

CORBA

Implementierung von Client und Server



© J. Heinzlreiter
WS 2003/04

Implementierung des Clients

- Initialisierung und Freigabe des ORBs.
- Mapping von Interfaces.
- Behandlung von Objektreferenzen.
- Verwaltung von Proxies (Reference-Counting).
- Parameterübergabe
 - Regeln für Speicherverwaltung
 - Verwendung von Smart-Pointer-Klassen

Initialisierung und Freigabe des ORBs

- Initialisierung und Übernahme der ORB-Parameter

```
using namespace CORBA;
```

```
int main(int argc, char* argv[]) {  
    ORB_var orb; // Smart-Pointer auf ORB  
    try {  
        orb = ORB_init(argc, argv);  
    } catch (Exception& e) {  
        return 1;  
    }  
}
```

- Freigabe

```
    orb->destroy();  
} // ~ORB_var(): release(orb);
```

Mapping von Interfaces

- IDL-Interfaces

```
interface X {  
    void f();  
};
```

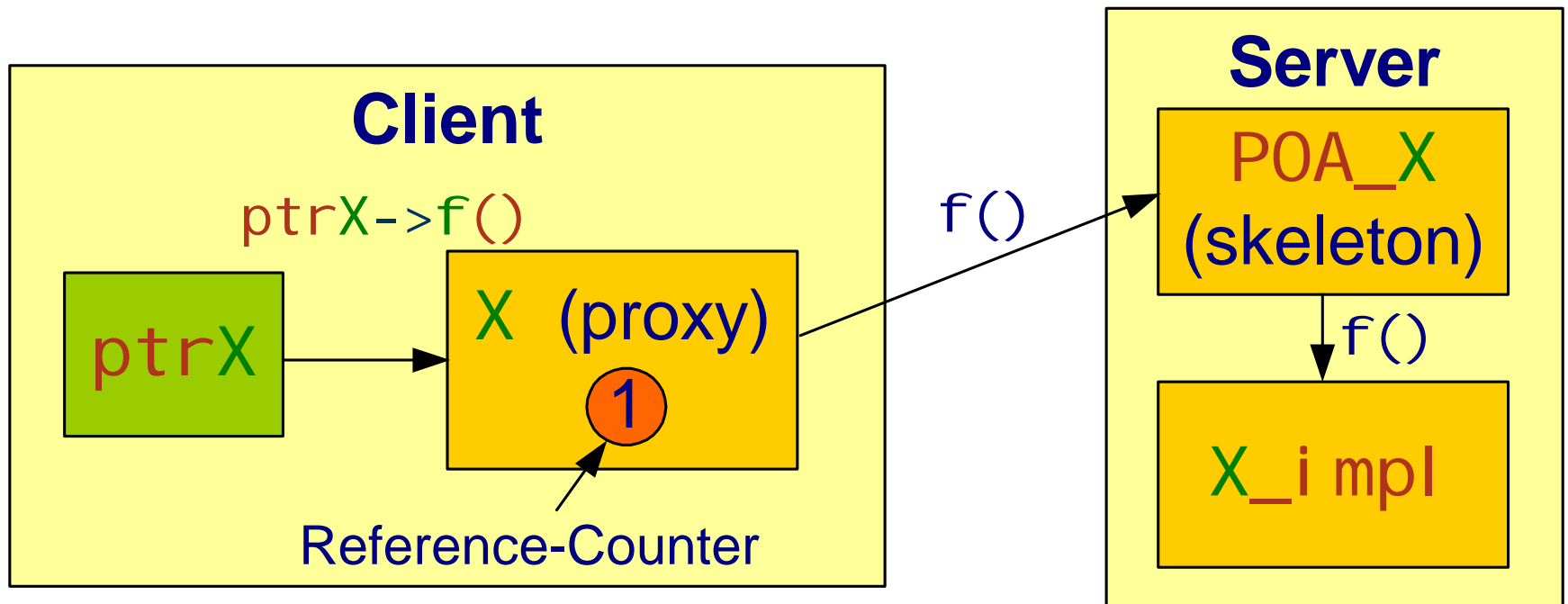
- werden auf C++-Klassen („proxies“) abgebildet.

```
typedef X* X_ptr;  
class X : public virtual Object {  
public:  
    static X_ptr _narrow(Object_ptr obj);  
    static X_ptr _duplicate(X_ptr);  
    static X_ptr _nil();  
    virtual void f();  
};
```

- Zusätzlich wird Smart-Pointer-Klasse **X_var** generiert.

