Developing a bioeconomic simulation tool of fisheries dynamics: a case study

[1]D. Versmisse, [2]C. Macher, [1]É. Ramat, [1]J.C. Soulié, and [2]O. Thébaud

[1] LIL - Laboratoire d'Informatique du Littoral
Maison de la Recherche Blaise Pascal
50, rue Ferdinand Buisson - BP 719
62228 CALAIS Cedex FRANCE
E-Mail: versmisse@lil.univ-littoral.fr,
ramat@lil.univ-littoral.fr,
soulie@lil.univ-littoral.fr

[2] Marine Economics Department - IFREMER
BP. 70
29280 Plouzané
E-mail: Claire.Macher@ifremer.fr,
Olivier.Thebaud@ifremer.fr

Keywords: *DEVS, Coupling models, Fisheries Dynamics, Bio-economy*

ABSTRACT

The purpose of this article is to present a simulation tool developed by our team in order to help answer problems of mixed fisheries modelling and simulation. This work deals with the context of the CHALOUPE Biodiversity project funded by the NRA (French National Research Agency). The aim is to model and simulate the Bay of Biscay nephrops-hakes fisheries.

In this article we will, first, present the technical part of our work. Indeed, we have to take into account different parameters: economic, technical, and biological factors. Difficulties relate, in particular, to the complex dependencies between the various components of the model. The goal, here, is to provide a tool that allows quick development, modification and use of fisheries models. In particular, the modelling tool includes the repetitive and complex tasks of parameterization using standard data format. Main technologies used are: DEVS formalism, the XML language, the C++ language, the embedded possibilities of PYTHON, and our own simulation platform: VLE (acronym for Virtual Laboratory Environment). Second, we present simulation preliminary results concerning the management of the Bay of Biscay hake-nephrops fishery. This fishery is characterized by technical interactions between trolling for nephrops and for the hake fishery. We simulate the potential impact changes in technical regulations concerning nephrops harvesting and analyze the implications for the fleet harvesting both hake and nephrops.

1 INTRODUCTION

The effects of fishing and of climate change have by now been identified as key factors in the biological evolution of marine populations and communities. The effect of this evolution on fisheries specifically has been that its development operates in a context of failed regulations on access to resources (leading to fleet overcapacity, increasing demand for fisheries products, and the deregulation of markets). The extent of these changes and the relative weight of different factors on a regional scale (*i.e* Bay of Biscay, for instance)have yet to be quantified. This is the principal objective of the CHALOUPE Project[1] funded by the NRA (French National Research Agency), which was launched in February 2006.

To carry out the objectives fixed, four tasks have been defined and will be carried out for three case studies. The four tasks are:

1. To measure the response of the communities, environmental temporal variations and impact of fishing;
2. To measure the response of fisheries to ecological, economic and institutional changes;
3. To Diagnose of the ecological and economic status and the conditions of viability of the exploited communities/fisheries systems;
4. To model and simulate possible evolutions in the bio-economic system.

The three case studies are: the Bay of Biscay that represents a temperate continental shelf, the Moroccan up-welling that represents a shelf under the influence of an up-welling, and the shelf of French Guyana that represents an amazonian tropical shelf.

[1]http://www.projet-chaloupe.fr/

This paper presents a work that takes place within the task four and applied to the Bay of Biscay case study. Our position within the CHALOUPE project is to provide a set of tools allowing quick writing and development models created jointly with the biologists and economists. Our team has worked for several years on the modeling and simulation problems and we have developed a framework named VLE (acronym of Virtual Laboratory Environment) [RP03, QDRT07]. VLE is a DEVS [Zei76] engine and is entirely written using C++. The simulation set up process is realized by charging the description of the model from a XML file[2] (number of atomic models, their names, their types, the connections between them, and their dynamics) and loads the appropriate plugins to run it. Each DEVS atomic model is a plugin written in C++ or in Python. Finally, VLE allows the user to write arbitrary XML part within the initialization file. These XML pieces are translated into a "vle compliant" XML thanks to another type of dedicated plugin: the *translators*.

