CCA
Common Component Architecture

Writing Components

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/
tutorial-wg@cca-forum.org

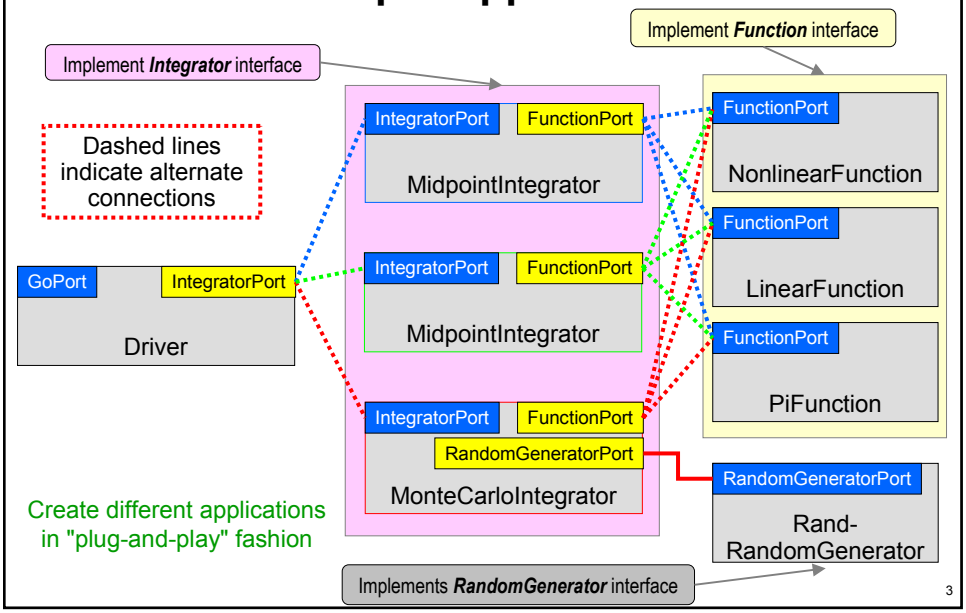

CCA
Common Component Architecture

Module Overview

- Goal: present a step-by-step approach to creating CCA components
- Example application
- Steps involved in writing CCA components
  1. Interface definition; ports
  2. Component implementation
     1. Framework interactions
     2. Component interactions: uses and provides ports
  3. Compiling
  4. Running

CCA Common Component Architecture

Example Applications



CCA Common Component Architecture

Interface Definition

- Component functionality:
  - Random number generator
    - Generates a pseudo-random number
  - Integrator
    - Computes the integral of a scalar function
  - Function
    - Computes a scalar function
  - Driver
    - Entry point into the application

CCA Common Component Architecture

MonteCarloIntegrator Component

1. Use Babel to generate C++ skeletons and implementation files from integrator.sidl
2. Fill in implementation details in integrator-component-c++/:
   - integrator_MonteCarloIntegrator_Impl.hh
   - integrator_MonteCarloIntegrator_Impl.cc
3. Create C wrapper functions (for component creation):
   - integrator_Integrator_wrapper_Impl.cc
4. Create makefile and build dynamic library
   - Makefile
   - libIntegrator-component-c++.so
5. Create integrator.cca (Ccaffeine-specific)

CCA Common Component Architecture

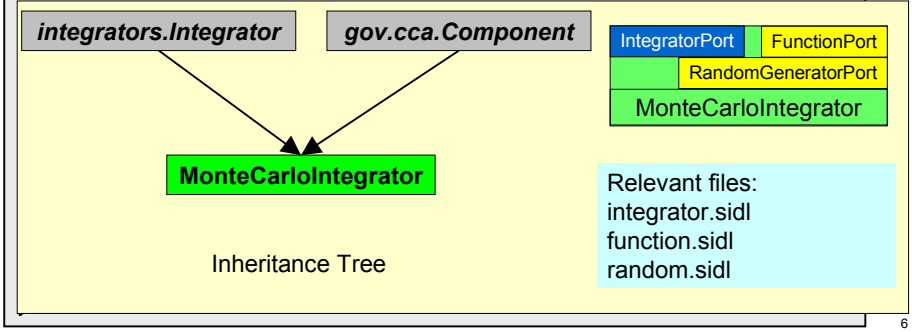
File: integrator.sidl

Integrator Port

```
version integrators 1.0;

package integrators {

  interface Integrator extends gov.cca.Port {
    double integrate(in double lowBound,
                     in double upBound, in int count);
  }
```



