INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

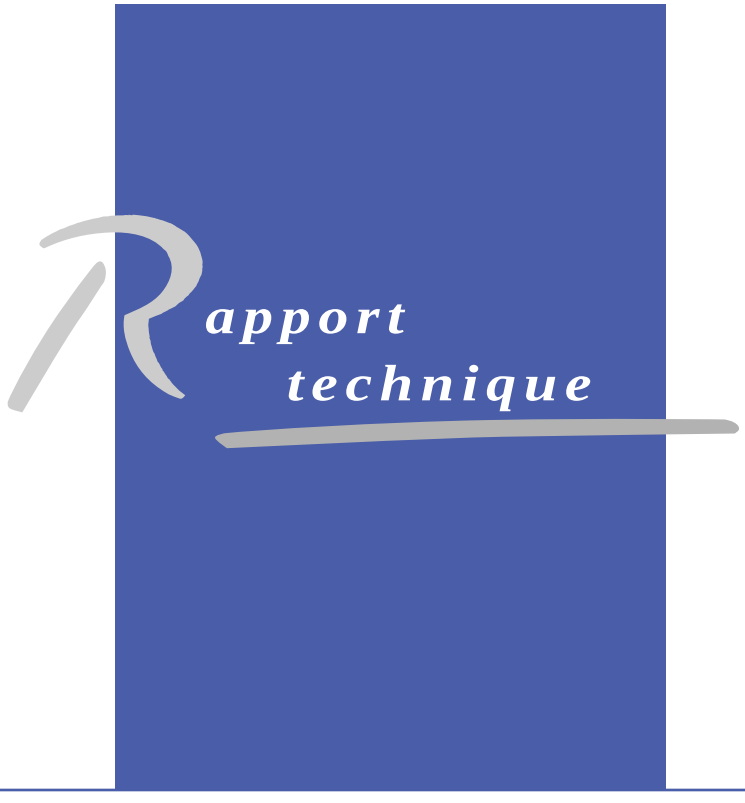
User guide of BioRica

Alice Garcia

N° 2 — version 2

initial version January 2011 — revised version Avril 2011

Computational Biology and Bioinformatics



ISSN 0249-0803 ISRN INRIA/RT--2--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
BORDEAUX - SUD OUEST

User guide of BioRica

Alice Garcia

Theme : Computational Biology and Bioinformatics
Équipe-Projet MAGNOME

Rapport technique n° 2 — version 2 — initial version January 2011 — revised version
Avril 2011 — 28 pages

Abstract: This document is the user guide of BioRica. This manual will be used by BioRica user to understand how it works and so how to use it. It allows user to discover Biorcia main features providing entry syntax and user cases.

Key-words: user guide,BioRica,biorica

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération 33405 Talence Cedex
Téléphone : +33 5 40 00 69 00

Manuel d’utilisation de BioRica

Résumé : Ce document est le manuel d’utilisation du projet BioRica. Il va être utilisé par les utilisateurs de BioRica pour comprendre comment fonctionne BioRica et donc comment l’utiliser. Il permet aux utilisateurs de découvrir les principales caractéristiques de BioRica en fournissant la syntaxe d’entrée et des cas d’utilisation présentant le fonctionnement.

Mots-clés : user guide,BioRica,biorica

Contents

Introduction

BioRica is a system to describe and simulate multi model. It's a software with a specific notation to create simulators.
The image in figure 1 presents the BioRica functioning. For a system describing models, the corresponding simulator is generated by BioRica. Then the system can be simulate running the simulator.

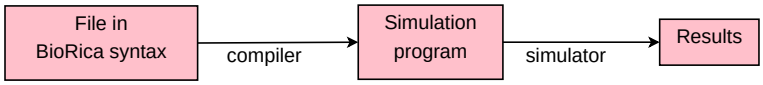


Figure 1: BioRica functioning with a compiler and a simulator

This manual presents the use of BioRica. The section 1 presents the BioRica syntax with examples. The section 2 defines some complex examples in BioRica syntax and presents the results of the simulation for these examples. The section 3 describes the installation and use on local machine.

1 BioRica syntax

This section provides BioRica users with a reference for the syntax of the language.

The notations used to described the language are :

- "::=" is used to define non terminal symbols and "|" denotes an alternative.
- A bracket expression matches any letter in the expression : "[abc]" means either "a", "b" or "c" ; "[a-z]" matches any lower-case letter from "a" to "z"."
- "t?", "t+" and "t*" match respectively 0 or 1, 1 or more and 0 or more repetitions of t

The section 1.1 presents the basic constituents such as numerical or boolean value, identifier, expressions or comments. The section 1.2 presents the biorica components.

1.1 Basic constituents

The comments in BioRica description are in the style of Python comments : any piece of text between '#' and the end of line is a comment. They are allowed anywhere in the description. Texts in comments are described by [a-zA-Z0-9_ -/\(){}[],;.:!?]*

A numerical constant is an integer or a float.
numerical_value ::= *integer* | *float*
integer ::= [-]?[0-9]+
float ::= [-]?[0-9]*.[0-9]+

A boolean value is True or False.
boolean_value ::= True | False

An idenfier is a sequence of letters, digits anc character "_"starting with a letter or "_". An identifier can't be a reserved word in ["BOOL", "INTEGER", "FLOAT", "POSITIVE_FLOAT", "NEGATIVE_FLOAT", "node", "edon", "flow", "state", "event", "sub", "trans", "assert", "sync", "extern", "or", "and", "True", "False", "not", "init", "choice", "law", "imply", "domain", "const", "min", "max", "in", "out", "eqdiff", "formula", "power", "exp", "log", "log1p", "log10", "sqrt", "root", "pow", "acos", "asin", "atan", "atan2", "cos", "sin", "tan", "hypot", "acosh", "asinh", "atanh", "cosh", "sinh", "tanh"].
identifier ::= [a-zA-Z_][a-zA-Z0-9_]*

