

When Scientific Software Meets Software Engineering

Software Engineering for Scientific Computing

Prof. Benoit Combemale
University of Rennes 1
DiverSE team (IRISA & Inria)

<http://combemale.fr> / [@bcombemale](#)

“Software Is Eating the World”

Digitalization of our society

- personal context (health, music, video, social networks...)
- professional context (digitalization of numerous processes and activities)



“Every company is a software company. You have to start thinking and operating like a digital company. It’s no longer just about procuring one solution and deploying one. It’s not about one simple software solution. It’s really you yourself thinking of your own future as a digital company.”

— Satya Nadella, CEO, Microsoft



(Scientific) Software Is Eating the World

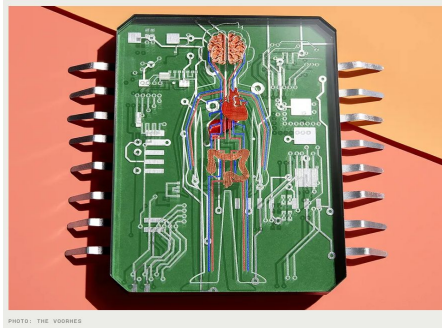
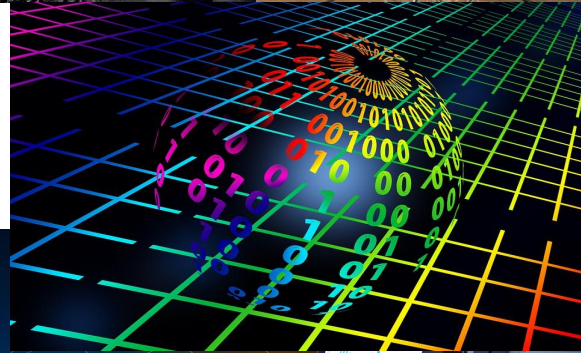
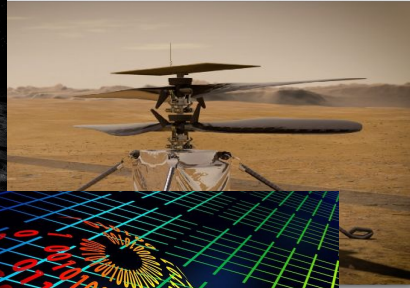
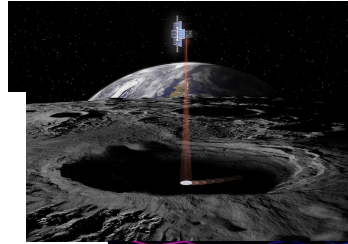
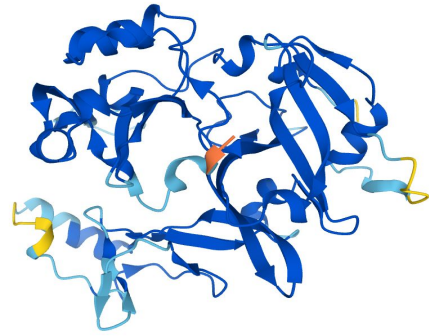
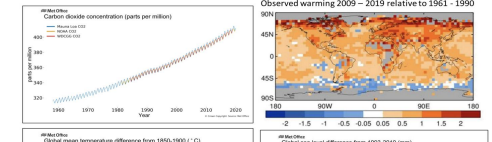


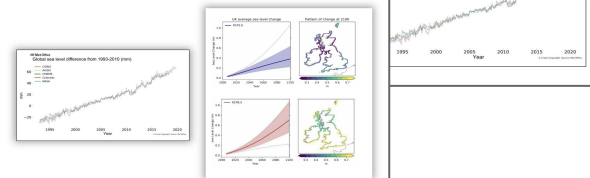
PHOTO: THE VOORDES



Climate is changing



The sea level is rising



Scientific Computing

In science

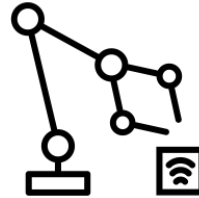
- holistic view of wicked problems
- what-if scenarios

In industry

- design of complex systems
- digital twins

In society

- decision and policy makers
- broader engagement
- education



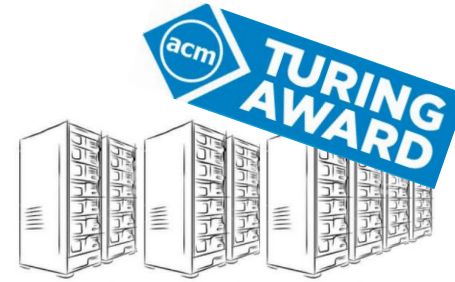
Sound
knowledge

Innovative
system

Smart
people

Scientific Computing

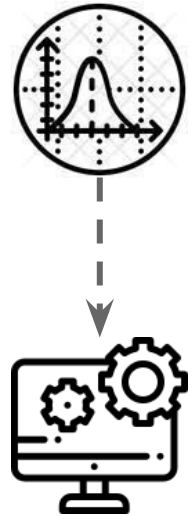
- Use of **advanced computing capabilities**
- To **understand** and **solve**
 - scientific problems (e.g., biological, physical, and social),
 - engineering problems, and
 - humanities problems.
- And **predict** the behavior or the outcome of a physical system, being natural or man-made.



Scientific Software

Rely on the development of **mathematical models** to understand physical systems through their **simulations**.

- Mathematical models belong to numerical models (continuous or discrete) and analytics.
- Simulations of mathematical models correspond to the execution of the computer programs containing these models, the so-called "simulation codes".
- Scientific software = software dedicated to scientific computing and simulation.

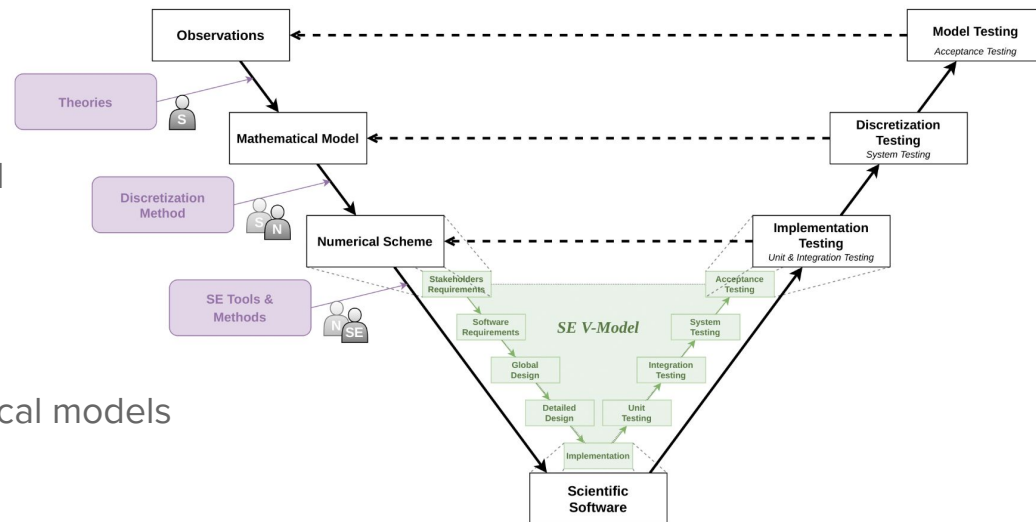


Scientific Software Specificities

- Specific **software**
 - highly iterative process,
 - large dataset,
 - complex compilation chains and deployments,
 - long lifetime application,
 - *usually, a means not an end.*
- Specific **stakeholders**
 - developers: scientists (physicist, mathematicians...), engineers, numerical analyst...
 - end-users: from the developer itself to decision makers and the general public
 - funders: possibly third parties, without any background on software development
 - ...

Scientific Software Development

- Scientific V-Model subsumes SE V-Model
- Multidisciplinary development
- Collaborative (time/space) development
- Highly configurable / variable mathematical models

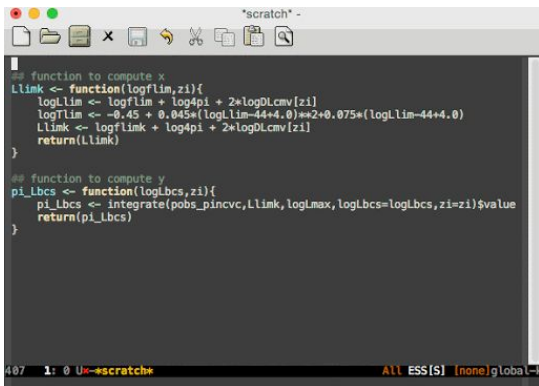


When Scientific Software Meets Software Engineering

D. Leroy, J. Sallou, J. Bourcier and B. Combemale. Computer, vol. 54, no. 12, pp. 60-71, 2021.
Preprint: <https://hal.inria.fr/hal-03318348v1>

From Scientific Coding...