This paper is organized as follows: first we present the specification of our model, second the solution we developed in order to simulate our model, and finally, we present a set of simulation results representing the Bay of Biscay nephrops-hakes fisheries.

2 THE NEEDS IN THE CHALOUPE PROJECT

In this section, we present the paradigm used in our model and the consequences due to this choice. Indeed, we have to consider two parts: the biological part and the economical part and we have also to represent the interactions between these two parts.

2.1 A large set of strongly recurrent equations

The core of our bioeconomic models relies on a large series of equations. For instance, in our current model representing the Bay of Biscay nephrops-hakes fisheries, we have the number of individuals N_a (for a given species, for a given age class), that is computed with:

- N_{a-1} of the number of individuals with are in the $a-1$ age class;
- Z the total mortality which results from fishing mortality and natural mortality for a given species, for a given class).

and the formula:

$$N_a := N_{a-1} \times e^{-Z}$$

[2]http://www.w3.org/XML/

Our model of the Bay of Biscay defines: nephrops and hakes species based on 9 classes of age, 4 métier, a price models, and a market model. Finally we define a total of 2130 recurrent equations like presented above. A variable (a model's output) depends on about 3.3 inputs. Given this large set of strongly recurrent equations a number of difficulties appear:

- How to manage all of these equations?
- How to describe the dependency between these equations? Given the fact that they are complex and the system is absolutely not linear;
- How to keep the same equation to make two different computations? For instance, with the natural growing equation, it is always the same equation but with different inputs;
- And the most important, how to keep all these computations efficient?

2.2 The VLE framework provides a DEVS context

In the framework of our simulator VLE, we have to describe all models like a DEVS (acronym of Discrete EVent system Specification) [Zei76, ZKP00] atomic model.

DEVS defines a model M as follows:

$$M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \tau >$$

Where X is the input space, S is the system state space, Y is the output space, $\delta_{int} : S \longrightarrow S$ is the internal state transition function, $\delta_{ext} : \mathcal{Q} \times S \longrightarrow S$ is the external transition function, $\mathcal{Q}$ is the "total state" (the set of all previous and actual model states), $\lambda : \mathcal{Q} \longrightarrow Y$ is the output function and τ is the time-advance function.

DEVS allows coordinates the components of a large complex system and makes coupled models manipulation easier. After their definitions, models will exchange timestamped messages in order to communicate.

So, it is necessery to translate the resolution of the equations via the DEVS mechanisms. This translation is a key point of our work because it can raise several problems whose principals ones are:

- It could be several time scales;
- Some outputs must be computed with the values of the entries at t_i, others with the values at t_{i-1};

- Some variables must be set up with a computation that depends on the initial data;
- Some equations are complex, it is difficult to describe them using XML and it is necessary to find a simple way to describe them;
- After having described the binding between the equations, the algorithm must choose the order it will evaluate these equations. This last point is the most difficult one.

It is often difficult to realize this translation from a formalism to the DEVS formalism, but it has to be realized. Indeed, when we have all the components translated into the DEVS formalism, we can mix them very more easily. This is why after the implementation of our *equations* tool, we can use it with all existing models (for instance: a differential equations solver, timers, an agent modeller, ...). The only work to do is to connect atomic or coupled models together.

Although VLE is written in C++, we can write a model (an atomic model in the DEVS language) in several languages (C++, Python, .Net, R and Java). It is simply a class that inherites from a pre-defined one with the DEVS functions (`Delta_ext`, `Delta_int`, ...). This feature has been developed in order to avoid the C++ difficulties (explicit pointer management, multiple inheritance, etc.). Indeed, some scientists not involved in implementation works could be discouraged by developing with C++ although they know other programming languages.

3 OUR SOLUTION TO THE MANAGEMENT OF RECURRENT EQUATIONS

Our solution is based on our VLE framework to run the simulation, the XML language for the description of the model and the Python language to write and evaluate the equations.

3.1 Description of our XML specification

We developed an XML syntax that allows to simplify the description of the model and its dependencies. Such class of syntaxes are based on tags that reprensent keywords in the grammar.

