CptS 464/564
Lecture 20

27th March 2003

CORBA any

A CORBA *any* can contain a value of any CORBA type. Along with the contained value an *any* contains a typecode describing that value. An *any* is not a CORBA object (it doesn't have an IOR, e.g.), but it is typically implemented as a *language* object. In C++ the corresponding language object goes by the name *Any.*

Using *any* and *Any*

To use an *Any* you move values into and out of it using the *insert* and *extract* operators which are spelled < <= and > >= respectively. Examples

```
CORBA::Any a;
a <<= (CORBA.Long) 99; //insert 99 into a
CORBA::Long val;
if (!(a >>= val)) abort(); //extract val from a
a <<= "Hi There"; // deep copy of unbounded string
const char * msg;
if (!(a >> msg)) abort(); // extract msg from a; a retains ownership
```

The last is interesting: extraction does *not* copy the value, so you must *not* free msg, change it, or extract directly into a String_var. But you can initalize a string var with the extracted value:

```
CORBA::String_var msg1 = msg; //ok
```

Some CORBA types may be mapped to the same C++ type leading to problems because the type codes used in *anys* should distinguish between values of different CORBA type. To overcome this problem there are a number of *Any help types* defined in C++. Example:

```
CORBA::Any::a;
CORBA::Boolean b = 0;
CORBA::Char c = 'z';
CORBA::Octet o = 0xff;
```

The Boolean, Char, and Octet may all have the same C++ type. Attempts to directly insert any of b, c, or o into a will result in a compile time error. The correct way:

```
a <<= CORBA::Any::from_boolean(b);
a <<= CORBA::Any::from_char(c);
a <<= CORBA::Any::from_octet(o);
```

On the other hand,

```
a <<= CORBA::Any::from_boolean(c);
```

will likely not produce a compile-time error, but will certainly not do what you want it to. Similarly for extraction:

```
if (a >>= CORBA::Any::to_boolean(b)) {
  // it was a boolean
  ...
} else if (a >>= CORBA::Any::to_char(c)) {
  // it was a char
  ...
} else if (a >>= CORBA::Any::to_octet(o)) {
  // it was an octet
  ...
}
```

Notice the use of the if/else if/else if/... style to extract a value of one of several different types from an *Any. (Hint:* this is the key to success with the sniffer programs for project 3.*)*

User-defined types

The IDL compiler generates insertion and extraction operators for user-defined types such as enumerations, structures, unions and sequences. Enumerations are simple enough:

```
Foo::Color c = Foo::blue;
Foo::Color cout;
a <<= c;
if (!(a>>=cout)) (abort());
```

There are no storage management issues for enumerations – all operations are by-value. Structures, unions and sequences raise issues similar to those for strings. Who owns the storage following an insert or extract operation.

Suppose I have a IDL definition

```
struct myData {
  long L1;
  string msg;
}
```

The compiler will generate overloaded insertion and extraction operators

```
void     operator<<=(CORBA::Any &, const myData &);
void     operator<<=(CORBA::Any &, myData *);
CORBA::Boolean operator>>=(const CORBA::Any &, myData * &)
```

The insertion operators differ in their memory-management affects: inserting by reference (the first case), copies the value and the Any takes ownership of the copy:

```
myData md;
a <<= md; //copy
```

But, inserting a pointer (the second case), just copies the pointer – the Any now owns the pointed-to memory and will deallocate it when it (the Any) goes out of scope. This is called a *consuming insertion.*

```
myData *mdp;
a <<= mdp; // do not delete mdp here!
```

Notice that extraction is always by pointer (the third case). The pointer refers to memory owned by the Any, so it must be treated as read-only and *not* deleted.

Object references

Object references can be inserted and extracted just as any other user-defined type. Likewise you can insert one Any into another. Similar memory management issues arise and I urge you to consult a reference such as Chapter 15 of *Advanced CORBA Programming with C++* before tackling this usage.

