Debugging Scientific Software

Dorian Leroy

Inria, DiverSE

December 15, 2021

Scientific software: what is it and why is it important?

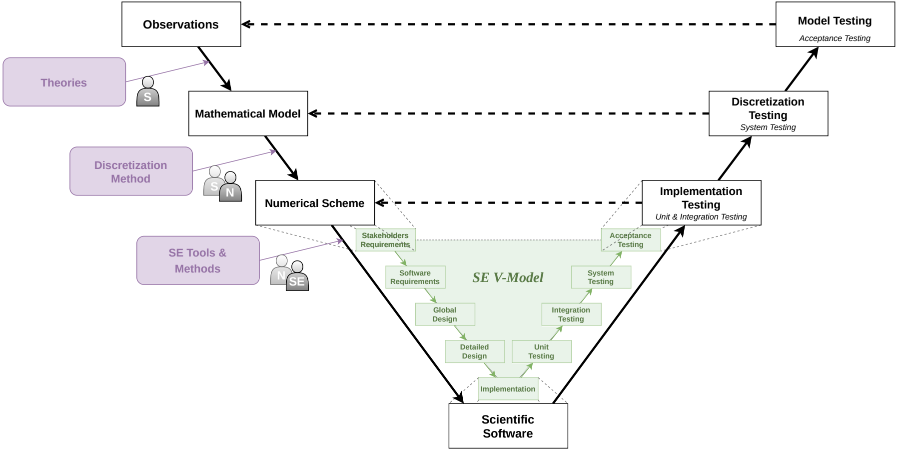
Scientific computing

Use of advanced computing capabilities to solve complex problems, aiming to predict the behavior or outcome of a system, man-made or otherwise.

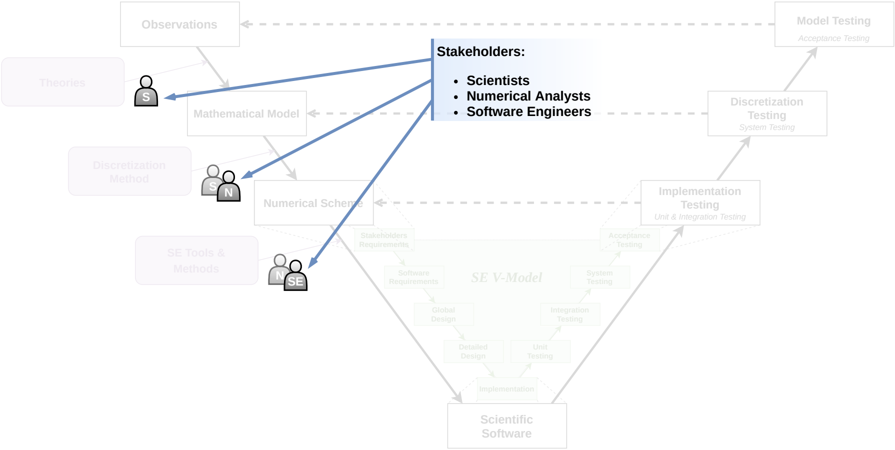
Why is it important?

Aerospace engineering, mechanical engineering, material science, chemistry, medicine and many more disciplines, but also...
Basis of scientific findings shaping **policy regarding wicked problems** such as COVID-19 or climate change.

The engineering of scientific software



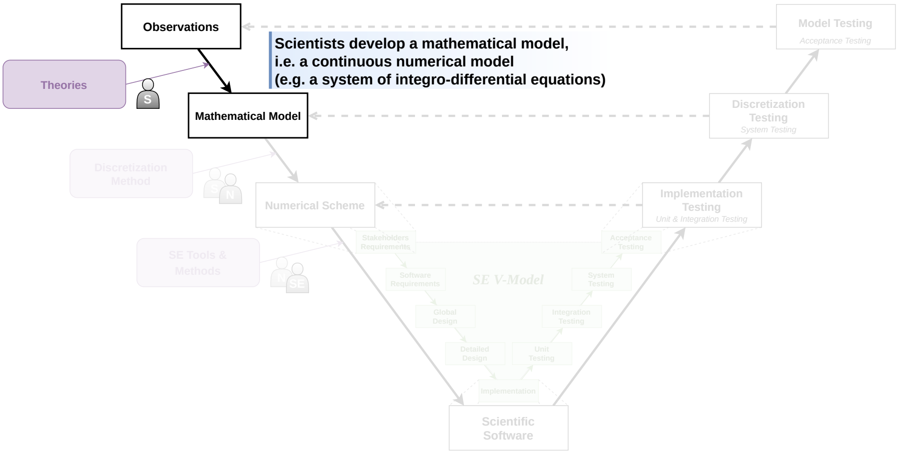
The engineering of scientific software



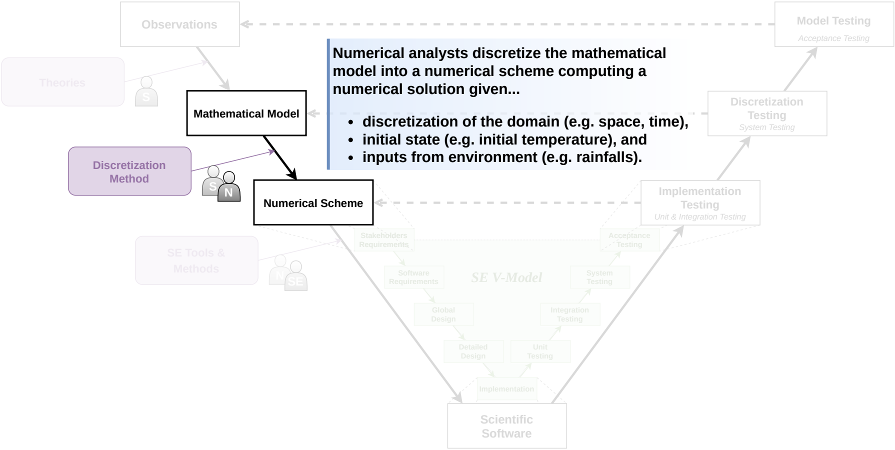
Stakeholders:

