



# Writing Components

**CCA Forum Tutorial Working Group**

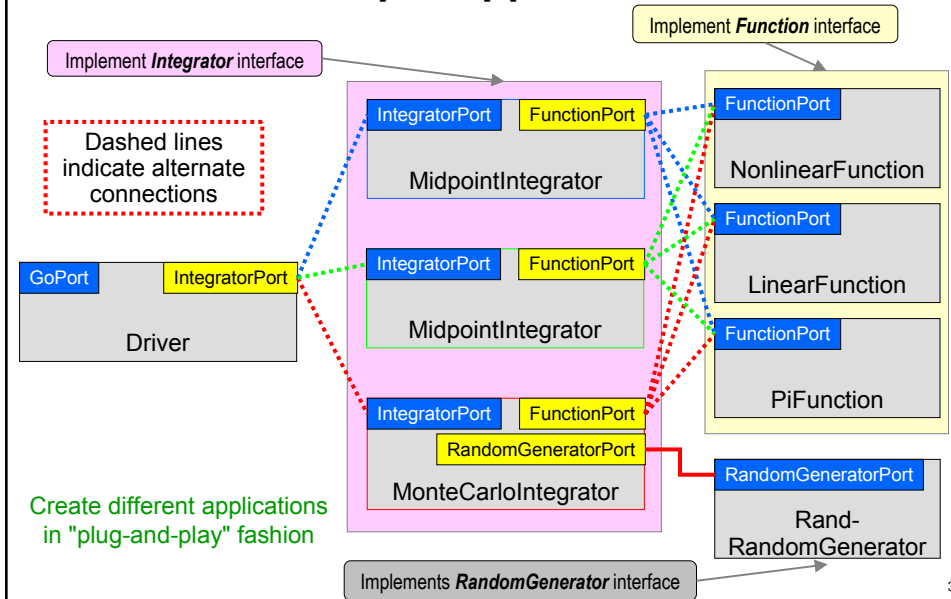
<http://www.cca-forum.org/tutorials/>  
[tutorial-wg@cca-forum.org](mailto:tutorial-wg@cca-forum.org)



## 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

## Example Applications



3

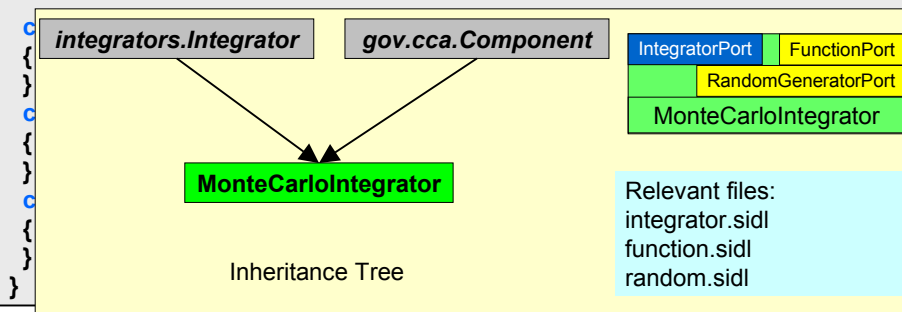
## Interface Definition

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

4

## Integrator Port

```
package integrators version 1.0 {  
  
  interface Integrator extends gov.cca.Port  
  {  
    double integrate(in double lowBound,  
                    in double upBound, in int count);  
  }  
}
```



5

## 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

```
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
```

6

## MonteCarloIntegrator Component

The next slides show two possible, semantically identical, implementations:

- C++
- Fortran 90

The two are shown for demonstration purposes; in real applications, the same functionality will not be re-implemented in multiple languages.

The tutorial example source code contains equivalent implementations of some components in C++, Fortran 90, Fortran 77, and Python.