The first tag we proposed is the `<PARAMETERS>` tag defining:

- The different time scales;
- The file format in which the models' outputs are stored. In our case this is CSV files. This file format is very often used because it can be read directly by the classical spreadsheets;
- The inputs in order to set up some part of our model. These inputs are stored into CSV files or databases.

These definitions give a name to reference tops, inputs and outputs in the rest of the XML, it is shorter and more practical. The different time scales are a multiple of a `STEP`.

The listing below shows the PARAMETERS part of our XML file. We can see the definition of three "`top`" with a time step of 0.5, 1 and 1.5. The definition of two outputs "`all`" and "`trainee`", stored into CSV files. And finally an input "`my_csv`" which is a CSV file too.

```
<PARAMETERS>
  <CLOCK STEP="0.5">
    <TOP NAME="top1" MULT="1"/>
    <TOP NAME="top2" MULT="2"/>
    <TOP NAME="top3" MULT="3"/>
  </CLOCK>
  <SCRIBE>
    <CSV NAME="all"
      FILE="output_all.csv"/>
    <CSV NAME="trainee"
      FILE="output_trainee.csv"/>
  </SCRIBE>
  <STREAM>
    <CSV NAME="my_csv"
      FILE="example.csv" TYPE="TITLE"/>
  </STREAM>
</PARAMETERS>
```

Listing 1. A part of our XML file

Second, we have the `<ROOT>` tag which contains the description of the model. It is a tree structure. The `<NODE>` and `<MODULE>` tags are the containers for others tags.

```
<ROOT>
  <NODE NAME="OFFICE">
    <CONSTANT NAME="K" VALUE="2"/>

    <MODULE NAME="The boss" PYTHON="example"
      CLASS="Boss">
      <IN PATH="K"/>
      <OUT NAME="A" CSV_NAME="Boss"/>
    </MODULE>

  </NODE>
</ROOT>
```

Listing 2. Example of `<ROOT>` node, It is a tree structure. Objects are in `<NODE>` and `<MODULE>`

A module defines an equation with its entries and its outputs. The entries are the `<IN>` tags. The `PATH` describes how and where to find the value. It could be an output of another module or a constant. Outputs (`<OUT>` in modules) and constant (`<CONSTANT>`) have got two names, a short and a long one. The short is for instance: given the `<CONSTANT NAME="K" VALUE="2"/>`, the name `K`, and the long is `ROOT:OFFICE:K`. So in a `PATH` tag, we can give:

- The long name of an entry;

- The short name, and the system will look for an output or a constant with this name in the higher branches of the tree. For instance K in the The Boss module;
- A relative path using the explicit notation "..". For instance A in the Trainee N1 module.

In order to handle the different time scales and the nature of the inputs, modules have a tag TOP and three MODE which are:

- LAZY: it computes its OUT only one time. This mode is used to evaluate some constants at the beginning of the computation;
- preevaluation: it computes its OUT with the values of its INPUT at the time $t-1$;
- postevaluation it computes its OUT with the values of its INPUT at the time t, in fact it must wait to have all of its INPUT.

The computation is made in a library loaded and written in C++. As presented above, it as also possible to use the Python language but it is less efficient in term of computation time. We use the Python language during the development process because it as an interpreted language and it avoids repetitive compilations that are time consuming. When the development process is ended, we translate the code written in Python into the C++ language. Consequently we can take advantage of the C++ efficiency.

```
<MODULE NAME="Trainee N1"
  PYTHON="example" CLASS="Trainee"
  TOP="top1"
  MODE="preevaluation">

  <IN PATH="..:The boss:A"/>

  <CONSTANT NAME="FACTOR"
    VALUE="my_csv#'Trainee'='1'|'Value'"/>

  <IN PATH="FACTOR"/>
  <OUT NAME="A" CSV_NAME="A1" SCRIBE="trainee"/>
</MODULE>
```

Listing 3. Example of <MODULE>. It is an equation, the inputs are A and FACTOR. The output (or result) is A. The computation is made into a Python file example.py and the used class to evaluate is Trainee

In OUT tag, we have the optional attribute INIT (for the initial value of this variable). Without the INIT the system tries itself to calculate the value. This computation is automatic, integrated in the DEVS model of a module.