- **Scientists**
- **Numerical Analysts**
- **Software Engineers**

The engineering of scientific software



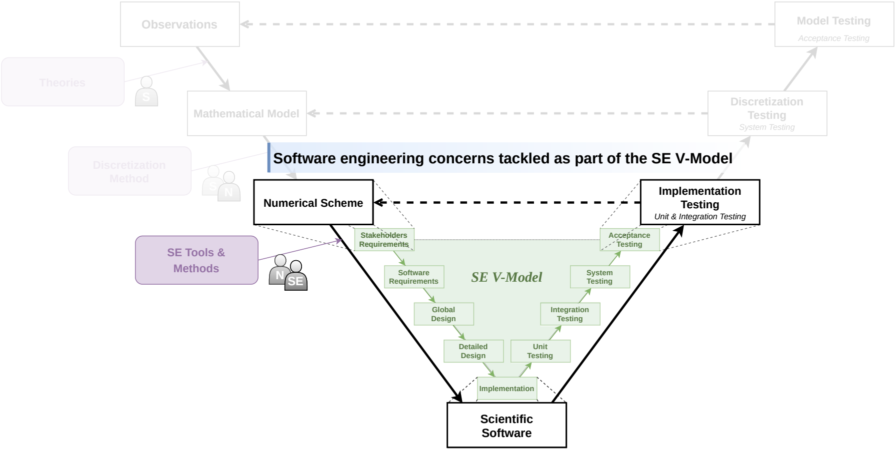
The engineering of scientific software



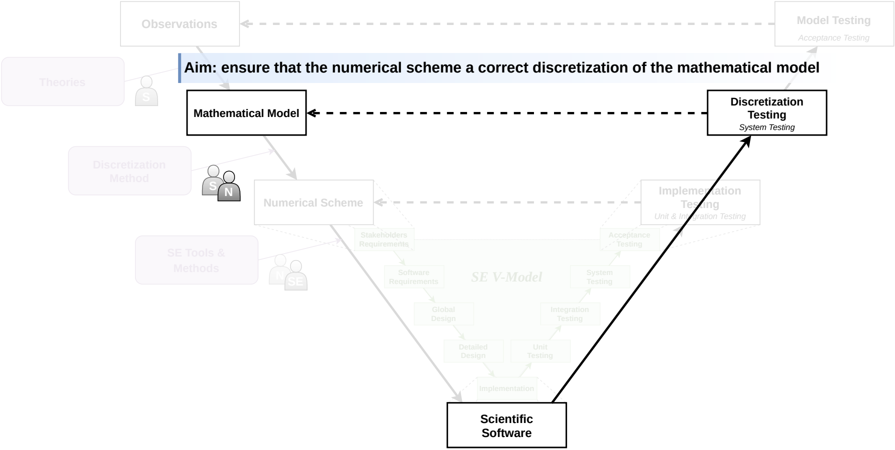
Numerical analysts discretize the mathematical model into a numerical scheme computing a numerical solution given...

- **discretization of the domain (e.g. space, time),**
- **initial state (e.g. initial temperature), and**
- **inputs from environment (e.g. rainfalls).**

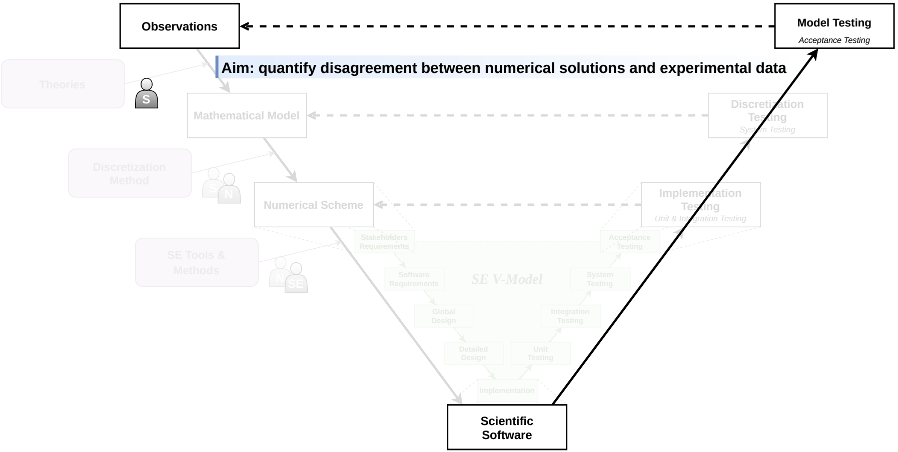
The engineering of scientific software



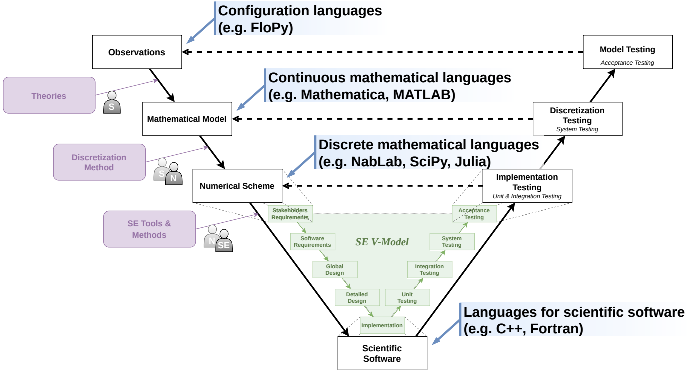
The engineering of scientific software



The engineering of scientific software

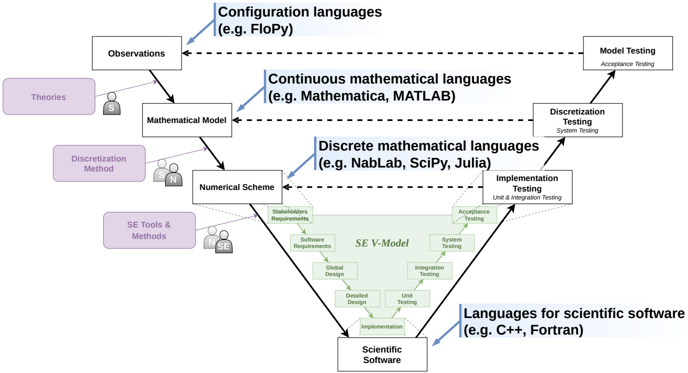


Language matters! Power comes with responsibility!



We investigated the impact of language choice through the **balance of responsibilities** between language users and language designers.

Language matters! Power comes with responsibility!



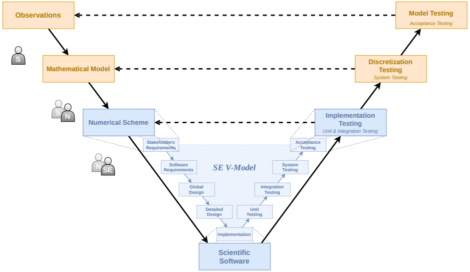
We investigated the impact of language choice through the **balance of responsibilities** between language users and language designers.

