The ERATO *Systems Biology Workbench* Project: A Simplified Framework for Application Intercommunication

Michael Hucka, Andrew Finney, Herbert Sauro, Hamid Bolouri

ERATO Kitano Systems Biology Project
California Institute of Technology, Pasadena, CA, USA

Principal Investigators: John Doyle, Hiroaki Kitano

Collaborators:

Adam Arkin (BioSpice), Dennis Bray (StochSim),
Igor Goryanin (DBsolve), Andreas Kremling (ProMoT/DIVA),
Les Loew (Virtual Cell), Eric Mjolsness (Cellerator),
Pedro Mendes (Gepasi/Copasi), Masaru Tomita (E-CELL)

Motivations

- **Observation: *proliferation of software tools***
- **No single package answers all needs**
  - Different packages have different niche strengths
  - Strengths are often complementary
- **No single tool is likely to do so in the near future**
  - Range of capabilities needed is large
  - New techniques ($\Rightarrow$ new tools) evolve all the time
- **Researchers are likely to continue using multiple packages for the foreseeable future**
- **Problems with using multiple tools:**
  - Simulations & results often cannot be shared or re-used
  - Duplication of software development effort

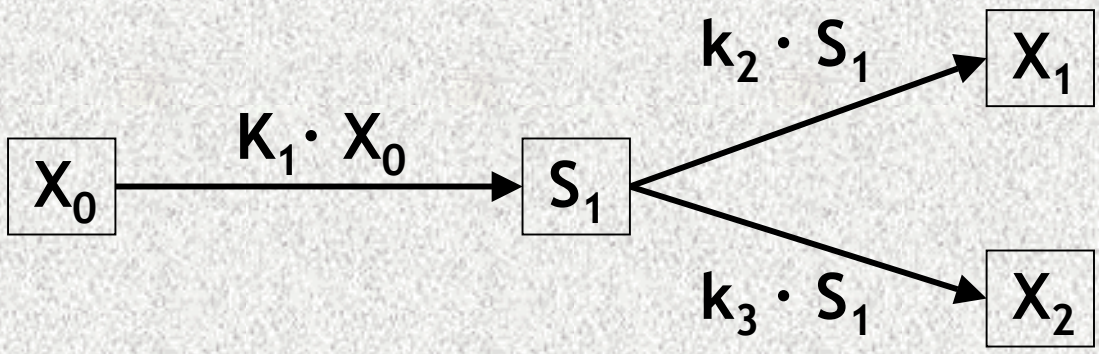
Project Goals & Approach

- **Develop software & standards that**
  - Enable sharing of modeling & analysis software
  - Enable sharing of models
- **Goal: make it easier to share tools than to reimplement**
- **Two-pronged approach**
  - Develop a common model exchange language
    - SBML: Systems Biology Markup Language
  - Develop an environment that enables tools to interact
    - SBW: Systems Biology Workbench

Systems Biology Markup Language (SBML)

- **Domain: biochemical network models**
- **XML with components that reflect the natural conceptual constructs used by modelers in the domain**
- **Reaction networks described by list of components:**
  - Beginning of model definition
    - » List of unit definitions (optional)
    - » List of **compartments**
    - » List of **species**
    - » List of parameters (optional)
    - » List of rules (optional)
    - » List of **reactions**
  - End of model definition

Example



Example (cont.)

