FACETS

FP6-2004-IST-FETPI 15879
Fast Analog Computing with Emergent Transient States

Deliverable D8-3

Second public release of the PyNN software and first public release of FacetsML validator and editor

Report Version: 1.0

Report Preparation: Andrew Davison (CNRS-UNIC); Theodore Papadopoulo (INRIA)

Classification: Public

Contract Start Date: 01/09/2005 Duration: 4 Years

Project Co-ordinator: Karlheinz Meier (Heidelberg)

Partners: U Bordeaux, CNRS (Gif-sur-Yvette, Marseille), U Debrecen, TU Dresden, U Freiburg, TU Graz, U Heidelberg, EPFL Lausanne, Funetics S.a.r.l., U London, U Plymouth, INRIA, KTH Stockholm

Project funded by the European Community

under the "Information Society Technologies" Programme

DELIVERABLES TABLE

Project Number: FP6-2004-IST-FETPI 15879

Project Acronym: FACETS

Title: Fast Analog Computing with Emergent Transient States

Del. No.	Revision	Title	Type[1]	Classification[2]	Due Date	Issue Date
D8-3	1.0	Second public release of the PyNN software and first public release of FacetsML validator and editor	O[a]	PU	29/02/2008	30/06/2008

[1] *R: Report; D: Demonstrator; P: Prototype; O: Other – Specify in footnote*

[2] *PU: Public*
PP: Circulation within programme participants, including the Commission Services
RE: Restricted circulation list (specify in footnote), including the Commission Services
CO: Confidential, only for members of the consortium, including the Commission Services

[a] Software

DELIVERABLE SUMMARY SHEET

Project Number: FP6-2004-IST-FETPI 15879

Project Acronym: FACETS

Title: Fast Analog Computing with Emergent Transient States

Deliverable N°: D8-3

Due date: 29/02/2008

Delivery Date: 30/06/2008

Short Description:

The second public release of PyNN (version 0.4) was on 30/05/2008. The software is available to download from http://neuralensemble.org/PyNN.

The first public release of the FacetsML validator and editor was on 30/06/2008. The software is available to download from http://neuralensemble.org/trac/FacetsML

This report summarizes the changes to PyNN since the previous release, and explains how to use FacetsML. The motivation for the development of PyNN and FacetsML was given in Deliverable D8-1. Full documentation is available at the websites given above.

Partners owning: CNRS-UNIC, INRIA

Partners contributed: CNRS-UNIC, CNRS-INCM, ALUF, EPFL, UHEI, TUG, INRIA

Made available to: Public

1 Introduction

PyNN is a Python package for simulator-independent specification of neuronal network models. FacetsML is the declarative equivalent to PyNN. It is an XML dialect, which can be easily translated to PyNN with an XSLT transformation. Automatic translation between FacetsML and NeuroML (http://www.neuroml.org) is in development.

The first public release of PyNN (version 0.3) was on 24/05/2007, corresponding to FACETS Milestone M8-2. Deliverable D8-1 gives more information about PyNN, including the motivation for the development of PyNN and FacetsML, and the Users' Guide for version 0.3.

The second public release of PyNN (version 0.4) was on 30/05/2008. The software is available from

http://neuralensemble.org/PyNN

The first public release of the FacetsML validator and editor was on 30/06/2008. The software is available from

http://neuralensemble.org/trac/FacetsML

This report summarizes the changes to PyNN since the previous release, and explains how to use FacetsML. The Users' Guide for this release of PyNN is available online at the above URL.

2 Revision history of PyNN

2.1 Release 0.4

Second public release.