When node, state variables, flow variables, event are embedded into hierarchies, the notation with "." is used.
variable ::= *identifier* | *hierarchy_path*.*identifier*
event ::= *identifier* | *hierarchy_path*.*identifier*
hierarchy_path ::= *identifier* | *identifier*.*hierarchy_path*

The syntax of the expression is :

exp ::= (*exp*)
::= *exp* `or` *exp*
::= *exp* (`and`|`&`) *exp*

::= (not|~) *exp*
::= *exp* = *exp*
::= *exp* > *exp*
::= *exp* < *exp*
::= *exp* <= *exp*
::= *exp* >= *exp*
::= *exp* != *exp*
::= *boolean_value*
::= max(*exp**)
::= min(*exp**)
::= *exp* + *exp*
::= *exp* - *exp*
::= *exp* * *exp*
::= *exp* / *exp*
::= - *exp*
::= *numerical_value*
::= *variable*
::= (*variable* | *numerical_value* | (*exp*)) ^ (*variable* | *numerical_value* | (*exp*))
::= (exp | log | log1p | log10 | sqrt | acos | asin | atan) (*exp*)
::= (cos | sin | tan | acosh | asinh | atanh | cosh | sinh | tanh) (*exp*)
::= (root | pow | atan2 | hypot) (*exp* , *exp*)

1.2 BioRica components

A BioRica description is made of a series of declarations of forumulas, constants, domains and nodes. It must contains at least the declaration of one node. The elements must be defined before being used.

biorica ::= (*formula_decl* | *constant_decl* | *domain_decl* | *node_decl*)*

1.2.1 Constant

The constants are used to defined constants used by constants, domains or in nodes. A constant is defined with a name and an expression of other constants, formulas or numerical values.

constant_decl ::= const (*identifier* = *exp* ;)+

The example below presents the definition of four constants C1 with the value 4 and C2 with the value C1+32 so C1 has the value 36, C3 with the value 15 and C4 with the value 5

```
const C1 = 4;
const C2 = C1 + 32; C3 = 15; C4 = 5;
```

1.2.2 Formula

The formulas are used to defined formulas used in the constant expressions, the node assertion expressions, the node transition conditions, the node transition assignments and the node differential equations. A formula is defined with a name and an expression. The expression can be composed by "?" and ":" statement to stand for a condi-

tional. The conditional "if cond : x = exp1 else x = exp2" corresponds to the formula "x = cond ? exp1 : exp2"

formula_decl ::= `formula` *identifier* = *formula_exp* `;`

forumula_exp ::= *exp*
::= *exp* `?` *exp* `:` *forumula_exp*

The example below presents the definition of a formula F and two constants C1 and C2. The formula F is used in the constant C2 expression.

```
formula F = 0.56452 / (15246 + C1);

const C1 = 32;
const C2 = F + 25631;
```

The second example below presents the definition of two formulas X and Y with conditionals using the constants C1 and C2. The formula Y says "if (C1 and C2): Y=1 else if C1: Y=2 else Y=3".

```
const C1 = True;
const C2 = False;
formula X = C1? 1 : 2;
formula Y = C1 and C2? 1 : not C1? 2 : 3;
```

1.2.3 Domain

A domain is either a numerical range, a set of symbolic constants or/and numerical values, a predefined domain or a named domain. Named domains are only used for the variables in node components to refer domains defined previously in components domain.

domain_decl ::= `domain` (*identifier* = *domain* `;`)+

domain ::= [*numerical_value*,*numerical_value*]
::= {(*numerical_value*|*identifier*)*} with *identifier* a constant name defined before
::= `BOOL` | `INTEGER` | `FLOAT` | `POSITIVE_FLOAT` | `NEGATIVE_FLOAT`
positive floats are floats positive or equal to zero (same for negative floats)
::= *identifier* with *identifier* a domain name defined before

The example below presents the definition of one constant C with the value 4 and four domains D1, D2, D3 and D4, respectively range, enumerated, float and named domains.

```
const C = 4;

domain D1 = [-40,40];
domain D2 = {20, -5, C, 45};
domain D3 = FLOAT; D4 = D1;
```

1.2.4 Node

A component node declaration is made of node elements and node fields which describe state variables, flow variables, events, sub-nodes, transitions, assertions, initial state values, external clauses and synchronisations.

node_decl ::= `node` *identifier node_elements node_fields* `edon`

The node definition depend on the node place in the hierarchy, if it's a leaf or not.

For the leaf :

node_elements ::= *state_decl*? *flow_decl*? *event_decl*?

node_fields ::= *transition_decl*? *assertion_decl*? *external_decl*? *init_decl*? *sync_decl*? *eqdiff_decl*?

For the branch :

node_elements ::= *flow_decl*? *event_decl*? *subnode_decl*?

node_fields ::= *assertion_decl*? *sync_decl*?

In each cases, *state_decl*, *flow_decl*, *event_decl* and *subnode_decl* clauses and *transition_decl*, *assertion_decl*, *external_decl*, *init_decl*, *sync_decl* and *eqdiff_decl* clauses can be given in any order.