Objektreferenzen - Proxies

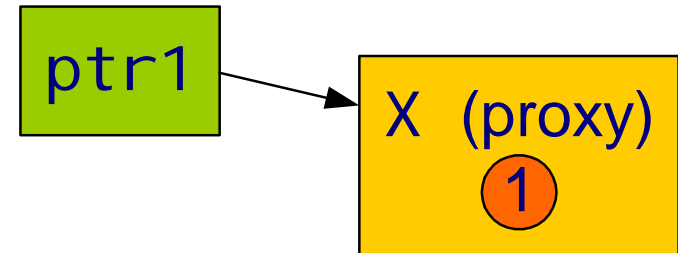
- Proxies werden vom ORB instanziiert:
`X_ptr ptrX = getX();`
- Entfernter Methoden-Aufruf:
`ptrX->f();`



Objektreferenzen – Reference-Counting

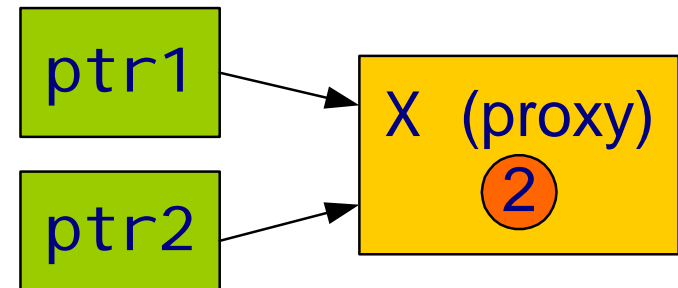
- ORB instanziiert Referenz
(Counter wird auf 1 gesetzt)

```
X_ptr ptr1 = getX();
```



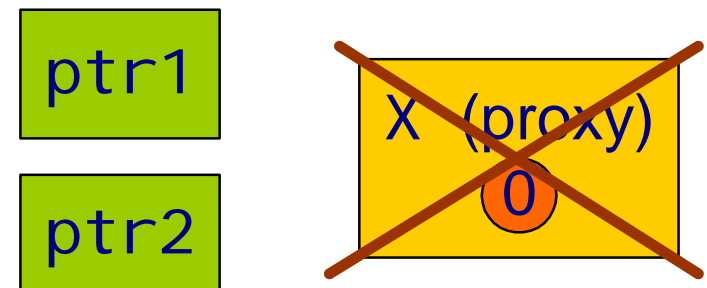
- „Kopieren“ der Referenz:

```
X_ptr ptr2 =  
X::_duplicate(ptr1);
```



- Freigabe der Referenzen:

```
CORBA::release(ptr1);  
CORBA::release(ptr2);
```



Smart-Pointer-Klasse für Referenzen (1)

```
class X_var {
private:
    X_ptr _ptr; // Pointer auf verwaltete Referenz
public:
    // Initialisierung mit nil-Referenz.
    X_var();
    // Besitz an Referenz übernehmen.
    X_var(X_ptr p);
    // Copy-Konstr.: Referenz kopieren (X::duplicate()).
    X_var(const X_var& pv);
    // Destruktor: Referenz freigeben (CORBA::release()).
    ~X_var();
    // Selektionsoperator: Delegation an _ptr.
    X_ptr& operator->() const;
```

Smart-Pointer-Klasse für Referenzen (2)

```
class X_var {  
    ...  
    // Zuweisungsoperatoren: Besitz übernehmen bzw. kopieren.  
    X_var& operator= (X_ptr p);  
    X_var& operator= (const X_var& pv);  
    // Konversionsoperator.  
    operator X_ptr& ();  
    // Funktionen zum direkten Zugriff auf _ptr. Für Compiler,  
    // die Konversionsoperatoren falsch implementieren.  
    X_ptr in() const;  
    X_ptr& inout();  
    X_ptr& out();  
    // Übergibt Besitz an Referenz an den Rufer.  
    X_ptr _retn();  
}
```


Verwendung der `_var` Klasse

- Zuweisung einer Referenz
`Object_var obj = orb->string_to_object(...);`
- Überprüfung auf nil-Referenz (Konversionsoperator)
`if (is_nil(obj)) { ... }`
- Downcast: `_narrow` inkrementiert den Ref-Counter
`X_var ptr1 = X::_narrow(obj);`
- Kopieren der Objektreferenz
`X_var ptr2 = ptr1;`
- Zugriff auf Funktion von Interface `X`
`ptr1->f();`
- Ref-Counter wird von `~X_var()` dekrementiert.

