Computational Exploratory Guide

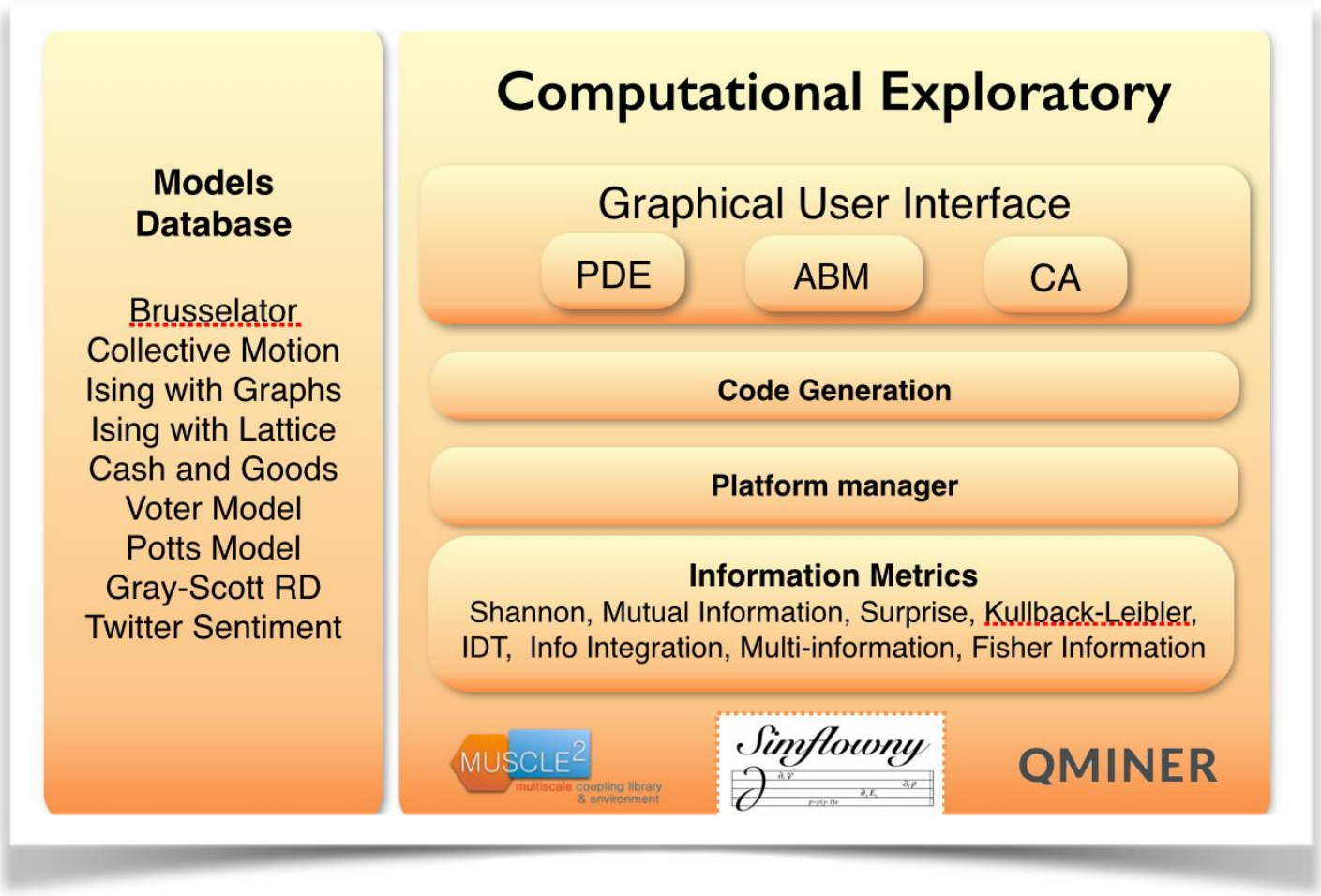

Table of Contents

CE 3
Installation guide 3
Problem generation 3
PDE 3
ABM/CA 4
Creation of a new document 4
Open new document 4
Save document 4
Edition 4
Validation 5
Code generation 5
Simulation 5
Monoprocessor 5
Multiprocessor 5
Using input data 5
Multiple simulations 6
Metrics 7
Single 8
Multiple 10
Metric parameters 10
Data types 16
Visualization of data 17

CE

The Computational Exploratory (CE) is part of the Sophocles project, which is a Future and Emerging Technologies Proactive Initiative funded by the European Commission under FP7

The Computational Exploratory (CE) is designed to analyze large datasets and includes a variety of benchmark models, encompassing metric and non-metric topologies, families and hierarchies, with support for Cellular Automata (CA), Agent Based Models (ABM) and Partial Differential Equations (PDE). It includes comprehensive information metrics analysis, parallel tools and integration with Simflowny, Muscle^2 and Qminer.