- The `NumpyRNG` class now has some extra options for running parallel simulations. If `parallel_safe` is `True` (and if the MPI rank and number of processes are correctly supplied), then the sequence of random numbers seen by the simulation script should be independent of the number of MPI processes (although not yet the number of threads).
- Added `num_processes()` and `rank()` functions to provide a uniform access. to MPI information.
- The headers of output files now contain the first and last ids in the Population (not fully implemented yet for recording with the low-level API)
- Global variables such as the time step and minimum delay have been replaced with functions `get_time_step()`, `get_min_delay()` and `get_current_time()`. This ensures that values are always up-to-date.
- Removed `cellclass` arg from `set()` function. All cells should now know their own cellclass.
- Added `get()` method to `Population` class.
- Default value for the `duration` parameter in `SpikeSourcePoisson` changed from 1e12 ms to 1e6 ms.
- Reimplemented `Population.set()`, `tset()`, `rset()` in a more consistent way which avoids exposing the translation machinery and fixes various bugs with computed parameters. The new implementation is likely to be slower, but several optimisations are possible.
- Added `simple_parameters()`, `scaled_parameters()` and `computed_parameters()` methods to the `StandardModelType` class. Their intended use is in making `set()` methods/functions more efficient for non-computed parameters when setting on many nodes.
- Multiple calls to `Population.record()` no longer record the same cell twice.
- Changed `common.ID` to `common.IDMixin`, which allows the type used for the id to vary (`int` for `neuron` and `nest1/2`, `long` for pcsim).
- In `common.StandardModelType`, changed most of the methods to be classmethods, since they do not act on instance data.
- Added a `ModelNotAvailable` class to allow more informative error messages when people try to use a model with a simulator that doesn't support it.
- hoc and mod files are now correctly packaged, installed and compiled with `distutils`.

- Added a check that argument names to `setup()` are not mis-spelled. This is possible because of `extra_params`.
- It is now possible to instantiate Timer objects, i.e. to have multiple, independent Timers
- Some re-naming of function/method arguments to conform more closely to Python style guidelines, e.g. `methodParameters` to `method_parameters` and `paramDict` to `param_dict`.
- Added `getWeights()` and `getDelays()` methods to `Projection` class.
- Added a `RoundingWarning` exception, to warn the user when rounding is occurring.
- Can now change the `spike_times` attribute of a `SpikeSourceArray` during a simulation without reinitialising. This reduces memory for long simulations, since it is not necessary to load all the spike times into memory at once.
- The `neuron` module now requires NEURON v6.1 or later.
- For developers, changes to the layout of the code: (1) Simulator modules have been moved to a `src` subdirectory - this is to make distribution/installation of PyNN easier. (2) Several of the modules have been split into multiple files, in their own subdirectories, e.g.: `nest2.py` → `nest2/__init__.py`, `nest2/cells.py` and `nest2/connectors.py`. The reason for this is that the individual files were getting very long and difficult to navigate.
- Added `index()` method to `Population` class
- Added `getSpikes()` method to `Population` class - returns spike times/ids as a numpy array.
- Added support for the Stage 1 FACETS hardware.
- Changed the default weight to zero (was 1.0 nA)
- New STDP API, with implementations for `neuron` and `nest2`, based on discussions at the CodeSprint.
- Distance calculations can now use periodic boundary conditions.
- Parameter translation system now fully supports reverse translation (including units). The syntax for specifying translations is now simpler, which makes it easier to define new standard cell models.
- All simulator modules now have a `list_standard_models()` function, which returns a list of all the models that are available for that simulator.
- Procedures for connecting `Populations` can now be defined as classes (subclasses of an abstract `Connector` class) rather than given as a string. This should make it easier for users to add their own connection methods. Weights and delays can also be specified in the `Connector` constructor, removing the need to call `setWeights()` and `setDelays()` after the building of the connections. We keep the string specification for backwards compatibility, but this is deprecated and will be removed in a future API version.
- Added new standard models: `EIF_cond_alpha_isfa_ista`, `IF_cond_exp_gsfa_grr`, `HH_cond_exp`.
- Version 2 of the NEST simulator is now supported, with the `nest2` module. The `nest` module is now called `nest1`.
- Changed the order of arguments in `random.RandomDistribution.__init__()` to put `rng` last, since this is the argument for which the default is most often used (moving it lets positional arguments be used for `distribution` and `parameters` when `rng` is not specified).
- Changes to `ID` class:
  - `_cellclass` attribute renamed to `cellclass` and is now a property.
  - Ditto for `_position` → `position`
  - Methods `setPosition()`, `getPosition()`, `setCellClass()` removed (just use the `position` or `cellclass` properties).
  - `set(param,val=None)` changed to `setParameters(**parameters)`.
  - Added `getParameters()`

  - `__setattr__()` and `__getattr__()` overridden, so that cell parameters can be read/changed using dot notation, e.g. `id.tau_m = 20.0`

  Note that one of the reasons for using properties is that it allows attributes to be created only when needed, hopefully saving on memory.

