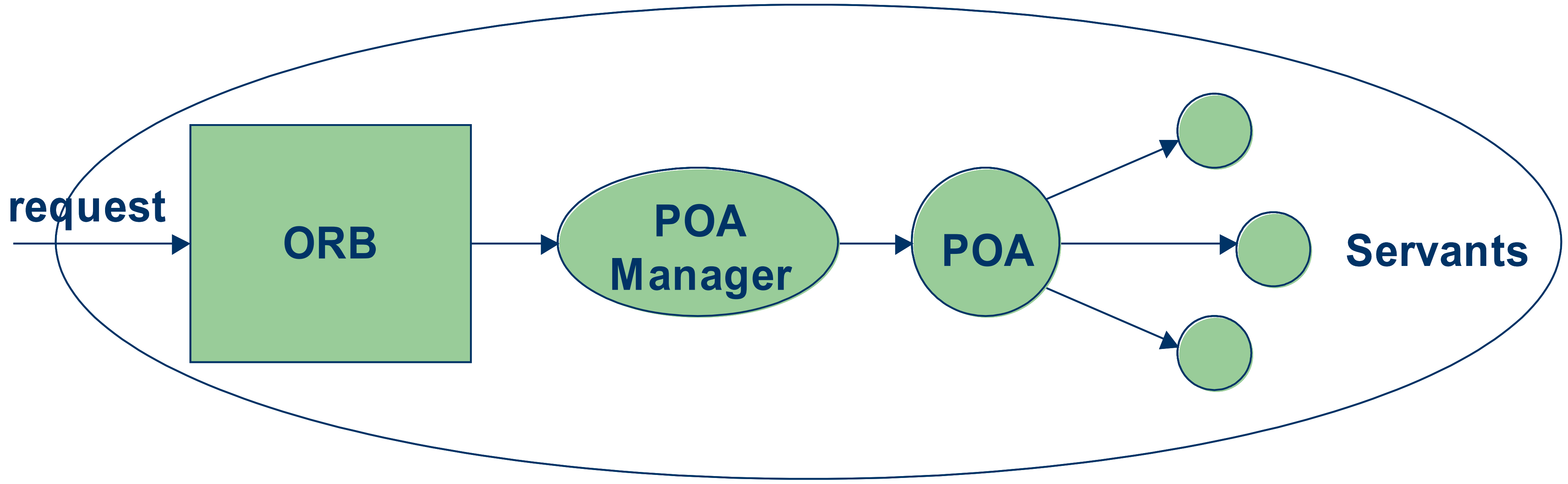
Server-Side C++ Mapping

Summary

- Basic concepts
- Parameter passing
- Exception throwing
- Tie classes

ORB, POA and Serwant



Mapping for Interfaces

- IDL:

```
Interface MyObject {
   long get_value( );
};
```

- Skeleton class header file:

```
class POA_MyObject : public virtual
PortableServer::ServantBase {
public:
   virtual CORBA::Long get_value( ) = 0;
};
```

- The name of the skeleton class is the name of the interface prefixed by **`POA_`**
  - **`MyObject -> POA_MyObject`**
  - **`Mod::MyObject -> POA_Mod::MyObject`**

The Servant Class

```
class MyObject_impl : public virtual
POA_MyObject {
public :
  MyObject_impl(CORBA::Long init_val) :
                     m_value(init_val) { }

  virtual CORBA::Long get_value()
            throw(CORBA::SystemException);
private:
  CORBA::Long m_value;

  MyObject_impl(const MyObject_impl &);
  void operator=(const MyObject_impl &);
}
```

The Servant Class (contd.)

```
CORBA::Long MyObject_impl::get_value()
throw(CORBA::SystemException)
{
  return m_value;
}
```

The Incarnation of The Object

```
// First create a servant instance
MyObject_impl servant(42);

//Next, create a new CORBA object and use our new servant
//to incarnate it
MyObject_var object = servant._this();
```

- Calling `_this()`:
  - Creates of the new CORBA object under Root POA.
  - Registers the servant in the Root POA as the implementation of a new object
  - Creates the reference to the new object
  - Returns the new reference

The Incarnation of The Object (contd.)

```
class POA_MyObject : public virtual
PortableServer::ServantBase {
   public:
      virtual CORBA::Long get_value( ) = 0;
      MyObject_ptr _this( );
   }
```

The `main()` Function

```
int main(int argc, char *argv[]) {
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    CORBA::Object_var obj = orb->resolve_initial_references(”RootPOA”);
    PortableServer::POA_var poa = PortableServer::POA::_narrow(obj);

    PortableServer::POAManager_var mgr = poa->the_POAManager();
    mgr->activate();

    MyObject_impl servant(42);
    MyObject_var object = servant._this();

    CORBA::String_var str = orb->object_to_string(object);
    cout << str << endl;

    orb->run();

    return 0;
}
```

