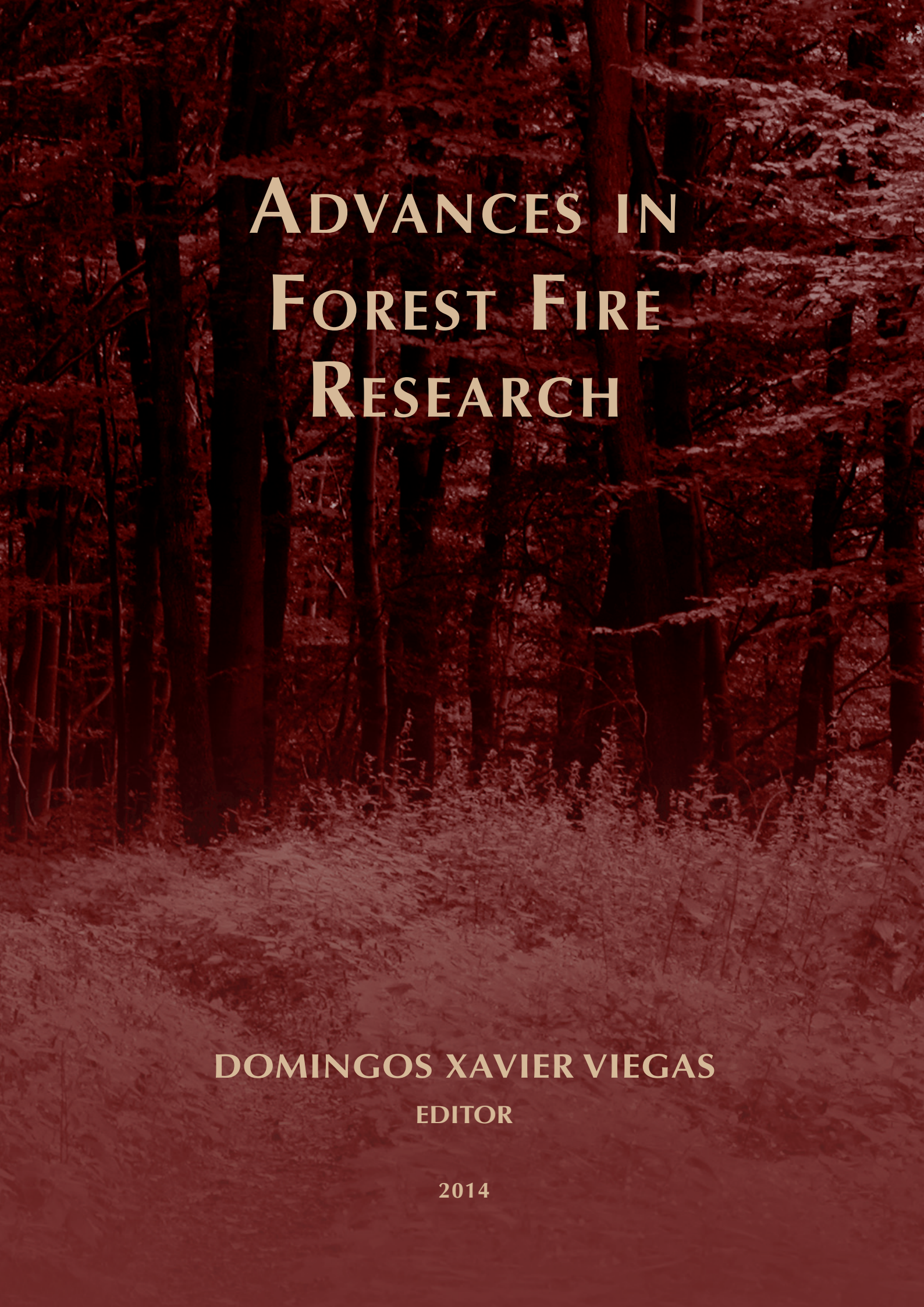

Advances in Forest Fire Research

DOMINGOS XAVIER VIEGAS

EDITOR

2014

ForeFire: open-source code for wildland fire spread models.

Jean-Baptiste Filippi, Frédéric Bosseur, Damien Grandi

SPE – CNRS UMR 6134, Campus Grossetti, BP52, 20250 Corte. filippi@univ-corse.fr

Abstract
Fore fire code has been developed to bridge the gap between research and operational code, it is open source, designed for large scale fire simulation, can be easily extended with any new model formulations and can take typical landscape data as input. The code is composed of a simulation engine that has been numerically tested with numerous bindings for different computer languages to be integrated into other scientific environments ranging from SciPy/Numpy to Fortran parallel coupled numerical weather forecast models. This paper presents the general software architecture and concepts, illustrated by examples and use cases in different context.