- Added `positions` property to `Population` class, which allows the positions of all cells in a population to be set/read at once as a numpy array.
- All positions are now in 3D space, irrespective of the shape of the `Population`.
- Threads can now be used in `nest` and `pcsim`, via the `extra_param` option of the `setup()` function.
- Removed `oldneuron` module.
- Added `__iter__()` (iterates over ids) and `addresses()` (iterates over addresses) to the `Population` class.

2.2 Release 0.3

First public release.

- `pcsim` is now fully supported, although there are still one or two parts of the API that are not implemented.
- The behaviour of the `run()` function in the `neuron` module has been changed to match the `nest` and `pcsim` modules, i.e., calling `run(simtime)` several times in succession will advance the simulation by `simtime` ms each time, whereas before, `neuron` would reset time to zero each time.
- PyTables is now optional with `pcsim`
- Change to `neuron` and `oldneuron` to match more closely the behaviour of `nest` and `pcsim` when the `synapse_type` argument is not given in `connect()`. Before, `neuron` would by default choose an excitatory synapse. Now, it chooses an inhibitory synapse if the weight is negative.
- `runtests.py` now runs tests for `pcsim` as well as `nest`, `neuron` and `oldneuron`.
- Minor changes to arg names and doc-strings, to improve API-consistency between modules.
- Added users' guide (in `doc` directory).
- Renamed `neuron` module to `oldneuron` and `neuron2` to `neuron`.
- PyNN can now be installed using `distutils`, although this doesn't install or compile hoc/mod files.
- Added a `compatible_output` argument to the `printX()` functions/methods, to allow choosing a simulator's native format (faster) or a format that is consistent across simulators.
- Temporary files used for saving spikes and membrane potential are now created using the `tempfile` module, which means it should be safe to run multiple PyNN simulations at the same time (before, they would all overwrite the same file).
- pygsl is no longer an absolute requirement but can be used if available
- Changed the behaviour of `Population` indexing in the `nest` module to be more consistent with the `neuron2` module, in two ways. (i) negative addresses now raise an Exception. (ii) Previously, an integer index `n` signified the `(n+1)`th neuron in the population, e.g., `p[99]` would be the same as `p[10,10]` for a 10x10 population. Now, `p[99]` is the same as `p[99,]` and is only valid for a 1D population.
- Addition of `ID` class (inherits from `int`), allowing syntax like `p[3,4].set('tau_m',20.0)` where `p` is a Population object.

2.3 Release 0.2

Only released within FACETS.

- `Population.tset()` now accepts arrays of arrays (e.g. conceptually a 2D array of 1D arrays, actually a 3D array) as well as arrays of lists.
- setup() now returns the node id. This can be used in a parallel framework to identify the master node.
- Unified output format for spikes and membrane potential for `nest` and `neuron` modules.
- Added first experimental version of `pcsim` module
- `neuron2` module now supports distributed simulations using NEURON compiled with both MPI and Python support.
- `Population[xx]` syntax for getting individual cell ids improved. You can now write `p[2,3]` instead of `p[(2,3)]`.
- `v_init` added as a parameter to the `IF_curr_alpha`, etc, models.
- Trying to access a hoc variable that doesn't exist raises a Python exception, `HocError`.
- If synaptic delay is not specified, delays are now set to `min_delay`, not zero.
- Random number API allows keeping control of the random numbers used in simulations, by passing an RNG object as an argument to functions that use RNGs. `random` module has wrappers for NumPy RNGs and GSL RNGs, as well as a stub class to indicate the simulator's native RNG should be used (i.e., the `Random` class in hoc).
- Translation of model and parameter names from standardised names to simulator-specific names now uses one class per neuron model, rather than a single class with one method per model. For users, the only difference is that you have to use, e.g., `create(IF_curr_alpha)` instead of `create('IF_curr_alpha')` i.e., pass the class instead of a string. For developers, it should now be easier to add new standard models.
- Added `neuron2` module, a reimplemtation of the PyNN API for NEURON, that uses more Python and less hoc.

2.4 Release 0.1

Version 0.1 of the API was never really released. At this point the project used the FACETSCOMMON svn repository.

First svn import of early stage of PyNN was on 9th May 2006.