~Syntactic support



```
## function to compute x
Llink <- function(logflim,zi){
  logLlim <- logflim + log4pi + 2*logDcmv[zi]
  logTlim <- -0.45 + 0.045*(logLlim-44+4.0)**2+0.075*(logLlim-44+4.0)
  Llink <- logflim + log4pi + 2*logDcmv[zi]
  return(Llink)
}

## function to compute y
pi_lbcs <- function(logLbcs,zi){
  pi_lbcs <- integrate(pobs_pincvc, Llink, logLmax, logLbcs=logLbcs, zi=zi)$value
  return(pi_lbcs)
}
```

From Scientific Coding...

~Syntactic support

```
function to compute x
LLink <- function(logflim,zi){
  logLlim <- logflim + log4pi + 2*logDlcmv[zi]
  logLlim <- -0.45 + 0.045*(logLlim-44+4.0)**2+0.075*(logLlim-44+4.0)
  LLink <- logflim + log4pi + 2*logDlcmv[zi]
  return(LLink)
}

function to compute y
pi_Lbcs <- function(logLbcs,zi){
  pi_Lbcs <- integrate(pobs_pincvc,LLink,logLmax,logLbcs=logLbcs,zi=zi)$value
  return(pi_Lbcs)
}
```

... to Structured and Sound Programming

- ▶ **Abstractions** (modularity, resources, computation...)
- ▶ **Automation** (dev/doc/test, compilation/integration, deployment, delivery...)
- ▶ **Validation & Verification**



The screenshot displays a code editor with R code on the left, a plot window showing two overlapping curves (one blue, one green) on the right, and a table of test results at the bottom. The code includes comments and function definitions. The plot shows two curves, one blue and one green, with the blue curve having a higher peak. The table below the code has columns for 'Test', 'Pass', 'Fail', 'Error', and 'Warning', with rows of test results.

From Scientific Coding...

~Syntactic support

```
function to compute x
LLink <- function(logflim,zi){
  logLlim <- logflim + log4pi + 2*logDcmv[zi]
  logLlim <- -0.45 + 0.045*(logLlim-44+4.0)
  LLink <- logflim + log4pi + 2*logDcmv[zi]
  return(LLink)
}

function to compute y
pi_Lbcs <- function(logLbcs,zi){
  pi_Lbcs <- integrate(pobs_pincvc, LLink, logLmax, logLbcs=logLbcs, zi=zi)$value
  return(pi_Lbcs)
}
```

Are they really fitting specificities of scientific software?

... to Structured and Sound Programming

- ▶ **Abstractions** (modularity, resources, computation...)
- ▶ **Automation** (dev/doc/test, compilation/integration, deployment, delivery...)
- ▶ **Validation & Verification**



The screenshot displays a code editor with R code for data analysis and plotting. A plot window shows two overlapping curves, one blue and one green. Below the code, a table lists test results for various components.

	True	False	True	False	False
1	True	False	True	False	False
2	True	False	False	False	False
3	True	False	False	False	False
4	True	False	False	False	False
5	True	False	False	False	False

Momentum

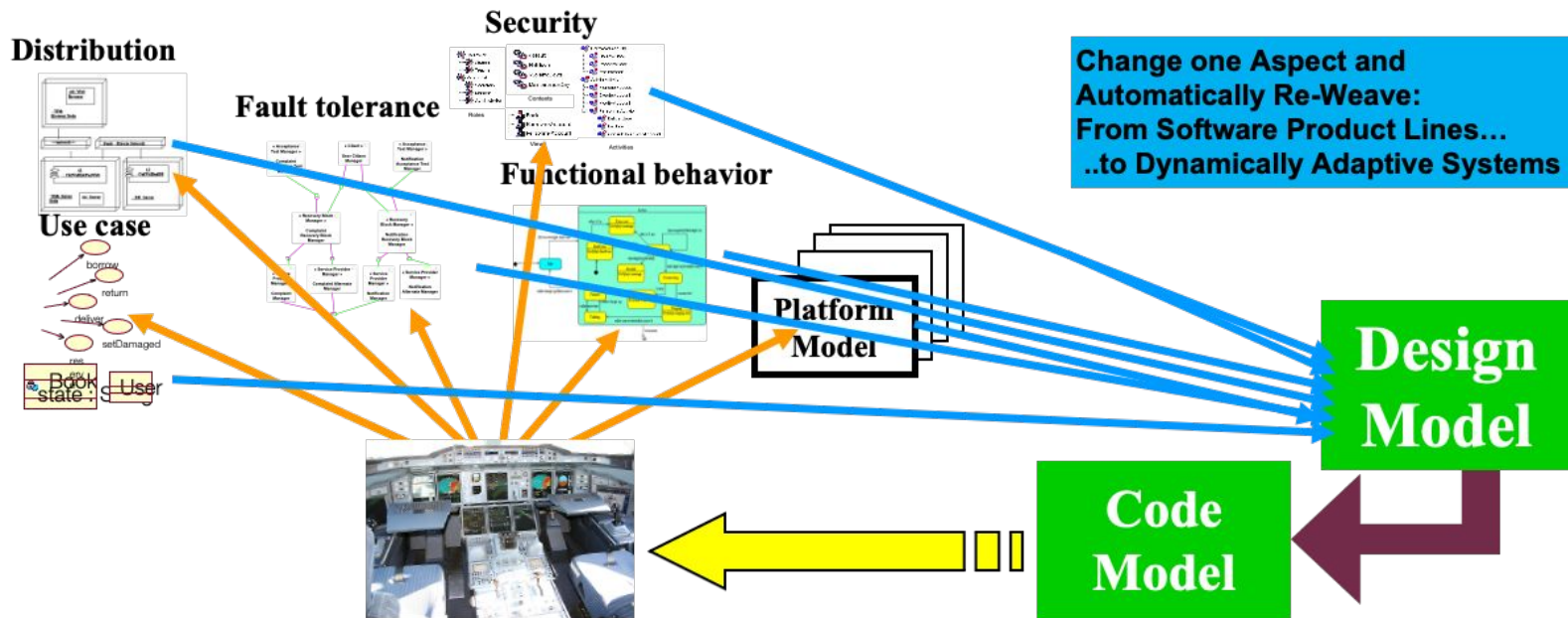
- Ever-increasing intrinsic complexity of mathematical models
- Broader engagement of various heterogeneous stakeholders
- Numerous scenarios to evaluate
- Ever more efficient and large simulations

⇒ It urges to establish the required software engineering foundations, tools and methods for scientific computing

⇒ The SE4Science Initiative!

On Software Languages

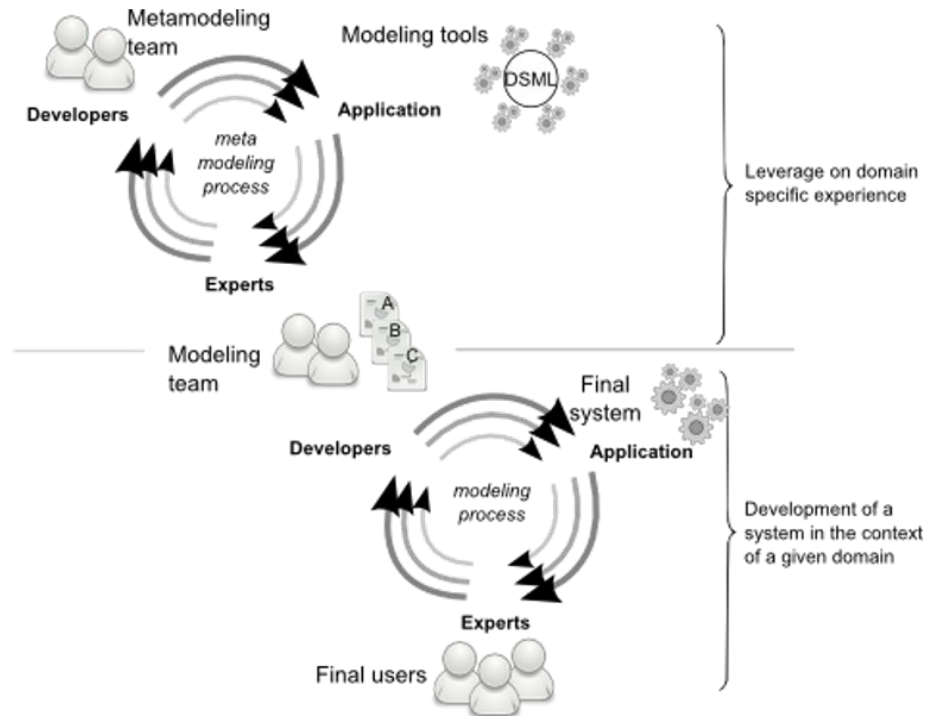
Model-Driven Engineering



Engineering Modeling Languages: Turning Domain Knowledge into Tools

Benoit Combemale, Robert B. France, Jean-Marc Jézéquel, Bernhard Rumpe, Jim R.H. Steel, and Didier Vojtisek
Chapman and Hall/CRC, pp.398, 2016. Companion website: <http://mdebook.irisa.fr>

