputs and list of available outputs
- For each submodel, submodels list divides into ’previous’ and ’further’ parts
- During compilation the compiler verifies (automatically), are all inputs of each submodel can be obtained from outputs of ‚previous’ part;
- If not, compilers throws an compilation error
- If program compiles successfully, configuration of submodels is valid

Exemplary code

```
// Definition of submodels structure
      typedef TEMPLATELIST_11(
        StressProxy,
        MockStressDerivatives,
        IvManager,
        StressThermalAthermalTaylor,
        StressThermalOrowanReduced,

StressAthermalDislocationsEq,
        DislocationWallEquation,
        DislocationRandomEquation,
        ShearModulusConst,
        PrecipitationsSwitcher,
        MacroModelState
        ) CaseTugMatCalcList;
```

```
// Definition of random dislocation equation model
 template<typename INPUT, typename STORAGE>
      class DislocationRandomEquation
        : public InputTypelist<INPUT>
        , public RandomDislocationsContract
        , public
ComputationsOnFirstCall<Coordinates2D>
      {
      public:
        typedef typename
StrainRateTemperatureStepInput Inputs;
        typedef typename
RandomDislocationsContract Contract;
        typedef STORAGE IvStorage;
        …
       }
```

Metamodelling solution

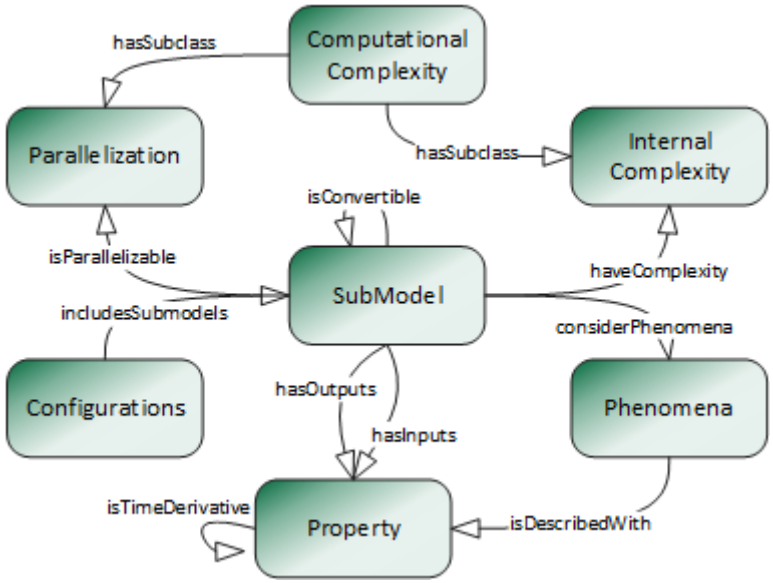
- Advantages
  - Submodels structure controlled with metamodeling is always valid
  - No additional computing costs; all work is done during compilation
- Disadvantages
  - Works only in C++ environment; submodels using other languages requires additional ‚wrappers'
  - Very high programming skills required

OWL solution

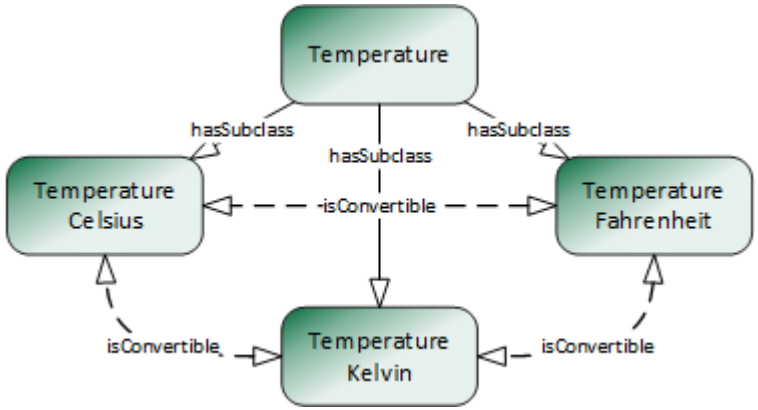
- An ontology is a model representing a fragment of knowledge domain, comprising a set of concepts and a set of predicates
- Ontology should provide knowledge interoperability and reusability
- Ontologies are capable of solving problems related to integration of knowledge
- Ontologies are usually developed using the OWL language which is a higher level language, based on XML and Resource Description Framework (RDF)
- Multiscale Modeling Formal Ontology (MMFO) was designed for describing multiscale models' data and structures

