The High Level SBW Libraries

Andrew Finney, Herbert Sauro, Mike Hucka and Hamid Bolouri

ERATO Kitano Systems Biology Workbench Project
Overview

- 3 Tutorials in Java and C++:
  - Calling a known service
  - Implementing a service
  - Finding services in a known category
- Assumes some understanding of Java or C++
Tutorial 1: Calling a known Service

- Assume we know there is
  - a module “edu.caltech.trigonometry”
  - with a service “Trig”
  - Which has the methods
    - `double sin(double)`
    - `double cos(double)`
- We will call the `sin` method
Tutorial 1: Accessing a service in C++ and Java

• In Java:

Module module = SBW.getModuleInstance("edu.caltech.trigonometry");
Service service = module.findServiceByName("Trig");

• In C++:

Module module = SBW::getModuleInstance("edu.caltech.trigonometry");
Service service = module.findServiceByName("Trig");
Tutorial 1: Calling a method in Java

- Create interface for service:

```java
Interface Trigonometry {
    double sin(double x) throws SBWException;
    double cos(double x) throws SBWException;
}
```

- Create proxy with this interface and call it:

```java
Trigonometry trig = (Trigonometry)service.getServiceObject(Trigonometry.class);
Double result = trig.sin(x);
```
Tutorial 1: Calling a method in C++

• Access Method object:

```cpp
Method method = service.getMethod("sin");
```

• Invoke method, marshalling arguments and result:

```cpp
DataBlockReader resultData = method.call(DataBlockWriter() << x);

SBWDouble result;

resultData >> result;
```
Tutorial 1: Exception Handling

- Most High Level API calls can potentially throw `SBWException`
- `SBWException` base class of hierarchy of exception types
- `SBWApplicationException` can be thrown by service implementations
Tutorial 1: Summary

- Start with information on a specific service
- Access module then service
- Call Method
  - In Java: use proxy object with defined interface
  - In C++: use call on Method object
- Exception Handling
Tutorial 2: Implementing a Service

• Implementation of the service called in Tutorial 1
• Service is an interface to a software resource inside a module (executable)
• Typically an executable operates in 3 modes:
  – Normal
  – Register
  – Module
class Trig
{
    public double sin(double x)
        throws SBWApplicationException
    {
        return Math.sin(x);
    }

    public double cos(double x)
        throws SBWApplicationException
    {
        return Math.cos(x);
    }
}
Tutorial 2: Java main method

ModuleImpl moduleImp =
    new ModuleImpl(
        "edu.caltech.trigonometry",
        "trig", SBW.STATIC, this.getClass());

moduleImp.addService(
    "trig funcs", "trigonometry", Trig.class);

if (args[0].equals("-sbwregister"))
    moduleImp.register();
else if (args[0].equals("-sbwmodule"))
    moduleImp.enableServices();
else
    // in normal mode - do something else
class Sin : public Handler
{
    DataBlockWriter receive(
        Module from, DataBlockReader args)
        throws SBWApplicationException
    {
        SBWDouble x;
        args >> x ;
        return DataBlockWriter() << sin(x);
    }
};

class Cos : public Handler ...
Tutorial 2: C++ main function

ModuleImpl moduleImp(
    "edu.caltech.trigonometry", "trig", SBW::STATIC, argv[0]);
moduleImp.addService("Trig", "trig funcs", "trigonometry");

if (strcmp(argv[1], "-sbwregister") == 0)
    moduleImp.register();
else if (strcmp(argv[1], "-sbwmodule") == 0)
{
    moduleImp.setHandler("trig", new Sin(), "double sin(double)");
    moduleImp.setHandler("trig", new Cos(), "double cos(double)");
    moduleImp.enableServicesAndWait();
}
else
    // normal mode – do something else
Tutorial 2: Summary

- Create Service implementation
  - In Java: use normal class
  - In C++: one class per method
- Supply module information in ModuleImpl constructor
- Supply service implementation via addService method on ModuleImpl
- Use the register method in register mode
- Use the enableServices method in module mode
Tutorial 3: Finding and using services in a given category

• Dynamically locate services in a given category
• We will locate services in the “Analysis” category
  – This category is an important standard for SBW
  – Has one method:
    • `void doAnalysis(string sbml)`
Tutorial 3: Finding services in Java

• Get information on services in category:

```java
ServiceDescriptor[] descriptors = SBW.findServices("Analysis");
```

• Access the humanly readable name of a service:

```java
descriptors[0].getDisplayName();
```

• Get a real service:

```java
Service service = descriptors[0].getServiceInModuleInstance();
```
Tutorial 3: Finding services in C++

- Get information on services in category:
  
  ```cpp
  std::vector<ServiceDescriptor> *descriptors = SBW::findServices("Analysis");
  ```

- Access the humanly readable name of a service:
  
  ```cpp
  (*descriptors)[0].getDisplayName();
  ```

- Get a real service:
  
  ```cpp
  Service service =
  (*descriptors)[0].getServiceInModuleInstance();
  ```
Tutorial 3: Summary

- Start with just a service category and its interface
- Use `findServices` to get static information on potential services
- Perhaps display list of potential services to user using `getDisplay`Name
- Access actual service, perhaps via user selection, using `getServiceInModuleInstance`
Summary

• Refer to “Programmer’s Manual for the Systems Biology Workbench (SBW)” Finney et al for more detail
  – [http://www.cds.caltech.edu/erato/sbw/docs/api](http://www.cds.caltech.edu/erato/sbw/docs/api)
• Subject to change: implementation not complete
• Scripting language High Level APIs similar to those described in this presentation