Keywords*: software, model, simulation, source code, open source, python, fire front, forecast*

1. Introduction

Fire front simulation tools have either been targeted with great success for research or for operations (Finney and Andrews 94). If most research oriented codes were designed to handle specific configuration with a complex and possibly time consuming set-up, operational tools are designed to handle almost readily available landscape data in order to provide simulation results relevant for real large scales fire. The ease of use of such software, and their relevance for large scale fire simulation has helped the field of forest fire operational research with application among which fire re-analysis, uncertainty estimation, land use, fuel management. Nevertheless, all these software are monolithic, they are not designed to be modular, extended or used in the context of multiple interactions with other scientific software (model coupling, data analysis, post-processing). An important aspect is also that unlike research purpose code operational software's are mostly closed source, available only as a compiled binary with little opportunity to add or change the numerical scheme or the formulation of the rate of spread of fire intensity.
Fore fire code has been developed to bridge the gap between research and operational code, it is open source, designed for large scale fire simulation, can be easily extended with any new model formulations and can take typical landscape data as input. The code is composed of a simulation engine that has been numerically tested (Filippi *et al*, 2009) with numerous bindings for different computer languages to be integrated into other scientific environments ranging from SciPy/Numpy to Fortran parallel numerical weather forecast models.
This paper will present first the general software architecture, illustrated in a second part by examples and use cases in different context. Finally the conclusion will also review the requirements and future code developments.

2. Time handling and numerical integration method

The forefire code is based on discrete events simulation, a complete description can be found in (Filippi et.al. 2009). Basically, each action (fire portion moving, wind change …) is performed by a component (fire portion, atmospheric data model) at a specific time. An action, from a component, is scheduled at this specific time by an event, these events are received and time-sorted in a global timetable. Events can be inserted by the user, and each component, when activated, can re-schedule events. Time is

continuous, meaning that there in no time step, and activation times are real numbers, simulation is run by activating the component with the most imminent event, scheduling its output events and moving to the next imminent event.
Discrete events time handling has several advantages, first it does not forbids the use of discrete time stepping methods (with evenly timed events), it is easily integrated in a scripted environment, can easily handle the introduction of heterogeneous components and most of all, can optimize computational time when components have very different characteristic time (such as a between a very slow moving backfire and a strong head-fire).
ForeFire software architecture is not tied to a specific numerical method to simulate fire front propagation, yet, only a Lagrangian front tracking method of markers is implemented.
In the front tracking method, the fire line is decomposed into a set of connected points, or markers. Each marker is a component that has a specific propagation direction and speed, such as shown in figure 1.
The speed at which the marker is traveling along its propagation vector is given by the rate of spread model with many available (isotropic, wind-aided, Balbi (2009), Rothermel (1972), other can be added with a single C++ file. The direction of propagation vector is usually (several methods available) taken as the bisector of the angle formed by the marker, and the location of the immediate left and right markers. Markers are redistributed along the front if separated by more than a resolution distance Δr and removed if separated by less than $\Delta r/4$. A fire line is defined as a full set of interconnected markers. If two points of different fire lines are separated by less than $\Delta r/4$ the two fronts are merging. The integration of a marker advance is performed in a discrete event fashion, with no global time step but specific activation time for markers. Each marker is advancing by the same distance Δq with an event time activation estimated as the time it will take to the parker to move by this distance given its local propagation.

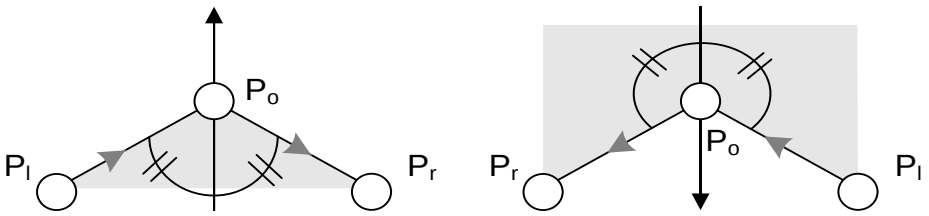

Figure 1. Front tracking and markers. Circles represent markers along the fire line. Arrows show the propagation vector (bisector of the local angle at the marker P0 between the point at left, Pl and point at right, Pr). Grey area represents the burned fuel.