Balance of responsibilities: languages for mathematical models

Example languages

Mathematica, MATLAB

Balance of responsibilities



Language users

- Model testing to assess mathematical model fidelity
- Discretization testing on the derived scientific software

Language designers

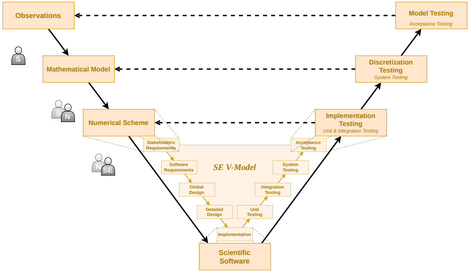
- Discretization testing of the provided continuous mathematical constructs
- Software engineering V&V concerns
- Providing tools for discretization testing

Balance of responsibilities: languages for scientific software

Example languages

C++, Fortran

Balance of responsibilities



Language users

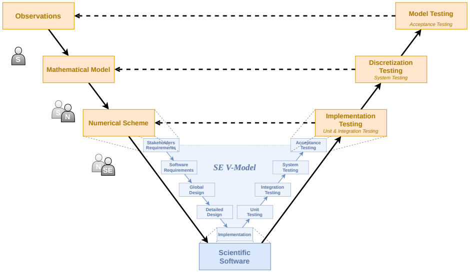
- Software engineering V&V concerns
- Implementation testing
- Discretization testing
- Model testing

Balance of responsibilities: languages for numerical schemes

Example languages

NabLab, Julia, SciPy, GNU Octave

Balance of responsibilities



Language users

- Implementation testing
- Discretization testing
- Model testing

Language designers

- Software engineering V&V concerns
- Implementation testing of the provided discrete mathematical constructs
- **Providing tools for implementation testing**

Example language for numerical schemes: NabLab

NabLab: Executable DSL (xDSL) for **numerical analysis**.

Abstract syntax: Metamodel reifying **domain concepts**, *e.g.*,

- Matrices, vectors, scalars
- Algebraic expressions over those
- Mathematical functions
- Iterative control structures

Operational semantics:

- Metamodel defining **model state** during the execution
- Set of execution rules $\Rightarrow$ **Interpreter**

Example NabLab model: IterativeHeatEquation

Example NabLab model: Solving the heat equation with iterative numerical method.

```
iterate n while (t^{n+1} < stopTime && n+1 < maxIterations),
        k while (residual > ε && check(k+1 < maxIterationsK));

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} +
    ∑{d∈neighbourCells(c)} (α{c, d} * u^{n+1, k}{d});
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));

ComputeTn: t^{n+1} = t^{n} + δt;
```

Example invariant to check: $\text{residual}_{n,k} > \text{residual}_{n,k+1}$

Example NabLab model: IterativeHeatEquation

Example NabLab model: Solving the heat equation with iterative numerical method.

```
iterate n while (t^{n+1} < stopTime && n+1 < maxIterations),
        k while (residual > ε && check(k+1 < maxIterationsK));

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} +
    ∑{d∈neighbourCells(c)} (α{c, d} * u^{n+1, k}{d});

ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));

ComputeTn: t^{n+1} = t^{n} + δt;
```

Example invariant to check: $\text{residual}_{n,k} > \text{residual}_{n,k+1}$

Example NabLab model: Solving the heat equation with iterative numerical method.

```
iterate n while (t^{n+1} < stopTime && n+1 < maxIterations),
        k while (residual > ε && check(k+1 < maxIterationsK));

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} +
    ∑{d∈neighbourCells(c)} (α{c, d} * u^{n+1, k}{d});
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));

ComputeTn: t^{n+1} = t^{n} + δt;
```

Example invariant to check: $\text{residual}_{n,k} > \text{residual}_{n,k+1}$

Example NabLab model: IterativeHeatEquation

Example NabLab model: Solving the heat equation with iterative numerical method.

```
iterate n while (t^{n+1} < stopTime && n+1 < maxIterations),
        k while (residual > ε && check(k+1 < maxIterationsK));

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} +
    ∑{d∈neighbourCells(c)} (α{c, d} * u^{n+1, k}{d});
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));

ComputeTn: t^{n+1} = t^{n} + δt;
```

Example invariant to check: $\text{residual}_{n,k} > \text{residual}_{n,k+1}$

Example NabLab model: IterativeHeatEquation

Example NabLab model: Solving the heat equation with iterative numerical method.

```
iterate n while (t^{n+1} < stopTime && n+1 < maxIterations),
        k while (residual > ε && check(k+1 < maxIterationsK));

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} +
    ∑{d∈neighbourCells(c)} (α{c, d} * u^{n+1, k}{d});
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));

ComputeTn: t^{n+1} = t^{n} + δt;
```

Example invariant to check: $\text{residual}_{n,k} > \text{residual}_{n,k+1}$

Example NabLab model: IterativeHeatEquation

Example NabLab model: Solving the heat equation with iterative numerical method.

```
iterate n while (t^{n+1} < stopTime && n+1 < maxIterations),
        k while (residual > ε && check(k+1 < maxIterationsK));

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} +
    ∑{d∈neighbourCells(c)} (α{c, d} * u^{n+1, k}{d});
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));

ComputeTn: t^{n+1} = t^{n} + δt;
```

Example invariant to check: $\text{residual}_{n,k} > \text{residual}_{n,k+1}$

Debugging NabLab models

Available facilities:

- Output capabilities ⇒ designed for **production use**, not debugging.

```
VtkOutput
{
    periodReferenceVariable = iterativeHeatEquation.n;
    outputVariables = iterativeHeatEquation.u as "Temperature";
}
```

- Interactive debugging ⇒ **impractical** for such highly iterative software.

```
UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} =
            u^{n}{c} + α{c, c} * u^{n+1, k}{c} +
            ∑{d∈neighbourCells(c)} (α{c, d} * u^{n+1, k}{d});
```

Preferred approach:

- **Logging and monitoring** of domain-specific properties (*e.g.*, physics conservation laws, numerical invariants).

Logging and monitoring for xDSLs

General obstacles to **domain-level** logging and monitoring facilities for xDSLs:

- Restricted DSL expressivity:
  - Introducing language constructs **goes against SoC** (*e.g.*, `printf`, `if`)
  - **Different expressivity** than offered by the DSL might be required
- Domain-specificness:
  - Cannot reuse libraries through **domain concepts** (*e.g.*, Apache log4x)
  - Additional development costs **for each DSL** to support

Logging and monitoring complementarities

Logging and monitoring are often **dependent on one another**:

- Monitoring can **operate on derived data** obtained through logging mechanisms
- Logging can be **triggered or altered** upon (in)validation of monitored properties

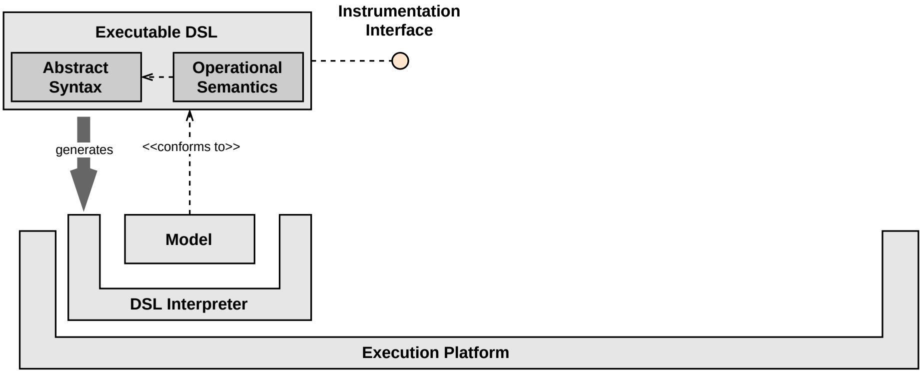
Yet, obstacles prevent **domain experts** from leveraging these complementarities:

