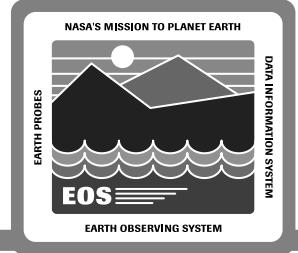


Code Walkthrough

Frank DeLuca

Developers Workshop
30 May 1995

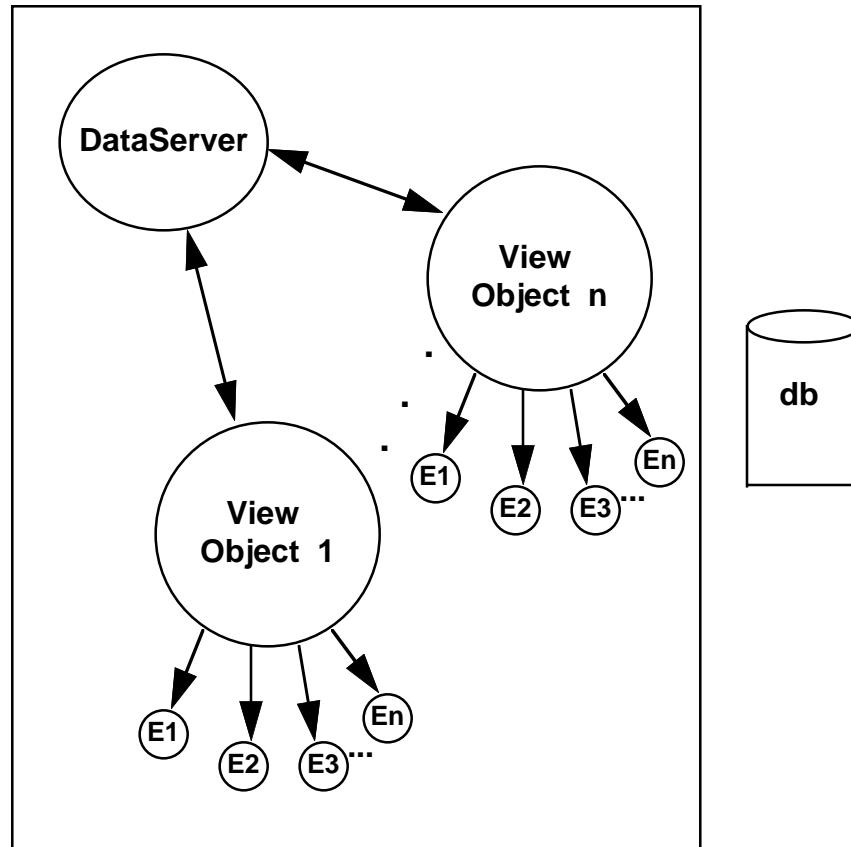
Example Problem Definition



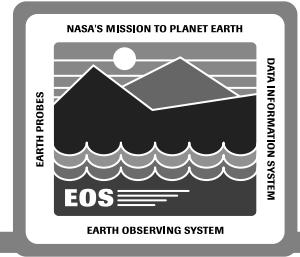
Client Application (main)

- Connect to a DataServer (view factory)
- A DataServer is up and running.
- Create a View , a container object.
- Populate the View with Esdt objects through a search criteria
- Bind to an Esdt object and obtain a service list.
- Invoke methods on the Esdt object.
- Destroy the objects when done.