The propagation velocity of the fire front given by the velocity model depends on the flame geometrical properties: tilt angle, curvature and front depth. The two firsts properties are assumed instantaneous in the model and may be observed from the front geometry or computed at a given time. On the contrary the front depth, thickness of the front where the flame is present, depends on front history and fuel properties (for how long and how fuel has been burning). When simulating the effect of a fire this history is also necessary to diagnose the location and intensity of different fire induces phenomenon (mass and heat fluxes for example, requiring user defined fluxed models). In the proposed method a high resolution matrix of arrival time is used to keep track of the fire history top perform instantaneous surface diagnostics. The matrix resolution is a key point and limitation of the method that requires to fix a physical lower limit, making the hypothesis of a minimal propagative front depth (if the front is thinner than this value the fire can't propagate and may go to extinction in non self-sustaining fire conditions). On large domains (tens of square kilometers) this two dimension matrix may contains millions of points at a typical resolution of one meter. Nevertheless, as in large Wildland

fires the active burning area is only a very small fraction of the total domain extension, arrival times is handled as a sparse matrix that may still be handled efficiently.

3. Software architecture

The general software architecture is presented in Figure 2. Basically, the software compiles in a shared library, that has optional bindings to different programming languages and data formats to use as input/outputs. In addition to these bindings, there is a binary interpreter with a ForeFire specific syntax and scripted commands that is compiled for standalone applications, parameterisation and text format data input/output.

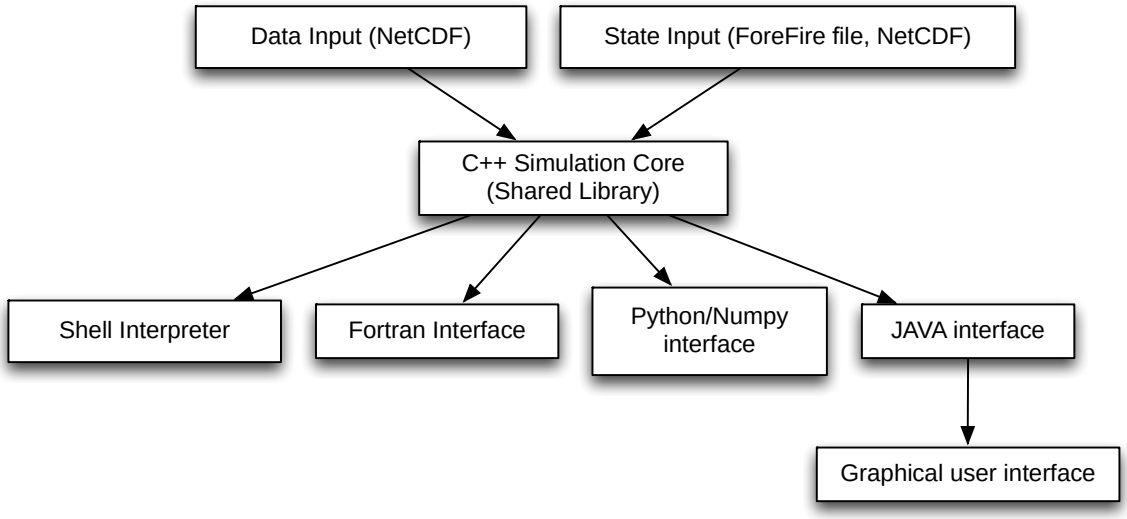

Figure 2. General software architecture and bindings

There are two different kind of input data, with sample format available from the software repository. The first is similar to Farsite's "Landscape" file but in the NetCDF file format, it defines, for a specific data domain a compilation of altitude, wind field, fuel distribution and optional diagnostic (fluxes) model distribution. Fuel state for each fuel classes is defined in a separated file and is highly dependent on the fire velocity model of use. The state input file is composed of the arrival time sparse matrix in NetCDF and ForeFire format for front input.