- Requires **DSL support** for logging and monitoring frameworks
- Requires **domain-level interoperability** between frameworks

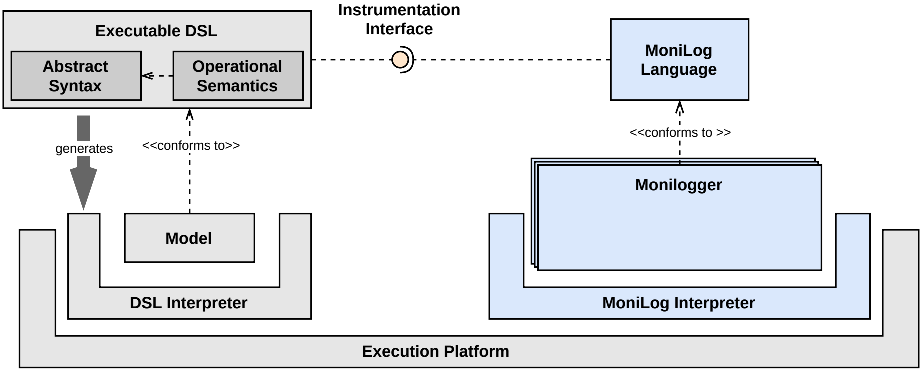
Proposed solution: MoniLog

- **Language-agnostic, unifying framework** for runtime monitoring and logging allowing to define loggers, runtime monitors and combinations of the two, a.k.a. ***moniloggers***.

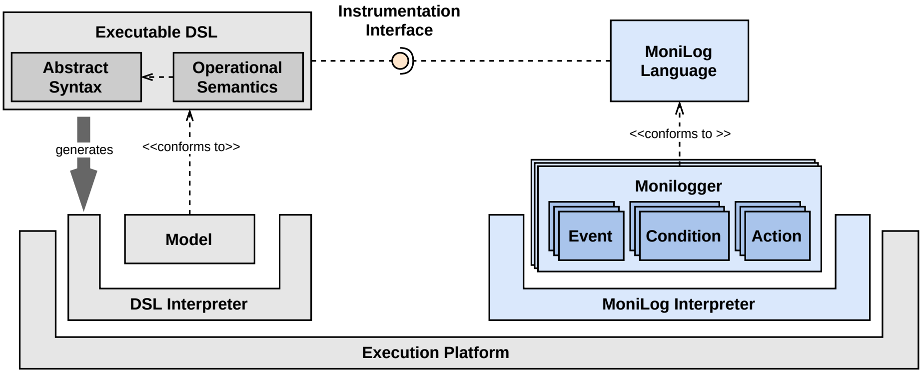
Overview of the approach



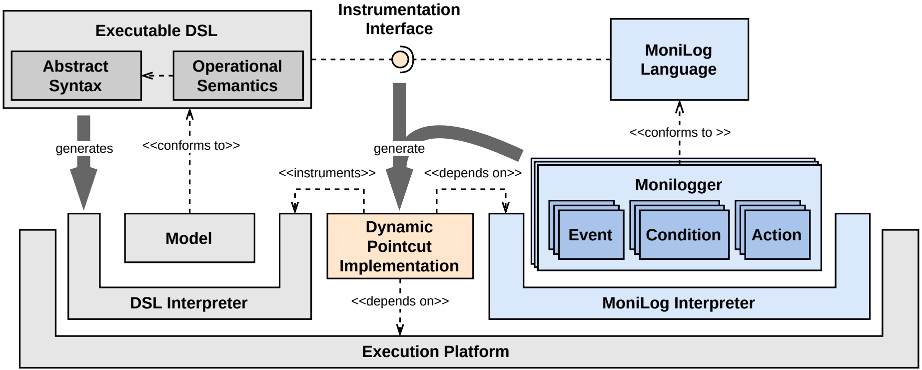
Overview of the approach



Overview of the approach



Overview of the approach



Overview of MoniLog

A MoniLog specification allows to

- define instrumentation-specific variables,
- declare the execution events of interest,
- register moniloggers to these events, which can
  - update instrumentation variables,
  - access the execution state of the running model,
  - evaluate expressions with languages available on the execution platform,
  - call various appenders (*e.g.*, file, message queue, console),
  - start/stop moniloggers

Example use on NabLab

IterativeHeatEquation-MonitorResidual.mnlg

```
package iterativeheatequation

import org.gemoc.monilog.stl.*
import fr.cea.nabla.monilog.nablalib.*
import IterativeHeatEquation.*

setup {
    prevResidual = 1.0;
}

event ComputeTnReturned {
    after call ComputeTn
}

event ResidualUpdated {
    after call ComputeResidual
}

monilogger correctResidual {
    when ResidualUpdated
    if (context(residual) > prevResidual)
    then {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "Incorrect residual! " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        correctResidual.stop();
        resetResidual.stop();
    } else {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        prevResidual = context(residual);
    }
}

monilogger resetResidual {
    when ComputeTnReturned
    {
        prevResidual = 1.0;
    }
}
```

IterativeHeatEquation.n

```
InitD: ∀c∈cells(), D{c} = 1.0;

ComputeDeltaTn: δt = Min{c∈cells()}(V{c}/D{c}) * 0.1;
ComputeV: ∀j∈cells(), V{j} = 0.5 * ∑{p∈nodesOfCell(j)}(det(X{p}, X{p+1}));
ComputeFaceLength: ∀f∈faces(), faceLength{f} = 0.5 * ∑{p∈nodesOfFace(f)}(norm(X{p]
ComputeFaceConductivity: ∀f∈faces(), faceConductivity{f} = 2.0 * ∏{c1∈cellsOfFace

// Assembling of the diffusion matrix
ComputeAlphaCoeff: ∀c∈cells(), {
    let ℝ αDiag = 0.0;
    ∀d∈neighbourCells(c), ∀f∈commonFace(c,d), {
        let ℝ αExtraDiag = δt / V{c} * (faceLength{f} *  faceConductivity{f}) / n
        α{c, d} = αExtraDiag;
        αDiag = αDiag + αExtraDiag;
    }
    α{c, c} = -αDiag;
}

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} + ∑{d∈n
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));
ComputeTn: t^{n+1} = t^{n} + δt;
```