We also added the possiblity to make loops, tests and define constants, and finally, we created a syntax to carry out the requests in CSV files.

```
<FOR VAR="trainee_number" IN="my_csv#|'Trainee'">
  <MODULE NAME="Trainee N$trainee_number$"
    PYTHON="$python_file$" CLASS="Trainee"
    TOP="top$(trainee_number-1)%3+1$"
    MODE="preevaluation">

    <IF EXPRESSION="$trainee_number .eq 1$">
      <IN PATH="..:The boss:A"/>
    </IF>

    <IF EXPRESSION="$trainee_number .neq 1$">
      <IN
        PATH="..:Trainee N$trainee_number-1$:A"/>
    </IF>

    <CONSTANT NAME="FACTOR" VALUE=
"my_csv#'Trainee'='$trainee_number$'|'Value'"/>

    <IN PATH="FACTOR"/>

    <OUT NAME="A" CSV_NAME="A$trainee_number$"
      SCRIBE="trainee"/>
  </MODULE>
</FOR>
```

Listing 4. This is the rewriting of the "Trainee" module with a loop and two tests

We can see in the last example that our translator has a little expressions interpreter. All expressions surrounded by $ are evaluated and switched for its result in the character string. Expressions could be complex, for instance this one:$(trainee_number-1)%3+1$ computes the good number for the TOP tag. We can see the reading in input my_csv, a stream defined in the <PARAMETERS> tag, by the string 'Trainee'='$trainee_number$'|'Value'.
It is the request in the CSV file. We have invent a small language for these requests in a CSV file. If the stream is a classic data base, we send directly the SQL request to the data base.

3.2 The DEVS model of the equation handler

The DEVS model of the simulator which evaluates an equation must take into account of all requirements presented in the first part. Our solution is a DEVS atomic model with six states: INIT_WAITING, INIT_SENDING, INIT_OK, WAITING, SENDING and IDLE. And, as defined above, three modes PREEVALUATION, POSTEVALUATION and LAZY. We described classic DEVS procedures (ta, λ, δ_{int} and δ_{ext}) of this atomic model in the below algorithms. They are sufficiently explicit, we will only clarify some points. The PREEVALUATION mode qualify equations whose outputs must be calculated with the values of the entries at t_{i-1}, the POSTEVALUATION mode waits that all input values (at $t = t_i$) are received and calculates the outputs and finally the LAZY mode computes only one time the outputs (at $t = 0$) and sends always the same values during the simulation. This last mode is for the calculation of constants at the beginning of the simulation.

The model designer should not rewrite or think in term of model DEVS. He only must overload, or accept the default behavior of a serie of functions/procedures whose are called by the DEVS model which are:

- $initialize()$ Called before the start of the simulation, by default, this procedure does nothing;
- $auto\_init()$ Called during the computation of all initial value for each variable. This procedure have a default behavior, which is: waiting for all the inputs of an equation and call $compute(0)$;
- $compute$(t) Called exactly at the good time by the DEVS model to ask an evaluation of the equation with the actual input values. This function must return the result of the computation;
- $finalize()$ Called after the end of te simulation.

Here is an extract of our model. This part computes the equation: $C_a = \frac{N_a \times (1 - e^{-Z_a})}{Z_a \times F_a}$

```
class Catch(Treq):
    def compute(self,t):
        self.set_out("C",
            self.get_in("N")*\\
            (1-exp(-self.get_in("Z")))/\\
            self.get_in("Z")*self.get_in("F"))
```

Listing 5. An example of complex equation written in a Python file and executed by VLE exactly at the good time according to the specification of the module that uses this code. We can use all the power of Python to describe the equation and its result.

This part computes C_a, it does not use an overloaded $auto\_init()$ function, so to calculate C_a at the beginning of the simulation, the program waits for N, Z and F, and calculates C_a with the call of $compute(0)$. As we can see, the DEVS model solves the problems of evaluation automatically: the correct control of the set up, *i.e* in which order is necessary to initialize the variables, and then at the time of simulation.

Let us see now the DEVS model of an atomic model that handles an equation.