Parameter Passing

- Location independence
  - Client and target object in one process or remote
- Efficiency
  - Especially important when server and client are in the same location

Parameter Passing - Simple Types

```
interface Foo {
long long_op(
         in long      l_in,
         inout long l_inout,
         out long    l_out);
}

CORBA::Long
Foo_impl::Long_op(
 CORBA::Long       l_in,
 CORBA::Long &    l_inout,
 CORBA::Long_out l_out){
   l_inout = l_in * 2;
   l_out = l_in / 2;
   return 99;
}
```

Parameter Passing - Fixed-length Compound Types

```
struct Fls {
  long l_mem;
  double d_mem;
};

interface Foo {
  Fls fls_op(
    in Fls fls_in,
    inout Fls fls_inout,
    out Fls fls_out );
};
```

```
Fls Foo_impl:: fls_op(
   const Fls & fls_in,
   Fls & fls_inout,
   Fls_out fls_out )
throw(CORBA::SystemException)
{
 fun1(fls_in.l_mem);
 fun2(fls_in.d_mem);
 fls_inout.l_mem *=2;
 fls_inout.d_mem /=2;
 fls_out.l_mem=1234;
 fls_out.d_mem = 5.67e8;
 Fls result = { 1234,-.87e6};
 return result;
}
```

Parameter Passing - Fixed-length Arrays

```
typedef double Darr[3];
interface Foo {
  Darr darr_op(
   in Darr darr_in,
   inout Darr darr_inout,
   out Darr darr_out);
};
```

```
Darr_slice *
Foo_impl::darr_op(
    const Darr darr_in,
    Darr darr_inout,
    Darr_out darr_out)
throw(CORBA::SystemException){
// Get the length of the array
const in len =
 sizeof(Darr)/sizeof(*Darr);
int i;
for (i = 0,i<len;i++)
 darr_inout[i] *= i;
for (i = 0,i<len;i++)
 darr_out[i] = i * 3.14;
Darr_slice * result = Darr_alloc();
for (i = 0,i<len;i++)
 result[i] = i * i;
return result;
}
```

Parameter Passing - Strings

```
interface Foo {
 string string_op(
 in string s_in,
 inout string s_inout,
 out string s_out );
};
```

```
char * Foo_impl::string_op(
   const char * s_i,
   char * & s_inout,
   CORBA::String_out s_out)
throw(CORBA::SystemException){
// Use s_in and s_inout (not shown)
const char *s = ”outgoing string”;
if(strlen(s_inout)<strlen(s)){
  CORBA::string_free(s_inout);
  s_inout = CORBA::string_dup(s);
  } else{ strcpy(s_inout, s);}
 s_out = CORBA::string_dup(s);
 return CORBA::string_dup(s);
}
```

Parameter Passing - Variable-length Compond Types and any Type

```
struct Vls {
  long      l_mem;
  string    s_mem;
};
interface Foo {
 Vls vls_op(
  in Vls vls_in,
  inout Vls vls_inout,
  out Vls vls_out );
};
```

```
Vls * Foo_impl::vls_op
throw(CORBA::SystemException)(
 const vls & vls_in,
 Vls & vls_inout,
 Vls_out vls_out){
 vls_inout.l_mem *= 2;
 vls.inout.s_mem = vls_in.s_mem;
 vls_out = new Vls;
 vls_out->l_mem = 1234;
 vls_out->s_mem = CORBA::string_dup("output"
   "string");
 Vls *result = new Vls;
 result->l_mem = vls_in.l_mem;
 result->s_mem = CORBA::string_dup("return"
   "string");
 return result;
}
```

Parameter Passing - Sequences

```
typedef sequence<long> LongSeq;
interface Foo {
 LongSeq seq_op();
};

LongSeq *
Foo_impl::seq_op()
throw(CORBA::SystemException)
{
  LongSeq * result =
                new LongSeq;
  result->length(2);
  result[0]=1234; // wrong
  result[1]=5678; // wrong
  return result;
};
```

```
LongSeq * Foo_impl::seq_op()
throw(CORBA::SystemException) {
  LongSeq * result = new LongSeq;
  result->length(2);
  (*result)[0]=1234; // correct
  (*result)[1]=5678; // correct
  return result;
};
LongSeq * Foo_impl::seq_op()
throw(CORBA::SystemException){
  LongSeq_var result = new LongSeq;
  result->length(2);
  result[0]=1234; // correct
  result[1]=5678; // correct
  return result._retn();
};
```