Austausch von Referenzen

- Konvertierung in Strings

```
interface ORB {  
    string object_to_string(in Object obj);  
    Object string_to_object(in string str);  
};
```

- Naming Service (in Modul CosNaming)

```
interface NamingContext {  
    (re)bind(in Name n, in Object o);  
    Object resolve(in Name n);  
};
```

- Methodenaufruf

```
interface X {  
    Y getY(); // Y ist ebenfalls ein IDL-Interface  
};
```

Regeln für Parameterübergabe

Die Art der Parameterübergabe hängt vom „directional attribute“ und vom Parametertyp ab:

IDL	in	inout	out	return
<i>simple</i>	<i>simple</i>	<i>simple</i> &	<i>simple</i> &	<i>simple</i>
string	const char*	char *&	char *&	char *
X	X_ptr	X_ptr &	X_ptr &	X_ptr
sequ	const sequ &	sequ &	sequ *&	sequ *
struct, fixed	const struct &	struct &	struct &	struct
struct, var.	const struct &	struct &	struct *&	struct *
array, fixed	const array	array_slice *	array_slice *	array_slice *
array, var.	const array	array_slice *	array_slice *&	array_slice *
Stack oder Heap		Heap: Freigabe	Heap: Allokierung und Freigabe	

Parameter mit fixer Länge

- IDL:

```
interface X {  
    long f(in    long lIn,  
          inout long lInOut,  
          out   long lOut);  
};
```
- C++:

```
typedef Long& Long_out;  
CORBA::Long f(CORBA::Long    lIn,  
              CORBA::Long&    lInOut,  
              CORBA::Long_out lOut);
```
- Verwendung:

```
X_var x = ...; // Hole Referenz auf Objekt  
CORBA::Long in    = 5;  
CORBA::Long inOut = 9;  
CORBA::Long out, ret;  
ret = x->f(in, inOut, out);  
cout << inOut << out << ret << endl;
```

Übergabe von Stringparametern

- IDL:

```
string f(in string sIn,           // interface X
         inout string sInOut,
         out string sOut);
```
- C++:

```
char* f(const char* sIn,
        char*& sInOut,
        String_out sOut);
```
- Verwendung:

```
X_var x = ...; // Hole Referenz auf Objekt
// i nOut muss mit dynamisch allokiertem Wert initialisiert werden,
String_var i nOut = string_dup("yyy");
String_var out, ret;
ret = x->f("xxx", i nOut, out);
cout << out << i nOut << ret << endl;
// String_out sorgt dafür, dass out vor zweitem Aufruf freigegeben wird.
ret = x->f("xxx", i nOut, out);
```

Parameter mit variabler Länge

- IDL:

```
typedef sequence<long> LongSeq;
LongSeq f(in LongSeq sIn,           // interface X
          inout LongSeq sInOut,
          out LongSeq sOut);
```
- C++:

```
typedef LongSeq*& LongSeq_out;
LongSeq* f(const LongSeq& sIn,
           LongSeq& sInOut,
           LongSeq_out sOut);
```
- Verwendung:

```
X_var x = ...; // Hole Referenz auf Objekt
LongSeq_var in = new LongSeq;
LongSeq_var inOut = new LongSeq;
LongSeq_var out, ret;
in->length(2); in[0] = 111; in[1] = 222;
inOut->length(1); inOut[0] = 333;
ret = x->f(in, inOut, out);
```

Übergabe von Objektreferenzen

- IDL:

```
interface Article { ... }  
Article f(in Article aIn,           // interface X  
          inout Article aInOut,  
          out Article aOut);
```
- C++:

```
Article_ptr f(Article_ptr aIn,  
              Article_ptr& aInOut,  
              Article_out aOut);
```
- Verwendung:

```
X_var x = ...; // Hole Referenz auf Objekt  
Article_var aIn = findArticleById("A001");  
Article_var aInOut = findArticleById("A002");  
Article_var aRet = x->f(aIn, aInOut, aOut);  
aInOut->setStock(100); // Entfernter Methodenaufruf  
aOut->setPrice(1000.0);
```