3.1. Command interaction

Interaction with the simulation is made with so-called commands, these can be either software calls (functions available in Fortran/Python/C++/Java interfaces) or command calls from the interpreter (a binary file launched from the command line). All commands are presented in Figure 3, all interaction with the simulation can be performed with this set of commands, with the exception of accessing the data arrays that cannot be made in text/console mode and can only be stored to files or accessed with programming interfaces.

```
FireDomain[date, location, extension] : creates a fire domain
FireFront[date] : creates a fire front with the following fire nodes
FireNode[location, velocity] : adds a fire node to the given location
step[duration] : advances the simulation for a user-defined amount of time
goTo[date] : advances the simulation till a user-defined time
setParameter[name=value] : sets a given parameter
setParameters[name=value;name=value] : sets a given list of parameters
getParameter[name] : return the parameter from a key
loadData[Netcdf File] : load a landscape file, define a domain
startFire[location, date] : start a triangular fire front
include[command file] : include commands from a file
print[output file(optional)] : prints the state of the simulation
save[NetCDF file] : saves the simulation state in NetCDF format
clear[] : resets the simulation
quit : quit and terminate
```

Figure 3. List of the scripted commands

3.2. Launching a simulation

If most of the commands are self-explanatory, some are notable, in particular the FireDomain, FireFront and FireNode commands that actually defines the fire mesh. A Node (marker) is container in a Front (interface) if it is defines just under it, with correct indentation. If it is only possible to define a single domain, it is possible in that way to define as many fronts with inner fronts as necessary. The propagation direction (contraction or dilatation) of the interface is given by the winding order. The Print command outputs the Domain/front/node structure in the same format, so it is directly readable with an Include command, Figure 4 is an example of the most simple isotropic front propagation.

```
1  setParameters[propagationModel=Iso;Iso.speed=1]
2  FireDomain[sw=(-10.,-10.,0.);ne=(10.,10.,0.);t=0.]
3      FireFront[t=0.]
4          FireNode[loc=(-3,-3,0.);vel=(-0.5,-0.3,0.);t=0.]
5          FireNode[loc=(0.,3,0.);vel=(0.2,1.2,0.);t=0.]
6          FireNode[loc=(3,0.,0.);vel=(0.7,0.1,0.);t=0.]
7  step[dt=10s]
8  print[sim1.ff]
9  clear[]
10 include[sim1.ff]
11 step[dt=5.2s]
12 quit[]
```

Figure 4. Simple simulation example

In this simulation, a domain is created between a southwest and a northwest point, and is added a front with three nodes (lines 3 to 7). The simulation is run for 10 seconds (line 8), then the structure is saved to a file, and reloaded (line 11), restarted for 5.2 more seconds then stopped. The “Iso” model is a simple isotropic model, taken as example for the simplicity of parameterisation here. More complex models are already available such as Balbi (Balbi *et al*, 2009) or Rothermel (1972), and the code design is such as it is very easily expandable with a minimum of coding, as one file is necessary (presented in Figure 5).

3.3. Adding new models

Adding a velocity model is simply done by adding a C++ file that define the “getSpeed” function, a name, and a constructor to specify the required parameters.

```
1  namespace libforefire {
2  /* model name */
3  const string ParametricROS::name = "WindSlope";
4  ParametricROS::ParametricROS(const int & mindex, DataBroker* db)
5          : PropagationModel(mindex, db) {
6      /* Locally interpolated values */
7      effectiveSlope = registerProperty("effectiveSlope");
8      normalWind = registerProperty("normalWind");
9      /* Global parameters */
10     windFactor = params->getDouble("WindAided.windFactor");
11     slopeFactor = params->getDouble("WindAided.slopeFactor");
12 }
13
14 /*  Velocity function */
15 double ParametricROS::getSpeed(double* localValue){
16     double speed = windFactor*localValue[normalWind]
17          + slopeFactor*(1.+localValue[effectiveSlope]);
18     if ( speed > 0. ) return speed;
19     return 0;
20 }
21 }
```

Figure 5. C++ fire velocity model

Once this file is added to the source list the model is available, callable by using the provided name (here “WindSlope”, line 3 of Figure 5). An important hidden object in ForeFire is the “DataBroker”, it is responsible to virtualize data access at the node location (here the “effectiveSlope” and the “normalWind”, both layers being a function applied on the elevation field and wind field given the node location). All data (fuel, wind, slope, flux, velocity) is in fact virtualizes as a Layer, that could be either data or a function, a Layer object has a name (eg. “effectiveSlope”) and can interpolate, compute or determine (index, table, vector, function, model) more layer types can be added and the reader is referred to the code for a complete description. The “getSpeed” function works by calling the DataBroker on each node at each activation, that will automatically compile the registered values interpolated at the node location (and with the node properties such as direction) from the different layers, in an array (“localValue”, line 15). Fluxes and diagnostic models can be added in a very similar fashion in the code.