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml level="1" version="1">
  <model name="simple">
    <listOfCompartments>
      <compartment name="c1" />
    </listOfCompartments>
    <listOfSpecies>
      <specie name="X0" compartment="c1"
                        boundaryCondition="true"
                        initialAmount="1"/>
      <specie name="S1" compartment="c1"
                        boundaryCondition="false"
                        initialAmount="0"/>
      <specie name="X1" compartment="c1"
                        boundaryCondition="true"
                        initialAmount="0"/>
      <specie name="X2" compartment="c1"
                        boundaryCondition="true"
                        initialAmount="0.23"/>
    </listOfSpecies>
```

Example (cont.)

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml level="1" version="1">
  <model name="simple">
    <listOfCompartments>
      <compartment name="c1" />
    </listOfCompartments>
    <listOfSpecies>
      <specie name="X0" compartment="c1"
                        boundaryCondition="true"
                        initialAmount="1"/>
      <specie name="S1" compartment="c1"
                        boundaryCondition="false"
                        initialAmount="0"/>
      <specie name="X1" compartment="c1"
                        boundaryCondition="true"
                        initialAmount="0"/>
      <specie name="X2" compartment="c1"
                        boundaryCondition="true"
                        initialAmount="0.23"/>
    </listOfSpecies>
```

Example (cont.)

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml level="1" version="1">
  <model name="simple">
    <listOfCompartments>
      <compartment name="c1" />
    </listOfCompartments>
    <listOfSpecies>
      <specie name="X0" compartment="c1"
                        boundaryCondition="true"
                        initialAmount="1"/>
      <specie name="S1" compartment="c1"
                        boundaryCondition="false"
                        initialAmount="0"/>
      <specie name="X1" compartment="c1"
                        boundaryCondition="true"
                        initialAmount="0"/>
      <specie name="X2" compartment="c1"
                        boundaryCondition="true"
                        initialAmount="0.23"/>
    </listOfSpecies>
```

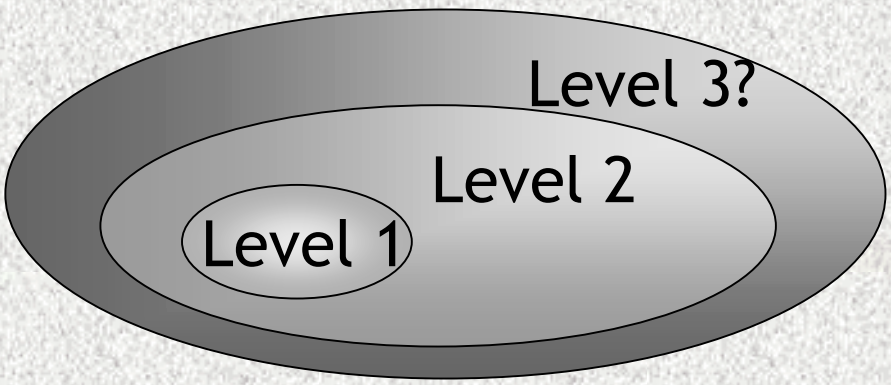
Example (cont.)

```
<listOfReactions>
   <reaction name="reaction_1" reversible="false">
      <listOfReactants>
         <specieReference specie="X0" stoichiometry="1"/>
      </listOfReactants>
      <listOfProducts>
         <specieReference specie="X0" stoichiometry="1"/>
      </listOfProducs>
      <kineticLaw formula="k1 * X0">
         <listOfParameters>
            <parameter name="k1" value="0"/>
         </listOfParameters>
      </kineticLaw>
   </reaction>