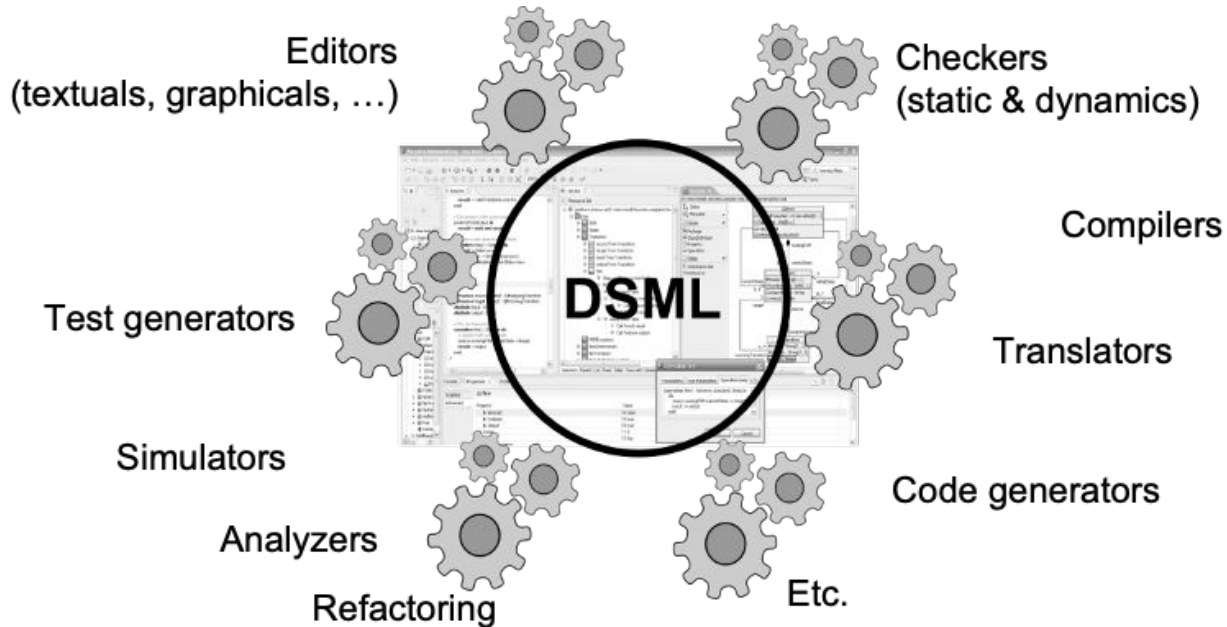
Model-Driven Engineering



Engineering Modeling Languages: Turning Domain Knowledge into Tools

Benoit Combemale, Robert B. France, Jean-Marc Jézéquel, Bernhard Rumpe, Jim R.H. Steel, and Didier Vojtisek
Chapman and Hall/CRC, pp.398, 2016. Companion website: <http://mdebook.irisa.fr>

Model-Driven Engineering



Engineering Modeling Languages: Turning Domain Knowledge into Tools

Benoit Combemale, Robert B. France, Jean-Marc Jézéquel, Bernhard Rumpe, Jim R.H. Steel, and Didier Vojtisek
Chapman and Hall/CRC, pp.398, 2016. Companion website: <http://mdebook.irisa.fr>

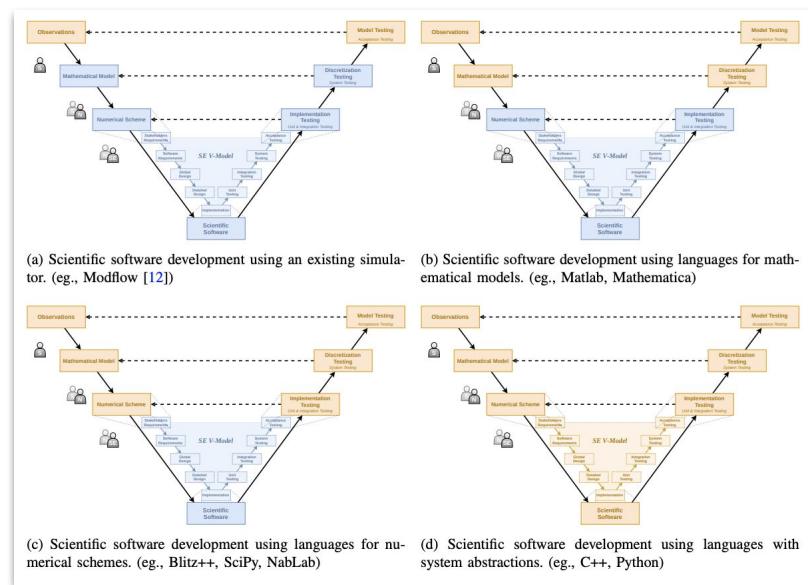
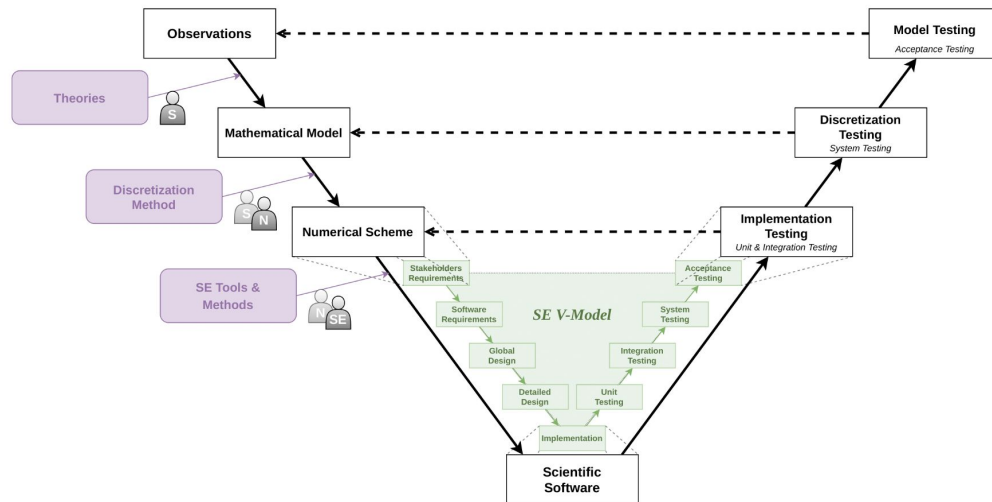
***"Software languages
are software too"***

Software Language Engineering

- Application of systematic, disciplined, and measurable approaches to the development, deployment, use, and maintenance of software languages
- Supported by various kind of "**language workbench**"
 - Eclipse EMF, Xtext, Sirius, Melange, GEMOC, Papyrus
 - JetBrains's MPS
 - Spoofox
 - MS DSL Tools
 - etc.
- Various shapes and ways to implement software languages
 - External, internal or embedded DSLs, Profile, etc.
 - Grammar, metamodel, ontology, etc.
- More and more literature, a dedicated Intl. conference (ACM SLE, cf. <http://www.sleconf.org>)...

Software Languages in Scientific Computing

The more general-purpose the language is the more flexibility it will provide, but also the more rigorous engineering principles and V&V activities it will require from the language user



When Scientific Software Meets Software Engineering

D. Leroy, J. Sallou, J. Bourcier and B. Combemale. Computer, vol. 54, no. 12, pp. 60-71, 2021. 20
Preprint: <https://hal.inria.fr/hal-03318348v1>

Software Languages in Scientific Computing

Language	Mathematical Model	Numerical Scheme	Scientific Software
Mathematica (Wolfram Language)	+++	++	
MATLAB	++	++	
R	+	+	
NabLab		+++	
Julia		++	+
SciPy		++	+
Python			+
Java			++
C/C++			+++
Fortran		++	+++