Variable

A component variable is a list of state variables or flow variables. Flow variable has property to describe the flow orientation.

state_decl ::= `state` (*variable_list* `:` *domain* `;`)+

flow_decl ::= `flow` (*variable_list* `:` *domain* (`: in`|`out`)? `;`)+

variable_list ::= *variable* (`,` *variable*)*

The example below presents the definition of a node main with two states s1 and s2 in float domain, two states s3 and s4 in range [0,40] and one flow f in boolean domain with in as property.

```
node main

    state s1,s2:FLOAT;
          s3,s4:[0,40];

    flow f:BOOL:in;
edon
```

Event

A component event is a list of events.

event_decl ::= `event` (*event* (`,` *event*)*`;`)+

The example below presents the definition of a node main with three events ev1, ev2 and ev3.

```
node main

    event ev1;
          ev2,ev3;

edon
```

Subnode

A component subnode is a list of subnodes. A subnode is given by a name and a node name.

subnode_decl ::= `sub` *subnode_list+*
subnode_list ::= *identifier* (`,` *identifier*)* `:` *identifier* `;`

The example below presents the definition of a node A and a node B with three subnodes a1 and a2 of the node A.

```
node A
edon

node B
    sub a1, a2 : A;
edon
```

Transition

A transition component is a list of transitions. A transition is a discrete step that can change the state of the system. It contains a guard, an event and some assignments. The guard may be satisfied for the transition achievement. The transition is labeled by an event. If the transition is passed, the assignments on variables are done.

transition_decl ::= `trans` *transition+*
transition ::= *exp* `|-` *event* `->` (*assignment_list*)? `;`
assignment_list ::= *variable* `:=` *exp* (`,` *variable* `:=` *exp*)*

For a transition with a guard on the time, use the key word “time”. See the example for the external directive for an example of time transition.

The example below presents the definition of a node counter with a state cpt to count and some events and transitions to increase and decrease the counter.

```
node counter
    state
        cpt1,cpt2:[0,40];
    event
        incr,decr;
    trans
        True |- incr -> cpt1:=cpt1+1,cpt2:=cpt2+2;
        True |- decr -> cpt1:=cpt1-1;
edon
```

Assertion

An assertion component is a list of assertions. An assertion is used to describe invariants on variables. For the moment, the assertions must be given in a logic order to

observe the flows.
The assertions can use the clause “if”and “else”.

assertion_decl ::= `assert` (*assertion*)+

assertion ::= *assertion_equal*
::= `if` *expression* `:` (*assertion_equal*)+ (`else` (*assertion_equal*)+)?

assertion_equal ::= *variable* (=|`imply`|=>) *exp*

The example below presents the definition of a node supply with a state _isOn, a flow isOn, an event failure to change the value of the state _isOn and an assertion to link the flow and the state values.

```
node supply
    state
        _isOn:BOOL;
    flow
        isOn:BOOL;
    event
        failure;
    trans
        _isOn |- failure -> _isOn:=failure;
    assert
        isOn=_isOn
edon
```

The second example below presents the definition of a node main with the flows input and output and an assertion to link the two flows.

```
node main
    flow
        input:BOOL;
        output:FLOAT;
    assert
        output=input;
        if input : output = 50, input=False else output = -20;
edon
```

External directive

An external directive component describes event information. The event weight is given by the `choice` clause and the event probability law by the `law` clause. A law affects a delay at an event. The event laws and weights are used in the events to schedule the events

external_decl ::= `extern` (*external_directive*)+

external_directive ::= `choice` < (*event*)* > `:` *integer* `;`
::= `law` < (*event*)* > `:` (*func*|*numerical_value*) `;`
func ::= `Exponential` {*numerical_value*}
::= (`Normal`|`Uniform`) {*numerical_value* `,` *numerical_value*}
::= *numerical_value*

The example below presents the definition of a node counter with a state cpt to count and some events and transitions to increase and decrease the counter. A constant law is defined for the event incr and an exponential for the event decr. Weights are defined for the events. There is a time transition for the incr event for time more than 20.

```
node counter
    state
        cpt:[0,40];
    event
        incr,decr;
    trans
        True |- incr -> cpt:=cpt+1;
        True |- decr -> cpt:=cpt-1;
        time > 20 |- incr -> cpt:=cpt+20;
    extern
        law<incr>:5;
        law<decr>:Exponential{0.1};
        choice<incr>:20;
        choice<incr>:50;
edon
```

Initialization

An initialization component is a list of initializations used to initialize variable values.

init_decl ::= `init` *init_assignment* (`,` *init_assignment*)* `;`
init_assignment ::= *variable* `:=` *exp*

The example below presents the definition of a node counter with a state c initialized at 15 by the component `init`.

```
node counter
    state
        cpt1,cpt2:[0,40];
    init
        cpt1:=15,cpt2:=20;
edon
```

Synchronization

A synchronization component matches the events synchronisation.

sync_decl ::= `sync` (< *event* (`,` *event*)+ > `;`)+

The example below presents the definition of a node counter with two states a and b and two events incr_a and incr_b to increase respectively a and b. In this node a synchronization is defined to synchronize the events incr_a and incr_b and so a and b has all the time the same values.

```
node counter
    state
        a,b:[0,40];
    event
        incr_a,incr_b;
    trans
        True |- incr_a -> a:=a+1;
        True |- incr_b -> b:=b+1;
    sync
        <incr_a,incr_b>;
edon
```

Differential equations system

A differential equations component defines a differential equation system.

eqdiff_decl ::= `eqdiff` (*eqdiff* | *formula_decl*)+
eqdiff ::= `d`*identifier* = *exp* `;`

The example below presents the definition of a node eqdiff with two states m and l and a differential equations system for the states m and l. A formula F2 is defined in the differential equations system definition. This system is equivalent to a system with the both equations : $\frac{d(m)}{dt} = 0.1 * m - 0.2 * m * l$ and $\frac{d(l)}{dt} = -0.3 * l + 0.2 * m * l$.

```
node eqdiff
    state
        m,l:[0,100];
    eqdiff
        formula F2 = 0.2*m*l;
        dm = 0.1*m + -F2;
        dl = (-0.3*l) + F2;
    init
        m:=10,l:=10;
edon
```

