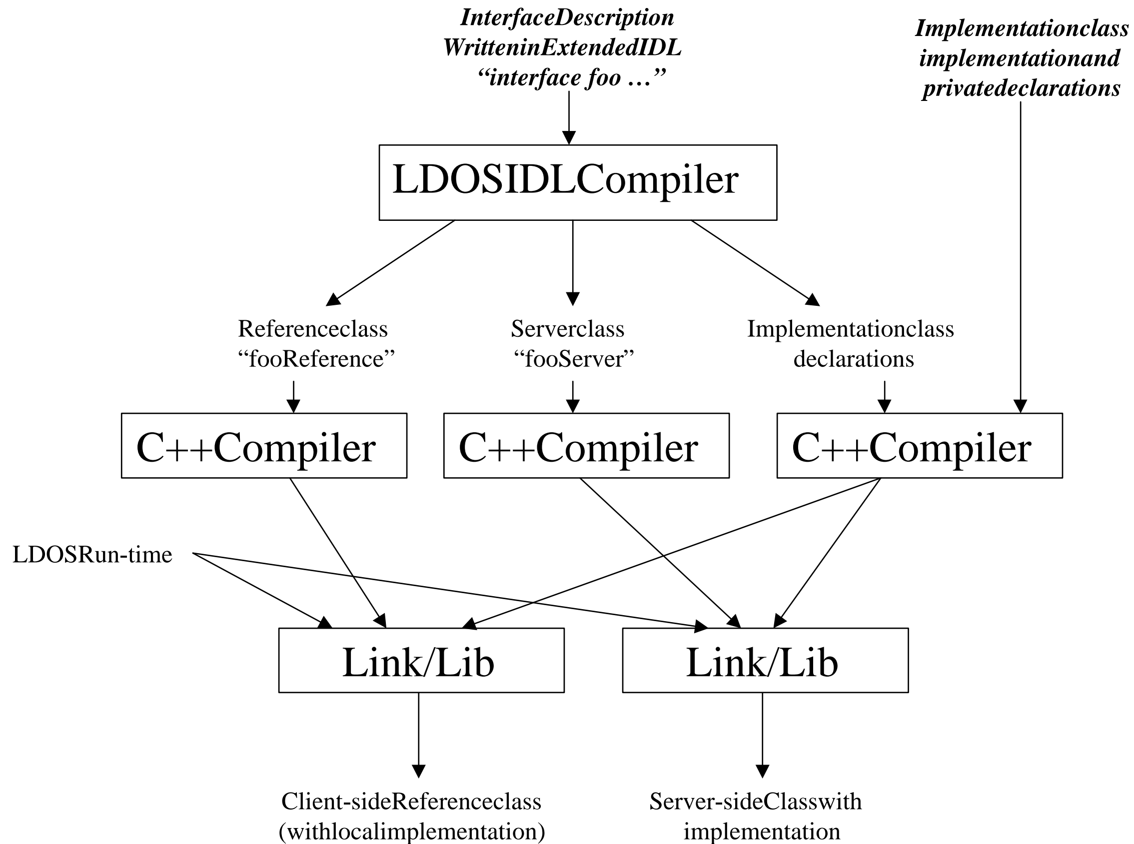
Programmer'sviewofLDOSsystem

PeterA.Dinda

LDOS(LightweightDistributedObjectSystem)letsyoubuil ddistributedobjectswithoutdealingwitht he detailsofcommunication. Theideaisthatyouwriteadescriptionofyour class'sinterfaceint heCORBA IDL(InterfaceDefinitionLanguage) anda local implementationofits class.TheLDOSI DLCompiler generatesthenecessary glue codeforyourimplementationto usetheLDOSrun -timesystem .Theend resultisa "server"classwhichwrapsyourimp lementationandmakesitpossibletobecalledfrom anywhereonthenetwork,anda "reference"classthroughwhichthosecallscanbemade. Thereference classisa "smartpointer "inthatitmakestheobjectbeingcalledappeartobelocal,whetheritis ornot.