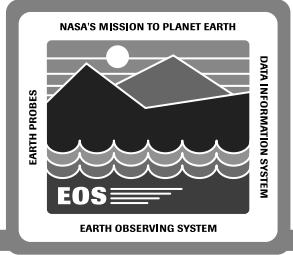
Server Application



IDL - DataServer Interface (User Defined)



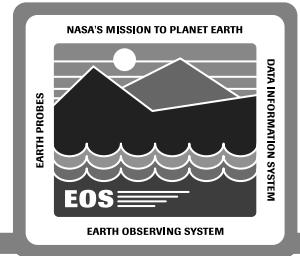
```
uuid(bab991ea-7c04-11ce-b0cf-080009701906),  
version(1.0)  
]  
interface DataServer  
{  
import "ObjRef.idl";  
  
DCEObjRefT *createView(  
    [in] handle_t h  
);  
  
void destroyView(  
    [in] handle_t h,  
    [in] uuid_t objUuid  
);  
}
```



IDL - rdacl Interface

```
[  
    uuid(47B33331-8000-0000-0D00-01DC6C000000)  
]  
  
interface rdaclif {  
    import "dce/aclbase.idl";  
  
    ... some sec_acl_typedefs...  
  
    void rdacl_lookup (  
        [in]     handle_t             h,  
        [in]     sec_acl_component_name_t component_name,  
        [in, ref]  uuid_t            *manager_type,  
        [in]     sec_acl_type_t       sec_acl_type,  
        [out]    sec_acl_result_t     *result  
    );  
  
    void rdacl_replace (...);  
    void rdacl_get_access (...);  
    boolean32 rdacl_test_access (...);  
    boolean32 rdacl_test_access_on_behalf (...);  
    void rdacl_get_manager_types (...);  
    ...  
}
```

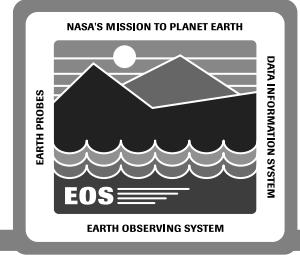
rdacl Manager Class



```
class rdaclif_0_0_Mgr : public rdaclif_0_0_ABS {
public:
    ...
    virtual void rdacl_lookup(
        /* [in] */ sec_acl_component_name_t component_name,
        /* [in] */ uuid_t *manager_type,
        /* [in] */ sec_acl_type_t sec_acl_type,
        /* [out] */ sec_acl_result_t *result
    );
    virtual void rdacl_replace(...);
    virtual void rdacl_get_access(...);
    virtual boolean32 rdacl_test_access(...);
    virtual boolean32 rdacl_test_access_on_behalf(...);
    virtual void rdacl_get_manager_types(...);
    virtual void rdacl_get_printstring(...);
    virtual void rdacl_get_referral(...);
    virtual void rdacl_get_mgr_types_semantics(...);

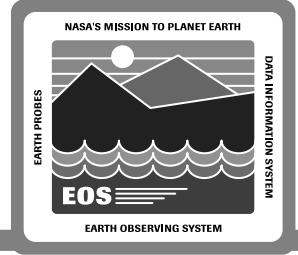
};
```

The Server Termination Thread



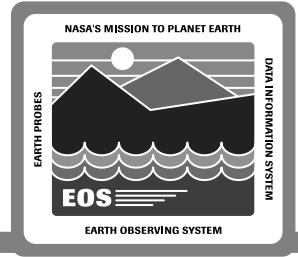
- **GSO provides ServerCleanup which**
 - listens for a termination signal and then
 - » cleans up anything registered with the DCE runtime
 - » cleans up anything entered into the endpoint map
 - » cleans up anything entered into the CDS
- **Frees you from DCE-style cleanup**
- **Runs in a separate thread**

Establishing a Server Identity



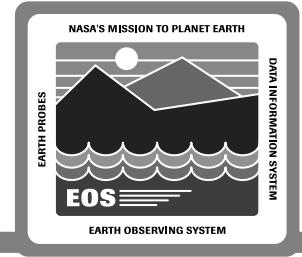
- **Server should have a unique identity (other than default)**
 - For manual start-up without logging into DCE
 - For automatic start-up at boot time
- **OODCE provides classes for acquiring a unique identity**
 - **DCELoginContext (abstract)** and **DCEStdLoginContext**
 - » Automatically refreshes login context
 - **DCEPassword (abstract)** and **DCEMemPassword**