For coupled equations, the symbol `d` can be used in the expression of the equations to specify the differential. The example below presents the definition of a node eqdiff with coupled equations for the states x, y and z. This system is equivalent to a system with the both equations : $\frac{d(x)}{dt} = 3 * x + y$, $\frac{d(y)}{dt} = 6 * y$ and $\frac{d(z)}{dt} = 5 * \frac{d(x)}{dt}$.

```
node eqdiff
    state
        x,y,z:FLOAT;
    eqdiff
        dx = 3*x+y;
        dy = 6*y;
        dz = 5*dx;
    init
        x:=0.1,y:=0.1,z:=0.1;
edon
```

When there are assertions and differential equations, formulas are added to the differential equations system.

The example below presents the definition of a node main with two states M and P, an assertion for the values of P and a differential equations system for the state M. A formula P is added to the differential equations system so the system is equivalent to a system with the equation $\frac{d(M)}{dt} = 4 * M + 2$.

```
node main
    state
        M,P:FLOAT;
    eqdiff
        dM = P;
    assert
        P = 4*M+2;
edon
```

2 BioRica simulation : Examples and simuation results

This section presents in the subsection 2.1 some examples of models in BioRica syntax. The subsection 2.2 describes the aim of the BioRica simulation and the subsection 2.3 gives some results for these examples.

2.1 The examples in the BioRica syntax

The three examples are chosen in order to match the different simulation ways in the case where there is a differential equation system and sub-nodes defined or not. All of these examples are in the folder example of the project.

2.1.1 Example 1 : a counter

The first example in the file example02_counter.br defines some nodes with a main node called main which is a counter systems. The code of it is given below :

```
node supply
    state

        _isOn:BOOL;
    flow
        isOn:BOOL:out;
    event
        failure;
    trans
        _isOn |- failure -> _isOn:=false;
    assert
        isOn=_isOn;
    extern
        law<failure>:Normal{20,0.5};
    init
        _isOn:=true;
edon

node counter
    state
        cpt:[0,40];
    flow
        isOn:BOOL:in;
    event
        incr,decr;
    trans
        isOn |- incr -> cpt:=cpt+1;
        isOn |- decr -> cpt:=cpt-1;
    extern
        law<incr>:Exponential{0.3};
        law<decr>:Exponential{0.1};
edon

node main
    sub
        c1,c2:counter;
        a:supply;
    assert
        c1.isOn=a.isOn;
        c2.isOn=a.isOn;
edon
```

2.1.2 Example 2 : a differential equation for the Mass growth

The second example in the file example14_eqdiff_mass.br defines a node main with a differential equation system representing the mass exponential growth. A transition is used to simulate the mass decrease by half when it reaches the value 3. The code of it is given below :

```
const mu = 0.04;

node main
    state
        Mass:FLOAT;
    event divide;
    eqdiff
        dMass = mu*Mass;
    init
        Mass:=0.5;
    trans
        Mass>=3 |- divide -> Mass:=Mass/2;
edon
```

2.1.3 Example 3 : a complex example of the Mass growth

The third example in the file example18_mass_coeff.br defines some nodes with a main node called main. This example defines a system with two sub-nodes, the first correspond to the main node of the previous example and the second is a counter to increment or decrement a variable called coeff. The main node specifies that the coefficient of the differencial equation is equal to the value of the counter. The code of it is given below :

```
node mass
    state
        Mass:FLOAT;
        coeff:[0.01,0.1];
    event divide;
    eqdiff
        dMass = coeff*Mass;
    init
        Mass:=0.5;
    trans
        Mass>=3 |- divide -> Mass:=Mass/2;
edon

node counter_coeff
    state
        coeff:[0.01,0.1];
    event
        incr,decr;
    trans
        True |- incr -> coeff:=coeff+0.02;
        True |- decr -> coeff:=coeff-0.01;
    extern
        law<incr>:Uniform{12,13};
        law<decr>:Uniform{5,6};
    init coeff:=0.04;
edon

node main
    sub
        c:counter_coeff;
        m:mass;
    assert
        m.coeff = c.coeff;
edon
```

2.2 The BioRica simulation

The simulation generates a trace giving the values of the system variables at each times. In the case without differential equations in the system, the times are the ones where transitions are passed, otherwise the times are the ones given at the equations resolution. If the system is hierarchical, the sub-nodes times are mixed.

In the node with differential equations, they are resolved. At each time, the simulation checks if a transition can be passed. In this case, the transition is passed and the resolutions in all of the system nodes with differential equations are restarted from this time. Then the same step are restarted.

At each time, the variables values are stocked and displayed as trace. The simulation is stopped when the maximal time or iteration number (1000) are reached.

The simulation, also create graph wanted in the current folder.

2.3 Simulation results examples

This section presents some results examples for the three examples considered.