CCA Common Component Architecture

Using Babel to Create The Repository

- A repository containing XML versions of the SIDL definition is created first; it will be used for name resolution later
- Makefile fragment (for all SIDL definitions in this example):

```
SIDLFILES = cca.sidl integrator.sidl function.sidl \
        random.sidl driver.sidl

.repository: $(SIDLFILES)
        rm -f repository/*.xml \
        babel --xml --repository-path=repository \
        --output-directory=repository $(SIDLFILES)
        touch .repository
```


CCA Common Component Architecture

Using Babel to Generate Code

- Makefile fragment (top-level directory):

```
.integrator-component-c++: integrator.sidl cca.sidl
        babel --server=C++ --repository-path=repository \
        --output-directory=integrator-component-c++ \
        --suppress-timestamp integrators \
        randomgen.RandomGenerator functions.Function
        touch .integrator-component-c++
```

- Important: the *randomgen.RandomGenerator* and *functions.Function* interfaces are referenced by the Integrator implementation(s) and are thus included in the command line for generating the sources for the integrators package.

Contents of integrator-component-c++/

SIDL.hh	gov_cca_ComponentID_IOR.c	integrators_Integrator.hh
SIDL_BaseClass.cc	gov_cca_ComponentID_IOR.h	integrators_Integrator_IOR.c
SIDL_BaseClass.hh	gov_cca_Component_IOR.c	integrators_Integrator_IOR.h
SIDL_BaseException.cc	gov_cca_Component_IOR.h	integrators_Integrator_wrapper_Impl.cc
SIDL_BaseException.hh	gov_cca_Port.cc	integrators_MidpointIntegrator.cc
SIDL_BaseInterface.cc	gov_cca_Port.hh	integrators_MidpointIntegrator.hh
SIDL_BaseInterface.hh	gov_cca_Port_IOR.c	integrators_MidpointIntegrator_IOR.c
SIDL_DLL.cc	gov_cca_Port_IOR.h	integrators_MidpointIntegrator_IOR.h
SIDL_DLL.hh	gov_cca_Services.cc	integrators_MidpointIntegrator_Impl.cc
SIDL_Loader.cc	gov_cca_Services.hh	integrators_MidpointIntegrator_Impl.hh
SIDL_Loader.hh	gov_cca_Services_IOR.c	integrators_MidpointIntegrator_Skel.cc
babel.make	gov_cca_Services_IOR.h	integrators_MonteCarloIntegrator.cc
functions_Function.cc	gov_cca_Type.hh	integrators_MonteCarloIntegrator.hh
functions_Function.hh	gov_cca_TypeMap.cc	integrators_MonteCarloIntegrator_IOR.c
functions_Function_IOR.c	gov_cca_TypeMap.hh	integrators_MonteCarloIntegrator_IOR.h
functions_Function_IOR.h	gov_cca_TypeMap_IOR.c	integrators_MonteCarloIntegrator_Impl.cc
gov_cca_CCAException.cc	gov_cca_TypeMap_IOR.h	integrators_MonteCarloIntegrator_Impl.hh
gov_cca_CCAException.hh	gov_cca_TypeMismatchException.cc	integrators_MonteCarloIntegrator_Skel.cc
gov_cca_CCAExceptionType.hh	gov_cca_TypeMismatchException.hh	integrators_ParallelIntegrator.cc
gov_cca_CCAExceptionType_IOR.c	gov_cca_TypeMismatchException_IOR.c	integrators_ParallelIntegrator.hh
gov_cca_CCAExceptionType_IOR.h	gov_cca_TypeMismatchException_IOR.h	integrators_ParallelIntegrator_IOR.c
gov_cca_CCAException_IOR.c	gov_cca_TypeMismatchException_Impl.cc	integrators_ParallelIntegrator_IOR.h
gov_cca_CCAException_IOR.h	gov_cca_TypeMismatchException_Impl.hh	integrators_ParallelIntegrator_Impl.cc
gov_cca_CCAException_Impl.cc	gov_cca_TypeMismatchException_Skel.cc	integrators_ParallelIntegrator_Impl.hh
gov_cca_CCAException_Impl.hh	gov_cca_Type_IOR.c	integrators_ParallelIntegrator_Skel.cc
gov_cca_CCAException_Skel.cc	gov_cca_Type_IOR.h	randomgen_RandomGenerator.cc
gov_cca_Component.cc	integrators.cca	randomgen_RandomGenerator.hh
gov_cca_Component.hh	integrators.hh	randomgen_RandomGenerator_IOR.c
gov_cca_ComponentID.cc	integrators_IOR.h	randomgen_RandomGenerator_IOR.h
gov_cca_ComponentID.hh	integrators_Integrator.cc	