The CE allows the generation, simulation and analysis of a different set of models and problems. From the definition of a model and its associated problem, the CE automatically generates parallel C++ code that allows the user to perform high performance computing simulations.

Installation guide

The CE is released in a VirtualBox Virtual Machine. Oracle VirtualBox can be downloaded from https://www.virtualbox.org/. For further information on virtual machine deployment visit the website.

Once the virtual machine is running, the user and password to access is by default "ce". For security reasons, the password should be changed.

In order to run PDE guided user interface follow the next steps:

1. Execute *Simflowny* icon from the desktop. It will open a terminal and start the server. To check if the server has successfully started, type *lb* in the command line from the opened terminal.
2. Once the server is running, double click *Simflowny.gui* icon placed below server launcher. A browser will open with the user interface URL.
3. In order to stop the server, type *stop 0* in the server terminal. It is very important stop it this way to avoid data corruption.

In order to run ABM guided user interface execute the same instructions than PDE using the ABM icons.

Problem generation

PDE

The PDE definition is aided by the existing capabilities of IAC³'s Simflowny software. This tool allows the user to create models governed by PDEs in a general manner to ensure their reutilization in different problems. A model is defined by equations, like Brusselator or Gray-Scott reaction diffusion system, while a problem defines the initial data, boundary conditions and the domain of the model. Documentation regarding Simflowny can be found in folder */home/ce/CE/simflowny/doc*, where a tutorial and users guide are available.

In order to create, generate code and simulate a PDE model or problem refer to the documentation.

ABM/CA

The ABM/CA definition is aided by the existing capabilities of IAC³'s Simflowny software. This tool allows the user to create models governed by ABMs in a general manner to ensure their reutilization in different problems. A model is defined by gather and update definitions, which are, basically, algorithms, like Voter Model or Collective Motion; while a problem defines the initial data, boundary conditions and the domain of the model.

Creation of a new document

The user interface provides the creation of a new document from scratch. Once selected the document type, a new empty skeleton is created to fill with the proper values. With the aid of contextual menu, the optional or repeatable elements can be added.

Open new document

A document can be opened for visualization or edition from the upload button on the top menu.

Save document

In order to save the current document, use the download button on the top menu. It will start a download of the file in XML format that can be saved in the local file system.

Edition

The idea of ABM/CA is similar to the PDE structure, there is a model and a problem separately definition, so a model can be reusable in different problems. The example files in */home/CE/problem_models* are expected to be clear enough for a new user to create a new model/problem by modifying an existing one. Nevertheless, there are some indications that may interest to a model/problem editor.

- Both ABM and CA use the same definition.
- Model
  - The algorithm definition for the ABM/CA is described with the gather-update paradigm. The gathers and updates for each field should be defined separately.
  - The order of execution for gathers and updates is specified within the *executionOrder* tag.
  - The code generation process will simplify the code from the gather and update blocks joining them together when possible.
- Problem
  - Most of the tags from a problem are optional and their usage depends on the kind of problem to resolve. For instance, Collective Motion needs a spatial domain for the agents to move, but Voter model runs on a non-spatial graph. It is strongly recommended to check the examples for orientation.

Validation

The models and problems can be validated using the interface. This validation returns the error, if any, related to the current document in the editor.

Code generation

The generation of simulation code is made automatically by the CE. The interface provides a button to create and compile a problem selected.

The result of the process indicates the system path where the files are created.

Simulation

This section describes the steps to simulate ABM/CA problems. The simulation of PDE problems is covered by Simflowny documentation, so basic simulation is not described in this guide. However, subsection Multiple simulations also applies to PDE problems.

Monoprocessor

To execute a single processor simulation it is needed an executable (See section Code generation) and a parameters file. A sample parameters file named *problem.input* is automatically in the problem folder created when generating code.