   <reaction name="reaction_2" reversible="false">
      <listOfReactants>
         <specieReference specie="S1" stoichiometry="1"/>
      </listOfReactants>
   . . .
```

Some Points about SBML

- **Users do not write in XML — software tools do!**
- **SBML is being defined incrementally**
  - SBML Level 1 covers non-spatial biochemical models
    - Kept **simple** for maximal compatibility
  - SBML Level 2 will extend Level 1 with more facilities



E.g.:
- Composition
- Geometry
- Arrays

… others

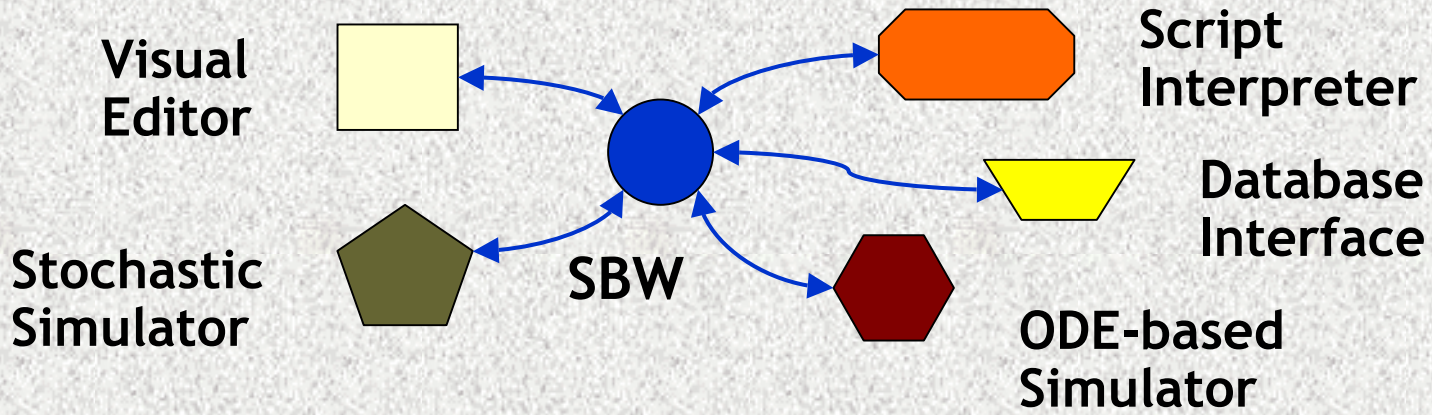
- **Defined in abstract form (UML) + textual descriptions**
  - Used to define XML encoding + XML Schema

Related Efforts

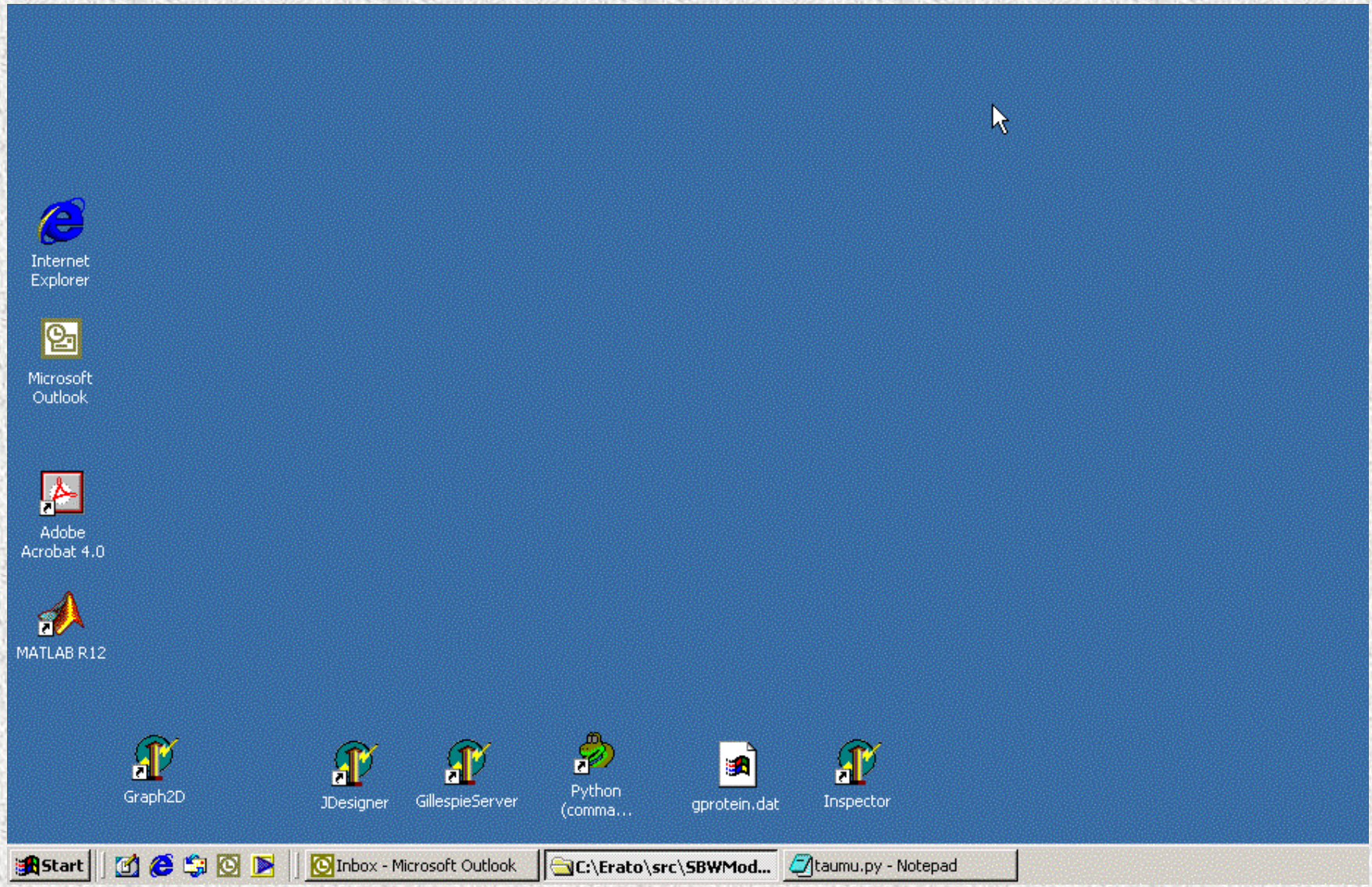
- **Similar in purpose to CellML**
  - CellML uses MathML, FieldML, etc.
    - SBML is simpler, easier for software developers to use
- **Both SBML and CellML teams are working together**
  - Striving to keep translatability between SBML and CellML

Systems Biology Workbench (SBW)

- **Simple framework for enabling application interaction**
  - Free, open-source (LGPL)
  - Portable to popular platforms and languages
  - Small, simple, understandable



SBW from the User's Perspective



SBW from the User's Perspective

- **SBW is almost invisible from the user's perspective**
- **Interaction & sequence is under user's control**
  - Each application takes center stage in turn
    - SBW is never in the forefront
  - Minimal disruption of normal tool interfaces
    - SBW has no interface of its own

From the Programmer's Perspective

- **Simple, lightweight, message-passing architecture**
  - Cross-platform compatible & language-neutral
  - Remote procedure call semantics
    - But can do message-passing semantics too
- **Uses well-known, proven technologies**
  - Communications via message-passing over plain sockets