## MonteCarloIntegrator Component (C++ Implementation)

1. Use Babel to generate C++ skeletons and implementation files for integrator.sidl
2. Fill in implementation details in integrator-component-c++/:
  - integrators\_MonteCarloIntegrator\_Impl.hh
  - integrators\_MonteCarloIntegrator\_Impl.cc
3. Create makefile and build dynamic library
  - Makefile.in
  - libIntegrator-component-c++.so
4. Create integrators.cca (Ccaffeine-specific)

## Using Babel to Generate Code

```
.integrator-component-c++: integrator.sidl cca.sidl
$(BABEL) --server=c++ --repository-path=repository \
--output-directory=integrator-component-c++ \
--exclude='^gov.*' --exclude='^SIDL.*' \
--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.
- **Note:** C++ client stubs are automatically generated

9

## Generated Files: C++ Server

### Contents of integrator-component-c++/

babel.make  
 functions\_Function.cc  
 functions\_Function.hh  
 functions\_Function\_IOR.c  
 functions\_Function\_IOR.h  
**integrators.cca**  
 integrators.hh  
 integrators\_Integrator.cc  
 integrators\_Integrator.hh  
 integrators\_Integrator\_IOR.c  
 integrators\_Integrator\_IOR.h  
 integrators\_IOR.h  
 integrators\_MidpointIntegrator.cc

integrators\_MidpointIntegrator.hh  
[integrators\\_MidpointIntegrator\\_Impl.cc](#)  
[integrators\\_MidpointIntegrator\\_Impl.hh](#)  
 integrators\_MidpointIntegrator\_IOR.c  
 integrators\_MidpointIntegrator\_IOR.h  
 integrators\_MidpointIntegrator\_Skel.cc  
 integrators\_MonteCarloIntegrator.cc  
 integrators\_MonteCarloIntegrator.hh  
[integrators\\_MonteCarloIntegrator\\_Impl.cc](#)  
[integrators\\_MonteCarloIntegrator\\_Impl.hh](#)  
 integrators\_MonteCarloIntegrator\_IOR.c  
 integrators\_MonteCarloIntegrator\_IOR.h  
 integrators\_MonteCarloIntegrator\_Skel.cc

integrators\_ParallelIntegrator.cc  
 integrators\_ParallelIntegrator.hh  
[integrators\\_ParallelIntegrator\\_Impl.cc](#)  
[integrators\\_ParallelIntegrator\\_Impl.hh](#)  
 integrators\_ParallelIntegrator\_IOR.c  
 integrators\_ParallelIntegrator\_IOR.h  
 integrators\_ParallelIntegrator\_Skel.cc  
 Makefile  
**Makefile.in**  
 randomgen\_RandomGenerator.cc  
 randomgen\_RandomGenerator.hh  
 randomgen\_RandomGenerator\_IOR.c  
 randomgen\_RandomGenerator\_IOR.h

10

## MonteCarloIntegrator Component (C++): 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",
    // Port type
    "integrators.Integrator", tm);
    // TypeMap reference
    // Ports used by MonteCarloIntegrator
    frameworkServices.registerUsesPort ("FunctionPort", "functions.Function",
    tm);
    frameworkServices.registerUsesPort ("RandomGeneratorPort",
    "randomgen.RandomGenerator", tm);
  }
  // DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.setServices)
}

```

## MonteCarloIntegrator Component (C++): 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)
}

```

## 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"); ← Get a Function reference

  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); ← Parallel environment details

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

## 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); ← Evaluate function
  }
  myresult = h * sum;

  MPI_Reduce(&myresult, &result, 1, MPI_DOUBLE, MPI_SUM, 0,
    MPI_COMM_WORLD); ← Compute integral in parallel
}

frameworkServices.releasePort("FunctionPort"); ← Release port
printf("result is %f\n", result);
return result; ← Return integral value

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

## MonteCarloIntegrator Component (F90 implementation)

1. Use Babel to generate Fortran 90 skeletons and implementation files from `integrator.sidl` in `integrator-f90` dir.
2. Use Babel to generate C++ client stubs in `integrator-component-f90` dir.
3. Fill in implementation details in `integrator-component-f90/`:
  - `integrator_MonteCarloIntegrator_Mod.F90`
  - `integrator_MonteCarloIntegrator_Impl.F90`
4. Create makefile and build dynamic libraries in the server and client subdirectories.
  - `Integrator-f90/libIntegrator-component-f90.so`
  - `Integrator-component-f90/libIntegrator-component-f90-c++client.so`
5. Create `integrator.cca` (Ccaffeine-specific)

15

## Using Babel to Generate Code