Example:AnAccumulatorC lass

Supposewewanttoimplementaclassthathasasumwhichcanbeaddedtoandread.First,wedefinethe interfaceinIDL:

```
interface Accumulator {
   void          Reset();
   void          Add(in unsigned long x);
   unsigned long Get();
};
```

IfyouareunfamiliarwithCORBAIDL,notethatthisissimilartoaC++classdefinition,exceptthat "class"isreplacedwith "interface",argumentshavedirection("in"),andnostateisdeclared.Atthispoint, wehavepureIDL ,whichcanbecompiledbyanyIDLcompiler. WecompilethefilewiththeLDOSIDL

compiler, which generates declarations and definitions for the classes AccumulatorServer and AccumulatorReference, and a macro for declaring the class AccumulatorImpl. The only class we need concern ourselves with now is AccumulatorImpl, which is the actual implementation of the class. First, we shall declare our implementation:

```
DECL_CLASS_Accumulator() {
DECL_IDL_GENERATED_Accumulator()
private:
      unsigned sum;
};
```

The DECL macros generate the appropriate declarations. There is no need to declare anything in the IDL file, which why we only add the declaration for the internal state, the sum. Next, we define the member functions:

```
void AccumulatorImpl::Reset()
{
      sum=0;
}

void AccumulatorImpl::Add(unsigned x)
{
      sum+=x;
}

unsigned AccumulatorImpl::Get()
{
      return sum;
}
```

That concludes the programmer's task for making Accumulator a class that LDOS can use. Notice the similarity to writing a header file (the IDL file) and an implementation file. To use the class locally, simply instantiate a server, and a reference to it, and make calls via the reference:

```
AccumulatorServer acc_s;
AccumulatorReference acc(&acc_s);

acc.Reset();
acc.Add(5);
unsigned x=acc.Get();
```

Notice that the AccumulatorReference acts as a pointer to the implementation. However, it is a very smart pointer, as we shall see.

Making A Remotely Accessible Accumulator

To make the accumulator accessible from a remote machine, we must add a network interface to it. To do this, we use an interface option, which is an LDOS extension to IDL:

```
interface Accumulator [TCP] {
```

In this case, we have endowed the Accumulator with a TCP/IP interface. Similarly, we can create UDP, HTTP (with HTML forms), and IP Multicast interfaces. After recompiling, we can instantiate an accumulator object in this way:

```
AccumulatorServer acc_s(portnum);
```

where portnumisth eportnumberatwhichtheserver willreside.Theobjectisinternally multithreaded andis immediately availablebothlocallyandremotely .

Areferencetothisobjectcanbeinstantiatedandusedonanymachineonthenetwork:

```
AccumulatorReference acc(hostname,port,objectid);

acc.Reset();
acc.Add(5);
unsigned x=acc.Get();
```

Notice thatthisdiffersfr omusingalocalserveronlyinhowthereferenceisinstantiated.Insteadofalocal pointer,ahostname,port,andobjectidentifieraresupplied.Thehostname andportidentifyanetwork interface,andtheobjectididentifiesanobjectservedbytha t interface.ByusingtheTCPinterfaceoption in the IDL,wehave requestedaprivateinterface.Foraprivateinterface,theobjectidisignoredandshould bezero.

NannyedObjects

Anetworkinterface canbesharedbymanyobjectsbyusinganObjectN anny.WecanremovetheTCP interfaceoptionfromtheaccumulatorIDL(ie,use "interfaceAccumulator{ "insteadof "interface Accumulator[TCP]{ ") andinstantiate two AccumulatorServers thatshareasinglenetworkinterfacein thisway :

```
AccumulatorServer acc_s1, acc_s2;  // Two accumulators, no server
ObjectNanny on(tcpportnum, udpportnum, httpportnum);

ObjectID id1=on.AddObject(&acc_s1);
ObjectID id2=on.AddObject(&acc_s2);
```

TheObjectNanny suppliesthe networkinterfaceandroutesrequeststotheapp ropriateobjectbasedonthe request'sobjectid.

DescribingObjectStateinIDL

Ifthe stateoftheobject andwhatstateeachmethoduses isdeclared intheIDLfile, LDOScanprovide manyadditionalservicesthanjustRPC.Statedeclar ationsareanLD OSex tensiontoCORBAIDL.For thea ccumulator,wecan rewritetheIDLthus:

```
interface Accumulator {
   state {
      unsigned long sum;
   } all_state;
   void          Reset() writes {all_state};
   void          Add(in unsigned long x) modifies {all_state};
   unsigned long Get() reads(all_state};
};
```

andwecanthenremovethestatedeclarationfromtheimplementation:

```
DECL_CLASS_Accumulator() {
DECL_IDL_GENERATED_Accumulator()
};
```

Thestateistheunionofthecontentsofallthestatedeclarations. Bypartitioning theobjectstateinto severalstatedeclarations,theprogrammercanspecifyatanarbitrarilyfinegranularityhowheaccessthe stateofhisobject.