You may also want to look at /net/niflab/orbacus/OB-4.1.2/ob/include/OB/Any.h to see what the interface to Any really looks like. Note that there are some truly dangerous operations in the Any interface that should *not be used.* For example,

```
const void * value();
```

allows extraction of a C++ (void *) – i.e. the C++ generic pointer. The problem is that to use this pointer you must cast it to some other pointer type; Such a cast is completely unsafe – there is no checking of the typecode. Another operation allows storing an arbitrary pointer and typecode in an *Any.* Again, there is no checking that the data and typecode agree. Don't do this!

Example of use of the Event Service and Any - Push Supplier

```
CosEventChannelAdmin::EventChannel_var e;
try {
   CORBA::Object_var obj =
           orb -> resolve_initial_references("EventService");
   e = CosEventChannelAdmin::EventChannel::_narrow(obj);
}
catch(const CORBA::ORB::InvalidName&){
   cerr << prog << ": can't resolve 'EventService'" << endl;
   exit(1);
}
if(CORBA::is_nil(e)){
   cerr << prog
        << ": 'EventService' is not an EventChannel object "
        << "reference"
        << endl;
   exit(1);
}

// Get a SupplierAdmin object from the EventChannel.
CosEventChannelAdmin::SupplierAdmin_var supplierAdmin =
      e->for_suppliers();
// Get a ProxyPushConsumer from the SupplierAdmin.
CosEventChannelAdmin::ProxyPushConsumer_var consumer =
      supplierAdmin->obtain_push_consumer();
// Connect to the ProxyPushConsumer as a PushSupplier
// (passing a nil PushSupplier object reference to it because
// we don't care to be notified about disconnects).
    consumer->connect_push_supplier(CosEventComm::PushSupplier::_nil());
// Create an event (just a string in this case).
const CORBA::String_var eventData = CORBA::string_dup("Hello, world.");
while (1) {
  // Insert the event data into an any.
  CORBA::Any any;
  any <<= eventData;
  // Now push the event to the consumer
```

```
  consumer->push(any);
  sleep...
}
```

Example of use of Event Service and Any – push consumer

For the consumer we have to derive a class from the the POA_CosEventCoMM::PushConsumer class. Remember, a push *consumer* is a CORBA object that is passed to the supplier proxy in the event channel and whose *push method* is invoked by the supplier proxy. So if we want something to happen in response to *push* we have to supply a class with an overridden push method. It might look something like this in the header file:

```
class EventConsumer_impl : public virtual POA_CosEventComm::PushConsumer
{
  ...
};
```

and in the implementation:

```
void EventConsumer_impl::push(
  const CORBA::Any & data)
 throw(CORBA::SystemException)
{
  // Extract event data from the any (assume string).
  const char* eventDataStr;
  if (data >>= eventDataStr)
  {
    cout << "push received event "
         << eventDataStr << endl;
  }
  else
  {
    cout << "Received something other than a string" << endl;
  }
}
```

In the main program then we have to instantiate an EventConsumer_impl object and hook it up to the supplier, thusly:

```
EventConsumer_impl* consumerImpl = new EventConsumer_impl();
EventConsumer_var consumer = consumerImpl -> _this();
// Get a ConsumerAdmin object from the EventChannel.
```

```
CosEventChannelAdmin::ConsumerAdmin_var consumerAdmin =
      e->for_consumers();
// Get a ProxyPushSupplier from the ConsumerAdmin.
CosEventChannelAdmin::ProxyPushSupplier_var supplier =
      consumerAdmin->obtain_push_supplier();
// Connect to the ProxyPushSupplier, passing our PushConsumer object
// reference to it.
supplier->connect_push_consumer(consumer);
// Activate the POA via its POAManager.
PortableServer::POAManager_var poa_manager = poa->the_POAManager();
poa_manager->activate();
cout << "Ready to receive events..." << endl;
// Enter the ORB event loop.
orb->run();
```

I will provide a simpler example than the one that comes with ORBacus and pointed to by the project handout. That example uses more complex features of the ORBacus ORB and the ORBacus event service than I (or you) want to get into right now. I'll send email to the class when it is ready, I hope by late afternoon tomorrow.