

Redi Quickstart

Lorenzo Dematté

November 27, 2008

1 About *Redi*

Redi is a prototype reaction-diffusion simulator, built to test new diffusion models and algorithms. *Redi* simulates biochemical systems at the *mesoscopic* scale of interaction, employing the space discretization and the Gillespie SSA adaptations as discussed by Bernstein [1]. As such, space in *Redi* is discretized into sub-volumes (called also cells or voxels) inside which chemicals and species react as in the Gillespie SSA. Diffusion happens between neighbor cells; entities (species instances) localization is approximated as the containing cell position.

Besides that, *Redi* is also a first attempt to write and code systems biology tools in a modular and re-usable way, with abstraction for species, simulation algorithms and data handlers. As it is, *Redi* is not meant to be super-fast - actually, it employs a variant of Elf's Next Subvolume Method [2] but it has not been optimized in any way.

2 License

As a pre-requisite for using *Redi*, a general user has to read and accept the terms of the license (CoSBI-SSLA) available on the CoSBI web site. By installing, copying, or otherwise using *Redi*, he/she agrees to be bound by the terms of this CoSBI-SSLA.

3 System Requirements

Redi is a .NET application. Version 2.0 of the CLR is required, and version 3.5 SP1 of the .NET Framework is recommended (required for the visualization plug-ins)

4 Usage

Redi is a command line tool; to invoke it, open a command prompt and type

```
redi.exe [params]
```

A list of parameters can be obtained by typing

```
redi.exe /help
```

Example:

```
redi.exe /steps:200000 /deltaSteps:1000
  /fileName:lotka1.txt /outputPrefix:lotka1_1\lotka1
  /x:32 /y:32 /z:4 /cellLength:700
  /writer:GlobalDataWriter /writer:AVFDataWriter
    /writer:Cosbi.DiffuseSim.ViewVolumeDataWriter,Simple3DView
  /writerSpecies:predator /writerSpecies:prey
```

This command launches a simulation of 200000 steps, recording info every 1000 steps using three writers: `GlobalDataWriter`, which records overall system concentrations in textual form, `AVFDataWriter`, which records concentrations of each species in each voxel in Virvo data format. The third writer is a custom data writer, provided in a plug-in included with the download package. It simply captures the output and writes it to the screen, using a 3D dithering view to show species concentrations in space. Notice that for this writer, as for any other writer provided in a plug-in, it is necessary to provide a fully qualified name in the form *Namespace.ClassName,AssemblyName* (Note: the assembly name is usually the name of the .dll file in which the plug-in is stored). In Table 1 it is possible to see a list of writers provided by CoSBI.

It is possible to specify which species will be handled by the writers with the `/writerSpecies` switch.

The `/fileName` switch, used to pass the input model file to the simulator, is always required.

4.1 Programming Redi: the input language

Models for the reaction-diffusion system are given in a simple input language. Named entities that appear in the system are declared in the first section of the input file. In addition, it is possible to specify a base rate for entities that are free to diffuse:

```
var Y : rate 10e-6;
var Yp : rate 10e-6;
var Z2 : weight 24.0;
var M;
var MYp;
```

In this example, Y and Yp are free to diffuse. Their basal diffusion rate is fixed, and it is specified after the *rate* keyword. Z2 is free to diffuse as well, but its basal diffusion rate is not fixed. Instead, it is computed using the algorithm in [3]. The algorithm needs to know the weight of the protein to compute its chemical activity; the weight is expressed in kDa after the *weight* keyword. M and MYp are not free to move, so they have no rate or weight associated to them.

Reactions can be specified in a very intuitive format:

Writer name	Location	Description
StandardDataWriter	Core	This data writer outputs a different ascii file with concentration at each grid cell for every entity in the system.
GlobalDataWriter	Core	This writer outputs a single file with global statistics (entity concentration) across the whole volume.
AVFDataWriter	Core	Writes concentration at each grid cell for every cell as a Virvo ascii file with multiple channels.
CoviseScalarSetWriter	Core	This writer outputs a covise ¹ [4] file for every entity. Each file is a covise set, with where each element of the set is a separate timestep
CoviseVectorSetWriter	Core	This writer outputs a covise file for every entity. Each file is a covise set, with where each element of the set contains the velocity and direction of diffusion at each separate timestep
CoviseVectorWriter	Core	Same as the previous one, but it outputs a different file for each timestamp instead of using a set file.
ViewVolumeDataWriter	Simple3DView plug-in	Displays on the screen the variation of concentration in space using a 3D dithering view.
ViewPlotDataWriter	Simple3DView plug-in	Displays on the screen the variation of global concentrations using a line-plot (available soon)
ViewVolumeDataWriter	VolumeRender plug-in	Displays on the screen the variation of concentration in space using 3D Volume Rendering, using a texture-based approach (available soon)