For the example example02_counter.br, an example of the results obtained by the simulation is given below.

time	*main.a._isOn*	*main.a.isOn*	*main.c2.cpt*	*main.c2.isOn*	*main.c1.cpt*	*main.c1.isOn*	*event*
0	*True*	*True*	*0*	*True*	*0*	*True*	.
0.629573556	*True*	*True*	*0*	*True*	*1*	*True*	*main.c1->incr*
2.146580216	*True*	*True*	*0*	*True*	*0*	*True*	*main.c1->decr*
5.168891767	*True*	*True*	*0*	*True*	*1*	*True*	*main.c1->incr*
6.764232625	*True*	*True*	*1*	*True*	*1*	*True*	*main.c2->incr*
8.076996242	*True*	*True*	*1*	*True*	*2*	*True*	*main.c1->incr*
8.898819747	*True*	*True*	*1*	*True*	*3*	*True*	*main.c1->incr*
9.737104797	*True*	*True*	*1*	*True*	*4*	*True*	*main.c1->incr*
13.27250053	*True*	*True*	*2*	*True*	*4*	*True*	*main.c2->incr*
15.604006578	*True*	*True*	*2*	*True*	*5*	*True*	*main.c1->incr*
15.845399739	*True*	*True*	*2*	*True*	*4*	*True*	*main.c1->decr*
18.11485548	*True*	*True*	*3*	*True*	*4*	*True*	*main.c2->incr*
18.751931859	*True*	*True*	*3*	*True*	*5*	*True*	*main.c1->incr*
19.749095024	*False*	*False*	*3*	*False*	*5*	*False*	*main.a->failure*

For the example example14_eqdiff_mass.br with 200 iterations, a part of the results obtained by the simulation is given below.

The figure 2 shows the values of the variable called "Mass" for 200 iterations. This graph is generated by the simulation when the compilation is done using the "- -graph" option.

```
time    main.Mass    event
0       0.5    .
0.2     0.504016053466    .
0.4     0.508064365751    .
0.6     0.512145171628    .

...

44.4    2.9530923684    .
44.6    2.97681185863    .
44.8    1.50036093289    main->divide
45.0    1.51241198391    .
45.2    1.52455982038    .

...

61.8    2.96152906179    .
62.0    2.98531632535    .
62.2    1.50464732562    main->divide
62.4    1.51673280537    .
62.6    1.52891534703    .

...

199.6    2.86488546292    .
199.8    2.88789647022    .
200.0    2.91109230528    .
```

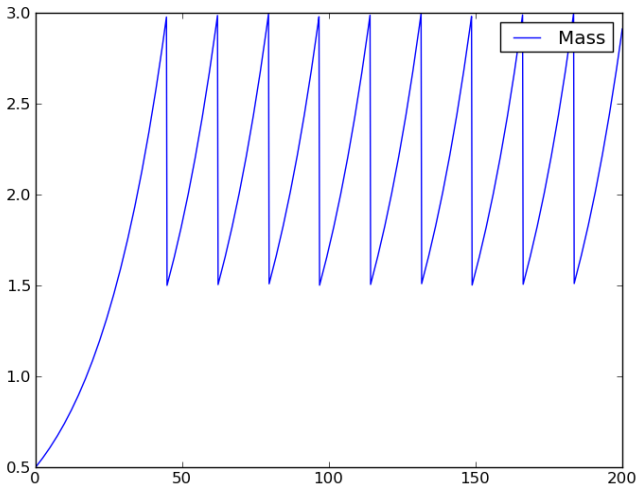


Figure 2: Graph of the variable Mass values for the exampe_eqdiff_mass.br for 200 iterations

For the example example18_mass_coeff.br with 100 iterations, a part of an example of the results obtained by the simulation is given below.

time	*main.c.coeff*	*main.m.Mass*	*main.m.coeff*	*event*
0	*0.04*	*0.5*	*0.04*	*.*
0.1	*0.04*	*0.502004009789*	*0.04*	*.*
0.2	*0.04*	*0.504016051688*	*0.04*	*.*
0.3	*0.04*	*0.50603615789*	*0.04*	*.*
0.4	*0.04*	*0.508064360718*	*0.04*	*.*
...				
5.1	*0.04*	*0.613149353929*	*0.04*	*.*
5.2	*0.04*	*0.615606868543*	*0.04*	*.*
5.207738303	*0.03*	*0.615797800678*	*0.03*	*main.c->decr*
5.307738303	*0.03*	*0.617647970253*	*0.03*	*.*
5.407738303	*0.03*	*0.619503698678*	*0.03*	*.*
...				
10.707738303	*0.03*	*0.726267838016*	*0.03*	*.*
10.807738303	*0.03*	*0.72844991573*	*0.03*	*.*
10.818563239	*0.02*	*0.72868683393*	*0.02*	*main.c->decr*
10.918563239	*0.02*	*0.730145666752*	*0.02*	*.*
11.018563239	*0.02*	*0.731607420162*	*0.02*	*.*
...				
12.218563239	*0.02*	*0.749378406932*	*0.02*	*.*
12.318563239	*0.02*	*0.750878664335*	*0.02*	*.*
12.417263905	*0.04*	*0.752362392876*	*0.04*	*main.c->incr*
12.517263905	*0.04*	*0.755377876076*	*0.04*	*.*
12.617263905	*0.04*	*0.758405445392*	*0.04*	*.*
12.717263905	*0.04*	*0.761445149265*	*0.04*	*.*
...				
99.583329515	*0.02*	*1.71908567452*	*0.02*	*.*
99.683329515	*0.02*	*1.72252728825*	*0.02*	*.*
99.783329515	*0.02*	*1.72597579209*	*0.02*	*.*
99.883329515	*0.02*	*1.72943119984*	*0.02*	*.*
99.979038984	*0.04*	*1.68167875382*	*0.04*	*main.c->incr*

3 Using BioRica locally

This section presents how to use BioRica on a local machine. The three examples example02_counter.br, example14_eqdiff_mass.br and example18_mass_coeff.br defined in the section 2.1 are used to show how to compile and simulate the both examples. The section 3.1 gives installation explanation. The section **??** presents how to convert a SBML file into a BioRica file, the section 3.3 how to compilate a model to generate the simulation file and the section 3.4 how to simulate a model using the generated file.