Implementierung des Servers

- Initialisierung und Freigabe des ORBs.
- Implementierung von Servants.
- Erzeugung Objektreferenzen.
- Parameterübergabe
 - Regeln für Speicherverwaltung
 - Verwendung der Smart-Pointer-Klassen

Initialisierung und Freigabe des ORBs (1)

```
using namespace CORBA;
```

- **Initialisierung des ORBs**

```
ORB_var orb;  
orb = ORB_init(argc, argv);
```

- **Referenz auf Object-Adapter (POA) holen**

```
Object_var poaObj =  
    orb->resolve_initial_references("RootPOA");  
PortableServer::POA_var rootPOA =  
    PortableServer::POA::_narrow(poaObj);
```

- **Aktivierung des POA-Managers**

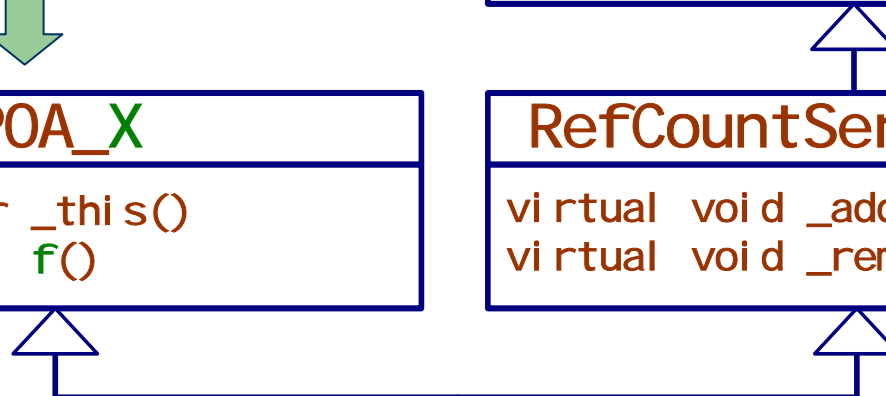
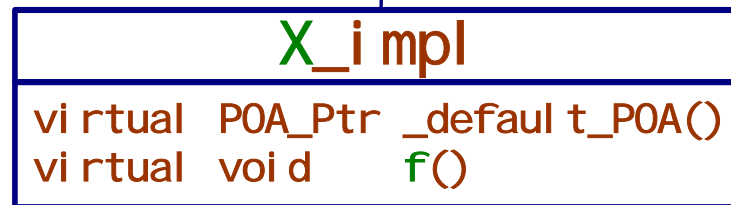
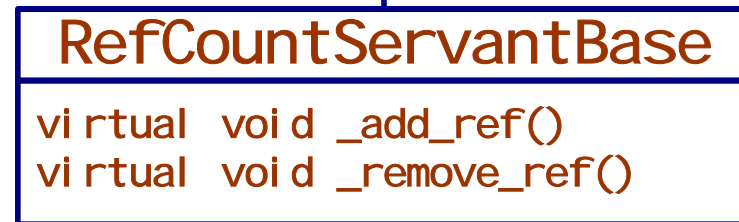
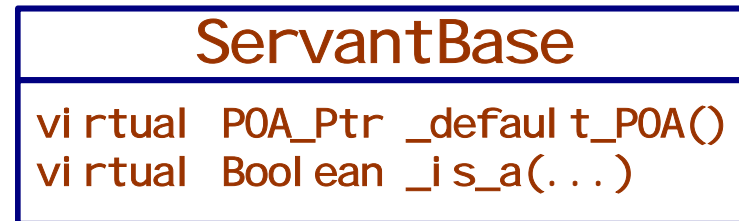
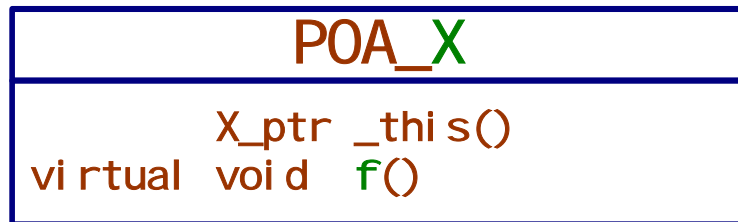
```
PortableServer::POAManager_var manager =  
    rootPOA->the_POAManager();  
manager->activate();
```

Initialisierung und Freigabe des ORBs (2)

- Instanziierung und Registrierung der Servants
`X_impl servantX(rootPOA);`
`X_var refX = servantX._this();`
- Starten der Dispatch-Schleife des ORBs
`orb->run();`
- Terminieren des ORBs
`orb->shutdown(wait_for_completion);`
- Freigabe des ORBs
`orb->destroy();`