When Scientific Software Meets Software Engineering

D. Leroy, J. Sallou, J. Bourcier and B. Combemale. Computer, vol. 54, no. 12, pp. 60-71, 2021. 21

Preprint: <https://hal.inria.fr/hal-03318348v1>



Domain-Specific Languages: NabLab

The screenshot shows the Eclipse IDE with the NabLab project open. The main editor displays Fortran code for a cell-centered Godunov scheme. A red callout box labeled "Textual Editor" points to the code. On the left, the Outline view shows a tree of code blocks, with a red callout box labeled "Outline" pointing to it. On the right, the Glace Ir Graph view shows a complex data flow graph, with a red callout box labeled "Data Flow Graph" pointing to it. At the bottom, the LaTeX View shows a mathematical expression for the time step calculation, with a red callout box labeled "Latex View" pointing to it.

```

InICenter:  $\forall j \in \text{cells}, \text{center}\{j\} = (1.0/4.0) * \sum_{r \in \text{nodes}\{j\}} X_{ic}(r);$ 
InIc:  $\forall j \in \text{cells}, \text{if } (\text{center}\{j\}.x < \text{option\_x\_interface}) \{$ 
   $\rho_{ic}\{j\} = \text{option\_p\_ini\_zg};$ 
   $\rho_{ic}\{j\} = \text{option\_p\_ini\_zg};$ 
 $\}$ 
else {
   $\rho_{ic}\{j\} = \text{option\_p\_ini\_zd};$ 
   $\rho_{ic}\{j\} = \text{option\_p\_ini\_zd};$ 
 $\}$ 

ComputeCjrIc:  $\forall j \in \text{cells}, \forall r \in \text{nodes}\{j\}, C_{ic}(j,r) = 0.5 * \text{perp}(X_{ic}(-r), X_{ic}(r));$ 
// pas 0.5 en dimension 3. 1/d
InIVic:  $\forall j \in \text{cells}, V_{ic}\{j\} = 0.5 * \sum_{r \in \text{nodes}\{j\}} (\text{dot}(C_{ic}(j,r), X_{ic}(r)));$ 
InIM:  $\forall j \in \text{cells}, m\{j\} = \rho_{ic}\{j\} * V_{ic}\{j\};$  // m est constant

// *****
// * Calcul des C(j,r) et des variables qui en decoulent
// *****
ComputeCjr:  $\forall j \in \text{cells}, \forall r \in \text{nodes}\{j\}, C(j,r) = 0.5 * \text{perp}(X(-r), X(r));$ 
ComputeAbsjr:  $\forall j \in \text{cells}, \forall r \in \text{nodes}\{j\}, \text{abs}(j,r) = \text{norm}(C(j,r));$ 
ComputeDtj:  $\forall j \in \text{cells}, \delta t\{j\} = 2.0 * V(j) / (c(j) * \sum_{r \in \text{nodes}\{j\}} (\text{abs}(j,r)));$ 

// *****
// * Règles EOS standards: m, p, c, p, e
// *****
ComputeDensity:  $\forall j \in \text{cells}, \rho\{j\} = m\{j\} / V(j);$ 
ComputeEOSp:  $\forall j \in \text{cells}, p\{j\} = (v-1.0) * \rho\{j\} * e\{j\};$ 
ComputeInternalEnergy:  $\forall j \in \text{cells}, e\{j\} = E(j) - 0.5 * \text{dot}(u(j), u(j));$ 
ComputeEOSc:  $\forall j \in \text{cells}, c\{j\} = \text{sqr}t(v * p\{j\} / \rho\{j\});$ 

// *****
// * Cell-centered Godunov Scheme for Lagrangian gas dynamics
// *****
ComputeAjr:  $\forall j \in \text{cells}, \forall r \in \text{nodes}\{j\}, A(j,r) = ((p\{j\} * c\{j\}) / \text{abs}(j,r)) * \text{tensProduct}(C(j,r), C(j,r));$ 
ComputeFjr:  $\forall j \in \text{cells}, \forall r \in \text{nodes}\{j\}, F(j,r) = p\{j\} * C(j,r) + \text{matVectProduct}(A(j,r), (u\{j\} - fu(r)));$ 
ComputeMr:  $\forall r \in \text{nodes}, M(r) = \sum_{j \in \text{cells}(r)} (A(j,r));$ 

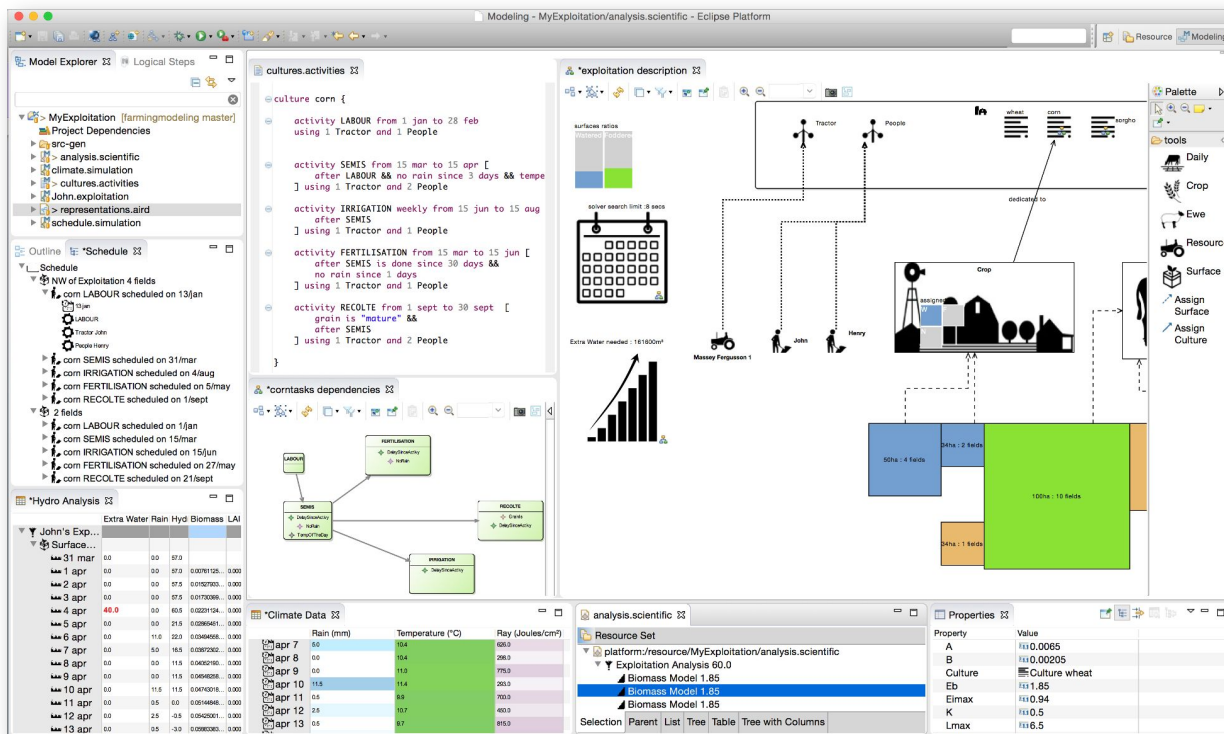
```

$$\forall j \in \text{cells}, \delta t_j = \frac{2.0 \cdot V_j}{(c_j \cdot \sum_{r \in \text{nodes}(j)} (\text{abs} C_{j,r}))}$$

Fostering metamodels and grammars within a dedicated environment for HPC: the NabLab environment (tool demo).

Benoît Lelandais, Marie-Pierre Oudot, Benoît Combemale. SLE 2018: 200-204

Domain-Specific Languages: “DSL de Vache”



The screenshot displays the Eclipse IDE with the DSL de Vache modeling environment. The main components are:

- Model Explorer:** Shows the project structure for 'MyExploitation [farmingmodeling master]', including 'Project Dependencies', 'analysis.scientific', 'climate.simulation', 'cultures.activities', 'John.exploitation', 'representations.aird', and 'schedule.simulation'.
- Logical Steps:** Shows the 'Schedule' outline with tasks like 'LABOUR', 'SEMIS', 'IRRIGATION', 'FERTILISATION', and 'RECOLTE' scheduled on various dates.
- Code Editor:** Contains the DSL code for 'cultures.activities', defining activities like 'LABOUR', 'SEMIS', 'IRRIGATION', 'FERTILISATION', and 'RECOLTE' with their respective resource requirements and conditions.
- Diagram Editor:** Visualizes the 'exploitation description' with a network diagram showing resources (Tractor, People, Ewe, Surface) and their relationships (dedicated to, search limit, etc.).
- Hydro Analysis:** A table showing 'Extra Water Rain Hyd Biomass LAI' over time.
- Climate Data:** A table showing 'Rain (mm)', 'Temperature (°C)', and 'Ray (Moules/cm²)' from April 7 to April 13.
- Properties:** A table showing properties for the 'Biomass Model 1.85' resource set.

MDE in Practice for Computational Science

Jean-Michel Bruel, Benoit Combemale, Ileana Ober, H el ene Raynal. ICCS, 2015.

Preprint: <https://hal.inria.fr/hal-01141393>

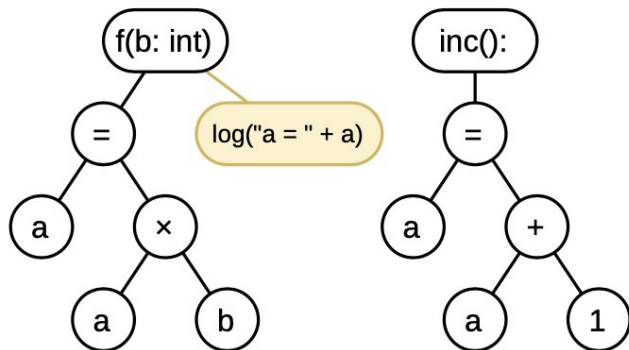
On Scientific Software Debugging

Debugging in Scientific Computing

- Common debugging facilities (i.e., step-by-step execution) are not suitable for numerical schemes with highly iterative processing
- Common **debug use cases** in scientific computing:
 - Conditional breakpoints
 - Constraint checking
 - Validation rules
 - Logging of values
- These use case each revolve around **runtime monitoring** and **logging**, respectively to:
 - determine when to perform an action (e.g., format a message), and
 - communicate the result to either the user, or a component (e.g., the debugger).