3.1 Installation

The BioRica project web page is : http://biorica.gforge.inria.fr/. The project is available on the software page : http://biorica.gforge.inria.fr/software.html.

The use of the project requires the installation of Python 2.6 and EasyInstall installer for Python projects. You can find information for these installations on the Python page http://www.python.org/ and the setuptools page http://pypi.python.org/pypi/setuptools, EasyInstall is include in the setuptools project.

BioRica depends on the numpy (http://numpy.scipy.org/), scipy (http://www.scipy.org/), matplotlib (http://matplotlib.sourceforge.net/) and ply (http://www.dabeaz.com/ply/) Python projects and the libSBML project (http://sbml.org/Software/libSBML).

The BioRica installation depends of the platform. The next sections present how to install BioRica and dependency on Unix and Windows.

3.1.1 BioRica installation on Unix and MacOS X

To install the requires on Unix, use the package installer or installer command of your distribution. For example, to install scipy, use the command "*urpmi python-scipy*" for Mandriva and "*sudo apt-get install python-scipy*" for Ubuntu.

You need to install in the same manner : Python 2.6, setuptools (python-setuptools), ply (python-ply), numpy (python-numpy), scipy (python-scipy) and matplotlib (python-matplotlib).

To install libSBML, you need to install first the "python-dev" package. Then, you need to download the sources provided in the libSBML download page and to build them using the command "./configure –with-python", "make" and "make install".

On MacOS X be careful to use the Python for python.org and not the Python provided by MacOS X. To install the requires on MacOS X, you can use .egg (Python eggs) or .dmg (MacOS X installer) provided on the components web-pages.

The BioRica installation is the same for Unix and MacOS X.

To install BioRica locally (this installation not need any rules), choose an emplacement like path_wanted/.

Then you need to specify this path to python in the PYTHONPATH variable. You can complete the PYTHONPATH permanently in your ".bashrc" or ".bash_export" file or temporally using the command in a terminal :
"*export PYTHONPATH:$ PYTHONPATH:path_wanted/*".

If you don't define the path permanently, you need to do this command again each time you open a new terminal to use BioRica.

Now, you can install BioRica, download the *BioRica-{BioRica version}.tar.gz* file (*BioRica-0.8.tar.gz* for the BioRica version 0.8) and install the archive using the python installer EasyInstall : *easy_install -d path_wanted/ BioRica-{BioRica version}.tar.gz*.

The installation generates a folder called *BioRica-{BioRica version}-py{python version}-egg*. For example for the versions 0.8 of BioRica and 2.6 of Python the folder is *BioRica-0.8-py2.6.egg*. The main folders generated are explained below :

- bin : the main programs, executables : "biorica_convert_sbml" to convert a SBML file into a BioRica file, "biorica_compil" to compile a BioRica or a SBML file and "biorica_simul" to simulate
- doc : this user guide and an explanation about the license chosen for BioRica, the CeCILL license.
- example : some examples of the BioRica syntax
- biorica_package : the BioRica sources

3.1.2 BioRica installation on Windows

To install BioRica on Windows, you need to define an environment variable PYTHONPATH to your Python installation. New variables can be added using the System Properties (Propriété système) dialog box.

Right-click on "My Computer" ("Poste de travail") and choose "Properties" ("Propriétés"). In the box that opens, click the "Advanced" ("Avancé"). Next, click the button "Environment Variables" ("Variables d'environnement"). Click on the "New" ("Nouveau") button in the "System variables" ("Variables système") part.

In the new windows opened, type "PYTHONPATH" as variable name and your path to your Python installation as variable value (generally, on windows default installation path are "C:\Program Files", so your Python installation should be "C:\Program Files\Python26").

After that, you need to install some Python libraries.

For ply, use the EasyInstall Python installer in a command prompt. Click on the Windows "Start" ("Démarrer") button, select "Programs" ("Programmes"), select "Accessories" ("Accessoires") and select "Command Prompt" ("Invite de commandes"). A new windows is opened. Type *%PYTHONPATH%\Scripts\easy_install.exe ply*, when installation is finished, close this windows.

Then install libSBML, numpy, scipy and matplotlib with executable files finding in the projects pages : http://sourceforge.net/projects/numpy/files/, http://sourceforge.net/projects/scipy/files/ and http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-1.0/

To install BioRica, download the *Biorica-{BioRica version}-win.zip* (*Biorica-0.8-win.zip* for the BioRica version 0.8), extract the archive and install the executable file in the extracted folder.

3.2 SBML conversion

The SBML conversion in BioRica is done using the program "biorica_convert_sbml" (executable on Windows and binary on Unix). This program can be used with some options :

- "- -help" : To optain the help
- "- -output" : To specify the name of the BioRica file in exit. For example "new_BioRica" for "new_Biorica.br", default is "{the SBML file name}.br".

The SBML conversion depends on the platform, the next sections present how to convert a SBML file on Unix, MacOS X and Windows.

3.2.1 SBML conversion on Unix or MacOS X

The SBML conversion on Unix or MacOS X is done in a Terminal. Go to your project directory, we will take the folder path_wanted/ created previously as an example. Your project is installed in the path_wanted/BioRica-0.8-py2.6.egg/ folder, go to this folder.

To convert a SBML file, use the program "biorica_convert_sbml" with the options given before : *./bin/biorica_convert_sbml [options]*.