```
.integrator-component-f90: integrator.sidl cca.sidl
$(BABEL) --server=f90 --repository-path=repository \
--output-directory=integrator-f90 \
--exclude='^gov.*' --exclude='^SIDL.*' \
--suppress-timestamp \
  integrators randomgen.RandomGenerator functions.Function
$(BABEL) --client=c++ --repository-path=repository \
--output-directory=integrator-component-f90 \
--exclude='^gov.*' --exclude='^SIDL.*' \
--suppress-timestamp integrators
touch .integrator-component-f90
```

- **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.

16



## Generated Files: Fortran90 Server

### Contents of integrator-f90/

babel.make	integrators_MidpointIntegrator_IOR.c	integrators_ParallelIntegrator_Mod.F90
functions_Function_array.F90	integrators_MidpointIntegrator_IOR.h	integrators_ParallelIntegrator_type.F90
functions_Function.F90	integrators_MidpointIntegrator_Mod.F90	Makefile.in
functions_Function_fAbbrev.h	integrators_MidpointIntegrator_type.F90	randomgen_RandomGenerator_array.F90
functions_Function_fStub.c	integrators_MonteCarloIntegrator_array.F90	randomgen_RandomGenerator.F90
functions_Function_IOR.c	integrators_MonteCarloIntegrator.F90	randomgen_RandomGenerator_fAbbrev.h
functions_Function_IOR.h	integrators_MonteCarloIntegrator_fAbbrev.h	randomgen_RandomGenerator_fStub.c
functions_Function_type.F90	integrators_MonteCarloIntegrator_fSkel.c	randomgen_RandomGenerator_IOR.c
integrators_Integrator_array.F90	integrators_MonteCarloIntegrator_fStub.c	randomgen_RandomGenerator_IOR.h
integrators_Integrator.F90	integrators_MonteCarloIntegrator_Impl.F90	randomgen_RandomGenerator_type.F90
integrators_Integrator_fAbbrev.h	integrators_MonteCarloIntegrator_IOR.c	SIDL_bool_array.F90
integrators_Integrator_fStub.c	integrators_MonteCarloIntegrator_IOR.h	SIDL_char_array.F90
integrators_Integrator_IOR.c	integrators_MonteCarloIntegrator_Mod.F90	SIDL_dcomplex_array.F90
integrators_Integrator_IOR.h	integrators_MonteCarloIntegrator_type.F90	SIDL_double_array.F90
integrators_Integrator_type.F90	integrators_ParallelIntegrator_array.F90	SIDL_fcomplex_array.F90
integrators_IOR.h	integrators_ParallelIntegrator.F90	SIDL_float_array.F90
integrators_MidpointIntegrator_array.F90	integrators_ParallelIntegrator_fAbbrev.h	SIDL_int_array.F90
integrators_MidpointIntegrator.F90	integrators_ParallelIntegrator_fSkel.c	SIDL_long_array.F90
integrators_MidpointIntegrator_fAbbrev.h	integrators_ParallelIntegrator_fStub.c	SIDL_opaque_array.F90
integrators_MidpointIntegrator_fSkel.c	integrators_ParallelIntegrator_Impl.F90	SIDL_string_array.F90
integrators_MidpointIntegrator_fStub.c	integrators_ParallelIntegrator_IOR.c	
integrators_MidpointIntegrator_Impl.F90	integrators_ParallelIntegrator_IOR.h	

17

## Generated Files: C++ Client

### Contents of integrator-component-f90/

babel.make	integrators_MonteCarloIntegrator.cc
integrators.cca	integrators_MonteCarloIntegrator.hh
integrators.hh	integrators_MonteCarloIntegrator_IOR.c
integrators_Integrator.cc	integrators_MonteCarloIntegrator_IOR.h
integrators_Integrator.hh	integrators_ParallelIntegrator.cc
integrators_Integrator_IOR.c	integrators_ParallelIntegrator.hh
integrators_Integrator_IOR.h	integrators_ParallelIntegrator_IOR.c
integrators_IOR.h	integrators_ParallelIntegrator_IOR.h
integrators_MidpointIntegrator.cc	Makefile
integrators_MidpointIntegrator.hh	Makefile.in
integrators_MidpointIntegrator_IOR.c	
integrators_MidpointIntegrator_IOR.h	

18

## MonteCarloIntegrator Component: F90 Module