Console

NabLab Console

```
[n=018, k=09] current residual: 1.1E-6, previous residual: 1.8E-6
[n=018, k=10] current residual: 7.0E-7, previous residual: 1.1E-6
[n=018, k=11] current residual: 4.4E-7, previous residual: 7.0E-7
[n=018, k=12] current residual: 2.8E-7, previous residual: 4.4E-7
[n=018, k=13] current residual: 1.8E-7, previous residual: 2.8E-7
[n=018, k=14] current residual: 1.2E-7, previous residual: 1.8E-7
[n=018, k=15] current residual: 8.0E-8, previous residual: 1.2E-7
[n=018, k=16] current residual: 5.3E-8, previous residual: 8.0E-8
[n=018, k=17] current residual: 3.6E-8, previous residual: 5.3E-8
[n=018, k=18] current residual: 2.5E-8, previous residual: 3.6E-8
[n=018, k=19] current residual: 1.7E-8, previous residual: 2.5E-8
[n=018, k=20] current residual: 1.2E-8, previous residual: 1.7E-8
[n=018, k=21] current residual: 8.5E-9, previous residual: 1.2E-8
[n=019, k=01] current residual: 9.0E-3, previous residual: 1.0E0
[n=019, k=02] current residual: 7.0E-4, previous residual: 9.0E-3
[n=019, k=03] current residual: 1.0E-4, previous residual: 7.0E-4
[n=019, k=04] current residual: 2.4E-5, previous residual: 1.0E-4
[n=019, k=05] current residual: 8.4E-6, previous residual: 2.4E-5
[n=019, k=06] current residual: 3.9E-6, previous residual: 8.4E-6
[n=019, k=07] current residual: 1.9E-6, previous residual: 3.9E-6
[n=019, k=08] current residual: 1.1E-6, previous residual: 1.9E-6
```

Example use on NabLab

IterativeHeatEquation-MonitorResidual.mnlg

```
package iterativeheatequation

import org.gemoc.monilog.stl.*
import fr.cea.nabla.monilog.nablalib.*
import IterativeHeatEquation.*

setup {
    prevResidual = 1.0;
}

event ComputeTnReturned {
    after call ComputeTn
}

event ResidualUpdated {
    after call ComputeResidual
}

monilogger correctResidual {
    when ResidualUpdated
    if (context(residual) > prevResidual)
    then {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "Incorrect residual! " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        correctResidual.stop();
        resetResidual.stop();
    } else {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        prevResidual = context(residual);
    }
}

monilogger resetResidual {
    when ComputeTnReturned
    {
        prevResidual = 1.0;
    }
}
```

IterativeHeatEquation.n

```
InitD: ∀c∈cells(), D{c} = 1.0;

ComputeDeltaTn: δt = Min{c∈cells()}(V{c}/D{c}) * 0.1;
ComputeV: ∀j∈cells(), V{j} = 0.5 * ∑{p∈nodesOfCell(j)}(det(X{p}, X{p+1}));
ComputeFaceLength: ∀f∈faces(), faceLength{f} = 0.5 * ∑{p∈nodesOfFace(f)}(norm(X{p
ComputeFaceConductivity: ∀f∈faces(), faceConductivity{f} = 2.0 * ∏{c1∈cellsOfFace

// Assembling of the diffusion matrix
ComputeAlphaCoeff: ∀c∈cells(), {
    ℝ αDiag = 0.0;
    neighbourCells(c), ∀f∈commonFace(c,d), {
        let ℝ αExtraDiag = δt / V{c} * (faceLength{f} *  faceConductivity{f}) / n
        α{c, d} = αExtraDiag;
        αDiag = αDiag + αExtraDiag;
    }
    α{c, c} = -αDiag;
}

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} + ∑{d∈n
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));
ComputeTn: t^{n+1} = t^{n} + δt;
```

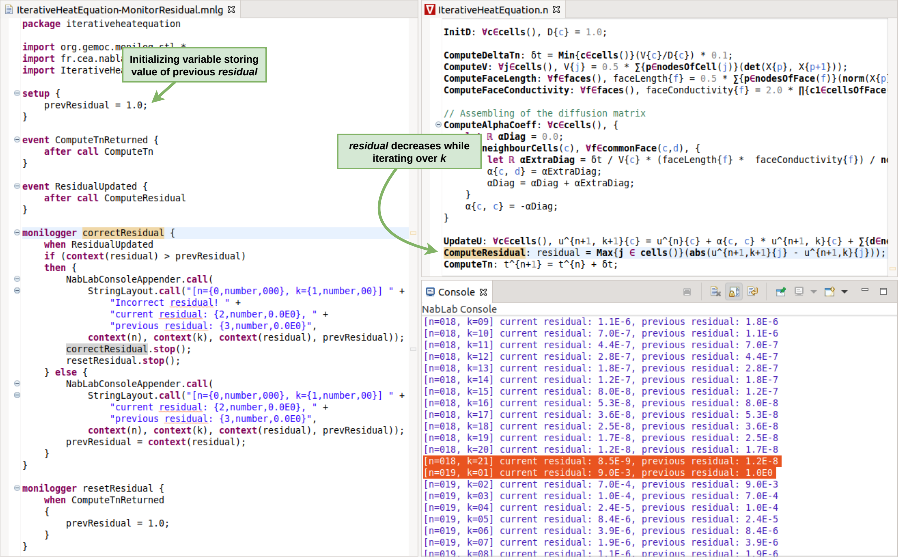
Console

NabLab Console

```
[n=018, k=09] current residual: 1.1E-6, previous residual: 1.8E-6
[n=018, k=10] current residual: 7.0E-7, previous residual: 1.1E-6
[n=018, k=11] current residual: 4.4E-7, previous residual: 7.0E-7
[n=018, k=12] current residual: 2.8E-7, previous residual: 4.4E-7
[n=018, k=13] current residual: 1.8E-7, previous residual: 2.8E-7
[n=018, k=14] current residual: 1.2E-7, previous residual: 1.8E-7
[n=018, k=15] current residual: 8.0E-8, previous residual: 1.2E-7
[n=018, k=16] current residual: 5.3E-8, previous residual: 8.0E-8
[n=018, k=17] current residual: 3.6E-8, previous residual: 5.3E-8
[n=018, k=18] current residual: 2.5E-8, previous residual: 3.6E-8
[n=018, k=19] current residual: 1.7E-8, previous residual: 2.5E-8
[n=018, k=20] current residual: 1.2E-8, previous residual: 1.7E-8
[n=018, k=21] current residual: 8.5E-9, previous residual: 1.2E-8
[n=019, k=01] current residual: 9.0E-3, previous residual: 1.0E0
[n=019, k=02] current residual: 7.0E-4, previous residual: 9.0E-3
[n=019, k=03] current residual: 1.0E-4, previous residual: 7.0E-4
[n=019, k=04] current residual: 2.4E-5, previous residual: 1.0E-4
[n=019, k=05] current residual: 8.4E-6, previous residual: 2.4E-5
[n=019, k=06] current residual: 3.9E-6, previous residual: 8.4E-6
[n=019, k=07] current residual: 1.9E-6, previous residual: 3.9E-6
[n=019, k=08] current residual: 1.1E-6, previous residual: 1.9E-6
```

residual decreases while iterating over *k*

Example use on NabLab



Example use on NabLab

IterativeHeatEquation-MonitorResidual.mnlg

```
package iterativeheatequation

import org.gemoc.monilog.stl.*
import fr.cea.nabl...
import IterativeHe...

setup {
    prevResidual = 1.0;
}

event ComputeTnReturned {
    after call ComputeTn
}

event ResidualUpdated {
    after call ComputeResidual
}

monilogger correctResidual {
    when ResidualUpdated
    if (context(residual) > prevResidual)
    then {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "Incorrect residual! " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        correctResidual.stop();
        resetResidual.stop();
    } else {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        prevResidual = context(residual);
    }
}

monilogger resetResidual {
    when ComputeTnReturned
    {
        prevResidual = 1.0;
    }
}
```

