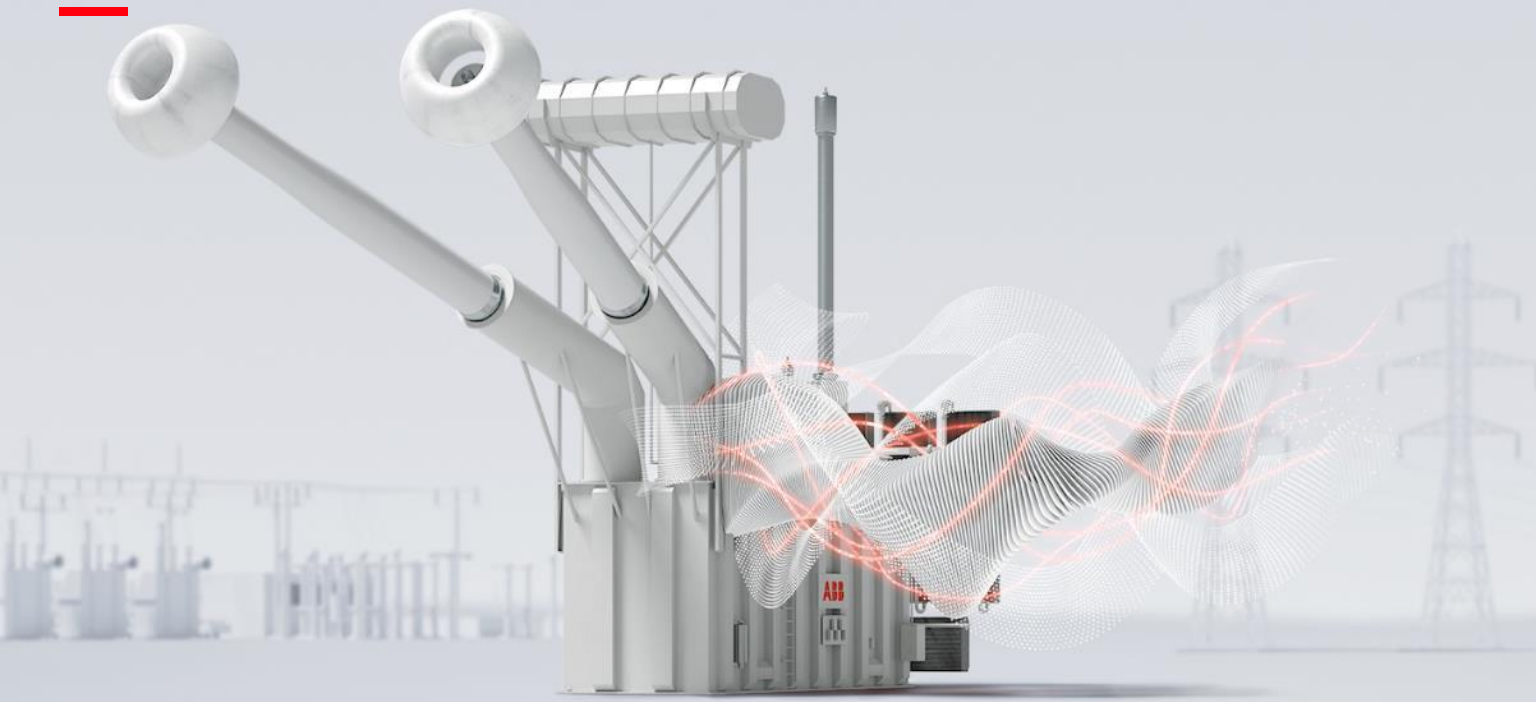


VECTORIZED MODELS, 04.02.2019

Vectorized models for new digital applications

R. Franke, ABB AG

Agenda

Motivation

Previous work

Extensions to new frontend

Extensions to backend

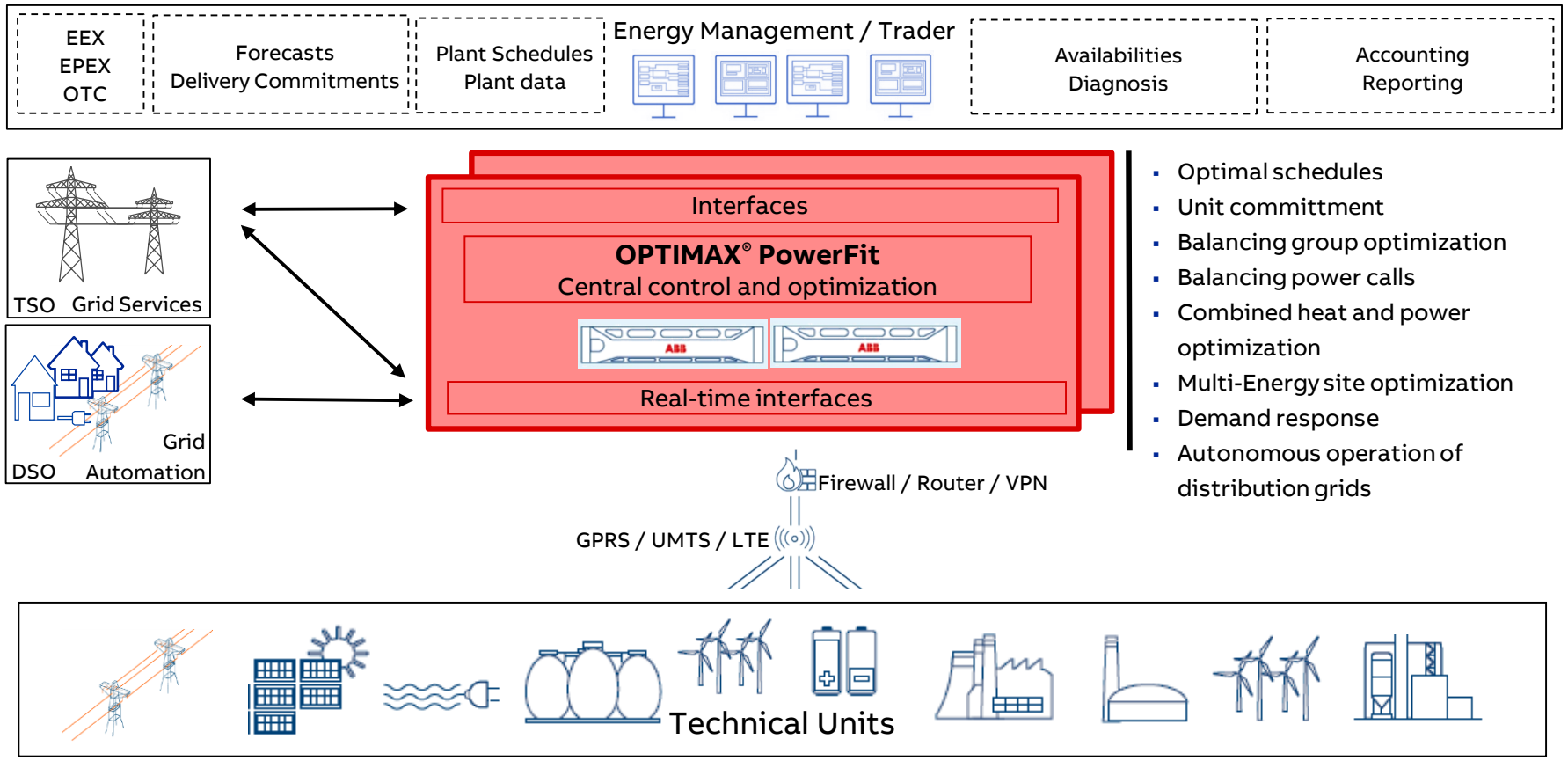
Example

Conclusions

This work was supported in parts by the Federal Ministry of Education and Research (BMBF) within the project PARADOM (PARallel Algorithmic Differentiation in OpenModelica) – BMBF funding code: 01IH15002E.