Establishing a Server Identity (cont'd)



- **Problem: DCEMemPassword**
 - still need to type in password
- **Solution: CSS-provided DCEFilePassword**
 - gets the secret key from a keytab file

Server: Starting Termination Thread and Establishing Identity



```
// server main

int main(int argc, char **argv)
{
    ...
    // termination thread

    DCEPthread *clean = new DCEPthread( theServer->ServerCleanup,
                                         NULL);

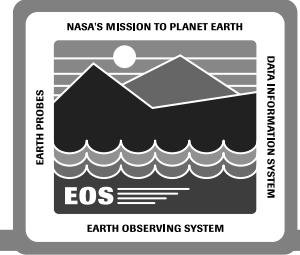
    // establishing server's own identity

    DCEFilePassword pswd(princName,
                         keyFile);

    DCEStdLoginContext loginCtx(&pswd);

    ...
}
```

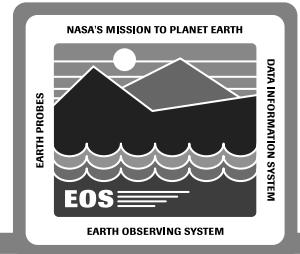
DCEFilePassword Class



- Takes the principal and keytab file as arguments

```
class DCEFilePassword : public DCEPassword {  
public:  
    ...  
    DCEFilePassword(char *pname, char *localKeyFile);  
  
    virtual sec_passwd_rec_t* GetPassword();  
    ...  
};
```

DCEFilePassword Class (cont'd)



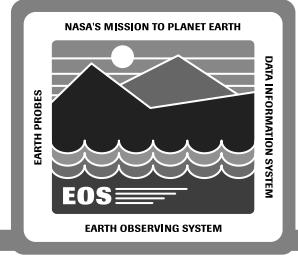
- Overrides **GetPassword** to get secret key from the keytab file

```
sec_passwd_rec_t *DCEFilePassword :: GetPassword()
{
    ...
    void *key;
    unsigned32 status;

    sec_key_mgmt_get_key( rpc_c_authn_dce_secret,
                          myKeyFile,
                          _principal_name,
                          0,
                          &key,
                          &status);

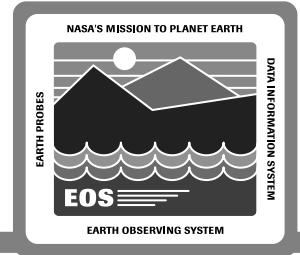
    ...
    return (sec_passwd_rec_t*)key;
}
```

Registering Authentication Information



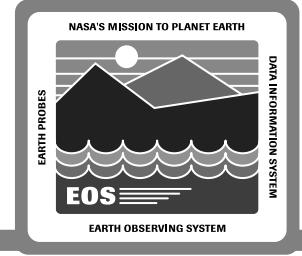
- The server registers service /principal pairs with RPC runtime
- The client decides whether or not to use authenticated RPC
 - Does not HAVE to use authentication just because the server has registered
 - If asking for an authentication service that is not registered, RPC request is rejected

Registering With the Cell Directory Service



- **The server registers its server-entry name with CDS**
 - manager object information automatically registered
- **The server can register under a group name**
 - enables clients to find interfaces only knowing group
- **The server can register under a profile**
 - enables clients to find interfaces with no information if default profile (`./:/cell-profile`) is used

Server: Registering Authentication and CDS Information



```
// server main
int main(int argc, char **argv)
{
    ...
    // register authentication information

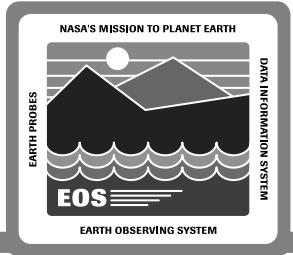
    theServer->SetAuthInfo((unsigned char*)princName,
                           rpc_c_authn_dce_secret,
                           (void*)keyFile);

    // register name information

    theServer->SetName((unsigned char*)cdsName);
    // theServer->SetGroupName((unsigned char*)grpName);
    // theServer->SetProfileName((unsigned char*)profile);

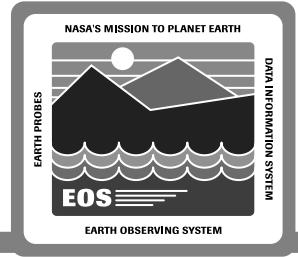
    ...
}
```