CCA Common Component Architecture

File: integrator-component-c++/integrators_MonteCarloIntegrator_Impl.hh

MonteCarloIntegrator Component:Implementation Header

```
namespace integrators {

  /**
   * Symbol "integrators.MonteCarloIntegrator" (version 1.0)
   */
  class MonteCarloIntegrator_impl
  {
  private:
    // Pointer back to IOR.
    // Use this to dispatch back through IOR vtable.
    MonteCarloIntegrator self;

    // DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator._implementation)
    // Put additional implementation details here...
    gov::cca::Services  frameworkServices;
    // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator._implementation)

    ...

  };  // end class MonteCarloIntegrator_impl

} // end namespace integrators
```

Reference to framework Services object

File: integrator-component-c++/integrators_MonteCarloIntegrator_Impl.cc

MonteCarloIntegrator Component: Framework Interaction

```
integrators::MonteCarloIntegrator_impl::setServices (
  /*in*/ gov::cca::Services services )
throw ()
{
  // DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.setServices)
  frameworkServices = services;
  if (frameworkServices._not_nil ()) {
     gov::cca::TypeMap tm = frameworkServices.createTypeMap ();
     gov::cca::Port p = self;     // Babel required cast

     // Port provided by all Integrator implementations
     frameworkServices.addProvidesPort (p, "IntegratorPort",
                    "integrators.Integrator", tm);
     // Ports used by MonteCarloIntegrator
     frameworkServices.registerUsesPort ("FunctionPort", "functions.Function",
                                         tm);
     frameworkServices.registerUsesPort ("RandomGeneratorPort",
                                          "randomgen.RandomGenerator",  tm);
  }
  // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.setServices)
}
```

Save a pointer to the Services object

Port name

Port type

TypeMap reference

File: integrator-component-c++/integrators_MonteCarloIntegrator_Impl.cc

MonteCarloIntegrator Component: integrate() Method

```
double
integrators::MonteCarloIntegrator_impl::integrate (
  /*in*/ double lowBound, /*in*/ double upBound,  /*in*/ int32_t count ) throw ()
{
  // DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.integrate)
   gov::cca::Port port;
   double sum = 0.0;
   functions::Function function_m;
   randomgen::RandomGenerator random_m;

   random_m = frameworkServices.getPort ("RandomGeneratorPort");
   function_m = frameworkServices.getPort ("FunctionPort");

   for (int i = 0; i < count; i++) {
     double x = random_m.getRandomNumber ();
     sum = sum + function_m.evaluate (x);
   }

   frameworkServices.releasePort ("RandomGeneratorPort");
   frameworkServices.releasePort ("FunctionPort");

   return (upBound - lowBound) * sum / count;
  // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.integrate)
}
```

Get a RandomGenerator reference

Get a Function reference

Get a random number

Evaluate function at random value

Release ports

Return integral value

Writing the C Wrapper

- At present, Ccaffeine requires some C functions for dynamic loading of components; example for two components:

```
#include "integrators.hh"
#include "gov_cca_Component.hh"
#include <stdio.h>

extern "C" {
  gov::cca::Component create_MonteCarloIntegrator() {
    ::gov::cca::Component ex = integrators::MonteCarloIntegrator::_create();
    return ex;
  }
  gov::cca::Component create_ParallelIntegrator() {
    ::gov::cca::Component ex = integrators::ParallelIntegrator::_create();
    return ex;
  }
  char **getComponentList() {
    static char *list[3] ;
    list[0] = "create_MonteCarloIntegrator integrators.MonteCarloIntegrator";
    list[1] = "create_ParallelIntegrator integrators.ParallelIntegrator";
    list[2] = 0;
    return list;
  }
}
```