Implementierung des Servants (1)

```
interface X {  
    void f();  
};
```



Implementierung des Servants (2)

- **ServantBase**: gemeinsame Funktionalität aller Servants.
 - Enthält Implementierungen von Methoden von CORBA::Object (z.B. `_is_a()`, `_non_existent()`).
 - `_default_POA()` liefert Referenz auf **RootPOA**.
- **RefServantBase**: Reference-Counting für Servants.
 - abgeleitete Servants müssen am Heap allokiert werden.
 - Freigabe mit `_remove_ref()` (und nicht mit `delete`).
- **POA_X**: Skeleton-Code
 - Wird vom IDL-Compiler für jedes IDL-Interface generiert
 - Enthält für jede Operation des IDL-Interfaces eine virtuelle Methode.
 - `_this()`: Aktivierung des Servants und Generierung einer Objektreferenz.
- **X_impl**: Servant
 - Implementierung aller Methoden des Skeletons.
 - `_default_POA()` überschreiben, falls POA mit spezifischen Policies benötigt wird.

Parameterübergabe

- Regeln für Parameterübergabe sind das Gegenstück zu den entsprechenden Regeln auf Client-Seite.
- Wenn Client einen Parameter freigeben muss, muss der Server diesen allokiieren.
- *Parameter mit fixer Länge* werden „by value“ oder „by reference“ übergeben.
- *i n-Parameter* dürfen nicht überschrieben oder freigegeben werden.
- *out-Parameter* und *Rückgabewerte mit variabler Länge* werden vom Server dynamisch allokiert.
- *i nout-Parameter mit variabler Länge* werden vom Client allokiert. Der Server kann diese überschreiben. Strings können auch freigegeben und neu allokiert werden.

Parameter mit fixer Länge

- IDL:

```
interface X {  
    long f(in    long lIn,  
          inout long lInOut,  
          out   long lOut);  
};
```
- Skeleton:

```
CORBA::Long f(CORBA::Long    lIn,  
              CORBA::Long&    lInOut,  
              CORBA::Long_out lOut) = 0;
```
- Implementierung:

```
CORBA::Long X_impl::f(...) {  
    cout << lIn << endl;  
    lInOut += 1;  
    lOut = 0;  
    return 1;  
}
```

Parameter mit variabler Länge

- IDL:

```
typedef sequence<long> LongSeq;
LongSeq f(in LongSeq sln,           // interface X
          inout LongSeq slnOut,
          out LongSeq sOut);
```
- Skeleton:

```
typedef LongSeq*& LongSeq_out;
LongSeq* f(const LongSeq& sln,
           LongSeq& slnOut,
           LongSeq_out sOut) = 0;
```
- Implementierung:

```
LongSeq* X_impl::f(...) {
    slnOut.length(sln.length());
    for (int i=0; i<sln.length(); i++) slnOut[i] = sln[i];
    sOut = new LongSeq;
    sOut->length(1); sOut[0] = 5;
    LongSeq_var sRet = new LongSeq;
    sRet->length(2); sRet[0] = 1; sRet[1] = 2;
    return sRet._retn();
}
```

Übergabe von Stringparametern

- IDL:

```
string f(in string sIn, // interface X
         inout string sInOut,
         out string sOut);
```
- Skeleton:

```
char* f(const char* sIn,
        char*& sInOut,
        String_out sOut) = 0;
```
- Implementierung:

```
char* X_impl::f(...) {
    string_free(sInOut);
    sInOut = string_dup(sIn);
    sOut = string_dup("OutString");
    return string_dup("ReturnString");
}
```


Übergabe von Objektreferenzen

- IDL:

```
interface Article { ... }
Article f(in Article aIn, // interface X
         inout Article aInOut,
         out Article aOut);
```
- Skeleton:

```
Article_ptr f(Article_ptr aIn,
              Article_ptr& aInOut,
              Article_out aOut) = 0;
```
- Implementierung:

```
Article_ptr X_impl::f(...) {
    if (!CORBA::is_nil(aIn)) aIn->setPrice(500.0);
    CORBA::release(aInOut);
    aInOut = Article::_duplicate(aIn);
    sOut = _this();
    Article_var aRet = stock->findArticleById("A001");
    return aRet->_retn();
}
```