Initializing variable storing value of previous *residual*

Declaring events of interest

residual* decreases while iterating over *k

IterativeHeatEquation.n

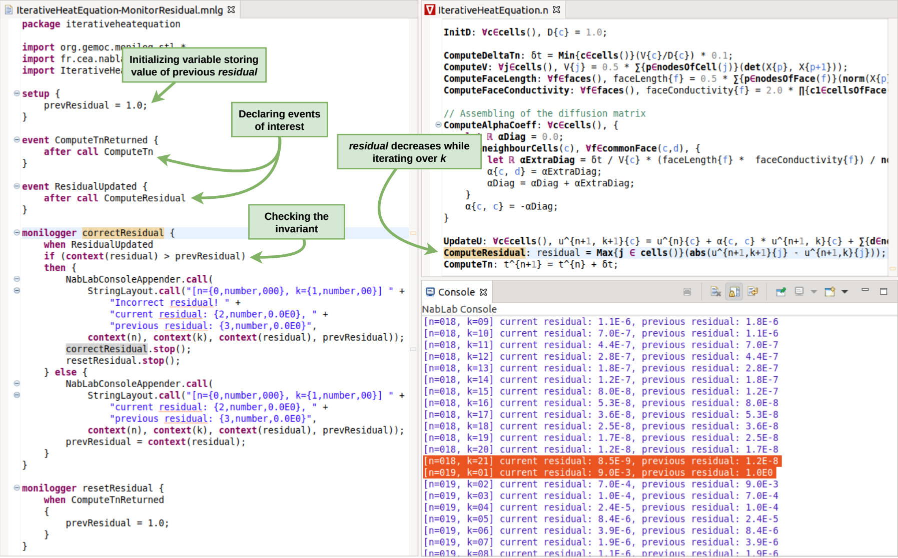
```
InitD: ∀c∈cells(), D{c} = 1.0;

ComputeDeltaTn: δt = Min{c∈cells()}(V{c}/D{c}) * 0.1;
ComputeV: ∀j∈cells(), V{j} = 0.5 * ∑{p∈nodesOfCell(j)}(det(X{p}, X{p+1}));
ComputeFaceLength: ∀f∈faces(), faceLength{f} = 0.5 * ∑{p∈nodesOfFace(f)}(norm(X{p]
ComputeFaceConductivity: ∀f∈faces(), faceConductivity{f} = 2.0 * ∏{c1∈cellsOfFace

// Assembling of the diffusion matrix
ComputeAlphaCoeff: ∀c∈cells(), {
    ... ℝ αDiag = 0.0;
    ... neighbourCells(c), ∀f∈commonFace(c,d), {
        let ℝ αExtraDiag = δt / V{c} * (faceLength{f} *  faceConductivity{f}) / n
        α{c, d} = αExtraDiag;
        αDiag = αDiag + αExtraDiag;
    }
    α{c, c} = -αDiag;
}

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} + ∑{d∈n
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));
ComputeTn: t^{n+1} = t^{n} + δt;
```

Console

NabLab Console

```
[n=018, k=09] current residual: 1.1E-6, previous residual: 1.8E-6
[n=018, k=10] current residual: 7.0E-7, previous residual: 1.1E-6
[n=018, k=11] current residual: 4.4E-7, previous residual: 7.0E-7
[n=018, k=12] current residual: 2.8E-7, previous residual: 4.4E-7
[n=018, k=13] current residual: 1.8E-7, previous residual: 2.8E-7
[n=018, k=14] current residual: 1.2E-7, previous residual: 1.8E-7
[n=018, k=15] current residual: 8.0E-8, previous residual: 1.2E-7
[n=018, k=16] current residual: 5.3E-8, previous residual: 8.0E-8
[n=018, k=17] current residual: 3.6E-8, previous residual: 5.3E-8
[n=018, k=18] current residual: 2.5E-8, previous residual: 3.6E-8
[n=018, k=19] current residual: 1.7E-8, previous residual: 2.5E-8
[n=018, k=20] current residual: 1.2E-8, previous residual: 1.7E-8
[n=018, k=21] current residual: 8.5E-9, previous residual: 1.2E-8
[n=019, k=01] current residual: 9.0E-3, previous residual: 1.0E0
[n=019, k=02] current residual: 7.0E-4, previous residual: 9.0E-3
[n=019, k=03] current residual: 1.0E-4, previous residual: 7.0E-4
[n=019, k=04] current residual: 2.4E-5, previous residual: 1.0E-4
[n=019, k=05] current residual: 8.4E-6, previous residual: 2.4E-5
[n=019, k=06] current residual: 3.9E-6, previous residual: 8.4E-6
[n=019, k=07] current residual: 1.9E-6, previous residual: 3.9E-6
[n=019, k=08] current residual: 1.1E-6, previous residual: 1.9E-6
```

Example use on NabLab



IterativeHeatEquation-MonitorResidual.mnlg

```
package iterativeheatequation

import org.gemoc.monilog.stl.*
import fr.cea.nabl
import IterativeHe

setup {
    prevResidual = 1.0;
}

event ComputeTnReturned {
    after call ComputeTn
}

event ResidualUpdated {
    after call ComputeResidual
}

monilogger correctResidual {
    when ResidualUpdated
    if (context(residual) > prevResidual)
    then {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "Incorrect residual! " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        correctResidual.stop();
        resetResidual.stop();
    } else {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        prevResidual = context(residual);
    }
}

monilogger resetResidual {
    when ComputeTnReturned
    {
        prevResidual = 1.0;
    }
}
```

Initializing variable storing value of previous *residual*

Declaring events of interest

Checking the invariant

Logging the values of interest and storing *residual* for next iteration

residual decreases while iterating over *k*

IterativeHeatEquation.n

```
InitD: ∀c∈cells(), D{c} = 1.0;

ComputeDeltaTn: δt = Min{c∈cells()}(V{c}/D{c}) * 0.1;
ComputeV: ∀j∈cells(), V{j} = 0.5 * ∑{p∈nodesOfCell(j)}(det(X{p}, X{p+1}));
ComputeFaceLength: ∀f∈faces(), faceLength{f} = 0.5 * ∑{p∈nodesOfFace(f)}(norm(X{p
ComputeFaceConductivity: ∀f∈faces(), faceConductivity{f} = 2.0 * ∏{c1∈cellsOfFace

// Assembling of the diffusion matrix
ComputeAlphaCoeff: ∀c∈cells(), {
    ℝ αDiag = 0.0;
    neighbourCells(c), ∀f∈commonFace(c,d), {
        let ℝ αExtraDiag = δt / V{c} * (faceLength{f} *  faceConductivity{f}) / n
        α{c, d} = αExtraDiag;
        αDiag = αDiag + αExtraDiag;
    }
    α{c, c} = -αDiag;
}

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} + ∑{d∈n
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));
ComputeTn: t^{n+1} = t^{n} + δt;
```