Create a MonteCarloIntegrator instance

Create a ParallelIntegrator instance

Return a list of components contained in this dynamic library

C wrapper function name

Component name

MonteCarloIntegrator: integrators.cca

- Ccaffeine-specific file specifying the name of the dynamic library and creation method for each component

```
!date=Thu Aug 15 14:53:23 CDT 2002
!location=
!componentType=babel
libIntegrator-component-c++.so
create_MonteCarloIntegrator integrators.MonteCarloIntegrator
create_ParallelIntegrator integrators.ParallelIntegrator
```

Component type: “babel” or “classic” (C++)

C wrapper function name

Component name

File: integrator-component-c++/integrators_ParallelIntegrator_Impl.cc

Parallel Example: MidpointIntegrator integrate() Method

```
double
integrators::ParallelIntegrator_impl::integrate (  /*in*/ double lowBound,  /*in*/ double upBound,
                                                    /*in*/ int32_t count ) throw ()
{
  // DO-NOT-DELETE splicer.begin(integrators.ParallelIntegrator.integrate)
    gov::cca::Port port;
    functions::Function function_m;

    // Get Function port
    function_m = frameworkServices.getPort("FunctionPort");

    int n, myid, numprocs, i;
    double result, myresult, h, sum, x;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Get_processor_name(processor_name, &namelen);

    fprintf(stderr, "Process %d on %s: number of intervals = %d\n", myid,
          processor_name, count);
    fflush(stderr);
    // … Continued on next page…
```

Get a Function reference

Parallel environment details

File: integrator-component-c++/integrators_ParallelIntegrator_Impl.cc

Parallel Example: MidpointIntegrator integrate() Method (continued)

```
// …
    MPI_Bcast(&count, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (count == 0) {
      return -1;
    } else {
      h = (upBound - lowBound) / (double) count;
      sum = 0.0;
      for (i = myid + 1; i <= count; i += numprocs) {
        x = h * ((double) i - 0.5);
        sum += function_m.evaluate(x);
      }
      myresult = h * sum;

      MPI_Reduce(&myresult, &result, 1, MPI_DOUBLE, MPI_SUM, 0,
                 MPI_COMM_WORLD);
    }

    frameworkServices.releasePort("FunctionPort");
    printf("result is %f\n", result);
    return result;

  // DO-NOT-DELETE splicer.end(integrators.ParallelIntegrator.integrate)
}
```

Compute integral in parallel

Evaluate function

Release port

Return integral value

RandRandomGenerator Component

1. Use Babel to generate C++ skeletons and implementation files for random.sidl
2. Fill in implementation details in random-component-c++/:
   - randomgen_RandRandomGenerator_Impl.hh
   - randomgen_RandRandomGenerator_Impl.cc
3. Create C wrapper functions (for component creation).
   - randomgen_RandomGenerator_wrapper_Impl.cc
4. Create makefile and build dynamic library
   - Makefile
   - libIntegrator-component-c++.so
5. Create random.cca (Ccaffeine-specific)

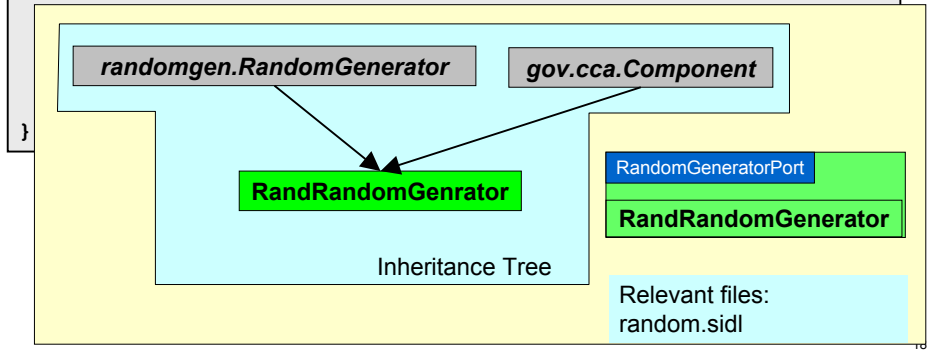
File: random.sidl

RandomGenerator Port

```
version randomgen 1.0;

package randomgen {

  interface RandomGenerator extends gov.cca.Port {
    double getRandomNumber();
  }
}
```