Debugging in Scientific Computing

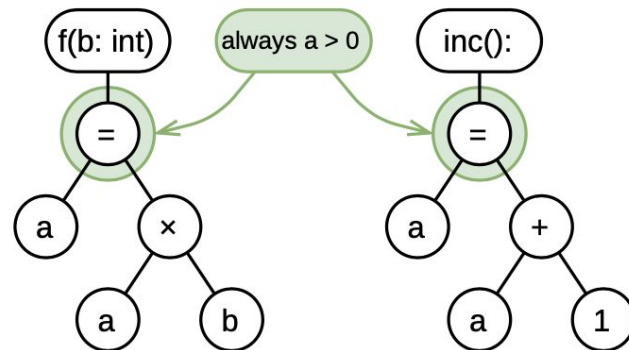
Logging:



Loggers are weaved in the AST to output context-specific messages.

- Requires **structural pointcuts**

Runtime monitoring:



Runtime monitors observe the execution to render a verdict on properties.

- Define **behavioral pointcuts**

Cross-fertilization of Logging & Runtime Monitoring?

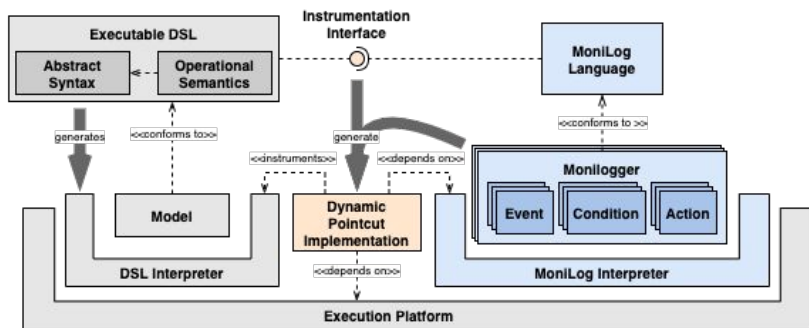
A few examples of cross-fertilization between logging and monitoring:

- Logging the steps in the evaluation of a temporal property
 - e.g., print the normal of a shock wave while checking it eventually reverses.
 - Logging benefits monitoring.
- log values in arbitrary complex situations:
 - e.g., log a message when the pressure becomes negative.
 - Monitoring benefits logging.

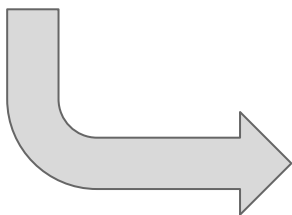
MoniLog (soon renamed '*SciHook*')

- Analyzing complex or data-intensive behaviors requires insightful data
 - alternative to debugging in scientific computing
- MoniLog: a unifying framework for defining:
 - **loggers**: extract data from program state and format it as messages
 - **runtime monitors**: evaluation of temporal properties on programs
 - **moniloggers**: combinations of loggers and monitors
- Moniloggers are defined in a language-agnostic way, relying on an instrumentation interface provided by DSLs
 - applied to any DSLs
 - keep monitoring and logging concerns out of domain concerns

MoniLog for Interpreted DSLs



Implementation on the JVM,
using either AspectJ or Truffle



```

package iterativeHeatEquation

import org.gemoc.monilog.*
import fr.ccea.nabl.*
import Iterative

@setup {
  prevResidual = 1.0;
}

@event ComputeTnReturned {
  after call ComputeTn
}

@event ResidualUpdated {
  after call ComputeResidual
}

@monilogger correctResidual {
  when ResidualUpdated
  if (context(residual) > prevResidual)
  then {
    NablConsoleAppender.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 {
    NablConsoleAppender.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;
  }
}
    
```

```

InitD: VcCells(), D{c} = 1.0;
ComputeDeltaTn: dt = Min(cCells()){|V{c}|/D{c}} * 0.1;
ComputeV: V{cCells()}, V{c} = 0.5 * Σ{pnodesOfCell(c)}{det(X(p), X(p+1))};
ComputeFaceLength: VfFaces(), faceLength{f} = 0.5 * Σ{pnodesOfFace(f)}{norm(X(p), X(p+1))};
ComputeFaceConductivity: VfFaces(), faceConductivity{f} = 2.0 * Σ{cCellsOffFace(f)}{cCellsOffFace(f)};

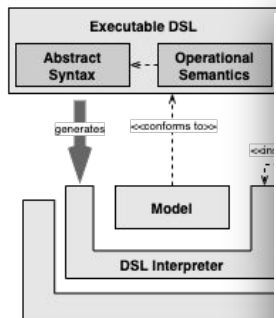
// Assembling the diffusion matrix
@ComputeAlphaCoeff: VcCells(), {
  R: aDiag = 0.0;
  neighbourCells(c), VfCommonFace(c,d), {
    let a: aExtraDiag = dt / V{c} * (faceLength{f} + faceConductivity{f}) / m;
    a(c, d) = aExtraDiag;
    aDiag = aDiag + aExtraDiag;
  }
  a(c, c) = -aDiag;
}

UpdateU: VcCells(), u^{n+1, k+1}(c) = u^n(n)(c) + a(c, c) * u^n(n+1, k)(c) + Σ{d∈Dn(c)}{d∈Dn(c)}{u^n(n+1, k+1)(d) - u^n(n+1, k)(c)};
ComputeTnResidual: Residual = Max{c ∈ cells()}{abs(u^{n+1, k+1}(c) - u^n(n+1, k)(c))};
ComputeTn: t^{n+1} = t^n + dt;
    
```

Annotations:

- Initializing variable storing value of previous residual:** `prevResidual = 1.0;`
- Declaring events of interest:** `@event ComputeTnReturned` and `@event ResidualUpdated`
- Checking the invariant:** `if (context(residual) > prevResidual)`
- Logging the values of interest and storing residual for next iteration:** `NablConsoleAppender.call(...)`
- Resetting stored residual to 1.0 after each iteration over n:** `prevResidual = 1.0;`

MoniLog for Interpreted DSLs



Implementation using either As

Instrumentation

Benchmark:

- Mean execution time overhead
- over 100 executions with and 100 executions without MoniLog
- in various scenarios.

Scenarios and results:

- log a message on each connectivity iteration (iterativeheatequation.nabla): $\times 2.51$;
- evaluate 6 expressions and log a message on each time loop iteration (iterativeheatequation.nabla): $\times 1.08$;
- evaluate an expression on each connectivity iteration, and log a message around half of the time (glace2d.nabla): $\times 1.17$.

```
context(n, context(k), context(residual), prevResidual);
prevResidual = context(residual);
}
}
@monilogger resetResidual {
  when ComputeReturned
  {
    prevResidual = 1.0;
  }
}
```

Resetting stored residual to 1.0 after each iteration over n

```
[n=018, k=18] current residual: 2.5E-8, previous residual: 1.8E-6
[n=018, k=19] current residual: 1.7E-8, previous residual: 1.1E-6
[n=018, k=20] current residual: 1.2E-8, previous residual: 7.0E-7
[n=018, k=21] current residual: 8.5E-9, previous residual: 4.4E-7
[n=019, k=01] current residual: 9.0E-3, previous residual: 2.0E-7
[n=019, k=02] current residual: 7.0E-4, previous residual: 1.8E-7
[n=019, k=03] current residual: 1.0E-4, previous residual: 1.2E-7
[n=019, k=04] current residual: 2.4E-5, previous residual: 8.0E-8
[n=019, k=05] current residual: 8.4E-6, previous residual: 5.3E-8
[n=019, k=06] current residual: 3.9E-6, previous residual: 3.6E-8
[n=019, k=07] current residual: 1.9E-6, previous residual: 2.5E-8
[n=019, k=08] current residual: 1.1E-6, previous residual: 1.7E-8
```

```
V(c)/D(c) * 0.1;
[|pNodesOfCell(f)|]det(X(p), X(p+1));
length(f) = 0.5 * [|pNodesOfFace(f)|]norm(X(p),
faceConductivity(f) = 2.0 * [|cellsOffFace
```

face(c,d), {
+ (faceLength(f) + faceConductivity(f)) / m

= u^n(n)(c) + a(c, c) * u^n(n+1, k)(c) + [|dEn
cells(f)|]abs(u^n(n+1, k+1)(f)) - u^n(n+1, k)(f));