4. Uses Cases

The originality of the code it its ability to be used in a variety of contexts, so the same developments (on a flux or velocity model, dataset, algorithm..) may be used to be tested in a pure analytic research mode, make its way to operations, and be compared easily with other approaches available from the same code. The open sourced code is essentially the simulation engine and bindings to other environments, there is no graphical user interface because it is intended to a research/developer community. Nevertheless, as a simulation engine, it is embedded in a system that does perform simulation for operational use, marking a clear separation between the simulation code and the user interface code. With this clear separation, any developments made in the research context can make its way to operational use in a very limited time.

Typical contexts of the code use are for research and testing purposes (using here scientific Python), operational use (used here with as a web-service) or even run in parallel embedded in a numerical

atmospheric model (Filippi et.al. 2013). Both scientific and operational use will be briefly presented here.

4.1. Simple simulation in a scientific python environment

The python interface is made with Numpy bindings, yet the interface is functional but minimalistic as it requires inputs from users to gain in usability. A sample script, performing a simulation is presented in Figure 6, only part of the code is shown to concentrate on the simulation aspects, with imports of the numpy and matplotlib libraries hidden.

```
1  ff = forefire.PLibForeFire()
2  ff.setString("propagationModel","Balbi")
3  ff.setString("fuelsTableFile","fuels.ff")
4  # Parameters for the global wind value
5  ff.setDouble("velU",0.)
6  ff.setDouble("velV",12.)
7  # Domain Definition
8  ff.execute("FireDomain[sw=(0.,0.,0.);ne=(300,300,0.);t=0.]")
9  # Adding layers
10 ff.addLayer("data","windU","velU")
11 ff.addLayer("data","windV","velV")
12 # Defining a specific fuel map
13 fuelmap = np.zeros((sizeX,sizeY,1), dtype=np.int32)
14 fuelmap[:,90:110,:] = 1
15 fuelmap[150:,90:110,:] = 3
16 ff.addIndexLayer("table","fuel",0 , 0, 0, sizeX, sizeY, 0, fuelmap)
17 ff.execute("    FireFront[t=0.]")
18 ff.execute("        FireNode[loc=(40,60,0.);vel=(-1.,-1.,0.);t=0.]")
19 ff.execute("        FireNode[loc=(40,65,0.);vel=(-1.,1.,0.);t=0.]")
20 ff.execute("        FireNode[loc=(260,65,0.);vel=(1.,1.,0.);t=0.]")
21 ff.execute("        FireNode[loc=(260,60,0.);vel=(1.,-1.,0.);t=0.]")
22 # Storing 20 fronts
23 for i in range(1,20):
24     ff.execute("goTo[t=%f]"%(i*20))
25     pathes += printToPathe( ff.execute("print[]"))
26 # Plotting the fuel map
27 CS = ax.imshow(ff.getDoubleArray("fuel"))
28 # Plotting the firelines
29 for path in pathes:
30     ax.add_patch(mpatches.PathPatch(path,edgecolor='red'))
31 plt.show()
```

Figure 6. ForeFire simulation in python

In this script a Forefire object is first created (line 1), it contains a simulation context, linked to one domain, parametrisations and data. In line 2 and 3 is defined the model and the comma separated array file that contains the fuel parameters that are relevant to the propagation model used.
The functions “addLayer” adds a parameter layer that will always return the value set in the corresponding parameter. A user defined indexLayer is then added as a numpy array (line 16). This layers contains the indices in the table (defined in the fuel file), so that all parameters linked to this indices may be passed to the velocity model, at the fire location. Finally, the simulation is initiated using the “execute” function, that calls the standards ForeFire commands, and simulation results are stored in an array (line 25).

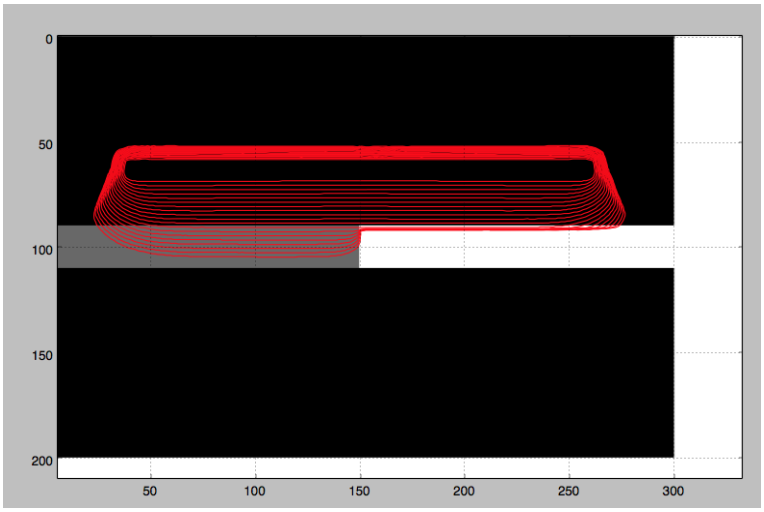

Figure 7. Output from the python script

Figure 7 presents the figure that is outputted from the script. Here MatPlotlib is used to visualize the front and fuel map using the "getLayer" function that return a numpy array.