Table 1: Writers available in the *Redi* package

```
Yp + Z2 -> Y + Z2 [1.90E-03];
M + Yp <- -> MYp [5.93E-03, 20];
```

In the first line, a simple bimolecular reaction is specified. Notice that the rate is given in scientific notation between square brackets. The second reaction is a shorthand to specify reversible reactions. In this case, it is necessary to specify a pair of rates between the square brackets.

After the reaction list, the last part of the file deals with cardinality and location of each species. For each species, the user have to provide a list of locations and of the number of entities of that species at that specific location.

For example:

```
Y [0, 0, 0, 500];
Y [0, 1, 0, 500];
Y [1, 0, 0, 500];
Y [1, 1, 0, 500];
```

After the species name, the user specifies between square brackets the position of a voxel on the 3D grid (in x, y, and z coordinates) and an integer number. Notice that there is a line (or, better, a tuple) for every different location onto the grid in which we would like to insert some number of a given entity. This input format may be verbose and cumbersome to enter, but allows for great control and precision. We are currently studying better ways of inputting both data (location and cardinality of entities) and geometry (that is, for now, fixed and with a regular polyhedric shape).

4.2 Programming Redi: writing a writer plug-in

The modular architecture of *Redi* allows the end-user to write its own output module in any .NET CLS compliant language (like VB, C#, F#, C++...). In fact the *ViewVolumeDataWriter* mentioned in the Usage section is implemented as a plug-in.

To implement a writer plugin, just add a reference to the DiffuseEngine assembly. This assembly contains the base class definition for output writers, namely *DataWriter*.

```
namespace Cosbi.DiffuseSim
{
    public abstract class DataWriter
    {
        protected int printStep = 0;

        public virtual void Init(string outputPrefix, int steps, List<int> entIdList) { }

        // write the file header
        public abstract void WriteHeader(IGlobalCellState cellSystem);

        // write a "frame" (the current cells configuration, total entity count, etc)
        // to the output (file(s), window, socket...)
        public abstract void WriteFrame(double timestamp, IGlobalCellState cellSystem);

        public abstract void Close();

        public virtual bool SetParams(List<string> list) { return true; }
    }
}
```

Deriving a new class from this base class is sufficient to create a new output module. The module is then called by the simulator, which provides informations about some relevant parameters (through the *Init* method, which is called to pass in the output file name common prefix, the expected number of steps and the list of entities to record), and a set of optional parameters passed on the command line (cfr. the */writerParams* switch, which tokens are passed to the *SetParams* method). During simulation, the *WriteHeader* is called once to let the writer initialize itself (i.e. by setting up an output window, or by

opening and writing some data to a file); the *WriteFrame* method is then called at every step to let the writer get the informations it finds relevant from the system (passed as a *IGlobalCellState* parameter) and output it. Finally, the *Close* method is called at the end of the simulation to let the writer perform some clean-up and finalization (close files, windows, etc.).

5 Documentation

A brief description of the simulator, the detailed method and algorithm to perform diffusion rate computation and a biologically relevant application of the method and simulator are described in [3].

References

- [1] D. Bernstein. Exact stochastic simulation of coupled chemical reactions. *PHYSICAL REVIEW E*, 71, April 2005.
- [2] J. Elf and M. Ehrenberg. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *Syst. Biol.*, 1(2), December 2004.
- [3] Paola Lecca, Lorenzo Dematteé, and Corrado Priami. Modeling and simulating reaction-diffusion systems with state-dependent diffusion coefficients. In *Proceedings of World Academy of Science, Engineering and Technology Int. Conference on Bioinformatics and Biomedicine*, volume 35, 2008.
- [4] J.P. Schulze, U. Wssner, S.P. Walz, and U. Lang. Volume rendering in a virtual environment. In Springer Verlag, editor, *Proceedings of 5th IPTW and Eurographics Virtual Environments*, pages 187–198, 2001.