MoniLog for Compiled DSLs

```
IterativeHeatEquation.n x
else
  u^{n}{c} = 0.0; // Initial circle in the center with value u0

InitD: VcCells(), D{c} = 1.0;

ComputeDeltaTn:  $\delta t = \text{Min}\{c\text{Cells}()\} \{V\{c\}/D\{c}\} * 0.1;$ 
ComputeV: VjCells(), V{j} = 0.5 *  $\sum\{p\text{NodesOfCell}\{j\}\}\{\text{det}\{X\{p\}, X\{p+1\}\}\};$ 
ComputeFaceLength: VfFaces(), faceLength{f} = 0.5 *  $\sum\{p\text{NodesOfFace}\{f\}\}\{\text{norm}\{X\{p\} - X\{p+1\}\}\};$ 
ComputeFaceConductivity: VfFaces(), faceConductivity{f} = 2.0 *  $\prod\{c\text{CellsOfFace}\{f\}\};$ 

// Assembling of the diffusion matrix
ComputeAlphaCoeff: VcCells(), {
  let R aDiag = 0.0;
  VdNeighbourCells(c), VfCommonFace(c,d), {
    let R aExtraDiag =  $\delta t / V\{c\} * \{\text{faceLength}\{f\} * \text{faceConductivity}\{f\}\} / \text{norm}\{X\{c\}, X\{d\}\};$ 
    a{c, d} = aExtraDiag;
    aDiag = aDiag + aExtraDiag;
  }
  a{c, c} = -aDiag;
}

UpdateU: VcCells(),  $u^{n+1, k+1}\{c\} = u^{n}\{c\} + a\{c, c\} * u^{n+1, k}\{c\} + \sum\{d\text{neig}\{c\}\}\{a\{c, d\} * u^{n+1, k}\{d\}\};$ 
ComputeResidual: residual =  $\text{Max}\{j \in \text{cells}()\}\{\text{abs}\{u^{n+1, k+1}\{j\} - u^{n+1, k}\{j\}\}\};$ 
ComputeTn:  $t^{n+1} = t^n + \delta t;$ 

plots x
frequency = 0.0001
last_update = 0

@before_call(ihe.ExecuteTimeLoopN)
def start(context):
  global fig, ax, quantiles_median_line, quantiles_fill
  fig, ax = plt.subplots()
  ax.set_xlim(quantiles_x_range)
  ax.set_ylim(quantiles_y_range)
  quantiles_median_line, = ax.plot(timesteps, second_quantile, color = 'xkcd:blue')
  quantiles_fill = ax.fill_between(timesteps, first_quantile, third_quantile, alpha=0.1)

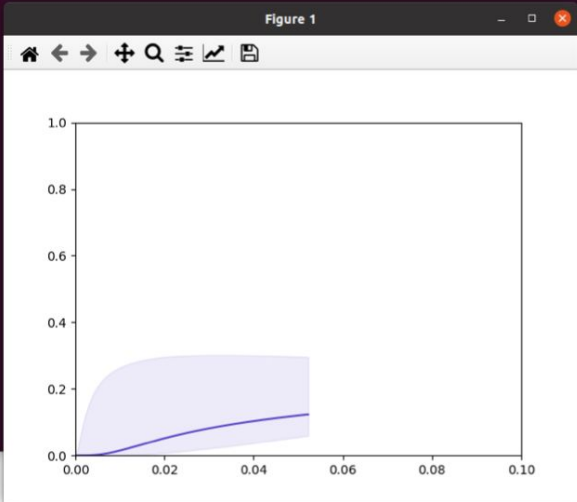
@after_call(ihe.ComputeTn)
def plotTemperature(context):
  global last_update, frequency
  if (context.t_n >= last_update + frequency):
    last_update = context.t_n
    plot_quantiles(context.u_n, context.t_n)
    plt.pause(0.000001)

@after_call(ihe.ExecuteTimeLoopN)
def stop(context):
  plt.show(block=True)
```

```
leroyd@un00302480: ~/eclipse-workspaces/nablab-debug/NablabWorkspace/NabLabExamples/src-gen-cpp/se...
leroyd@un00302480: ~
leroyd@un00302480: ~/ec...
leroyd@un00302480: ~/jgl...

(nablab-venv) leroyd@un00302480:~/eclipse-workspaces/nablab-debug/NablabWorkspace/NabLabExamples/src-gen-cpp/sequentia
l/iterativeheatequation/build$ ./iterativeheatequation ../../../../src/iterativeheatequation/IterativeHeatEquation.js
n
Starting IterativeHeatEquation ...

[TOPOLOGY] HMLOC unavailable cannot get topological informations
[OUTPUT] VTK files stored in output directory
[t=0] t = 5.22500000e-02 [CPU: 20ms, IO: 4ms] 4s950ms
```



MoniLog for Compiled DSLs

```
IterativeHeatEquation.n x
else
    u^{n}{c} = 0.0; // Initial circle in the center with value u0

InitD: VcCells(), D{c} = 1.0;

ComputeDeltaTn: dt = Min{cCells()}{V{c}/D{c}} * 0.1;
ComputeV: VjCells(), V{j} = 0.5 * \sum{pNodesOfCell{j}}{det(X{p}, X{p+1})};
ComputeFaceLength: VfFaces(), faceLength{f} = 0.5 * \sum{pNodesOfFace{f}}{norm(X{p} -
ComputeFaceCo

// Assemb
= ComputeA
  Let
  = VdEn
}
}
a{c,
}

UpdateU:
ComputeR
ComputeT

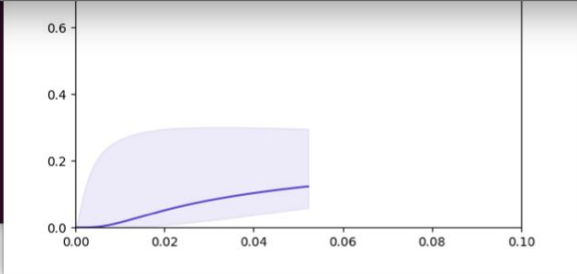
plots x
frequency
last_upda

@before call(ihe.ExecuteTimeLoopN)
@def start(context):
    global fig, ax, quantiles_median_line, quantiles_fill
    fig, ax = plt.subplots()
    ax.set_xlim(quantiles_x_range)
    ax.set_ylim(quantiles_y_range)
    quantiles_median_line, = ax.plot(timesteps, second_quantile, color = 'xkcd:blue')
    quantiles_fill = ax.fill_between(timesteps, first_quantile, third_quantile, alph

@after call(ihe.ComputeTn)
@def plotTemperature(context):
    global last update, frequency
    if (context.t_n >= last update + frequency):
        last update = context.t_n
        plot_quantiles(context.u_n, context.t_n)
        plt.pause(0.000001)

@after call(ihe.ExecuteTimeLoopN)
@def stop(context):
    plt.show(block=True)
```

- Monilog as a Python-based framework
- C++ generated code exposes a Python object
 - global variables, and
 - local variables of the current scope



MoniLog for Compiled DSLs

```
IterativeHeatEquation.n x
else
  u^{n}{c} = 0.0; // Initial circle in the center with value u0

InitD: VcCells(), D(c) = 1.0;

ComputeDeltaTn: dt = Min{cCells()}(V(c)/D(c)) * 0.1;
ComputeV: VjCells(), V{j} = 0.5 * sum{pNodesOfCell{j}}(det(X{p}, X{p+1}));
ComputeFaceLength: VfFaces(), faceLength{f} = 0.5 * sum{pNodesOfFace{f}}(norm(X{p} -
ComputeFaceCond

// Assemb
= ComputeA
  Let
  VdEn
}
} a{c,
}
UpdateU:
ComputeR
ComputeT

plots x
frequenc
last_upd

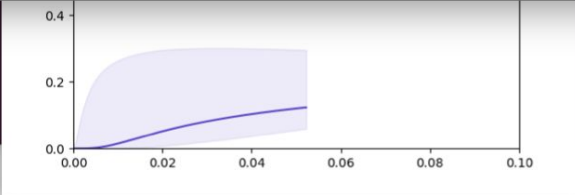
@before
@def star
globa
fig, ax
ax.set_xe
ax.set_ylim(quantiles_y_range)
quantiles_median_line, = ax.plot(timesteps, second_quantile, color = 'xkcd:blue')
quantiles_fill = ax.fill_between(timesteps, first_quantile, third_quantile, alph

@after call(ihe.ComputeTn)
@def plotTemperature(context):
  global last update, frequency
  if (context.t.n >= last update + frequency):
    last update = context.t.n
    plot_quantiles(context.u.n, context.t.n)
    plt.pause(0.000001)

@after call(ihe.ExecuteTimeLoopN)
@def stop(context):
  plt.show(block=True)
```

Encompasses several scenarios:

- Logging
- Monitoring
- Conditional debugging
- Design-by-contract
- Unit testing
- Lightweight simulation processes
- Explore alternatives of code/data
- Code coupling