ABB OPTIMAX PowerFit

General System Architecture



Example: solar plants with collector grid

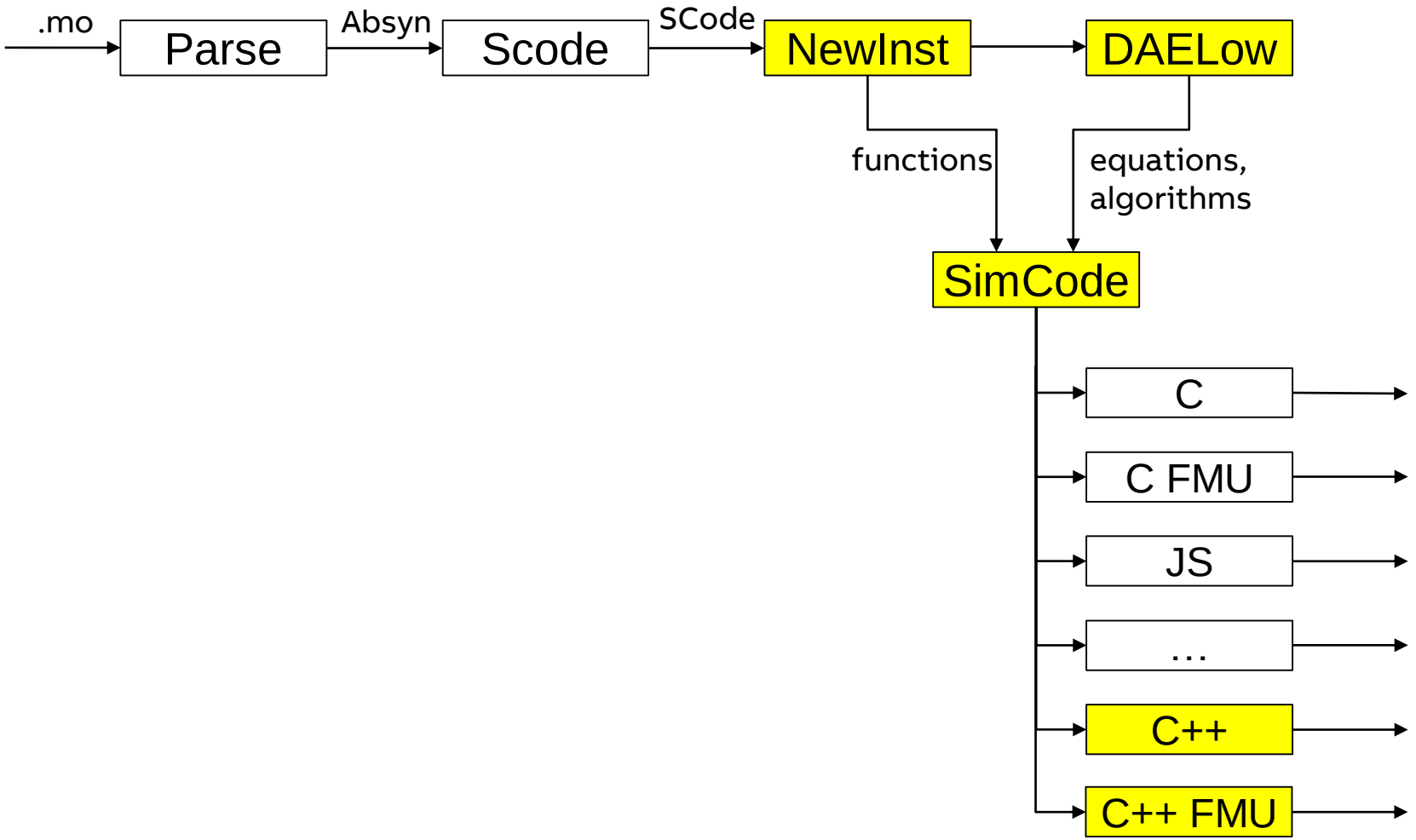
```
package Vectorized
  import SI = Modelica.SIunits;

  connector Terminal
    SI.Voltage v;
    flow SI.Current i;
  end Terminal;

  model Collector
    parameter Integer n;
    parameter SI.Voltage V = 1000;
    output SI.Power P_grid;
    Terminal terms[n];
  equation
    for i in 1:n loop
      terms[i].v = V;
    end for;
    0 = P_grid + terms.v * terms.i;
  end Collector;

  model SolarPlant
    input Boolean on "Plant status";
    input SI.Power P_solar "Solar power";
    parameter Real eta = 0.9 "Efficiency";
    Terminal term;
  equation
    term.v * term.i =
      if on then eta * P_solar else 0;
  end SolarPlant;

  model SolarSystem
    parameter Integer n = 1000;
    SolarPlant plant[n](each on = true,
                        P_solar = 100:100:n*100);
    Collector grid(n = n);
  equation
    connect(plant.term, grid.terms);
  end SolarSystem;
end Vectorized;
```

Motivation

Unable to compile models starting from array dimensions of 50 (larger models than SolarPlant), due to

- Excessive use of memory – 64bit OpenModelica shifted the boundary, but hits limits too
- Excessive translation times (both: omc and gcc)
- Huge compiled models

Overview of OpenModelica Compiler – touched parts in yellow



Previous work

Previous work in backend and code generation

- Collapse arrays for C++ variable declarations
- Collapse equations to for loops
- SimCode.SES_FOR_LOOP for collapsed equations
- Treatment of different kinds of arrays and array slices in Cpp runtime