RandRandomGenerator Component:Implementation Header

```
namespace randomgen {

 /**
  * Symbol "randomgen.RandRandomGenerator" (version 1.0)
  */
 class RandRandomGenerator_impl
 {
 private:
   // Pointer back to IOR.
   // Use this to dispatch back through IOR vtable.
   RandRandomGenerator self;

   // DO-NOT-DELETE splicer.begin(randomgen.RandRandomGenerator._implementation)
   // Put additional implementation details here...
   gov::cca::Services  frameworkServices;
   // DO-NOT-DELETE splicer.end(randomgen.RandRandomGenerator._implementation)

   …

 };  // end class RandRandomGenerator_impl

} // end namespace randomgen
```

Reference to framework Services object

RandRandomGenerator Component: Framework Interaction

```
randomgen::RandRandomGenerator_impl::setServices (
  /*in*/ gov::cca::Services services )
throw ()
{
  // DO-NOT-DELETE splicer.begin(randomgen.RandRandomGenerator.setServices)
  frameworkServices = services;
  if (frameworkServices._not_nil ()) {
     gov::cca::TypeMap tm = frameworkServices.createTypeMap ();

     gov::cca::Port p = self;     // Babel required cast

     // Port provided by RandomGenerator implementations
     frameworkServices.addProvidesPort (p, "RandomGeneratorPort",
                "randomgen.RandomGenerator", tm);
     // No ports are used by this RandomGenerator implementation
   }
   // DO-NOT-DELETE splicer.end(randomgen.RandRandomGenerator.setServices)
}
```

Save a pointer to the Services object

Port name

Port type

TypeMap reference

PiFunction Component

1. Use Babel to generate C++ skeletons and implementation files for function.sidl
2. Fill in implementation details in function-component-c++/:
   - functions_PiFunction_Impl.hh
   - functions_PiFunction_Impl.cc
3. Create C wrapper functions (for component creation).
   - functions_Function_wrapper_Impl.cc
4. Create makefile and build dynamic library
   - Makefile
   - libFunction-component-c++.so
5. Create functions.cca (Ccaffeine-specific)

File: function.sidl

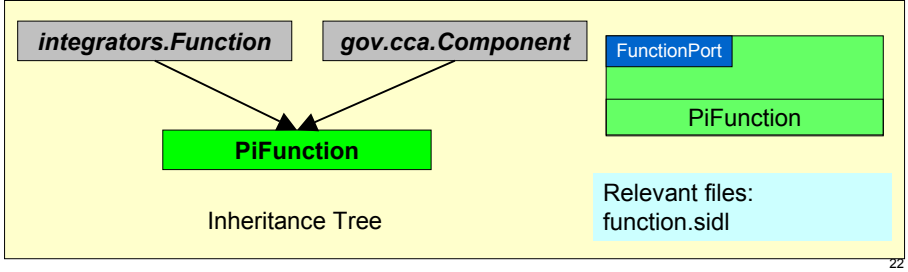
Function Port

```
version functions 1.0;

package functions {

  interface Function extends gov.cca.Port {
    double evaluate(in double x);
  }

  class PiFunction implements-all Function,
```



PiFunction Component:Implementation Header

```
namespace functions {

 /**
  * Symbol "function.PiFunction" (version 1.0)
  */
 class PiFunction_impl
 {
 private:
   // Pointer back to IOR.
   // Use this to dispatch back through IOR vtable.
   PiFunction self;

   // DO-NOT-DELETE splicer.begin(functions.PiFunction._implementation)
   // Put additional implementation details here...
   gov::cca::Services  frameworkServices;
   // DO-NOT-DELETE splicer.end(functions.PiFunction._implementation)

   …

 };  // end class PiFunction_impl

} // end namespace functions
```

Reference to framework Services object

PiFunction Component: Framework Interaction

```
functions::PiFunction_impl::setServices (
  /*in*/ gov::cca::Services services )
throw ()
{
  // DO-NOT-DELETE splicer.begin(functions.PiFunction.setServices)
  frameworkServices = services;
  if (frameworkServices._not_nil ()) {
     gov::cca::TypeMap tm = frameworkServices.createTypeMap ();

     gov::cca::Port p = self;     // Babel required cast

     // Port provided by Function implementations
     frameworkServices.addProvidesPort (p, "FunctionPort",
                "functions.Function", tm);
     // No Ports are used by this Function implementation
   }
   // DO-NOT-DELETE splicer.end(functions.PiFunction.setServices)
}
```