Serializability,Mobility,andPersistence

An classwhosestateisdeclaredinIDLcanreque stthat itbemadeserializable:

```
interface Accumulator [Serializable] {
```

Thismeans thatinstancesofthisclasscansave themselvesto andrestorethemselves fromLDOSstreams:

```
AccumulatorServer x,y;

x.Serialize(somestream);        // x saves itself
y.UnSerialize(somestream);   // y restored from x
```

LDOSstreamsincludenetworkstreams,filestreams,andmemorystreams(buffers). Serializable objects canalsobepersistent(existlongerthantheprogramthatcreatedthem),andmobile (abletomovefromsite to site).Thesefeaturesareexpressedviainterfaceoptions:

```
interface Accumulator [Serializable, Persistent, Mobile] {
```

Notethatper sistenceand mobilityarenotcurrentlyimplemented.However,theserializationinterfaceand streamscanbeusedforthi spurpose currently.

ReplicableObjects

Anobjectwithnostatecanbereplicated.Replicabilityisaninterfaceoption:

```
interface Adder [Replicable] {
      unsigned long Add(unsigned long x, unsigned long y);
};
```

TotreatmultipleAdderServersasasingle ,replicatedobject,agroupiscreated:

```
GroupID gid=theGM->NewGroup();
```

andAdderServ ersjoinit:

```
AdderServer add_s(tcpport);
ObjectAddress add_oa={ToIPAddress(gethostname()), tcpport, udpport,
                      httpport, objectid};
theGM->AddAddress(gid,&add_oa);
```

Tocreateareferencetoareplicatedobject,weexecute:

```
AdderReference ar(theGM,gid); //identifythegroupmanager,andthegrouponthatmanager.
```

Now,whenacallismadeviathereference ,themappingofthatcalltoaspecificinstance isdet erminedby a “memberselector ”.Theprogrammercaninstallhisownmemberselector eitheratcompiletimeby subclassingAdderReference andoverridingAdderReference::MemberSelector()oratruntimeby usingthe InstallExternalMemberSelectorcall:

```
unsigned mem_select(GroupMangerReference *gm,  unsigned max_choice);

ar.InstallExternalMemberSelector(mem_select);
```

DistributedObjects

Distributedo bjectsareobjectswithstatewhichmayhavemorethanoneinstance.Inordertomakethis possible,relevant componentsofobjectstatearemoved bytheLDOSrun -time tosatisfy calls.Toc reate suchanobject,we relyonseveralID Lextensions.W e tagitsinterface spec withtheDistributedopti on, specifyitsstate,anddefine howthemethodsintheinterfac eusetheobject 'sstate.Hereisa simple example of twoaccumulatorswrapped inasingleobject:

```
interface TwoAccumulators [Distributed] {
   state {
      unsigned long acc1;
   } acc1;

   state {
      unsigned long acc2;
   } acc2;

   void          Reset1() writes {acc1};
   void          Add1(in unsigned long x) modifies {acc1};
   unsigned long Get1() reads(acc1};

   void          Reset2() writes {acc2};
   void          Add2(in unsigned long x) modifies {acc2};
   unsigned long Get2() reads(acc2};
};
```

Wewritetheimplementat ionofTwoAccumulatorsinpreciselythesamewayasforthesingleaccumulator exampleabove. Tocreateaninstance,wecreateagroup,justlikewithareplicatedobject:

```
GroupID gid=theGM->NewGroup();
```

andTwoAccumulator sServersjoinit:

```
TwoAccumulatorsServer acc2_s(tcpport);
ObjectAddress acc2_oa={ToIPAddress(gethostname()), tcpport, udpport,
                      httpport, objectid};
theGM->AddAddress(gid,&acc2_oa);
```

Wemustalsoinitiallyassignthestatetooneinstance:

theGM->SetAllStateOwnership(gid,membernumber);

Tocreateareferencetoareplicatedobject,weexecute:

```
TwoAccumulatorsReference acc(theGM,gid);
```

justlikeforareplicatedobject,and,similarly,wecaninstallourownmemberselectoraswell.TheLDOS run-time,andthecodetheIDLcomp ilergeneratedconspiretoassurethatallstatenecessarytocompletea callisaccessible,ineffectperformingli keasmalltimedistributedsharedmemorysystem.