We present here some SBML conversion call examples in a Terminal :

- *./bin/biorica_convert_sbml - -help*
  To obtain the help
- *./bin/biorica_convert_sbml example_SBML.xml*
  To convert the SBML file "example_SBML.xml" into the BioRica file "example_SBML.br", the main node in the BioRica file will be named "example_SBML"
- *./bin/biorica_convert_sbml example_SBML.xml - -output=my_BioRica_file*
  To convert the SBML file "example_SBML.xml" into the BioRica file "my_BioRica_file.br", the main node in the BioRica file will be named "my_BioRica_file"

3.2.2 SBML Conversion on Windows

To convert a SBML file on Windows, you need to use the command prompt. Click on the Windows "Start" ("Démarrer") button, select "Programs" ("Programmes"), select "Accessories" ("Accessoires") and select "Command Prompt" ("Invite de commandes"). A new windows is opened.

Move in your examples directory, we will take the folder BioRica-0.8-win extracted previously as an example. In the command prompt, use the command "cd" to move into directories, "cd .." is used to go out a folder and return to the parent folder and "cd Documents" to go in the "Documents" folder.

To convert a SBML file, use the program "biorica_convert_sbml.exe" with the options given before : *"%PYTHONPATH%"\Scripts\biorica_convert_sbml.exe [options]*.

We present here some SBML conversion call examples running the Windows command prompt :

- *"%PYTHONPATH%"\Scripts\biorica_convert_sbml.exe - -help*
  To obtain the help

- *"%PYTHONPATH%"\Scripts\biorica_convert_sbml.exe example_SBML.xml*
  To convert the SBML file "example_SBML.xml" into the BioRica file "example_SBML.br", the main node in the BioRica file will be named "example_SBML"
- *"%PYTHONPATH%"\Scripts\biorica_convert_sbml.exe example_SBML.xml - -output=my_BioRica_file*
  To convert the SBML file "example_SBML.xml" into the BioRica file "my_BioRica_file.br", the main node in the BioRica file will be named "my_BioRica_file"

3.3 Compilation

The compilation in BioRica is done using the program "biorica_compil" (executable on Windows and binary on Unix). Both BioRica and SBML file can be compiled by the BioRica compilation. This program can be used with some options :

- "- -help" : To optain the help
- "- -main" : To specify the name of the system main node, default is "main"
- "- -output" : To specify the name of the simulation file generated by the compilation, default is "simul_file" for 'simul_file.brs'
- "- -config" : To specify the name of the configuration file, default is "config_simul_file.cfg". If this file already exists the values are not changed, else it is generated by the compilation.

The compilation depends on the platform, the next sections present how to compile a BioRica file on Unix, MacOS X and Windows.

To illustrate the compilation, we will used our examples example02_counter.br, example14_eqdiff_mass.br, example18_mass_coeff.br.

3.3.1 Compilation on Unix or MacOS X

The compilation on Unix or MacOS X is done in a Terminal. Go to your project directory, we will take the folder path_wanted/ created previously as an example. Your project is installed in the path_wanted/BioRica-0.8-py2.6.egg/ folder, go to this folder.

To compile a BioRica example, use the program "biorica_compil" with the options given before : *./bin/biorica_compil [options]*.

We present here some compilation call examples in a Terminal :

- *./bin/biorica_compil - -help*
  To obtain the help
- *./bin/biorica_compil example/example02_counter.br*
  To compile the first example without option
- *./bin/biorica_compil example/example02_counter.br - -main=main*
  To compile the first example specifying that the main node is the node called "main"
- *./bin/biorica_compil example/example14_eqdiff_mass.br*
  To compile the second example without option

- *./bin/biorica_compil example/example18_mass_coeff.br*
  To compile the third example without option
- *./bin/biorica_compil example/example18_mass_coeff.br - -output=my_simul_file*
  To compile the third example specifying that the simulation file generated by the compilation will be the file "my_simul_file.brs"
- *./bin/biorica_compil example/example18_mass_coeff.br - -config=my_config_file.cfg*
  To compile the third example specifying that the configuration simulation file is the file "my_config_file.cfg"
- *./bin/biorica_compil example_SBML.xml*
  To compile an SBML example called "example_SBML.xml" with no option. The BioRica file "example_SBML.br" will be created by the compilation
- *./bin/biorica_compil example_SBML.xml - -config=my_config_file.cfg*
  To compile an SBML example called "example_SBML.xml" specifying that the configuration simulation file is the file "my_config_file.cfg"

3.3.2 Compilation on Windows

To compile a BioRica example, you need to use the command prompt. Click on the Windows "Start" ("Démarrer") button, select "Programs" ("Programmes"), select "Accessories" ("Accessoires") and select "Command Prompt" ("Invite de commandes"). A new windows is opened.

Move in your examples directory, we will take the folder BioRica-0.8-win extracted previously as an example. In the command prompt, use the command "cd" to move into directories, "cd .." is used to go out a folder and return to the parent folder and "cd Documents" to go in the "Documents" folder.

To compile a BioRica example, use the program "biorica_compil.exe" with the options given before : *%PYTHONPATH%\Scripts\biorica_compil.exe [options]*.

We present here some compilation call examples running the Windows command prompt :

- *"%PYTHONPATH%"\Scripts\biorica_compil.exe - -help*
  To obtain the help
- *"%PYTHONPATH%"\Scripts\biorica_compil.exe example\example02_counter.br*
  To compile the first example without option
- *"%PYTHONPATH%"\Scripts\biorica_compil.exe example\example02_counter.br - -main=main*
  To compile the first example specifying that the main node is the node called "main"
- *"%PYTHONPATH%"\Scripts\biorica_compil.exe example\example14_eqdiff_mass.br*
  To compile the second example without option
- *"%PYTHONPATH%"\Scripts\biorica_compil.exe example\example18_mass_coeff.br*
  To compile the third example without option