Parameter Passing - Variable-length Arrays

```
struct Vls {
   long   number;
   string name;
};
typedef Vls varr[3];
interface Foo {
 Varr varr_op(
  in Varr varr_in,
  inout Varr varr_inout,
  out Varr varr_out);
};
```

```
Varr_slice * Foo_impl::varr_op(
 const Varr varr_in,
 Varr_slice * varr_inout,
 Varr_out varr_out)
throw(CORBA::SystemException){
const int len =
        sizeof(Varr)/sizeof(*Varr);
int i;
varr_inout[0]=varr_in[0];
varr_out = Varr_alloc();
const char * brothers[] =
           {"John","Jim","Rich"};
for ( i = 0; i < len; i++){
 varr_out[i].number = i+1;
 varr_out[i].name = brothers[i];}
Varr_slice * result = Varr_alloc();
for ( i = 0; i < len; i++){
 result[i].number = i;
 result[i].name = brothers[i];}
return result;}
```

Parameter Passing - Object References

```
interface Foo {
 Foo  ref_op(
  in Foo ref_in,
  inout Foo ref_inout,
  out Foo ref_out);
 void say_hello();
};
Foo_ptr
Foo_impl::ref_op( Foo_ptr ref_in, Foo_ptr & ref_inout, Foo_out ref_out)
throw(CORBA::SystemException)
{
 if(!CORBA::is_nil(ref_in))
   ref_in->say_hello();
 if(!CORBA::is_nil(ref_inout))
   ref_inout->say_hello();
 CORBA::release(ref_inout);
 ref_inout = _this();
 // Ensure the servant is allocated in the heap!
 Foo_impl * new_servant = new Foo_impl;
 ref_out = new_servant->_this();
 return Foo::_nil();
}
```

Exceptions - IDL

```
pragma prefix "acme.com"
module CCS {
 typedef short TempType;
 interface Thermometer {/* … */};
 interface Thermostat : Thermometer {
  struct BtData {
   TempType    requested;
   TempType    min_permitted;
   TempType    max_permitted;
   string      error_msg;
  };
  exception BadTemp { BtData details; };
  TempType    get_nominal();
  TempType    set_nominal(in TempType new_temp)raises(BadTemp);
 };
};
```

Exceptions - Class Header

```
namespace POA_CCS{
 class Thermostat: public virtual Thermometer {
 public:
 //...
 virtual CCS::TempType set_nominal(CCS::TempType new_temp) = 0;
 //...
 };
}
// note that all exceptions can be thrown
```

Exceptions - Implementation

```
CCS::TempType
Thermostat_impl::set_nominal(CCS::TempType new_temp)
throw(CORBA::SystemException, CCS::Thermostat::BadTemp)
{
  const CCS::TempType MIN_TEMP =50, MAX_TEMP =90;
  if(new_temp < MIN_TEMP || new_temp > MAX_TEMP){
   BtData bt;
   bt.requested = new_temp;
   bt.min_permitted = MIN_TEMP;
   bt.max_permitted = MAX_TEMP;
   bt.error_msg = CORBA::string_dup("temperature out of range");
      throw CCS::Thermostat::BadTemp(bt);
   }
   //...
}
```

Exception Specification

- The ORB and skeleton surround the invocation of servant's method with the **`try ... catch`** block catching all exceptions
- If we add C++ exception specification, throwing exception from outside this specification will end the server program
- All exceptions violating the IDL specification are replaced with **`CORBA::UNKNOWN`**

Throwing System Exceptions

- Makes debugging harder
  - We do not know: problem with ORB or problem with the server implementation
- Exceptions
  - **`CORBA::NO_MEMORY`**
  - **`CORBA::OBJECT_NOT_EXIST`**

Memory Management

- After the exception is thrown, the ORB
  - Frees memory allocated for **`in`** and **`inout`** parameters
  - Ignores **`out`** parameters and return values
  - Passes the exception to the client

Memory Management (contd.)

```
exception SomeException {};
interface SomeObject {
  string string_op() raises(SomeException);
};
struct Vls {
 long l_mem;
 string s_mem;
};
interface Foo {
 Vls op(in SomeObject obj, out Vls vls_out)
  raises(SomeException);
};
```

Memory Management (contd.)

```
Vls* Foo_impl::op(SomeObject_ptr obj, Vls_out vls_out)
throw(CORBA::SystemException, SomeException)
{
 vls_out = 0;
 Vls * result = 0;
 try {
 vls_out = new Vls;
 vls_out->l_mem = 1234;
 vls_out->s_mem = obj->string_op();
 result = new Vls;
 result->l_mem = 5678;
 result->s_mem = obj->string_op();
 }
 catch (...) {
  delete vls_out.ptr();
  delete result;
  throw;
 }
 return result;
}
```