Modify the parameters file, open a terminal and execute: *./execName problem.input*

Multiprocessor

The CE brings the possibility to run parallel simulations. All generated code is ready to be run in parallel with Message Passing Interface (MPI). In order to execute a simulation in parallel, the following steps should be taken into account:

- Be sure the virtual machine of the exploratory is configured to use more than one host processors, otherwise the parallelization will not perform as expected. For further information check Virtual Box (or the virtualization tool in use) documentation.
- Open a terminal and execute a MPI daemon if not already executing: *mpd &*. To check if a daemon is already running: *mpdtrace*
- Execute with the proper MPI parameters: *mpirun -np X ./execName problem.input*

Using input data

Some of the problems could require using external data as the initialization of the fields (See Twitter sentiment example). There are some indications that should be considered:

- If the input file is the result of a parallel graph simulation, the file cannot be used directly. A python script is provided in order to convert multiprocessor graph output to mono-processor graph input.

The script is located at */home/ce/CE/CE1.1a/scripts/multi2monoDOT.py*. Usage: *python multi2monoDOT.py -i inputFile -o outputFile*

- When a simulation does not use a field that is declared in the input graph an execution will throw an error. To solve this problem, the CE provides an script to remove fields from DOT graphs. The script is located at */home/ce/CE/CE1.1a/scripts/removeFieldsGraph.py*. Edit the script conveniently to remove the fields not used in the simulation.

Multiple simulations

It is often desirable to launch multiple simulations with different initial conditions. When the number of combinations is huge it is unacceptable to launch them manually. The CE provides a system to automatize the execution of multiple simulations simultaneously, even if the simulations are parallel.

Before using the script, it is necessary to understand how it works. To generate all the combinations there should be defined a set of wildcards, with format **#Name#**, that will be used inside both the simulation and script parameters files. The script will replace the wildcards with their respective values to execute all the simulations.

The script can be found at */home/ce/CE/CE1.1a/scripts/exec_multiple.py*. Its usage requires a parameters file, one example can be found in the same folder. The parameter must follow the following structure:

- **[Execution]**
  - **exe.** Name of the executable to run.
  - **param-file.** Name of the parameter file for the simulation.
- **[Parallel]**
  - **nprocs-exec.** Number of processors used for each simulation. If the value is one, the simulations will be mono-processor, otherwise each simulation will be multiprocessor.
  - **nprocs-reps.** Number of processors used to launch the simulations. This parameter and the previous one allows to run, for instance, 4 simulations to be executed in two processors in parallel, using a total of 4 (nprocs-reps) x 2 (nprocs-exec) processors of the machine.
  - **Set of Wildcard#=Values.** After the two previous parameters there is an area where to specify the values for the wildcards to be used each simulation. Each line represents a wildcard, followed by a dash symbol (#) and the list of values for that wildcard. This list can be set in two different formats:
    - Space separated list. For instance, *R# = 1 18 40 54 62 154 164 200*, to represent the list of ECA rules wanted to simulate.
    - Range of values. Useful for long lists. For instance *T# = 0..25..5*, meaning from 0 to 25 in increments of 5. The upper limit is not inclusive. The increment part is optional, being also valid *T# = 0..25*, resulting in a list of [0, 25).
- **[Extra-Files]**
  - **Set of FileDescriptor=FileName.** When the simulation uses external input files that provide the initialization this is the way to generate the combinations of initial states. Each line represents a file

descriptor (only necessary for format purposes) and a file name, which could contain wildcards.

Metrics

Once the simulations have finalized, the CE brings the possibility to perform the computation of set of information metrics. The list of metrics is as follows:

1. Shannon Entropy
2. Mutual Information
3. Surprise
4. Kullback-Leibler
5. IDT
6. Information Integration
7. Multi-information
8. Fisher Information
9. Hellinger Distance

Each metric is isolated from the others and the range of this list could be extended with little effort. The folder */home/ce/CE/CE1.1a/metrics* contains the python files of the metrics. The next relase of the CE will provide a new and improved interface to use these metrics.

The CE simulations generate different output formats. PDE problems output HDF5 mesh and meshless files and ABM/CE problems also generate DOT graph files. All of them can be processed in the CE metric processor, even simultaneously, so it is possible to get metrics from simulations with different output format.

The metric script uses wildcards similarly to the ones used in Multiple simulations, with format **#Name#**, to execute multiple metric calculations at the same time. But there is a second kind of wildcards, **[P]**, **[T]** and **[G]**, whose usage is explained in the following paragraphs.