- *"%PYTHONPATH%"\Scripts\biorica_compil.exe example\example18_mass_coeff.br - -output=my_simul_file*
  To compile the third example specifying that the simulation file generated by the compilation will be the file "my_simul_file.brs"
- *"%PYTHONPATH%"\Scripts\biorica_compil.exe example\example18_mass_coeff.br - -config=my_config_file.cfg*
  To compile the third example specifying that the configuration simulation file is the file "my_config_file.cfg"
- *"%PYTHONPATH%"\Scripts\biorica_compil.exe example_SBML.xml*
  To compile an SBML example called "example_SBML.xml" with no option. The BioRica file "example_SBML.br" will be created by the compilation
- *"%PYTHONPATH%"\Scripts\biorica_compil.exe example_SBML.xml - -config=my_config_file.cfg*
  To compile an SBML example called "example_SBML.xml" specifying that the configuration simulation file is the file "my_config_file.cfg"

3.4 Simulation

The simulation is done using the program "biorica_simul" and the Python file generated by the compilation.
The simulation, also create graph wanted in the current folder. For example, for the first example "example02_counter.br" compiled with the option "- -graph=[c1.cpt+c2.cpt, c1.isOn+c2.cpt]", the simulation will create two files named "result_c1-cpt_c2-cpt.png" and "result_c1-isOn_c2-isOn.png" respectively for the variable "cpt" of the nodes "c1" and "c2" sub-node of the main node and the variable "isOn" of the nodes "c1" and "c2" sub-node of the main node.

The Biorica simulation program "biorica_simul" can be used with some options :

- "- -help" : To optain the help
- "- -iter" : To specify the number of simulation step wanted, default is 1000
- "- -graph" : To specify the variables wanted to be plotted. If more than one state are wanted in the same graph, the notation + is used (- -graph=[X,Y] for plotting a graph for X and one for Y, - -graph=[X+Y,Z] for plotting a graph with the values of X and Y and one for Z). By default no graph are shown. Hierarchies are described by '.' to show or print only some variables in sub-nodes (- -graph=[A.B.X,A.C.Y] to show the variable X of the node B sub-node of the node A and the variable Y of the node C sub-node of the node A).
- "- -input" : To specify the name of the simulation file to use for the simulation, default is "simul_file.brs".

The options can be used all in the same time and in any order.

To illustrate the simulation, we will used our examples example02_counter.br, example14_eqdiff_mass.br, example18_mass_coeff.br. The simulation depends on the platform, the next sections present how to run the simulation on Unix and Windows.

3.4.1 Simulation on Unix or MacOS X

Your project is installed in the path_wanted/BioRica-0.8-py2.6.egg/ folder, go to this folder.

Compile your BioRica example, then to simulate the simulation file created by the compilation, use the next command :
./bin/biorica_simul

We present here some simulation call examples on Unix in a Terminal :

- *./bin/biorica_simul - -help*
  To obtain the help
- *./bin/biorica_simul*
  To simulate the model just compiled without options
- *./bin/biorica_simul - -iter=2000*
  To simulate the model just compiled with 2000 iterations instead of 1000
- *./bin/biorica_simul - -graph=[c1.cpt]*
  To simulate the model just compiled and print a graph for the variable "cpt" of the node "c1" sub-node of the main node. This example supposed that you just compile the example example02_counter.br
- *./bin/biorica_simuly - -graph=[c1.cpt+c2.cpt,c1.isOn+c2.cpt]*
  To simulate the model just compiled and print a graph for the variable "cpt" of the nodes "c1" and "c2" sub-node of the main node and a graph for the variable "isOn" of the nodes "c1" and "c2" sub-node of the main node. This example supposed that you just compile the example example02_counter.br
- *./bin/biorica_simul - -iter=200 - -graph=[Mass]*
  To simulate the model just compiled with 200 iterations and to print a graph for the variable Mass of the main node. This example supposed that you just compile the example example14_eqdiff_mass.br
- *./bin/biorica_simul - -input=my_simul_file.brs*
  To simulate the model in the simulation file "my_simul_file.brs"

3.4.2 Simulation on Windows

Compile your BioRica example, then to simulate the simulation file created by the compilation, use the next command in the command prompt :
"%PYTHONPATH%"\Scripts\biorica_simul.exe

We present here some simulation call examples running the Windows command prompt :

- *"%PYTHONPATH%"\Scripts\biorica_simul.exe - -help*
  To obtain the help
- *"%PYTHONPATH%"\Scripts\biorica_simul.exe*
  To simulate the model just compiled without options
- *"%PYTHONPATH%"\Scripts\biorica_simul.exe - -iter=2000*
  To simulate the model just compiled with 2000 iterations instead of 1000

- *"%PYTHONPATH%"\Scripts\biorica_simul.exe - -graph=[c1.cpt]*
  To simulate the model just compiled and print a graph for the variable "cpt" of the node "c1" sub-node of the main node. This example supposed that you just compile the example example02_counter.br
- *"%PYTHONPATH%"\Scripts\biorica_simul.exe - -graph=[c1.cpt+c2.cpt,c1.isOn+c2.cpt]*
  To simulate the model just compiled and print a graph for the variable "cpt" of the nodes "c1" and "c2" sub-node of the main node and a graph for the variable "isOn" of the nodes "c1" and "c2" sub-node of the main node. This example supposed that you just compile the example example02_counter.br
- *"%PYTHONPATH%"\Scripts\biorica_simul.exe - -iter=200 - -graph=[Mass]*
  To simulate the model just compiled with 200 iterations and to print a graph for the variable Mass of the main node. This example supposed that you just compile the example example14_eqdiff_mass.br
- *"%PYTHONPATH%"\Scripts\biorica_simul.exe - -input=my_simul_file.brs*
  To simulate the model in the simulation file "my_simul_file.brs"

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex (France)

Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803