3 The FacetsML Editor and Validator

3.1 Introduction and history

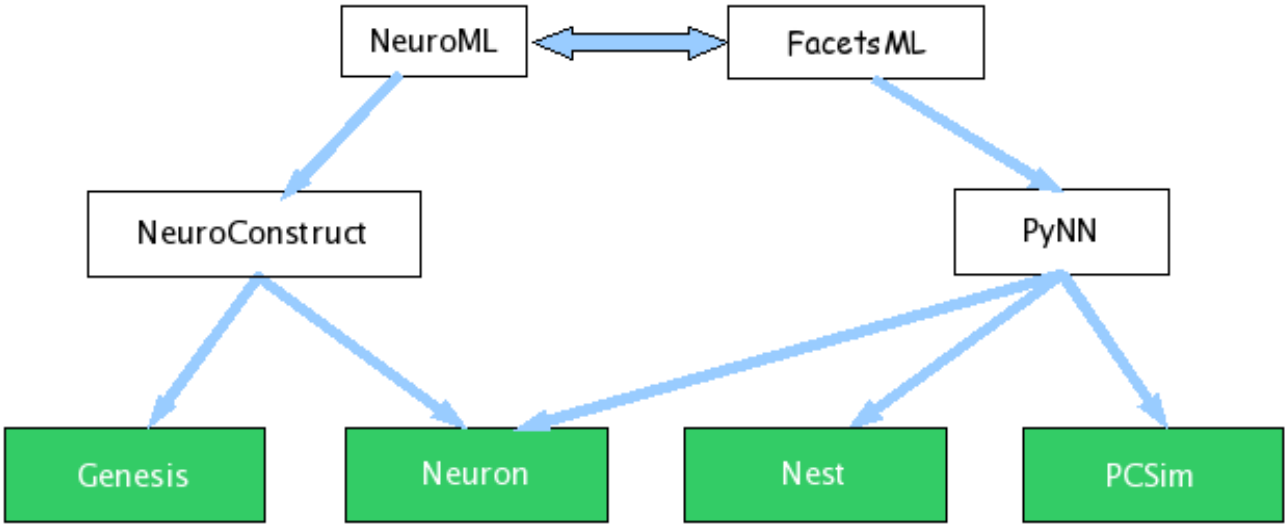
The initial goals of FacetsML were:

- To be an extension of the NeuroML standards that provided missing building blocks such as network models and integrate-and-fire (IF) neuron models that were needed by the FACETS community.
- To provide a simple declarative procedure for designing and describing neural network models at the network level.

However, since then the NeuroML group has release level 3 of the standards, which includes support for networks and some support for IF models. The NeuroML developers incorporated many of the proposed FACETS extensions (see FACETS report D23) into more recent versions of their standards. At the same time within FACETS, a lot of work has gone into PyNN, thus the goal of FacetsML has shifted to that of functioning as a declarative equivalent to PyNN, and as a basis for conversion between PyNN and NeuroML descriptions of network models. As NeuroML, FacetsML is a XML dialect, which can be easily translated to PyNN or NeuroML (with some restrictions) with XSLT transformations.

3.2 Main characteristics

FacetsML is an attempt to create a pathway between NeuroML and PyNN that keeps the ease-of-use of PyNN for describing neural networks at the network level and using IF models for neurons while allowing some leveraging of the tools build around NeuroML such as NeuroConstruct (a graphical interface for building neural network descriptions) or Genesis (a neural network simulator which is not yet directly supported by PyNN). The interactions between NeuroML, PyNN and FacetsML are depicted schematically in the following diagram.



FacetsML may be translated to PyNN with an XSLT transformation (see below). Translation to NeuroML using XSLT transformations is also possible: populations and projections are easily handled due to their common roots, translation of cells is currently limited to the PyNN `IF_cond_exp` model. Translation from a subset of NeuroML to FacetsML is also possible (albeit with the same limitations).

3.3 Tools and Downloads

- The definition of the FacetsML dialect is provided as a set of RelaxNG schemas. Those are available at http://neuralensemble.org/trac/FacetsML/browser/trunk/schemas/.
- The various XSLT transformations between FacetsML and PyNN and NeuroML are available at http://neuralensemble.org/trac/FacetsML/browser/trunk/transformations.
- The RelaxNG schemas are also the base of a web form that can be used to create FacetsML files. This form is the base of an integrated web service available at https://facets.inria.fr/simulator.html, that takes FacetsML descriptions (previous stored descriptions or new ones entered in the system using the html form for FacetsML descriptions), translates it to PyNN and finally launches the simulation of the depicted neural network model.

For more information about FacetsML, including the current schemas, see http://neuralensemble.org/trac/FacetsML. All the above are public downloads.