The following considerations are essential to understand the metric parameter file:

- The input data is stored in multi-dimensional variables, each dimension with an specific grouping meaning.
- The metric calculation is applied on a set of elements depending on the input format (cell, agent or graph node). This way, each element can obtain a metric value separately from the others. This grouping is called the **N** dimension.
- The basic unit of metrics is the PDF calculation. As a result, the input data need a population grouping **P**, who can use wildcard **[P]**.
- The data is classified by:
  - Spatial data. The population is taken from the neighbours of an element. Each element could have a different number of neighbours, so the population could have different length for the elements. In this case, the usage of wildcard **[P]** is not necessary.

- Statistical data. The population is taken from different simulations. The occurrence of the same element over different experiments is grouped in the population dimension. The usage of the wildcard **[P]** is mandatory.

- The number of dimensions for the variables could be two, three or four with the following groupings:
  - Bidimensionals: **N**x**P**
  - Tridimensionals: **T**x**N**x**P**, **T** is the temporal dimension for metrics in which there is a time series calculation. Actually, it is not mandatory to use time series for this dimension, can be used with an arbitrary meaning.
  - 4-dimensionals: **G**x**T**x**N**x**P**, **G** is an unespecified grouping index useful for metrics that requires an extra dimension for its calculations (e.g. information integration).

The HDF5 output is stored within folders whose name is similar to *visit_dump.00000*. In order to get the information the name should be, for instance, specified as *visit_dump.**[T]*** in the parameters file. When the script replaces the wildcards would obtain *visit_dump.1*, *visit_dump.2*, *visit_dump.3*, etc. that do not matches with the output naming. In order to solve this problem, the CE provides an script to automatically rename the folders to cope the CE metric system. The scripts can be found at */home/ce/CE/CE1.1a/scripts*, to execute it: *python outputRename.py -d directoryName*

Single

Similarly to the execution of simulations, there is the possibility to perform metrics a single time or running multiple metrics at a once. The metric script is located at */home/ce/CE/CE1.1a/workflow.py* and a parameter file is required to run. The file must follow the following format:

- **Set of [VariableName]**. The beginning of the file is used to specify the multidimensional array variables that will be used as the input for the metrics. Each variable is specified in a section whose name is used to identify the variable, so the names cannot be repeated. The parameter configuration for a section depends on the kind of variable.
  - Mandatory parameters:
    - **dimensions:** 2, 3 or 4
    - **population_type:** spatial or statistical
    - **input_files**. Path for the input file. Wildcards could be needed depending from the population_type and dimensions. If population_type is statistical, wildcard **[P]** should be used.
      - For 3 dimensions, also **[T]** is needed.
      - For 4 dimensions, also **[G]** is needed.
    - **Field.** Used to set the field to use in this variable. CE simulation output stores all the fields together.
  - Additional parameters depending on the kind of input:
    - Graphs:
      - Population_type: spatial

        - **field_type:** node or edge
    - Mesh:
        - Population_type: spatial
            - **stencil**. Number of adjacent cells in each direction to be considered neighbours and be introduced in the population list.
            - **periodical**. Boolean that controls if the domain should be considered periodical. In a periodical domain, the cells at the beginning of the domain are connected to the ones at the other extreme.
    - Meshless:
        - Population_type: spatial
            - **radius**. Distance in domain units in which agents are considered neighbours from each other.
            - **periodical**. If the domain, limited by computational restrictions, should be considered periodical. In a periodical domain, the agents at the beginning of the domain are connected to the ones at the other extreme.
- Additional parameters depending on the number of dimensions:
    - 2 dimensions:
        - Population_type: statistical
            - **stat_from.** Number of the initial experiment to get population from.
            - **stat_to.** Number of the last experiment to get population from.
    - 3 dimensions:
        - **time_from.** Number of the initial time step for time grouping.
        - **time_to.** Number of the last time step for time grouping.
    - 4 dimensions:
        - **group_from.** Number of the initial value for the extra grouping.
        - **group_to.** Number of the last value for the extra grouping.
- **[Metrics].** This section contains all the metrics and auxiliary operations to be performed sequentially. The output variable of a metric could be the input of another.
    - **Set of varName=metricName(metricParameters).** The result of each metric calculation is stored in a variable, which can be used as input in following metrics. The metrics are calculated sequentially depending on the order specified on the file. Check Metric parameters section for further detail of metrics available.