Console — NabLab Console

```
[n=018, k=09] current residual: 1.1E-6, previous residual: 1.8E-6
[n=018, k=10] current residual: 7.0E-7, previous residual: 1.1E-6
ent residual: 4.4E-7, previous residual: 7.0E-7
ent residual: 2.8E-7, previous residual: 4.4E-7
ent residual: 1.8E-7, previous residual: 2.8E-7
[n=018, k=14] current residual: 1.2E-7, previous residual: 1.8E-7
[n=018, k=15] current residual: 8.0E-8, previous residual: 1.2E-7
[n=018, k=16] current residual: 5.3E-8, previous residual: 8.0E-8
[n=018, k=17] current residual: 3.6E-8, previous residual: 5.3E-8
[n=018, k=18] current residual: 2.5E-8, previous residual: 3.6E-8
[n=018, k=19] current residual: 1.7E-8, previous residual: 2.5E-8
[n=018, k=20] current residual: 1.2E-8, previous residual: 1.7E-8
[n=018, k=21] current residual: 8.5E-9, previous residual: 1.2E-8
[n=019, k=01] current residual: 9.0E-3, previous residual: 1.0E0
[n=019, k=02] current residual: 7.0E-4, previous residual: 9.0E-3
[n=019, k=03] current residual: 1.0E-4, previous residual: 7.0E-4
[n=019, k=04] current residual: 2.4E-5, previous residual: 1.0E-4
[n=019, k=05] current residual: 8.4E-6, previous residual: 2.4E-5
[n=019, k=06] current residual: 3.9E-6, previous residual: 8.4E-6
[n=019, k=07] current residual: 1.9E-6, previous residual: 3.9E-6
[n=019, k=08] current residual: 1.1E-6, previous residual: 1.9E-6
```

Example use on NabLab

IterativeHeatEquation-MonitorResidual.mnlg

```
package iterativeheatequation

import org.gemoc.monilog.stl.*
import fr.cea.nabl
import IterativeHe

setup {
    prevResidual = 1.0;
}

event ComputeTnReturned {
    after call ComputeTn
}

event ResidualUpdated {
    after call ComputeResidual
}

monilogger correctResidual {
    when ResidualUpdated
    if (context(residual) > prevResidual)
    then {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "Incorrect residual! " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        correctResidual.stop();
        resetResidual.stop();
    } else {
        NabLabConsoleAppender.call(
            StringLayout.call("[n={0,number,000}, k={1,number,00}] " +
                "current residual: {2,number,0.0E0}, " +
                "previous residual: {3,number,0.0E0}",
            context(n), context(k), context(residual), prevResidual));
        prevResidual = context(residual);
    }
}

monilogger resetResidual {
    when ComputeTnReturned
    {
        prevResidual = 1.0;
    }
}
```

Initializing variable storing value of previous *residual*

Declaring events of interest

Checking the invariant

Resetting stored *residual* to 1.0 after each iteration over *n*

IterativeHeatEquation.n

```
InitD: ∀c∈cells(), D{c} = 1.0;

ComputeDeltaTn: δt = Min{c∈cells()}(V{c}/D{c}) * 0.1;
ComputeV: ∀j∈cells(), V{j} = 0.5 * ∑{p∈nodesOfCell(j)}(det(X{p}, X{p+1}));
ComputeFaceLength: ∀f∈faces(), faceLength{f} = 0.5 * ∑{p∈nodesOfFace(f)}(norm(X{p
ComputeFaceConductivity: ∀f∈faces(), faceConductivity{f} = 2.0 * ∏{c1∈cellsOfFace

// Assembling of the diffusion matrix
ComputeAlphaCoeff: ∀c∈cells(), {
    ℝ αDiag = 0.0;
    neighbourCells(c), ∀f∈commonFace(c,d), {
        let ℝ αExtraDiag = δt / V{c} * (faceLength{f} *  faceConductivity{f}) / n
        α{c, d} = αExtraDiag;
        αDiag = αDiag + αExtraDiag;
    }
    α{c, c} = -αDiag;
}

UpdateU: ∀c∈cells(), u^{n+1, k+1}{c} = u^{n}{c} + α{c, c} * u^{n+1, k}{c} + ∑{d∈n
ComputeResidual: residual = Max{j ∈ cells()}(abs(u^{n+1,k+1}{j} - u^{n+1,k}{j}));
ComputeTn: t^{n+1} = t^{n} + δt;
```

residual* decreases while iterating over *k

Console

NabLab Console

```
[n=018, k=09] current residual: 1.1E-6, previous residual: 1.8E-6
[n=018, k=10] current residual: 7.0E-7, previous residual: 1.1E-6
ent residual: 4.4E-7, previous residual: 7.0E-7
ent residual: 2.8E-7, previous residual: 4.4E-7
ent residual: 1.8E-7, previous residual: 2.8E-7
[n=018, k=14] current residual: 1.2E-7, previous residual: 1.8E-7
[n=018, k=15] current residual: 8.0E-8, previous residual: 1.2E-7
[n=018, k=16] current residual: 5.3E-8, previous residual: 8.0E-8
[n=018, k=17] current residual: 3.6E-8, previous residual: 5.3E-8
[n=018, k=18] current residual: 2.5E-8, previous residual: 3.6E-8
[n=018, k=19] current residual: 1.7E-8, previous residual: 2.5E-8
[n=018, k=20] current residual: 1.2E-8, previous residual: 1.7E-8
[n=018, k=21] current residual: 8.5E-9, previous residual: 1.2E-8
[n=019, k=01] current residual: 9.0E-3, previous residual: 1.0E0
[n=019, k=02] current residual: 7.0E-4, previous residual: 9.0E-3
[n=019, k=03] current residual: 1.0E-4, previous residual: 7.0E-4
[n=019, k=04] current residual: 2.4E-5, previous residual: 1.0E-4
[n=019, k=05] current residual: 8.4E-6, previous residual: 2.4E-5
[n=019, k=06] current residual: 3.9E-6, previous residual: 8.4E-6
[n=019, k=07] current residual: 1.9E-6, previous residual: 3.9E-6
[n=019, k=08] current residual: 1.1E-6, previous residual: 1.9E-6
```

Logging the values of interest and storing *residual* for next iteration

Implementation – AspectJ

- Applicable to **Java-based** interpreters
- **Non-intrusive** w.r.t. language definition
- Instrumentation interface = aspects **weaved into the interpreter**:

```
pointcut interpreteJob(Job job, Context context) :
call(public static void fr.cea.nabla.ir.interpreter.JobInterpreter.interprete(Job, Context)) &&
        args(job, context);

after(Job job, Context context) : interpreteJob(job, context) {
    notifyAfter(job.getName(), null, context);
}

before(Job job, Context context) : interpreteJob(job, context) {
    notifyBefore(job.getName(), context);
}
```

Implementation – Truffle language implementation framework

- Applicable to languages with a **Truffle-based** interpreter (*e.g.*, Python, R)
- Can evaluate expressions in **any language** installed on the GraalVM
- Instrumentation interface **part of language definition**:

```
public abstract class NablaWriteVariableNode
        extends NablaInstructionNode
        implements InstrumentableNode, TruffleObject {
    @Override
    public boolean hasTag(Class<? extends Tag> tag) {
        return tag.equals(StandardTags.WriteVariableTag.class) || super.hasTag(tag);
    }
```

Evaluation

Research questions:

RQ#1 How does the proposed approach allow to combine runtime monitoring and logging to extract relevant data from running models?