3.4 Limitations and Caveats

As said above, only part of FacetsML can be translated to or from NeuroML. No attempt has been made yet to have a PyNN to FacetsML translator.

3.5 FacetsML web services

As of 30/06/2008, the first public release (0.5) of the web service has been made available, at http://facets.inria.fr/Public_simulator.html (the public counterpart of https://facets.inria.fr/simulator.html). Figure 1 gives a global view of this web service. The service uses the tools depicted in section 3.3 to create an application that allows for:

- Editing/creating a FacetsML specification using a web form such as the one shown in figure 2.

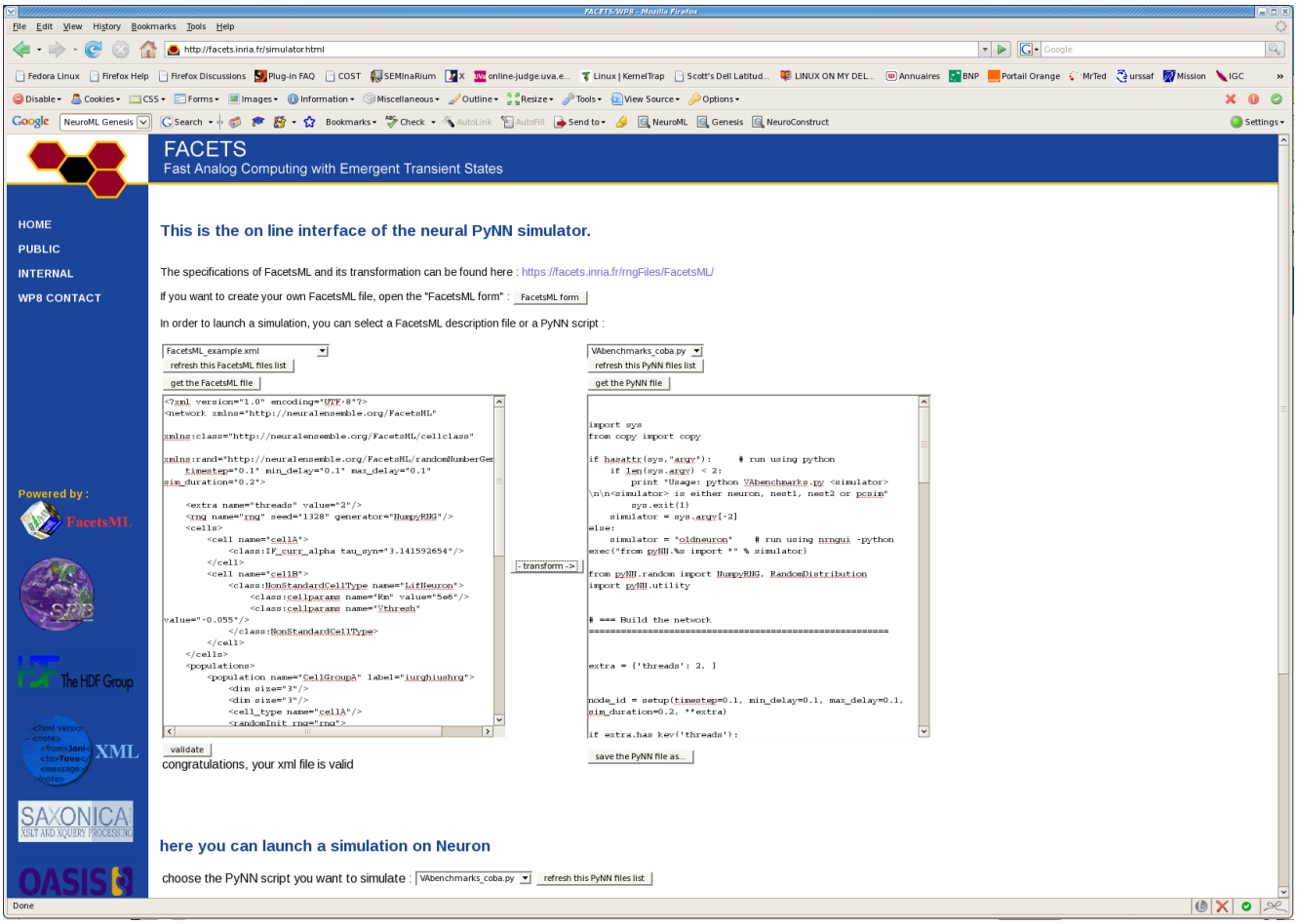


Figure 1. View of the web service interface.