Save a pointer to the Services object

Port name

Port type

TypeMap reference

Driver Component

1. Use Babel to generate C++ skeletons and implementation files for driver.sidl
2. Fill in implementation details in driver-component-c++/:
   - tutorial_Driver_Impl.hh
   - tutorial_Driver_Impl.cc
3. Create C wrapper functions (for component creation).
   - tutorial_Driver_wrapper_Impl.cc
4. Create makefile and build dynamic library
   - Makefile
   - libDriver-component-c++.so
5. Create driver.cca (Ccaffeine-specific)

Driver SIDL Definition

- Driver implements standard gov.cca.ports.GoPort
- No additional interfaces defined

```
version tutorial 1.0;

package tutorial {

  class Driver implements-all gov.cca.ports.GoPort,
                              gov.cca.Component
  {
  }

}
```

Driver Component: Framework Interaction

```
tutorial::Driver_impl::setServices (
  /*in*/ gov::cca::Services services )
throw ()
{
  // DO-NOT-DELETE splicer.begin(tutorial.Driver.setServices)
  frameworkServices = services;
  if (frameworkServices._not_nil ()) {
     gov::cca::TypeMap tm = frameworkServices.createTypeMap ();

     gov::cca::Port p = self;     // Babel required cast

     // Port provided by Function implementations
     frameworkServices.addProvidesPort (p, "GoPort",
                                "gov.cca.ports.GoPort", tm);
     // Port used by the Driver component
     frameworkServices.registerUsesPort ("IntegratorPort",
                                "integrators.Integrator", tm);
   }
   // DO-NOT-DELETE splicer.end(tutorial.Driver.setServices)
}
```

Save a pointer to the Services object

Port name

Port type

TypeMap pointer

Driver Component: GoPort implementation

```
int32_t
tutorial::Driver_impl::go () throw ()
{
  // DO-NOT-DELETE splicer.begin(tutorial.Driver.go)
  double value;
  int count = 100000;   // number of intervals/random samples
  double lowerBound = 0.0, upperBound = 1.0;

  // Ports
  ::gov::cca::Port port;
  ::integrators::Integrator integrator;

  port = frameworkServices.getPort("IntegratorPort");
  integrator = port;

  value = integrator.integrate (lowerBound, upperBound, count);

  fprintf(stdout,"Value = %lf\n", value);

  frameworkServices.releasePort ("IntegratorPort");
  return 0;

  // DO-NOT-DELETE splicer.end(tutorial.Driver.go)
}
```

Get an Integrator reference

Invoke the integrate method

Output integration result

Release ports

CCA Common Component Architecture

Build Issues

- Dynamic (shared) libraries
  - For each component or a set of components, build a dynamic library
  - No linking of libraries for components on which current component depends
  - Non-component libraries on which a component depends directly (e.g., BLAS), must be linked explicitly when the shared library is created

CCA Common Component Architecture

File: integrator-component-c++/Makefile

Complete Makefile for MonteCarloIntegrator

```
include babel.make

WRAPPERS = integrators_Integrator_wrapper_Impl.cc

INCLUDES = -I$(BABEL_ROOT)/include  -I. -I$(MPI_HOME)/include

all: libIntegrator-component-c++.so

.c.o:
	gcc -g -fPIC $(INCLUDES) -c $< -o $(<:.c=.o)
.cc.o:
	g++ -g -fPIC $(INCLUDES) -c $< -o $(<:.cc=.o)

OBJS = $(IMPLSRCS:.cc=.o) $(IORSRCS:.c=.o) $(SKELSRCS:.cc=.o) \
	$(STUBSRCS:.cc=.o)  $(WRAPPERS:.cc=.o)

LIBS = -Wl,-rpath,$(BABEL_ROOT)/lib -L$(BABEL_ROOT)/lib -lsidl

libIntegrator-component-c++.so: $(OBJS)
	g++ -shared $(INCLUDES) $(OBJS) -o $@  $(LIBS)

clean:
	$(RM) *.o libIntegrator-component-c++.so
```

Running the Example

Next: Using the Ccaffeine framework