Memory Management (contd.)

```
Vls* Foo_impl::op(SomeObject_ptr obj, Vls_out vls_out)
throw(CORBA::SystemException, SomeException)
{
 Vls_var temp_out = new Vls;
 temp_out->l_mem = 1234;
 temp_out->s_mem = obj->string_op();
 Vls_var result = new Vls;
 result->l_mem = 5678;
 result->s_mem = obj->string_op();
 // no exception occured - return
 vls_out = temp_out._retn();
 return result._retn();
}
```

Tie Servants

```
// Create a C++ class instance to be out tied object
// Assume MyLegacyClass also supports the get_value method
MyLegacy Class * tied_object = new MyLegacyClass;

// Create aninstance of the tie class template, using
// MyLegacyClass as the template parameter. Pass our tied_object
// pointer to set the tied object. The release parameter defalults to true,
// so the tie_servant adopts the tied object
POA_MyObject_tie<MyLegacyClass> tie_servant(tied_object);

// Create our object and register out tie_servant as its servant
MyObject_var my_object = tie_servant._this();

// adaptation of legacy class by template specialization
class MyLegacyClass
{
public:
 unsigned short counter_value();
 // ...
};

template <> CORBA::Long POA_MyObject_tie<MyLegacyClass>::
get_value() throw(CORBA::SystemException)
{
  return _tied_object()->counter_value();
};
```

The `RefCountServantBase` Class

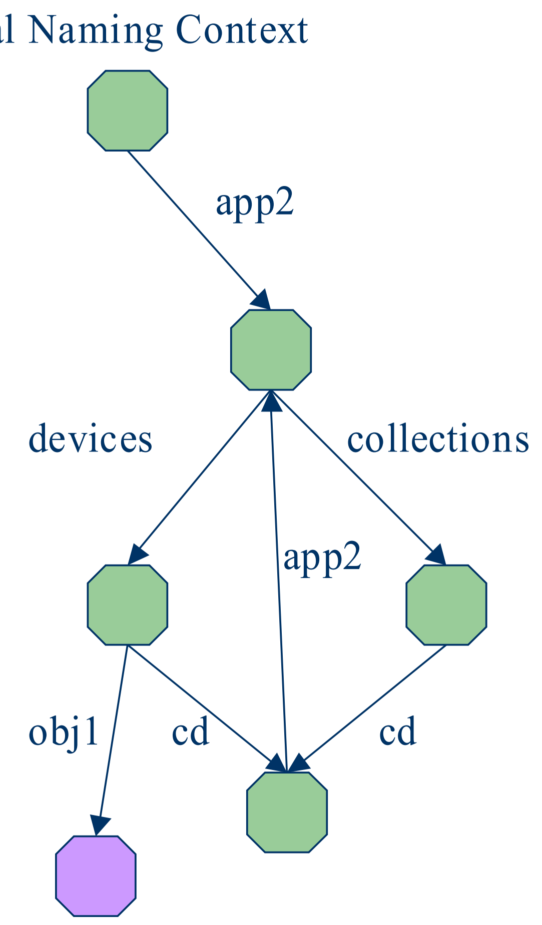
```
namespace PortableServer {
 class RefCountServantBase : public virtual ServantBase {
 public:
  RefCountServantBase() : m_ref_count(1) {}
  virtual void _add_ref();
  virtual void _remove_ref();
 private:
  CORBA::ULong m_ref_count;
  // ...
};

class Echo_i : public POA_Echo,
           public PortableServer::RefCountServantBase
{ /* ... */ };
Echo_i* myecho = new Echo_i();
obj = myecho->_this();
myecho->_remove_ref();
```

- In the present version of the standard, the **`RefCountServantBase`** is empty, retained for the backward compatibility, its functionality is contained in the skeleton class

The OMG Naming Service

```
module CosNaming {
 typedef string Istring;
 struct NameComponent { Istring id; Istring kind; };
 typedef sequence<NameComponent> Name;
 interface NamingContext {
 NamingContext new_context();
 NamingContext bind_new_context(in Name n) raises(
   NotFound, CannotProceed, InvalidName, AlreadyBound);
 void destroy() raises(NotEmpty);
 void bind(in Name n, in Object obj) raises(
   NotFound, CannotProceed, InvalidName, AlreadyBound);
 void bind_context(in Name n, in NamingContext nc) raises(
   NotFound, CannotProceed, InvalidName, AlreadyBound);
 void rebind(in Name n, in Object obj) raises(
   NotFound, CannotProceed, InvalidName, AlreadyBound);
 void rebind_context(in Name n, in NamingContext nc) raises(
   NotFound, CannotProceed, InvalidName, AlreadyBound);
 Object resolve(in Name n) raises(
   NotFound, CannotProceed, InvalidName, AlreadyBound);
 };
};
```


Initial Naming Context
app2
devices
collections
app2
obj1
cd
cd