First, the $init$ and the $finalize$ functions which are called at the beginning and at the end of the simulation.

```
call initialize()
call auto_init()
//iteration stores the number of "top"
iteration←0
state←INIT_SENDING
```

Listing 6. The DEVS model of an equation: The $init$ function

```
call finalize()
```

Listing 7. The DEVS model of an equation: the $finalize$ function

Second, the Ta function of our solution with the various returns according to the state of the module. Ta specifies the duration of a state.

```
if state=INIT_WAITING
   return step
else if state=INIT_SENDING
   return 0
else if state=INIT_OK
   return 0
else if state=WAITING
   return step
else if state=SENDING
   return 0
else if state=IDLE
   return step
end if
```

Listing 8. The DEVS model of an equation: the Ta function

Third, the λ function. It is the output of a DEVS atomic model. An output is an event which is send to the good modules. With the `PATH` tag in each input from a module, the translator knows which is connected to a module and where it must send the event which is the new value of a variable (its long name and its new value).

```
if state=INIT_WAITING
   return "No event"
else if state=INIT_SENDING
   if size(out_dict)=total_out
      return "All outputs"
      state←INIT_OK
   else
      return "No Event"
   end if
else if state=IDLE
   return "No event"
else if state=WAITING
   return No event
else if state=SENDING
   return "All outputs"
end if
```

Listing 9. The DEVS model of an equation: the λ function. `total_out` stores the number of `OUT` tag in the module

The δ_{int} function is called by the DEVS engine when a state is finished. According to its last state the

atomic model moves to a new state and can make some computations.

```
if state=INIT_WAITING
   error "This module has not
      received enought
      data to make its
      self initialization"
else if state=INIT_SENDING
   state←INIT_WAITING
else if state=INIT_OK
   state←IDLE
else if state=IDLE
   iteration←iteration+1
   if iteration mod mult=0
      mode=PREEVALUATION
      call compute(t)
      state←SENDING
   else if mode=POSTEVALUATION
      if total_input ≠ 0
         received_input←0
         state←WAITING
      else
         call compute(t)
         state←SENDING
      end if
   else if mode=LAZY
      state←SENDING
   end if
else if state=WAITING
   nop
else if state=SENDING
   state←IDLE
end if
```

Listing 10. The DEVS model of an equation: the δ_{int} function. `total_in` stores the number of `IN` tag in the module which are not constants

We can remark that the model is able to detect an error in the graph of dependencies of the equations: if an equation could not compute its result due to it did not receive all its inputs, an error is raised. Generally, it is because some variables do not have an initialisation value and it is impossible to evaluate it with the given datas.

Finally, the δ_{ext} function. It is called when an event (so a new value for a variable) comes.

```
call set_in(name,value)
if state=INIT_WAITING
   call auto_init()
   state←INIT_SENDING
else if state=INIT_SENDING
   call auto_init()
else if state=INIT_OK
   nop
else if state=IDLE
   nop
else if state=WAITING
   received_input←received_input+1
   if total_input=received_input
      call compute(t)
      state←SENDING
   else if state=SENDING
      nop
   end if
end if
```

Listing 11. The DEVS model of an equation: the δ_{ext} function. The event (the new value of a external variable) has two argument its `name` and its `value`

This model is implemented in our VLE framework and all the equations (the code with the functions: $auto\_init$, $compute$, ...) are stored into a Python file. The data are stored into a `ods` file (an openDocument spreadsheet format[3]). A small program converts this file in several CSV files for VLE. So, we can change a value in the spreadsheet or change an equation implementation in the Python file and rerun directly the simulation. The results can also be read by a spreadsheet. A run roughly takes a minute to be carried out with 50 time steps.