4.2. Simulation in a operational context

Figure 8 presents a simulation results in a web application that provides a graphical user interface to easily launch simulations (available at http://forefire.univ-corse.fr/sim/dev/). This context is very different, user does not have to take time to prepare data, so everything is already available (a NetCDF file for each small region with all data layers) and the simulation is just run by using the "loadData", then "addFire", and "print" function in a loop on the server and sent back to the web browser.

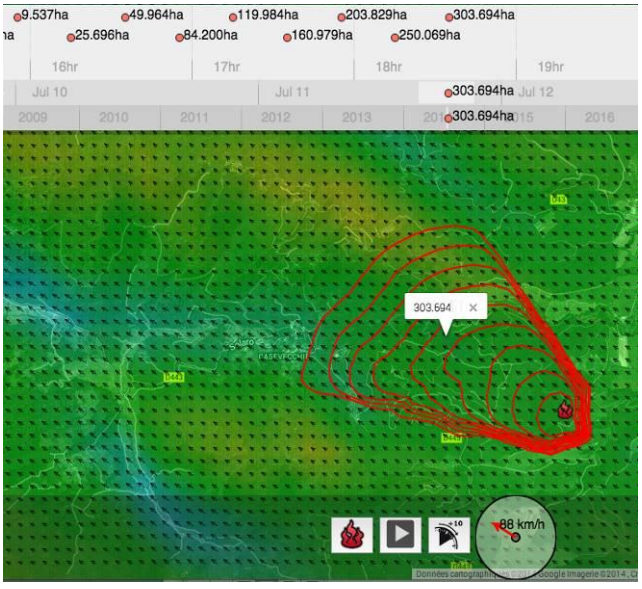

Figure 8. ForeFire in command line, launched on a server trough a web interface

5. Requirements and conclusion

Forefire library, python, java and command line interpreter is available on GitHub at address https://github.com/forefireAPI/firefront. The minimum requirement is to have NetCDF library

installed with legacy C++ interface. The SWIG (swig, 2014) build platform is used to automate the compilation. Target platform is a Unix compatible system with a C++ compiler available.
This paper presented the philosophy of the software as well as showing its usage in a scripted, command line and a coupled environment [Filippi *et al* 2013]. The same C++ compiled shared library is used here in a graphical user interface, a high performance computing environment and a scripted, scientific oriented environment (Python/Numpy).
The philosophy is typical of both scientific and operational software, with sources available, self compiling and expandable code but also having connections with data formats that allows to simulate real fire on existing data in a very limited time. Further work will be focused to enhance the python interface, optimize the code, add more components and interaction methods.

6. Acknowledgement

This research was developed within the projects ANR-09-COSI-006-01 IDEA

7. References

Filippi et.al. 2009. Jean Baptiste Filippi; F. Morandini; Jacques-Henri Balbi; David Hill Discrete event front tracking simulator of a physical fire spread model, hal-00438619, Simulation, 2009, 86 (10), pp. 629-644, DOI : 10.1177/0037549709343117

Balbi et. al. 2009 Jacques Henri Balbi; Frédéric Morandini; Xavier Silvani; Jean Baptiste Filippi; Frédéric Rinieri. A Physical Model for Wildland Fires, hal-00593608, Combustion and Flame, 2009, 156 (12), pp. 2217-2230 . DOI : 10.1016/j.combustflame.2009.07.010

Finney and Andrews, 1994 Finney, M., Andrews, P., 1994. The farsite fire area simulator: Fire management applications and lessons of summer 1994. In: in Proceedings of Interior West Fire Council Meeting and Program. pp. 209–216, coeur Alene, USA.

Filippi et.al. 2013 Assessment of ForeFire/Meso-NH for wildland fire/atmosphere coupled simulation of the FireFlux experiment, J.-B. Filippi, X. Pialat, C.-B. Clements, Proceedings of the Combustion Institute, Volume 34, Issue 2, Pages 2633–2640, 2013.

Swig, 2014. http://www.swig.org