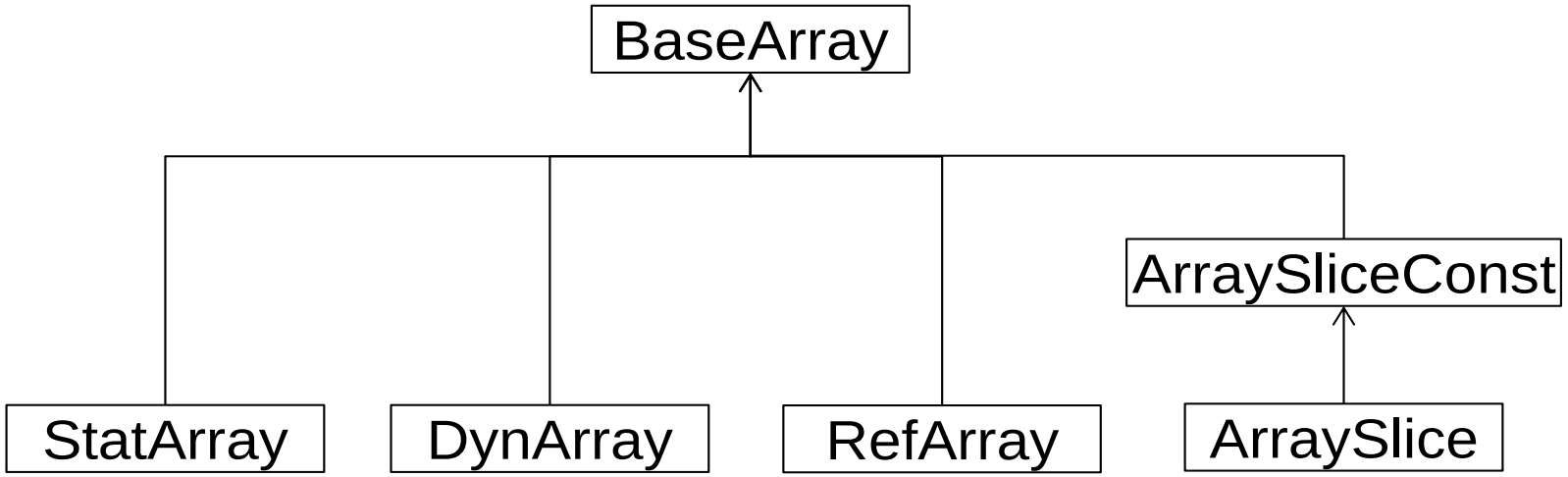
References

Joseph Schuchart, Volker Waurich, Martin Flehmig, Marcus Walther, Wolfgang E. Nagel, and Ines Gubsch.
Exploiting repeated structures and vectorization in Modelica.
In Proceedings of 11th International Modelica Conference. Modelica Association, Paris, France, 2015.

Rüdiger Franke, Marcus Walther, Niklas Worschech, Willi Braun, and Bernhard Bachmann.
Model-based control with FMI and a C++ runtime for Modelica.
In Proceedings of the 11th International Modelica Conference. Modelica Association, Paris, France, 2015.

Array implementations

Optimizing runtime performance for different uses of Modelica arrays



"I have never seen a perfect matrix class. In fact, given the wide variety of uses of matrices, it is doubtful whether one could exist" (Stroustrup, 2014)

Mapping of Modelica arrays to C++

Introductory examples

Modelica

```
// inside a regular model
Real[3] a;
String[4] s;


// e.g. inside a function
Real[:] b;
String[:] t;
```

C++

```
template <typename T, size_t nelems>
class StatArray {
  T data[nelem]; // reserve on stack
};


template <typename T>
class DynArray {
  T *data;
  DynArray(size_t nelems) {
    data = new T[nelems]; // on heap
  }
  ~DynArray() {
    delete [] data;
  }
};
```

New Frontend with -d=newInst,-nfScalarize

Extensions to new frontend

- Keep array declarations and array equations
- Keep for-loops (NFFlatten.mo, NFConvertDAE.mo, DAE.mo, …)
- Vectorize arrays of components by introducing array equations (NFFlatten.mo, NFSections.mo)
- Keep arrays in connect equations, including flows and overconstrained connectors (NFConnectionSets.mo, NFConnections.mo, NFConnector.mo)
- Remove array equations of dimension zero (NFInst.mo, NFVariable.mo, NFConnector.mo)

New Frontend with -nfScalarize

Implications on Backend

Equation count, for-equations, treatment of array equations

- Count array variables and equations as 1 if -nfScalarize
- Convert DAE.FOR_EQUATION to BackendDAE.FOR_EQUATION (BackendDAECreate.mo)
- Treat for-equations in backend (BackendDAETransform.mo, BackendDAEUtil.mo, BackendEquation.mo)
- Generalize BackendDAE.FOR_EQUATION from left=right to any equation
- Convert BackendDAE.FOR_EQUATION to SimCode.SES_FOR_LOOP (SimCodeUtil.mo)
- Inline for-equations (BackendInline.mo)
- Enhance solution of array equations (SolveSimpleEquations.mo, ExpressionSolve.mo, Expression.mo)