  - Modular, distributed, broker-based architecture

SBW Design

- **API provides two styles:**
  - "Low-level": fundamental call/send operations
  - "High-level": object-oriented interface layered on top
- **Native data types supported in messages:**
  - Byte Boolean String Integer Double
  - Array (homogeneous) List (heterogeneous)

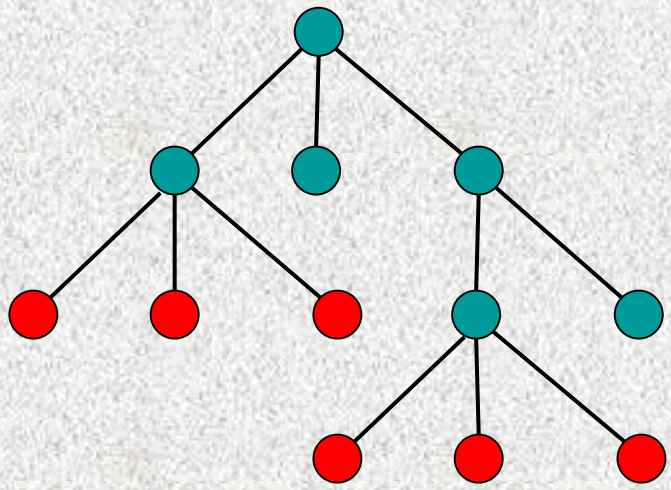
  - You can send XML, but are not limited to XML
  - You can send arbitrary binary data, or structured data

Features of SBW

- ***Modules*** **are separately-compiled executables**
  - A module defines *services* which have *methods*
  - SBW native-language libraries provide APIs
    - C, C++, Java, Delphi, Python available now
    - … but can be implemented for any language
  - APIs hide protocol, wire transfer format, etc.
    - Programmer usually **doesn't care** about this level
- ***SBW Broker*** **acts as coordinator**
  - Remembers services & modules that implement them
  - Provides directory
  - Starts modules on demand
    - Broker itself is started automatically
  - Notifies modules of events (startup, shutdown, etc.)

The SBW Broker's Registry

- **Registry records information about modules**
  - Module name
  - How to start module
  - What services the module provides
  - The categorization of those services
- **Hierarchy of service categories**



Service Categories **(Interface Hierarchy)**

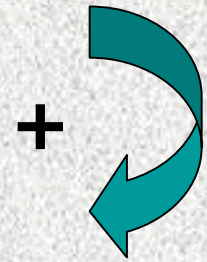
Services **(Interfaces)**

Example of Service Categories

Service	Methods

Simulation	`void loadModel(string SBML)`
	`void setStartTime(double time)`
	`void setEndTime(double time)`
	`void run()`

ODESimulation	`void setIntegrator(int method)` `void setNumPoints(int num)`



Service Categories Group Applications

- **Clients can be written to interact with classes of modules in a generic way**

(Java)