To execute the metrics: *python workflow -p parameter-file -o output-file*

The results of the metrics are stored in an HDF5 file.

Multiple

It is possible to perform the same metrics simultaneously with different configurations. The same script is used, but a new section should be added to the parameter file.

- **[Parallel]**
  - **nprocs.** Number of processors which will execute the metrics in parallel.
  - **Set of Wildcard#=Values.** It works the same way as in Multiple simulations section. Visit the section for further detail.

As the output file is the same for the parallel executions, the user should be sure that the naming of the metric variables do not overlap. There is an example of parameter file located in */home/ce/CE/CE1.1a/ElementaryCellularAutomata/multi-information.params.*

Metric parameters

The following list describes the parameters for the metrics and auxiliary functions that could be used in the parameter files. **Please, note that the next version of the CE will provide a new interface and this list will be deprecated.**

pdf(data, bin_values, continuous_bins)
Calculates the probability density function. **Parameters:** ▪ **data –** data with format NxP. N = elements; P = population ▪ **bin_values –** list of values to obtain the bins ▪ **continuous_bins** - true if the values of the bins are continuous **Returns:** data with format NxB. N = elements; B = bins
pdf_joint(dataA, bin_valuesA, continuous_binsA, dataB, bin_valuesB, continuous_binsB)

Calculates the joint probability density function from two sets of data.

Parameters:

- **dataA –** data with format NxP. N = elements; P = population
- **bin_valuesA –** list of values to obtain the bins
- **continuous_binsA** - true if the values of the bins are continuous
- **dataB –** data with format NxP. N = elements; P = population
- **bin_valuesB –** list of values to obtain the bins
- **continuous_binsB** - true if the values of the bins are continuous

Returns: data with format NxBAxBB N = elements; BA = bins A; BB = bins B

Preconditions:

- The number of elements from dataA must be the same as dataB.

mutual_information(pdfA, pdfB, joint_pdf, logbase="log2")

Calculates the mutual information between two PDFs.

Parameters:

- **pdfA –** pdf variable NxB. N = elements; B = bins
- **pdfB –** pdf variable NxB. N = elements; B = bins
- **joint_pdf –** joint pdf variable with format NxBAxBB N = elements; BA = bins A; BB = bins B
- **logbase –** base for the logarithm ("log2", "log", "log10")

Returns: data with format N. N = elements

Preconditions:

- The number of elements from pdfA must be the same as pdfB.
- The number of bins BA and BB from joint_pdf variable must be the same than the bins from pdfA and pdfB respectively.
-

shannon(pdf, logbase="log2")

Calculates the shannon entropy of a PDF.

Parameters:

- **pdf –** pdf variable NxB. N = elements; B = bins
- **logbase –** base for the logarithm ("log2", "log", "log10")

Returns: data with format N. N = elements

kullback-leibler(pdf_p, pdf_q, logbase="log2")

Calculates the Kullback-Leibler divergence of two PDFs.

Parameters:

- **pdf_p –** pdf variable NxB. N = elements; B = bins
- **pdf_q –** pdf variable NxB. N = elements; B = bins
- **logbase –** base for the logarithm ("log2", "log", "log10")

Returns: data with format N. N = elements

hellinger-distance(pdf_p, pdf_q)

Calculates the Hellinger distance of two PDFs.

Parameters:

- **pdf_p –** pdf variable NxB. N = elements; B = bins
- **pdf_q –** pdf variable NxB. N = elements; B = bins
- **logbase –** base for the logarithm ("log2", "log", "log10")

Returns: data with format N. N = elements

surprise(prob)

Calculates the surprise of a PDF.

Parameters:

- **prob –** pdf variable NxB. N = elements; B = bins

Returns: data with format N. N = elements

idt(initial, time_series, epsilon, dt, bin_values, continuous_bins, logbase="log2")

Calculates the idt requiring backward simulations.

Parameters:

- **initial –** initial data with format NxP. N = elements; P = population
- **time_series –** time series data with format TxNxP. T= time series; N = elements; P = population
- **epsilon –** epsilon parameter
- **dt –** time increment between time snapshots
- **bin_values -** list of values to obtain the bins
- **continuous_bins** - true if the values of the bins are continuous
- **logbase –** base for the logarithm ("log2", "log", "log10")

Returns: data with format N. N = elements

idt_individual(initial, time_series, dt, bin_values, continuous_bins, sample_state_0, sample_state_t, sample_time, logbase="log2")

Calculates the individual idt with forward simulations.