- Answered through demonstration cases similar to the provided example.

RQ#2 How is the overhead induced by the approach affected by different scenarios?

- Answered through quantitative evaluation

RQ#1: Demonstration case (Coarsen Interval)

IterativeHeatEquation-CoarsenInterval.mnlg

```
package heatequation

import org.gemoc.monilog.stl.*
import fr.cea.nabla.monilog.nablalib.*
import IterativeHeatEquation.*

import "/home/vagrant/workspace/NabLabExamples/Utils.js" as jsutils

setup {
    currentTime = 0.0;
    outputInterval = 0.0001;
    stdev = 1.0;
    dateString = js(jsutils.getDate);
    filePath = "/home/vagrant/workspace/NabLabExamples/dumps/";
}

event ComputeTnReturned {
    after call ComputeTn
}

monilogger LogTemperature {
    when ComputeTnReturned
    if (currentTime + outputInterval <= context(t_n))
    then {
        stdev = js(jsutils.stdev(context(u_n)));
        FileAppender.call(
            StringLayout.call(
                "{0,number,0.0000}, {1,number,0.000E0}",
                context(t_n),
                stdev),
            StringLayout.call("{0}iterativeheatequation-{1}.csv", filePath, dateString));
        currentTime = currentTime + outputInterval;
    }
}

monilogger CoarsenInterval {
    when ComputeTnReturned
    if (stdev <= 0.20)
    then {
        outputInterval = 0.01;
        CoarsenInterval.stop;
    }
}
```

- Log standard deviation of `u_n` to file at interval of `0.0001`
- When standard deviation less than `0.2`, increase interval to `0.01`

- Derived data leveraged by monitor
- Monitor affects logger behavior

RQ#2: Quantitative evaluation

Setup:
CPU: Intel® Core™ i7-9850H CPU @ 2.60GHz $\times$ 12
OS: Ubuntu 20.04.2, VM: GraalVM 21.1.0

Overhead induced by 3 MoniLog specifications over simulation times from 0.2 to 1.0

AspectJ:

baseline:	from ≈42.6s	to ≈134.89s
absolute:	from ≈8.75s	to ≈18.27s
relative:	from ≈26%	to ≈16%

Truffle:

baseline:	from ≈10.21s	to ≈29.76s
absolute:	from ≈2.75s	to ≈4.85s
relative:	from ≈36.5%	to ≈19.5%

- Suitable for **debugging** as absolute overhead reasonably low on shorter execution times, and relative overhead decreases by 40 to 50% on longer execution times.

MoniLog for compiled DSLs (ongoing work)

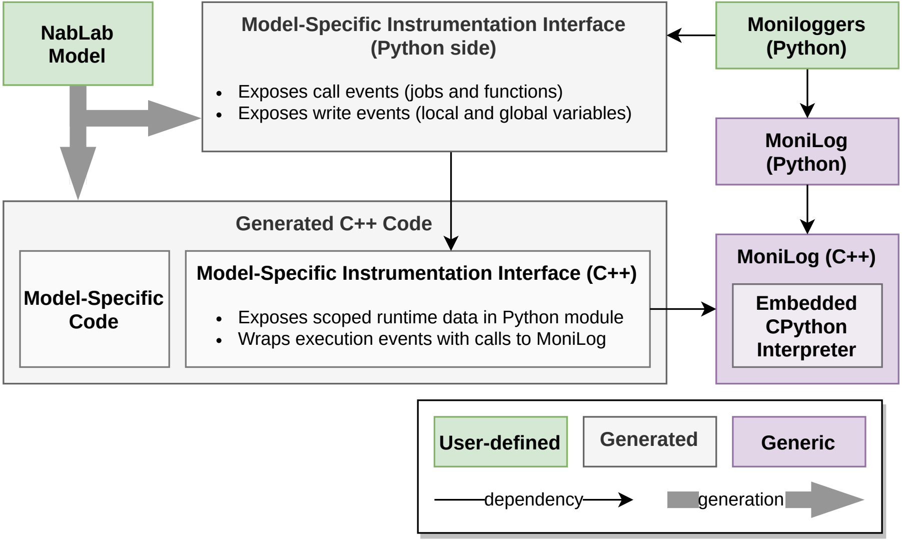
Prerequisites:

- MoniLog host language interpreter embeddable in target language
- Extend code generators to generate model-specific instrumentation interface

The generated model-specific instrumentation interface is split between:

- the target language of the DSL's code generator, to expose runtime data and events of the model, and
- the host language of MoniLog, to execution events of the model from moniloggers.

MoniLog for compiled DSLs (ongoing work)



Example use of Python-based MoniLog

src > iterativeheatequation > logStdev.py > ...

```
import iterativeheatequation as ihe
from monilog import *
import statistics

currentTime = 0
outputInterval = 0.0001
stdev = 1.0

@after(ihe.ComputeTn)
def logTemperature(context):
    global currentTime
    global outputInterval
    global stdev
    currentTime += outputInterval
    if (currentTime <= context.t_n):
        stdev = statistics.stdev(context.u_n)
        print("[" + str(context.t_n) + "] stdev=" + str(stdev))

@after(ihe.ComputeTn)
def coarsenInterval(context):
    global stdev
    global outputInterval
    if (stdev <= 0.20):
        outputInterval = 0.01
        coarsenInterval.stop()
```

Conclusion

- High-level languages allow scientists and numerical analysts to focus on their area of expertise and **associated V&V concerns**.
- Designers of high-level languages must **guarantee correctness and performance** of derived scientific software.
- Designers must furthermore **give tools to address the V&V concerns** corresponding to the level of abstraction of the language.
- In the context of **languages for numerical schemes** such as NabLab, MoniLog is particularly suited to this thanks to:
  - its combination of **logging and monitoring**,
  - its ability to use the **best suited languages** for the task (*e.g.*, Python for data analysis)

Thank you for your attention!

When Scientific Software Meets Software Engineering
Leroy, Dorian, Sallou, June, Bourcier, Johan, Combemale, Benoit Computer 2021

Monilogging for executable domain-specific languages
Leroy, Dorian, Lelandais, Benoît, Oudot, Marie-Pierre, Combemale, Benoit In Proceedings of the 14th ACM SIGPLAN International Conference on Software Language Engineering 2021