```
interface ODESimulation {
        void loadModel(string SBML)
        void setStartTime(double time)
        void setEndTime(double time)
        void run()
        void setIntegrator(int method)
        void setNumPoints(int num)
}
```

- **User menus can be grouped by categories**
- **Need help from community to define common categories of interfaces**

Java Code Example: Implementing a Module

- **Suppose you have an existing package:**

```
package math;

import java.lang.Math;

class Trig {

  public double sin(double x) {
    return Math.sin(x);
  }

  public double cos(double x) {
    return Math.cos(x);
  }
}
```

(Note absense of any SBW-specific code)

Java Code Example: Implementing a Module

```
package math;

import edu.caltech.sbw;

class TrigApplication {

  public static void main(String[] args) {
    try {
      ModuleImpl moduleImp = new ModuleImpl("Math");

      moduleImp.addService("Trig", "trig functions",
                           "Math", Trig.class);
      moduleImp.run(args);

    } catch (SBWException e) {
      e.handleWithException();
    }
  }
}
```

Java Code Example: Implementing a Module

```
package math;

import ...

class TrigApplication {

  public static void main(String[] args) {
    try {
      ModuleImpl moduleImp = new ModuleImpl("Math");

      moduleImp.addService("Trig", "trig functions",
                           "Math", Trig.class);
      moduleImp.run(args);

    } catch (SBWException e) {
      e.handleWithException();
    }
  }
}
```

Name of module

Name for human display

Java Code Example: Implementing a Module

```
package
import edu.caltech.s
class TrigApplication {
  public static void main tring[] args)
    try {
      ModuleImpl moduleImp = new ModuleIm ("Math");
      moduleImp.addService("Trig", "trig functions",
                           "Math", Trig.class);
      moduleImp.run(args);
    } catch (SBWExcep ) {
      e.
    }
  }
}
```

Service name

Name for human display

Service category

Class for implementation

Java Code Example: Implementing a Module

```
package math;

import edu.caltech.sbw;

class TrigApplication {

  public static void main(String[] args) {
    try {
      ModuleImpl moduleImp = new ModuleImpl("Math");

      moduleImp.addService("Trig", "trig functions",
                           "Math", Trig.class);
      moduleImp.run(args);

    } catch (SBWException e) {
      e.handleWithException();
    }
  }
}
```

Java Code Example: Calling a Known Module

