BIOINFORMATICS APPLICATIONS NOTE Vol. 21 no. 6 2005, pages 823–824
doi:10.1093/bioinformatics/bti110

Systems biology

The SBW–MATLAB interface

Cameron Wellock*, Vijay Chickarmane and Herbert M. Sauro

Keck Graduate Institute, Claremont, CA 91711, USA

Received on May 12, 2004; revised and accepted on July 6, 2004
Advance Access publication November 5, 2004

ABSTRACT

Summary: The SBW–MATLAB Interface allows MATLAB users to take advantage of the wide variety of tools available through SBW, the Systems Biology Workbench (Sauro *et al*. (2003) *OMICS*, **7**, 355–372). It also enables MATLAB users to themselves create SBW-enabled tools which can be freely distributed.

Availability: The software is available for download from the author's web page at http://www.public.kgi.edu/~cwellock/. Source code is available as well; the software is released under an MIT (open-source) license.

Contact: cwellock@kgi.edu

INTRODUCING SBW

The Systems Biology Workbench (SBW) is a messaging system that programs can use to exchange data and to accomplish tasks. SBW-enabled programs ('modules') provide 'services' to other SBW client applications; an SBML (Hucka *et al*., 2003) parser program for example might provide services to extract parts of SBML documents. Another program such as a simulator could use these services to load a model, without having to implement its own parser.

SBW is designed to be simple to use, portable and fast. SBW is supported by a number of programming languages and tools, including Java, C/C++, Perl, Python, Delphi and now MATLAB. It works on Windows, Linux and FreeBSD.

A number of SBW-enabled modules have been written, which provide services such as interactive model editing, simulation, optimization and SBML reading and writing. Through SBW, your applications can make use of these abilities quickly and easily.

CONNECTING MATLAB TO SBW

One of the most popular tools for scientific computing today is MATLAB. While C, FORTRAN and Perl have their advantages, few tools can turn high-level mathematical thought into high-speed computation as well as MATLAB. MATLAB has a few weaknesses, however: chief among these is the difficulty in integrating MATLAB programs with other software. This is particularly chafing in a field like systems biology, where the ability to write interchangeable 'software components'—simulators, optimizers and the like—is sorely needed.

FEATURES

The SBW–MATLAB interface provides two key capabilities: firstly, the ability to access existing SBW modules from inside of MATLAB,

```
% connect to SBW
sbwconnect;

% get handles for module, service, and method
j = sbwgetmoduleinstance('Jarnac');
sim = sbwgetservice(j, 'sim');
loadSBML = sbwgetmethod(j, sim, 'void loadSBML(string)');
getJac = sbwgetmethod(j, sim, 'double[][] getJacobian()');

% load a model into Jarnac and get the Jacobian
model = loadfile('C:\models\lorenz.xml');
sbwcall(loadSBML, model);
jac = sbwcall(getJac);

% calculate the eigenvalues for jac
eig(jac)

ans =

  -10.9552
    6.9545
   -0.9994
```

Fig. 1. Sample code illustrating MATLAB accessing an SBW module.

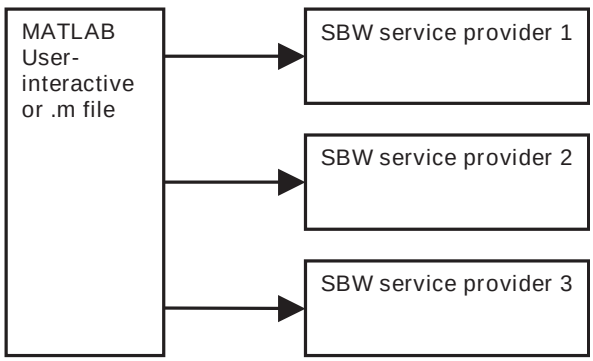

Fig. 2. Schematic showing MATLAB accessing SBW modules.

and secondly, the ability to create new SBW modules entirely inside of MATLAB.

Access to SBW modules is provided through a set of MATLAB MEX-files which behave as any other MATLAB function. These functions can be used to initialize the SBW subsystem, connect to specific modules and send and receive messages. A simple example is the 'Jarnac' module, which can calculate Jacobians for the steady

*To whom correspondence should be addressed.

```
function rval = add2(a, b)

% A matlab function to add two integers. We'll create a SBW module from it called "add2".

%SBW module add2 help="functions to add 2 numbers"
%SBW service add2 help="the only service in the module; coincidendally same name"
%SBW method add service=add2 signature="int add2(int, int)"

rval = a + b;
```

Fig. 3. Sample code illustrating SBW directives in a MATLAB script.

state of a given model, amongst other things. The module has a service named 'sim' and two methods we need: 'loadSBML' and 'getJacobian'. The MATLAB code in Figure 1 illustrates how we might access this module to get the Jacobian and calculate the eigenvalues of a model.

These SBW-related commands can be used from the MATLAB interactive interface or from a .m script file; they can even be used inside of compiled MATLAB functions. The SBW–MATLAB bridge transparently handles the technical details of the calls, converting external data types into MATLAB matrices, and providing robust exception-propagation support (Fig. 2).

The other way the SBW–MATLAB interface can be used is to create new SBW modules inside of MATLAB. The MATLAB compiler is required in order to use this functionality, but SBW modules built in this manner can be distributed freely—end users will not need MATLAB to use the modules.

Before a MATLAB script (or collection of scripts—it is possible to put many MATLAB functions in a single SBW module) can be built into an SBW module, some additional information is needed: the name of the module, the service names and the method signatures—because SBW methods are strictly and explicitly typed. This information is added through extra comments in your MATLAB file, such as that given in Figure 3.

To compile the module, one should invoke the new 'sbwbuild' command. For the example in Fig. 3, we would use (Fig. 4):

```
sbwbuild add2
```

Under Windows this would create a DLL named add2.dll. The sbwbuild command uses the extra information provided in the '%SBW' comments to construct the module.

Running a built module requires the use of the 'sbwmatch' utility, a simple 'stub' executable which loads the DLL and connects it to SBW. For example, from the Windows command line,

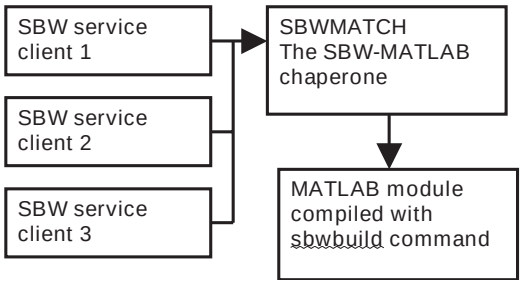

Fig. 4. Schematic of the operation of an SBW module built in MATLAB.

we would use:

```
sbwmatch add2.dll –sbwmodule
```

to start the add2 module as a running server in SBW. Similarly, to register the new add2 module with SBW, we would use:

```
sbwmatch add2.dll –sbwregister
```

ACKNOWLEDGEMENTS

We acknowledge the generous support of the DARPA/IPTO BioCOMP program, whose assistance made this work possible, through contract number MIPR 03-M296–01.

REFERENCES

Hucka,M. *et al*. (2003) 'The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models.' *Bioinformatics*, **19**, 524–531.