4 PRESENTATION OF THE MODEL AND THE RESULTS

The French Nephrops trawler fishery in the Bay of Biscay is characterized by a high level of discards of many species especially Nephrops and Hake. Talidec *et al.* [TRBM05] estimated that Nephrops trawlers discard about half of their Nephrops catches in numbers, and a third in weight. Discarding mainly occurs on the younger ages of Nephrops and Hake that are under the Minimum Landing Size (MLS) fixed by the European Commission in the framework of the CFP. Because of high mortality rates on discards, these discarding behaviours lead to an important waste for the stocks and for the fleets. Nephrops trawlers themselves as well as other fleets targeting discarded species like the hake netters are affected by the low selectivity of the trawlers. The simulation tool enables us to study impacts of implementing management measures on the status of the stocks (in terms of biomass and age structure) and on several indicators for the fleets (such as the gross return from the catches or the return to be shared). The model is age-structured and deterministic. It is close to the model described in [MGTB06] but includes the dynamic of hake and two netter fleets. A scenario of improving selectivity on the three first age groups of Nephrops and Hake caught by the Nephrops trawler fleets was tested using the "VLE" model. This scenario corresponds to the case of the adoption by the Nephrops trawler fleet of a selective device according to the regulation on MLS: no hake and no Nephrops under MLS are caught neither discarded.

The biological parameters for Nephrops and Hake

[3] http://opendocument.xml.org/

stock dynamic are provided by the ICES working groups [ICE04a, ICE04b]. Six Nephrops trawler fleets practicing the métiers of Nephrops single bottom trawling and Nephrops twin bottom trawling and two Hake netter fleets were parameterized for their cost and revenue structure and their exploitation pattern for the two stocks using the data collected by IFREMER (SIH). The potential benefits of the scenario implemented in simulated year 30 are addressed. The model enables to provide the results at different level of aggregation: per fleets, per métiers or per fleet-métier, for the whole stocks or detailed per age-group. At present the model is within a validation stage. The preliminary results are very encouraging, but we want to confront them with the known data before advancing a series of curves and interpretations.

5 CONCLUSION

In this article, we have presented a tool and a way to handle a model of complex fisheries. The design of our solution allows to add more complexity in the model: we can add new equations or news DEVS models and connect them together. In the past, we have already realized a model of fisheries relies on a DEVS equation differential solver, and so now, we can mix all the point of view.

We have also seen that this tool is enough effective to handle the complex model of the Bay of Biscay nephrops-hakes fisheries. It is easy to write a new model or modify an old one: the data are brought together in a spreadsheet file, the complex equations in a Python file and the description of the model in a XML file. This last file is the more difficult to do, so we currently have developed a graphical tool to write this file automatically.

6 ACKNOWLEDGEMENT

The authors would like to thank the CHALOUPE project(agreement n° ANR-05-BDIV-001-06, French Research National Agency - Biodiversity Program) for the stimulating intellectual environment it provided to us, as well as the financial support for this work.

7 REFERENCES

ICES. Report of the working group on nephrops stocks. ICES CM ACFM:19, 2004.

ICES. Report of the working group on the assessment of southern shelf stocks of hake,monk and megrim. ICES CM ACFM:02, 2004.

C. Macher, O. Guyader, C. Talidec, and M. Bertignac. A cost-benefit analysis of improving trawl selectivity: the nephrops norvegicus fishery in the bay of biscay. AMURE PUBLICATIONS, ISSN 1951-641X, Working Papers Series N° D-20, 2006.

G. Quesnel, R. Duboz, É. Ramat, and M. K. Traouré. A multimodeling and simulation environment, proceedings of the summer computer simulation conference. In Summer Computer Simulation Conference ACM, editor, *Moving Towards the Unified Simulation Approach*, San Diego, USA, july 2007.

E. Ramat and P. Preux. Virtual Laboratory Environment (VLE): a software environment oriented agent and object for modeling and simulation of complex systems. In *Simulation Modelling Practice and Theory*, volume 11, pages 45–55, 2003.

C. Talidec, M.J. Rochet, M. Bertignac, and C. Macher. Discards estimates of nephrops and hake in the nephrops trawl fishery of the bay of biscay: methodology and preliminary results for 2003 and 2004. ICES Working Group on the Assessment of Southern Shelf Stocks of Hake, Monk and Megrim, WGHMM, 2005.

B. P. Zeigler. *Theory Of Modeling and Simulation*. Wiley Interscience, 1976.

B. P. Zeigler, D. Kim, and H. Praehofer. *Theory of modeling and simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000.