```
interface Trig {
  double sin(double x) throws SBWException;
  double cos(double x) throws SBWException;
}

...
try {
  Module module = SBW.getModuleInstance("math");
  Service service = module.findServiceByName("Trig");
  Trig trig = (Trig)service.getServiceObject(Trig.class);

  double result = trig.sin(0.5);
  . . .
} catch (SBWException e) {
  e.handleWithDialog();
}
```

Java Code Example: Calling a Known Module

```
interface Trig {
  double sin(double x) throws SBWException;
  double cos(double x) throws SBWException;
}

...
try {
  Module module = SBW.getModuleInstance("math");
  Service service = module.findServiceByName("Trig");
  Trig trig = (Trig)service.getServiceObject(Trig.class);

  double result = trig.sin(0.5);
  . . .
} catch (SBWException e) {
  e.handleWithDialog();
}
```

Java Code Example: Calling a Known Module

```
interface Trig {
  double sin(double x) throws SBWException;
  double cos(double x) throws SBWException;
}

...
try {
  Module module = SBW.getModuleInstance("math");
  Service service = module.findServiceByName("Trig");
  Trig trig = (Trig)service.getServiceObject(Trig.class);

  double result = trig.sin(0.5);
  . . .
} catch (SBWException e) {
  e.handleWithDialog();
}
```

Java Code Example: Calling a Known Module

```
interface Trig {
  double sin(double x) throws SBWException;
  double cos(double x) throws SBWException;
}

...
try {
  Module module = SBW.getModuleInstance("math");
  Service service = module.findServiceByName("Trig");
  Trig trig = (Trig)service.getServiceObject(Trig.class);

  double result = trig.sin(0.5);
  . . .
} catch (SBWException e) {
  e.handleWithDialog();
}
```

Java Code Example: Calling a Known Module

```
interface Trig {
  double sin(double x) throws SBWException;
  double cos(double x) throws SBWException;
}

...
try {
  Module module = SBW.getModuleInstance("math");
  Service service = module.findServiceByName("Trig");
  Trig trig = (Trig)service.getServiceObject(Trig.class);

  double result = trig.sin(0.5);
  . . .
} catch (SBWException e) {
  e.handleWithDialog();
}
```

Java Code Example: Calling a Known Module

```
interface Trig {
  double sin(double x) throws SBWException;
  double cos(double x) throws SBWException;
}

...
try {
  Module module = SBW.getModuleInstance("math");
  Service service = module.findServiceByName("Trig");
  Trig trig = (Trig)service.getServiceObject(Trig.class);

  double result = trig.sin(0.5);
  . . .
} catch (SBWException e) {
  e.handleWithDialog();
}
```

Why?

- **Why not use CORBA as the underlying mechanism?**
  - Complexity, size, compatibility
  - Could not find fully-compliant open-source CORBA ORB that supports all required programming languages
    $\Rightarrow$ Would have to deal with multiple ORBs
  - SBW scheme does not require a separately compiled IDL
  - But: want to have CORBA gateway
- **Why not use SOAP or XML-RPC?**
  - Performance, data type issues, protocol issues
  - But: want to have SOAP interface
- **Why not Java RMI?**
  - Java-specific
- **Why not COM?**
  - Microsoft-specific, low portability

SBW Status & Future

- **Beta release: http://bioinformatics.org/sbw**
  http://www.cds.caltech.edu/erato
  - **Java, C, C++, Delphi, Python libraries**
  - **Windows 2000 & Linux**
  - **Developer's manuals & tutorials, examples**
  - **Modules:**
    - SBML Network Object Model
    - MATLAB model generator
    - Plotting module
    - Jarnac ODE simulator
    - Optimization module
    - Stochastic simulator
    - JDesigner visual editor
- **Spring 2002: production release 1.0**
  - Perl and (hopefully) C# libraries
  - Secure distributed operation
  - CORBA gateway