```
#include"integrators_MonteCarloIntegrator_fAbbrev.h"
module integrators_MonteCarloIntegrator_impl

! DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.use)
! Insert use statements here...
! Framework Services module
use gov_cca_Services
! DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.use)

type integrators_MonteCarloIntegrator_private
sequence
! DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.private_data)
! integer :: place_holder ! replace with your private data
type(gov_cca_Services_t) :: myservices ← Reference to framework Services object
! DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.private_data)
end type integrators_MonteCarloIntegrator_private

type integrators_MonteCarloIntegrator_wrap
sequence
type(integrators_MonteCarloIntegrator_private), pointer :: d_private_data
end type integrators_MonteCarloIntegrator_wrap

end module integrators_MonteCarloIntegrator_impl
```

19

## MonteCarloIntegrator Component (F90): Framework Interaction

```
recursive subroutine MonteC_setServicesucff4xebul_mi(self, services)
use integrators_MonteCarloIntegrator
use gov_cca_Services
use integrators_MonteCarloIntegrator_impl
! DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.setServices.use)
! Insert use statements here...
use gov_cca_TypeMap
use gov_cca_Port
use SIDL_BaseException
! DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.setServices.use)
implicit none
type(integrators_MonteCarloIntegrator_t) :: self
type(gov_cca_Services_t) :: services
! DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.setServices)
type(gov_cca_TypeMap_t) :: myTypeMap
type(gov_cca_Port_t) :: integratorPort
type(SIDL_BaseException_t) :: excpt
! Access private data
type(integrators_MonteCarloIntegrator_wrap) :: pd
external integrators_MonteCarloIntegrator_get_data_m
call integrators_MonteCarloIntegrator_get_data_m(self, pd)
! Set my reference to the services handle
pd%d_private_data%myservices = services ← Save a handle to the Services object
call addRef(services)
! Create a TypeMap with my properties
call createTypeMap(pd%d_private_data%myservices, myTypeMap, excpt)
```

20

## MonteCarloIntegrator Component (F90): Framework Interaction (Continued)

```

call cast(self, integratorPort) ← Explicit cast to Port

! Register my provides port
call addProvidesPort(pd%d_private_data%myservices, integratorPort, &
  'IntegratorPort', 'integrators.Integrator', &
  TypeMap → myTypeMap, excpt) ← Exception ← Port type
if (not_null(excpt)) then
  write(*, *) 'Exception: MonteCarloIntegrator:setServices addProvidesPort'
end if

! The ports I use
call registerUsesPort(pd%d_private_data%myservices, &
  Port name → 'FunctionPort', 'functions.Function', &
  myTypeMap, excpt) ← Port type

call registerUsesPort(pd%d_private_data%myservices, &
  'RandomGeneratorPort', 'randomgen.RandomGenerator', &
  myTypeMap, excpt)

call deleteRef(myTypeMap)

! DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.setServices)
end subroutine MonteC_setServicesucff4xebul_mi
  
```

21

## MonteCarloIntegrator Component (F90): integrate() Method

```

recursive subroutine MonteCar_integrateni9nnumrzd_mi(self, lowBound, upBound, &
  count, retval)
  use integrators_MonteCarloIntegrator
  use integrators_MonteCarloIntegrator_impl
  ! DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.integrate.use)
  ! Insert use statements here...
  use functions_Function
  use randomgen_RandomGenerator
  use gov_cca_Services
  use gov_cca_Port
  use sidl_BaseException
  ! DO-NOT-DELETE splicer.end(integrators.MonteCarloIntegrator.integrate.use)
  implicit none
  type(integrators_MonteCarloIntegrator t) :: self
  real(selected_real_kind(15, 307)) :: lowBound
  real(selected_real_kind(15, 307)) :: upBound
  integer(selected_int_kind(9)) :: count
  real(selected_real_kind(15, 307)) :: retval

  ! DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.integrate)
  ! Insert the implementation here...
  
```

22

## MonteCarloIntegrator Component (F90): integrate() Method (Cont.)