MoniLog for Compiled DSLs

```
IterativeHeatEquation.n x
else
  u^{n}(c) = 0.0; // Initial circle in the center with value u0

InitD: VcCells(), D(c) = 1.0;

ComputeDeltaTn: dt = Min{cCells()}(V(c)/D(c)) * 0.1;
ComputeV: VjCells(), V(j) = 0.5 * sum{pNodesOfCell(j)}{det(X(p), X(p+1))};
ComputeFaceLength: VfFaces(), faceLength(f) = 0.5 * sum{pNodesOfFace(f)}{norm(X(p) -
ComputeFaceCo

// Assemb
ComputeA
  let
  VdEn
}
a(c,
}

UpdateU:
ComputeR
ComputeT

plots x
frequency
last_upda

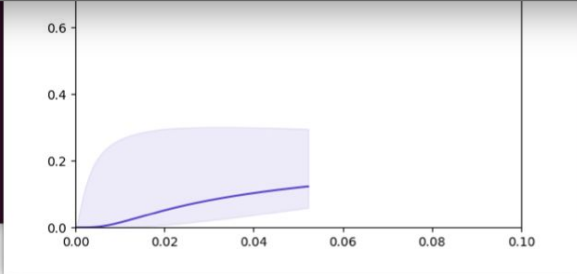
@before call(ihe.ExecuteTimeLoopN)
def start(context):
  global fig, ax, quantiles_median_line, quantiles_fill
  fig, ax = plt.subplots()
  ax.set_xlim(quantiles_x_range)
  ax.set_ylim(quantiles_y_range)
  quantiles_median_line, = ax.plot(timesteps, second_quantile, color = 'xkcd:blue')
  quantiles_fill = ax.fill_between(timesteps, first_quantile, third_quantile, alph

@after call(ihe.ComputeTn)
def plotTemperature(context):
  global last update, frequency
  if (context.t.n >= last update + frequency):
    last update = context.t.n
    plot_quantiles(context.u.n, context.t.n)
    plt.pause(0.000001)

@after call(ihe.ExecuteTimeLoopN)
def stop(context):
  plt.show(block=True)
```

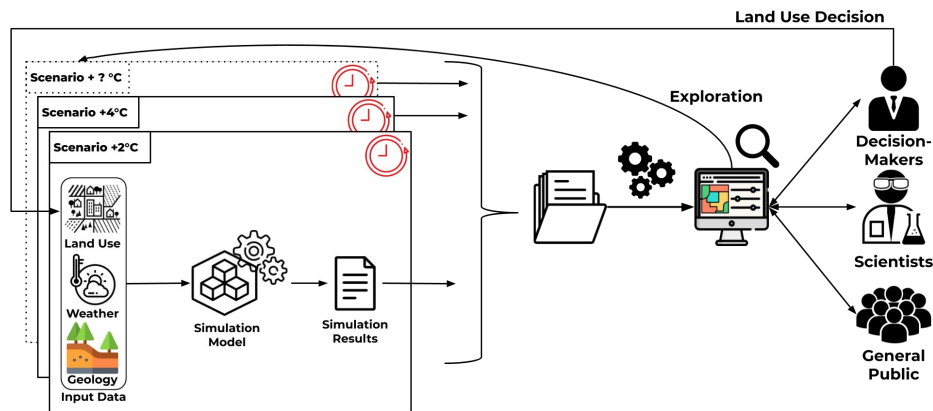
In the meantime...:

- The Python debugger (pdb) can be used to add breakpoints on moniloggers
- When the execution is paused, any function can be applied on the current execution context



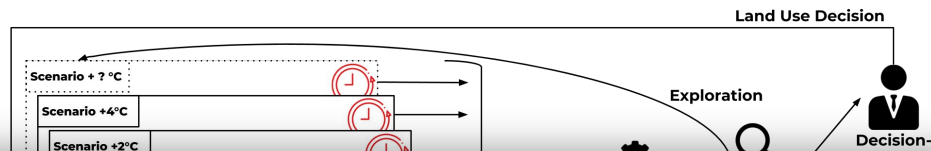
On Trade-off Analysis

Tradeoff Analysis in Scientific Computing



- Integrated environment for scientific computing
 - Flexible, agile, collaborative, distributed & adaptive
- Support for trade-off analysis and decision making
 - Exploration of scenarios
 - Integrity of projections
 - interactivity
 - Generalisation
- Application to environmental sciences
 - in collaboration with OSUR (UR1)
 - other collaborations with Lancaster University (e.g., Data Science of the Natural Environment)

Tradeoff Analysis in Scientific Computing



How to tailor scientific software to support decision-making?

- Integ
 -
- Support for trade-off analysis and decision making
 - Exploration of scenarios
 - Integrity of projections
 - interactivity
 - Generalisation
- Application to environmental sciences
 - in collaboration with OSUR (UR1)
 - other collaborations with Lancaster University (e.g., Data Science of the Natural Environment)

Tradeoff Analysis in Scientific Computing

High Performance Computing (HPC)

Increase and improve the computational hardware and infrastructure



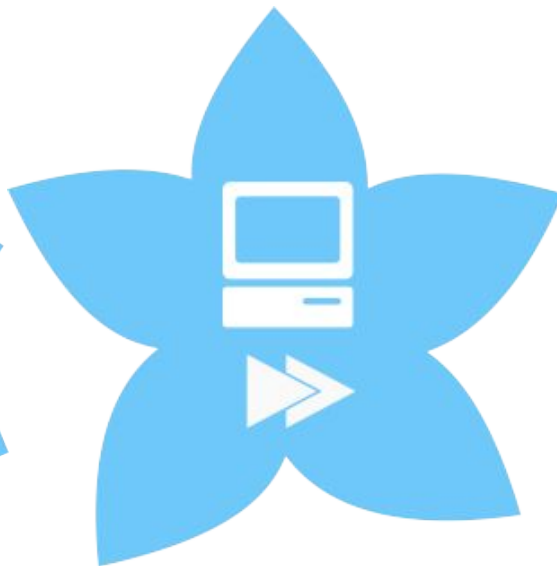
Data-driven Approach

Build an analytic model



Model Reduction

Simplify the mathematical model



Tradeoff Analysis in Scientific Computing

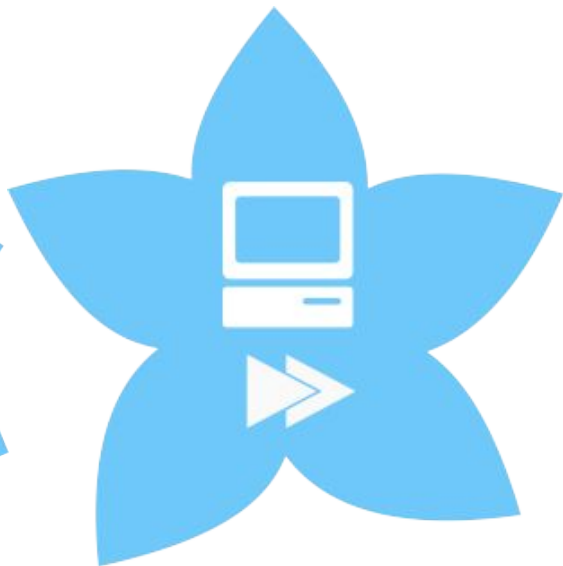
High Performance Computing (HPC)

Increase and improve the computational hardware and infrastructure



Data-driven Approach

Build an analytic model



Model Reduction

Simplify the mathematical model



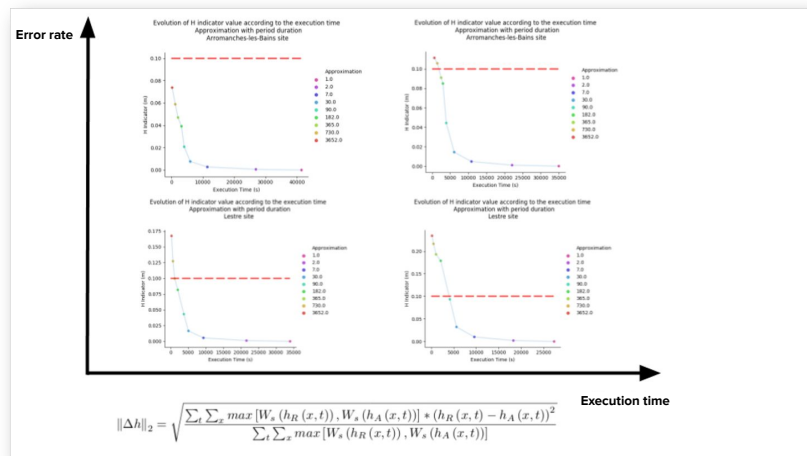
Approximate Computing

Approximate the computation. Trade-off between accuracy and time execution



Approximate Scientific Computing

- Reduce the simulation time to better support trade-off analysis and decision making
- Application of approximate computing to scientific computing
- Work on the simulation code (white box) or the input data (black box)
- Require a domain-specific interpolation operator
- Challenge: infer the approximation bound



Loop Aggregation for Approximate Scientific Computing

June Sallou, Alexandre Gauvain, Johann Bourcier, Benoît Combemale, Jean-Raynald de Dreuzy. *ICCS 2020: 141-155.*

SE4Science: Call For Actions!