Parameters:

- **initial –** initial data with format NxP. N = elements; P = population
- **time_series –** time series data with format TxNxP. T= time series; N = elements; P = population
- **dt –** time increment between time snapshots
- **bin_values -** list of values to obtain the bins
- **continuous_bins -** true if the values of the bins are continuous
- **sample_state_0 -** percentage of elements to choose as a sample for initial state
- **sample_state_t -** percentage of elements to choose as a sample for state t
- **sample_time -** percentage of elements to choose as a sample for time series
- **logbase –** base for the logarithm ("log2", "log", "log10")

Returns: data with format N. N = elements

information_integration(initial, group, dt, bin_values, continuous_bins, sample_N1, sample_N2, sample_G, sample_t, logbase="log2")

Calculates the information integration.

Parameters:

- **initial –** initial data with format NxP. N = elements; P = population
- **group –** input data with format GxTxNxP. G = groups; T = time series; N = elements; P = population
- **dt –** time increment between time snapshots
- **bin_values -** list of values to obtain the bins
- **continuous_bins -** true if the values of the bins are continuous
- **sample_N1 -** percentage of elements to choose as a sample for initial state
- **sample_N2** - percentage of elements to choose as a sample for state t
- **sample_G** - percentage of groups to choose as a sample
- **sample_t** - percentage of elements to choose as a sample for time series
- **logbase –** base for the logarithm ("log2", "log", "log10")

Returns: data with format TxN (sampled). T = time series; N = elements

multi_information(data, bin_values, continuous_bins, sample_var, sample_elems, sample_pop, logbase="log2")

Calculates the multi-information between the variables.

Parameters:

- **data –** initial data with format VxNxP. V = variables; N = elements; P = population
- **bin_values -** list of values to obtain the bins
- **continuous_bins -** true if the values of the bins are continuous
- **sample_var** - percentage of variables to choose as a sample
- **sample_elems** - percentage of elements to choose as a sample
- **sample_pop** - percentage of population to choose as a sample
- **logbase –** base for the logarithm ("log2", "log", "log10")

Returns: data with format N. N = elements

swap_axes(data, axis0, axis1)

Auxiliary function that swaps two axes from a variable.

Parameters:

- **data –** any kind of data with at least 2 dimensions

Returns: data with different dimension order

Preconditions:

- The number of dimensions of the data should be 2 or more

add_dimension(data, dimNumber)

Auxiliary function that adds an extra dimension to the data in the given position.

Parameters:

- **data –** any kind of data

Returns: data with and added dimension of length 1.

count(data)

Auxiliary function that counts the population number.

Parameters:

- **data –** data with format NxP. N = elements; P = population

Returns: data with format N. N = elements

Data types

The CE metric system handles the following data types as parameter inputs:

- **Numeric constant**. E.g., 4, 6.87, etc.
- **Boolean constant**. E.g., true, false.
- **Numeric list**. Numbers separated by spaces. E.g., 0 1, 4.56 6.8 9 23.45, etc.
- **Variables**. Variables used to load input data or metric variables. There are some options:
  - **Simple variables**. E.g., spin, cash, etc.
  - **Slice of variable**. Python style. E.g., spin[2], cash[:, 9], etc.
  - **Temporal repeating variable**. A metric containing this kind of variable will be executed as many times as the length of dimension **T**. It is only applicable to statistic-temporal data **T**x**N**x**P**. E.g., {prob}, {var_t}, etc
  - **Slice of a temporal variable**. Combination of the two last variable types. The slicing must not be applied to the temporal dimension of the variable. E.g., {prob[3]}

Visualization of data

The CE is provided with third party tools that allow the visualization of the data involved in the whole process, from the creation of models to the result from metrics. The following list describes the recommended usage:

- **Xmlcopyeditor.** XML editing tool that ease the modification and validation of models.
- **Visit.** A visualization tool for scientific data. All simulation result, apart from graphs simulations, can be easily analyzed with this tool.
- **Gephi.** Graph visualization tool. Additionally, it provides some analysis on the graphs.
- **HDFView.** Specific HDF file format viewer. Can be used to show the results of the metric process.