```
! DO-NOT-DELETE splicer.begin(integrators.MonteCarloIntegrator.integrate)
! Insert the implementation here...
type(gov_cca_Port_t) :: generalPort
type(functions_Function_t) :: functionPort
type(randomgen_RandomGenerator_t) :: randomPort
type(SIDL_BaseException_t) :: excpt
type(integrators_MonteCarloIntegrator_wrap) :: pd
external integrators_MonteCarloIntegrator__get_data_m
real (selected_real_kind(15, 307)) :: sum, width, x, func
integer (selected_int_kind(9)) :: i
! Access private data
call integrators_MonteCarloIntegrator__get_data_m(self, pd)
retval = -1
if (not_null(pd%d_private_data%myservices)) then
! Obtain a handle to a FunctionPort
call getPort(pd%d_private_data%myservices, 'FunctionPort', generalPort, excpt)
call cast(generalPort, functionPort)
! Obtain a handle to a RandomGeneratorPort
call getPort(pd%d_private_data%myservices, 'RandomGeneratorPort', generalPort, excpt)
call cast(generalPort, randomPort)
! Compute integral
sum = 0
width = upBound - lowBound
do i = 1, count
call getRandomNumber(randomPort, x)
x = lowBound + width*x
call evaluate(functionPort, x, func)
sum = sum + func
enddo
retval = width*sum/count
call deleteRef(generalPort)
call deleteRef(randomPort)
call releasePort(pd%d_private_data%myservices, 'RandomGeneratorPort', excpt)
call deleteRef(functionPort)
call releasePort(pd%d_private_data%myservices, 'FunctionPort', excpt)
endif ! End of implementation
```

Access component's data

Get a Function reference

Get a RandomGenerator reference

Return integral value

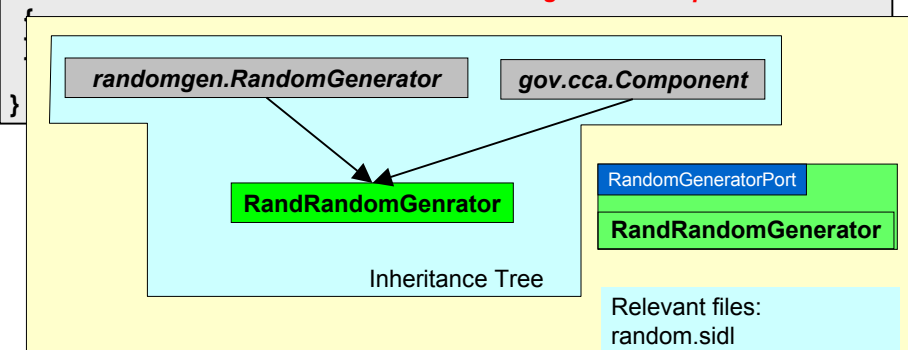
Release ports

## 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 makefile and build dynamic library
  - Makefile
  - libRandom-component-c++.so
4. Create randomgen.cca (Ccaffeine-specific)

## RandomGenerator Port

```
package randomgen version 1.0 {
  interface RandomGenerator extends gov.cca.Port
  {
    double getRandomNumber();
  }
  class RandRandomGenerator implements-all RandomGenerator,
    gov.cca.Component
}
```



## RandRandomGenerator Component: C++ 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 (C++): 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)
}

```

Annotations in the code:

- Save a pointer to the Services object**: Points to `frameworkServices = services;`
- Port name**: Points to the string `"RandomGeneratorPort"`
- Port type**: Points to the string `"randomgen.RandomGenerator"`
- TypeMap reference**: Points to the variable `tm`

27

## 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 makefile and build dynamic library
  - Makefile
  - libFunction-component-c++.so
4. Create functions.cca (Ccaffeine-specific)