MMFO

Metaontology – describes what relationships are valid

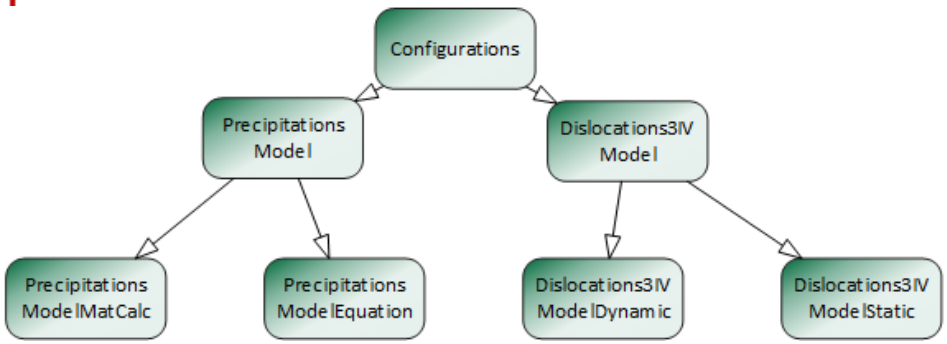


Dependencies between *Properties*

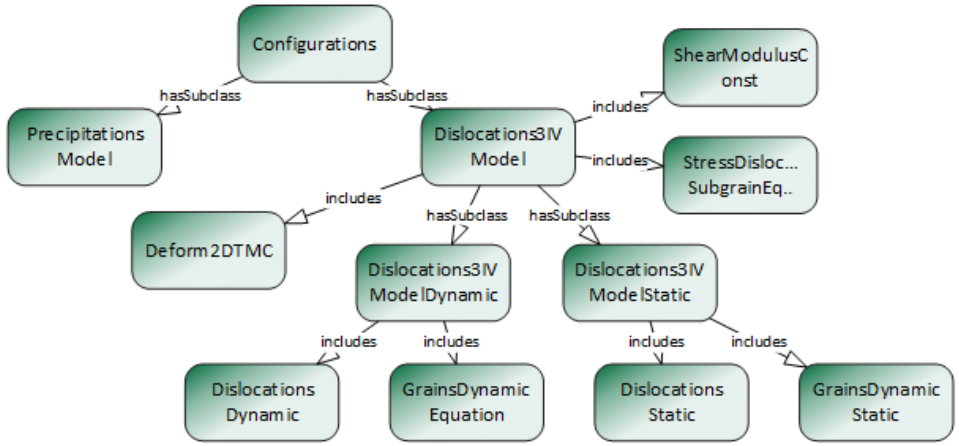


MMFO

An exemplary hierarchy of Configuration subclasses

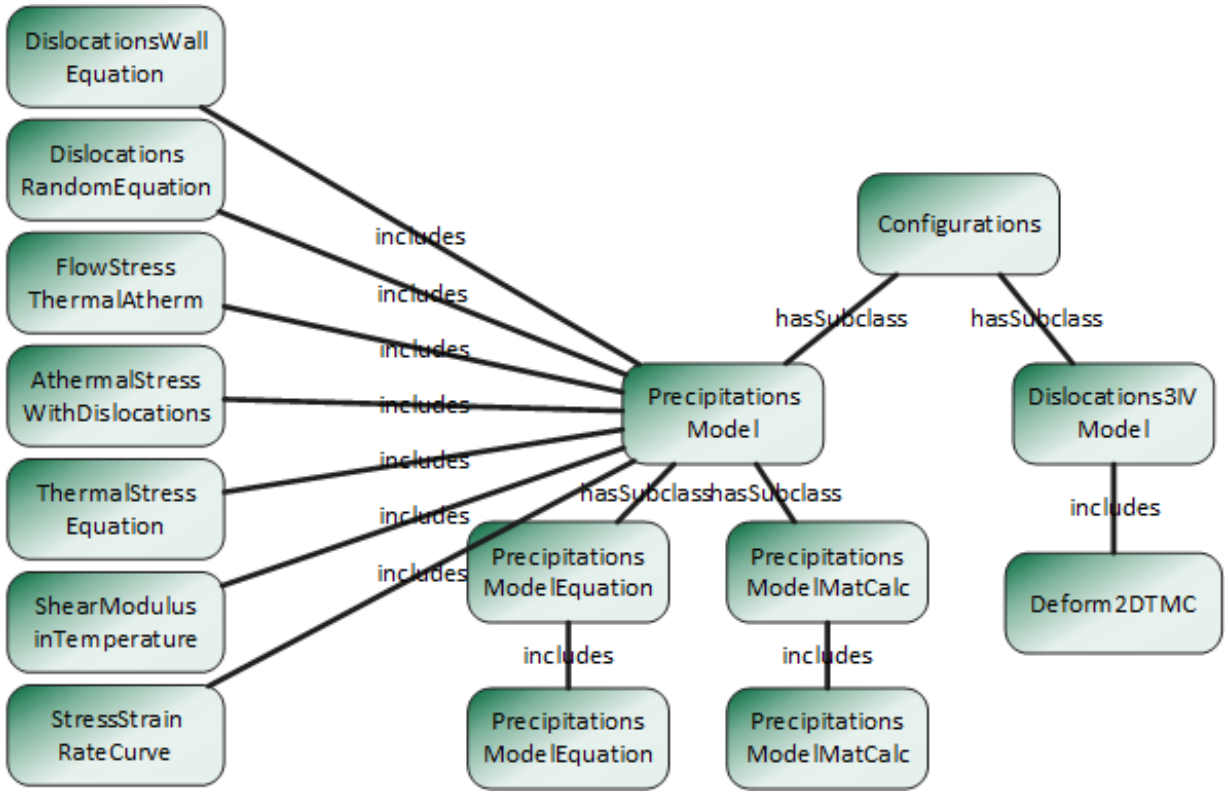


Configuration of exemplary model: the three-internal-variable



MMFO

Configurations of exemplary model: precipitation kinetic model



OWL solution

- Advantages
  - Human-readable
  - ***Metaontology*** gives capability of validation of syntactic correctness of the submodels description
  - Defines dependencies between submodels and properties/variables (including more complex ones, like *Velocity isTimeDerivative* of *Coordinates*)
  - OWL language is easily convertible to relational databases structures and common formats of data representation (e.g. JSON)
  - OWL based ontology (MMFO) can be easily used a common standard
- Disadvantages
  - The are no direct tools to validate the semantic correctness of submodels structure (but since OWL classes are well defined, it is ease to develop such tools)
  - Developing of ontology is tedious

Conclusions

- » A tool for verification of submodels structure is strongly recommended
- » Metamodeling
  - – can be used to ensure correctness of submodels structure, but
  - – it is not human-friendly
  - – cannot be used as a common standard
- » Ontology
  - – is human-friendly
  - – Can be a common standard, but
  - – Cannot be directly used for automatic validation of submodels structure
- » The goal for further works → combine both solution into flexible framework for semantic description of multiscale models and validation of their correctness