- Validation of any FacetsML file: the RelaxNG schemas are used to verify that the proper FacetsML syntax has been used for a given input file.
- Translation to PyNN syntax: the XSLT based transformations are used to convert a FacetsML description file to its PyNN equivalent. The result can be stored on the FACETS server. Alternatively, the user can also type PyNN code directly into the form.
- Launching a PyNN simulation with the NEURON backend and retrieving the results.

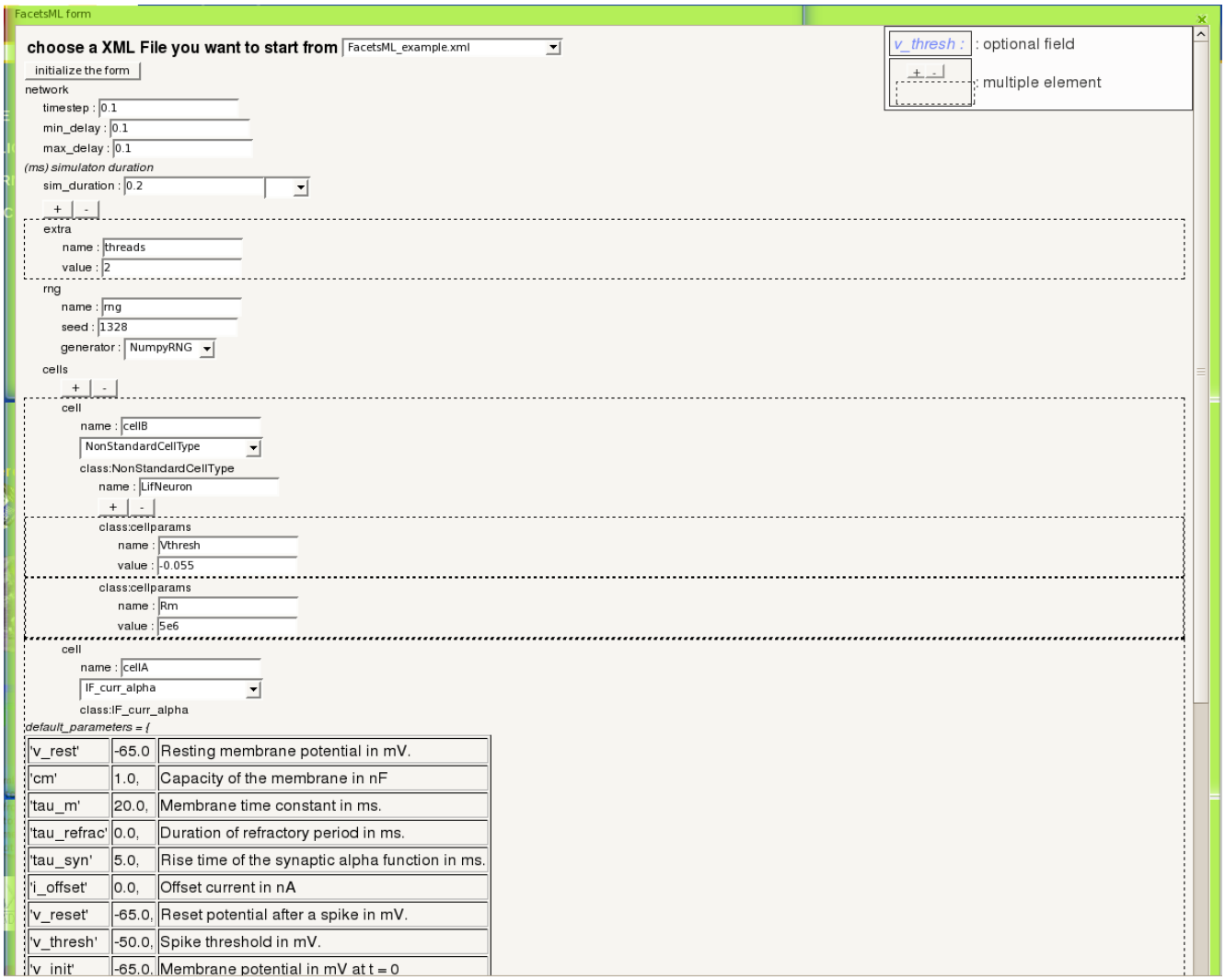


Figure 2. View of the web form used for entering FacetsML descriptions.