28

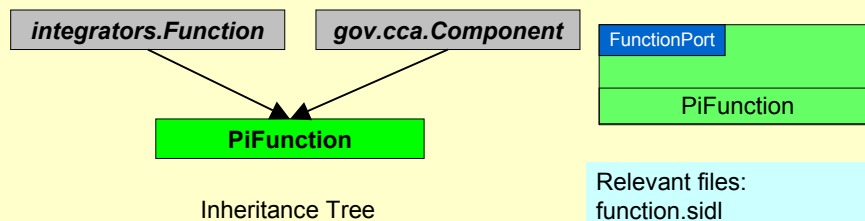
## Function Port

```
package functions version 1.0 {

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

  class PiFunction implements-all Function, gov.cca.Component
  {

```



29

## PiFunction Component (C++): 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

30

## PiFunction Component (C++): 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)
}
  
```

Annotations for the code above:

- Save a pointer to the Services object (points to `frameworkServices = services;`)
- Port name (points to `"FunctionPort"`)
- Port type (points to `"functions.Function"`)
- TypeMap reference (points to `tm`)

31

## 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 makefile and build dynamic library
  - Makefile
  - libDriver-component-c++.so
4. Create driver.cca (Ccaffeine-specific)

32



## Driver SIDL Definition

- Driver implements standard interface

*gov.cca.ports.GoPort*

No additional interfaces defined

```
package tutorial version 1.0 {

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

33

## Driver Component (C++): 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 (C++): 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

## Building components

- Dynamic (shared) libraries
  - For each component or a set of components, build a dynamic library
  - Babel components and Ccaffeine: build a shared library for the implementation (server) and a shared library for the C++ client (if the implementation was in a language other than C++); for example
    - integrator-f90/libIntegrator-component-f90.so
    - integrator-component-f90/libIntegrator-component-f90-c++client.so
  - No linking of libraries for implementations of 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

## Complete Makefile for MonteCarloIntegrator (C++)

```
include ../Makefile.Vars
include babel.make

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$(BABEL_ROOT)/lib -lsidl \
        -L$(CCATUT_SIDL_ROOT)/cca-stuff-c++ -lcca-stuff-c++ \
        -Wl,-rpath,$(CCATUT_SIDL_ROOT)/cca-stuff-c++

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

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

37

## Complete Makefile for MonteCarloIntegrator (C++ client of F90 server)

```
include $(CCATUT_HOME)/Makefile.Vars
include babel.make
COMPONENT_DIR = $(CCATUT_ROOT)/integrator-f90
CXXCLIENT_LIB = libIntegrator-component-f90-c++client.so

CCATUT_ROOT=/home/norris/cca/tutorial/src/sidl
INCLUDES = -I$(BABEL_ROOT)/include -I. -I$(CCATUT_SIDL_ROOT)/cca-stuff-c++
all: $(CXXCLIENT_LIB)

.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)

LIBS = -Wl,-rpath,$(BABEL_ROOT)/lib -L$(BABEL_ROOT)/lib -lsidl \
        -L$(COMPONENT_DIR) -lIntegrator-component-f90 \
        -Wl,-rpath,$(COMPONENT_DIR) \
        -L$(CCATUT_ROOT)/cca-stuff-f90 -lcca-stuff-c++ \
        -Wl,-rpath,$(CCATUT_ROOT)/cca-stuff-c++

$(CXXCLIENT_LIB): $(OBJS)
    gcc -shared $(INCLUDES) $(OBJS) -o $@ $(LIBS)

clean:
    $(RM) *.o $(WRAPPER_LIB)
```

38

## MonteCarloIntegrator (F90): integrators.cca

- Ccaffeine-specific file giving the type of component (e.g., “babel”), name of the dynamic library (classic only) and creation

```
!date=Thu Aug 15 14:53:23 CDT 2002
```

```
!location=
```

```
!componentType=babel
```

```
dummy_libIntegrator-component-f90-c++client.so
```

```
dummy_create_MonteCarloIntegrator integrators.MonteCarloIntegrator
```

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

C wrapper function name

Component name

*Note: This mechanism is expected to change soon*

39

## Running the Example

40