Each qualifier – implicitly apply fill assuming that dimensions have been checked by frontend

- Accept initialization of arrays with scalars – roll out only to assign actual values to memory in runtime
- Solution of array equations with scalars

States

Continuous-time states

- not considered

Discrete-time states

- Treat arrays of clocked states (SynchronousFeatures.mo)
- Extend inline integration to treat arrays of states, exploiting for-equation (SynchronousFeatures.mo)

Cpp code generation and FMI model description

Cpp code generation – minor enhancements and fixes

- Initialization of arrays with each
- Exploit operator= for array assignments (to simplify code generation)
- Fix code generation for array equations and slices
- Consider array slices with less subscripts than dimensions
- Skip subscripts in variable names for arrays if -nfScalarize

FMI model description

- Need to roll out XML model description (SimCodeUtil.mo)
- No dependencies yet

Balanced array models

Two options

a) Standard variables/equation counting

- Count arrays of dimension n as n variables
- Full flexibility for model formulation
- Array dimension must be known at compile time

b) Array variables/equations counting

- Count arrays as 1 variable
- Must formulate 1 array equation for 1 array variable
  Note: for-loop counts as 1 equation too
- May change array dimension at runtime

➔ went for option b) so far
(knowing that there is no benefit as of today,
but preparing possible future change of dimension
at runtime)

Example: solar plants with collector grid

```
package Vectorized
  import SI = Modelica.SIunits;

  connector Terminal
    SI.Voltage v;
    flow SI.Current i;
  end Terminal;

  model Collector
    parameter Integer n;
    parameter SI.Voltage V = 1000;
    output SI.Power P_grid;
    Terminal terms[n];
  equation
    for i in 1:n loop
      terms[i].v = V;
    end for;
    0 = P_grid + terms.v * terms.i;
  end Collector;

  model SolarPlant
    input Boolean on "Plant status";
    input SI.Power P_solar "Solar power";
    parameter Real eta = 0.9 "Efficiency";
    Terminal term;
  equation
    term.v * term.i =
      if on then eta * P_solar else 0;
  end SolarPlant;

  model SolarSystem
    parameter Integer n = 1000;
    SolarPlant plant[n](each on = true,
                        P_solar = 100:100:n*100);
    Collector grid(n = n);
  equation
    connect(plant.term, grid.terms);
  end SolarSystem;
end Vectorized;
```

Flattened array model (Vectorized.SolarSystem)

-d=newInst,-nfScalarize (simulate with: --simCodeTarget=Cpp)

New Frontend still expands dimensions

```
class SolarSystem
  parameter Integer n = 1000;
  Real[1000] plant.term.i;
  Real[1000] plant.term.v;
  parameter Real[1000] plant.eta = 0.9;
  Real[1000] plant.P_solar =
  (100:100:100000);
  Boolean[1000] plant.on = true;
  parameter Integer grid.n = 1000;
  parameter Real grid.V = 1000.0;
  Real grid.P_grid;
  Real[1000] grid.terms.i;
  Real[1000] grid.terms.v;
```

Array connections: Dimensions Checked by Frontend

Array of components

```
equation
  plant.term.v = grid.terms.v;
  plant.term.i + grid.terms.i = 0.0;
  for $i in 1:1000 loop
    plant[$i].term.v * plant[$i].term.i =
      if plant[$i].on then
        plant[$i].eta * plant[$i].P_solar
      else
        0.0;
  end for;
  for i in 1:1000 loop
    grid.terms[i].v = grid.V;
  end for;
  0.0 = grid.P_grid +
        grid.terms.v * grid.terms.i;
end SolarSystem;
```

Collector model

Conclusions

Many new applications require the treatment of vast numbers of loosely coupled components, e.g.

- Small renewable plants
- Autonomous cars

Modelica language has a rich array syntax; so far all tools unroll arrays during flattening

This work was motivated by dramatic performance problems for commercial array models

Model	Equations	Dym (s)	OMC NF/CF (s)
Vectorized.SolarSystem(n=10000) from section 4	60001	146.30	34.12 / 314.8 (**02.95**)
Vectorized.SolarSystem(n=100000) from section 4	600001	14458.68	2450.57 / 19760.42 (**02.95**)

Test by Adrian Pop et al

Coverage:

- Array variables, equations and components
- Arrays of connectors, including flows and overconstrained

It serves as proof of concept for a more rigorous implementation

Most missing as of today

- **Jacobian and model structure**
- Refactoring of Backend (cf. New Frontend) and then support more features