SE4Science: Opportunities

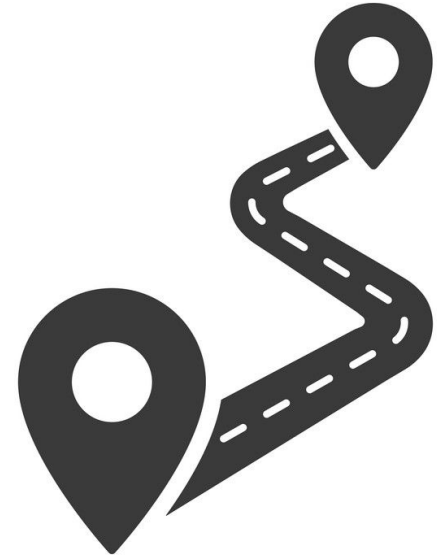
- **Increasing demand**
 - Science-driven society
 - Acceleration of natural phenomena
 - Breakthroughs in engineering required innovative thinking
- **Technology evolution**
 - Continuous evolution on computing capabilities
 - Progresses on M&S tools and methods
 - Breakthroughs on development environments (e.g., language servers)
 - Cloud infrastructure and web-based technologies as key enablers

SE4Science: Objectives

- **International leadership** within the scientific community
 - proper programming foundations
 - tools and methods for collaborative, effective and reliable development
- **Competitiveness** of the French industry
- **Broader engagement** with decision and policy makers, and the general public

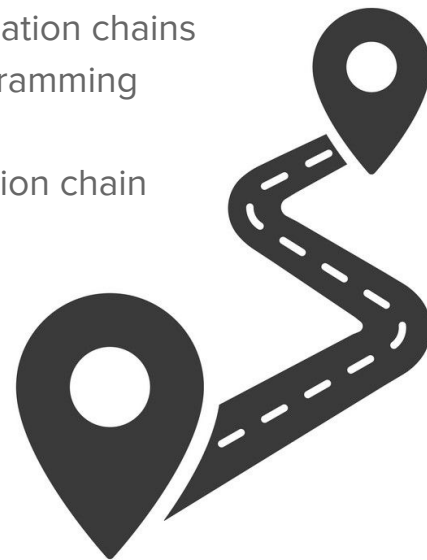
SE4Science: Roadmap

- Programming Foundations
- Scientific Software Validation and Verification
- AI-enhanced Scientific Computing
- Scientific Computing Virtual Lab



SE4Science: Roadmap

- **Programming Foundations**
 - Domain-Specific Languages for simulation processes, and compilation chains
 - Sound combination of literate, exploratory, live and polyglot programming
 - Language constructs for heterogeneous model coupling
 - Variability management for input data, source code, and compilation chain
 - Reproducibility and deep variability management
- Scientific Software Validation and Verification
- AI-enhanced Scientific Computing
- Scientific Computing Virtual Lab



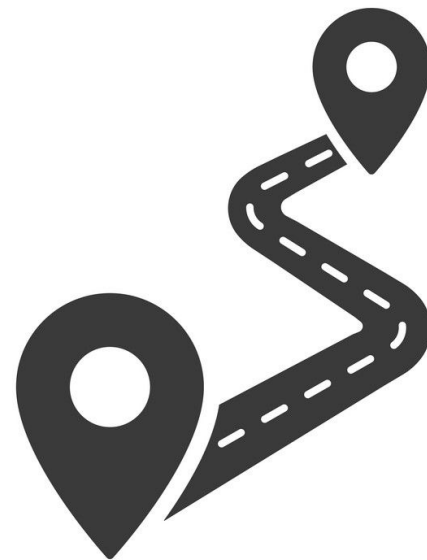
SE4Science: Roadmap

- Programming Foundations
- **Scientific Software Validation and Verification**
 - Testing framework and design-by-contract
 - Tradeoff with privacy
 - Experimental frame and correctness envelope for scientific models
 - Compiler V&V
- AI-enhanced Scientific Computing
- Scientific Computing Virtual Lab



SE4Science: Roadmap

- Programming Foundations
- Scientific Software Validation and Verification
- **AI-enhanced Scientific Computing**
 - AI-based simulation
 - Predictive models for design-space exploration
 - AI-based assistants/recommenders and tradeoff analysis tools
 - for scientific software development
 - for model calibration, compilation/execution parameters
 - for deployment and delivery
 - Transfer learning for reuse
 - Multi-fidelity and frugality
- Scientific Computing Virtual Lab



SE4Science: Roadmap

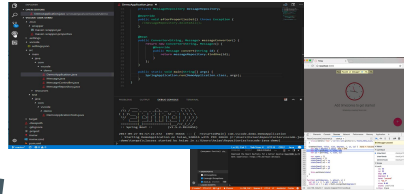
- Programming Foundations
- Scientific Software Validation and Verification
- AI-enhanced Scientific Computing
- **Scientific Computing Virtual Lab**
 - Programming interface (e.g., computational notebooks)
 - Web-based, cloud-native, collaborative and distributed IDE
 - Digital twin framework



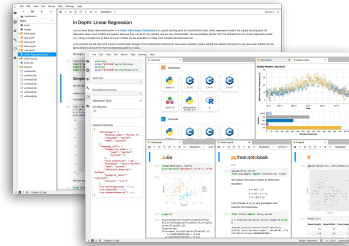
Virtual Lab



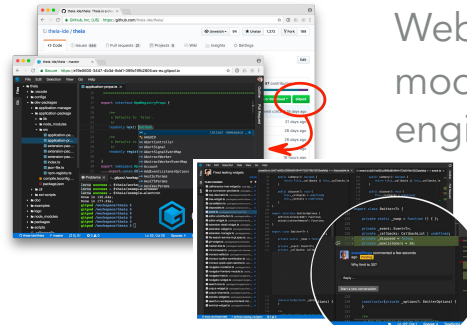
Socio-technical
coordination



Lightweight, modular, customizable,
distributed and self-adaptable scientific
and engineering platforms...



Polyglot, literate
programming



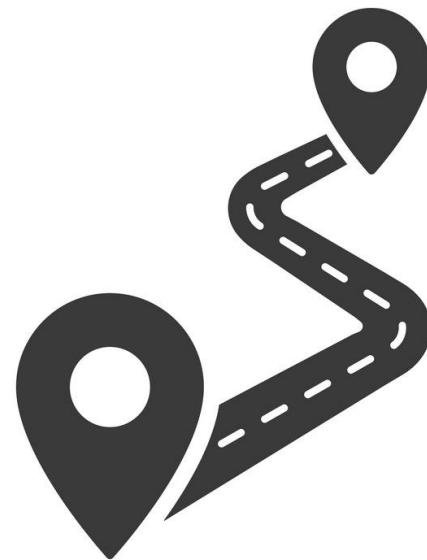
Web-based, Collaborative
modeling, modeling flow, social
engineering



Exploratory and live programming, digital twin

SE4Science: Roadmap

- **Programming Foundations**
 - Domain-Specific Languages for simulation processes and compilation chains
 - Sound combination of literate, exploratory, live and polyglot programming
 - Language constructs for heterogeneous model coupling
 - Variability management for input data, source code, and compilation chain
 - Reproducibility and deep variability management
- **Scientific Software Validation and Verification**
 - Testing framework and design-by-contract
 - Tradeoff with privacy
 - Experimental frame and correctness envelope for scientific models
 - Compiler V&V
- **AI-enhanced Scientific Computing**
 - AI-based simulation
 - Predictive models for design-space exploration
 - AI-based assistants/recommenders and tradeoff analysis tools
 - Transfer learning for reuse
 - Multi-fidelity and frugality
- **Scientific Computing Virtual Lab**
 - Programming interface (e.g., computational notebooks)
 - Web-based, cloud-native, collaborative and distributed IDE
 - Digital twin framework



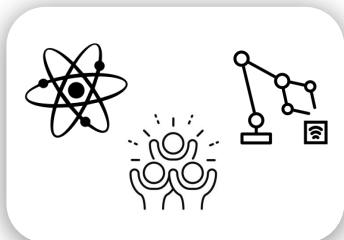
SE4Science: Expected Impact

- Establish **software engineering principles** for engineering scientific software
- Consolidate research activities into **open-source tools** accessible to scientists, engineers, decision makers and the general public
- Establish a strong **leadership** in the scientific, technological and industrial communities

SE4Science: Expected Impact

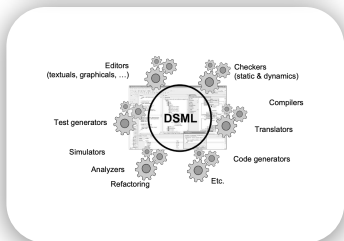


Take Away Messages



Scientific Computing

- ▶ In science, engineering and society
- ▶ Complex and specific SDLC
- ▶ Requires proper software engineering principles



(Domain-Specific) Software Languages

- ▶ Domain-specific abstractions
- ▶ Support for coordination, debugging, approximation...



SE4Science: a vision with associated roadmap

- ▶ Timely and relevant
- ▶ Impactful
- ▶ Unique opportunity for the French community