Requesting Authenticated RPC

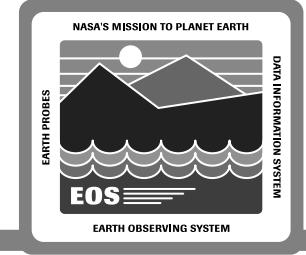


- Client makes request on a per object basis
- Request consists of
 - Authentication service
 - » No authentication
 - » DCE shared, secret-key authentication (default)
 - Communication protection level
 - » Default (can be set by cell administrator)
 - » No protection
 - » Connect level: Encrypted handshake during connect
 - » Call level: Verifier attached to each call and response (n/a with tcp/ip)

Requesting Authenticated RPC (cont'd)



- » **Packet level:** Verifier attached to each message
- » **Packet integrity level:** Cryptographic checksum for each message
- » **Privacy level:** Encryption of each RPC argument
 - Authorization level
 - » Name-based
 - » PAC-based (required for ACLs)



Client: Requesting Security Level

```
void DataServer_cl :: setSecurity(const char *princName)
{
    ...
    SetAuthInfo((unsigned char*)princName,
                rpc_c_protect_level_pkt_integ,
                rpc_c_authn_dce_secret,
                (rpc_auth_identity_handle_t)NULL,
                rpc_c_authz_dce);
}
```

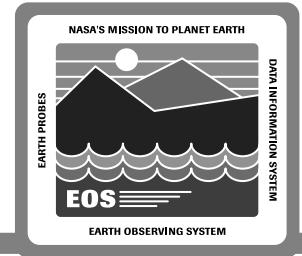
None
Connect
Call
Packet
Packet integrity
Privacy

None
Secret-key

Name
PAC

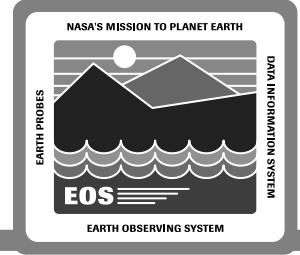
Note: DataServer_cl is derived from DataServer_1_0, the IDL generated surrogate object

Attaching a Reference Monitor to a Manager Object



- Attached on a per object basis for fine control
- Forces a client to request certain security levels
- Automatically invoked from EPV code
- OODCE provides
 - DCERefMon (abstract)
 - » Specify required security levels in constructor
 - DCEStdRefMon (concrete), which checks that
 - » Client's requested authentication service \geq required
 - » Client's requested protection level \geq required
 - » Client's requested authorization service \geq required

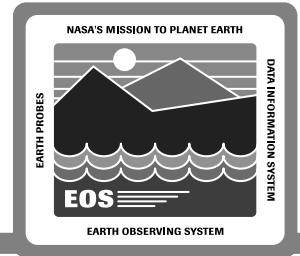
Registering With the GSO



- GSO puts manager object on internal list
- GSO automatically (at the appropriate time)
 - Registers the object with the DCE runtime
 - Creates an entry in the endpoint map for the object with interface UUID, object UUID, and binding information (protocol, endpoint)
 - Adds the object's interface UUID and, optionally, its object UUID to CDS server entry

Note: There are some issues with registering object UUIDs in the CDS

Server: Instantiate Manager and Register with GSO



```
// server main
int main(int argc, char **argv)
{
    ...
    // instantiate manager object and attach reference monitor

    DataServerSrv obj(objUuid);

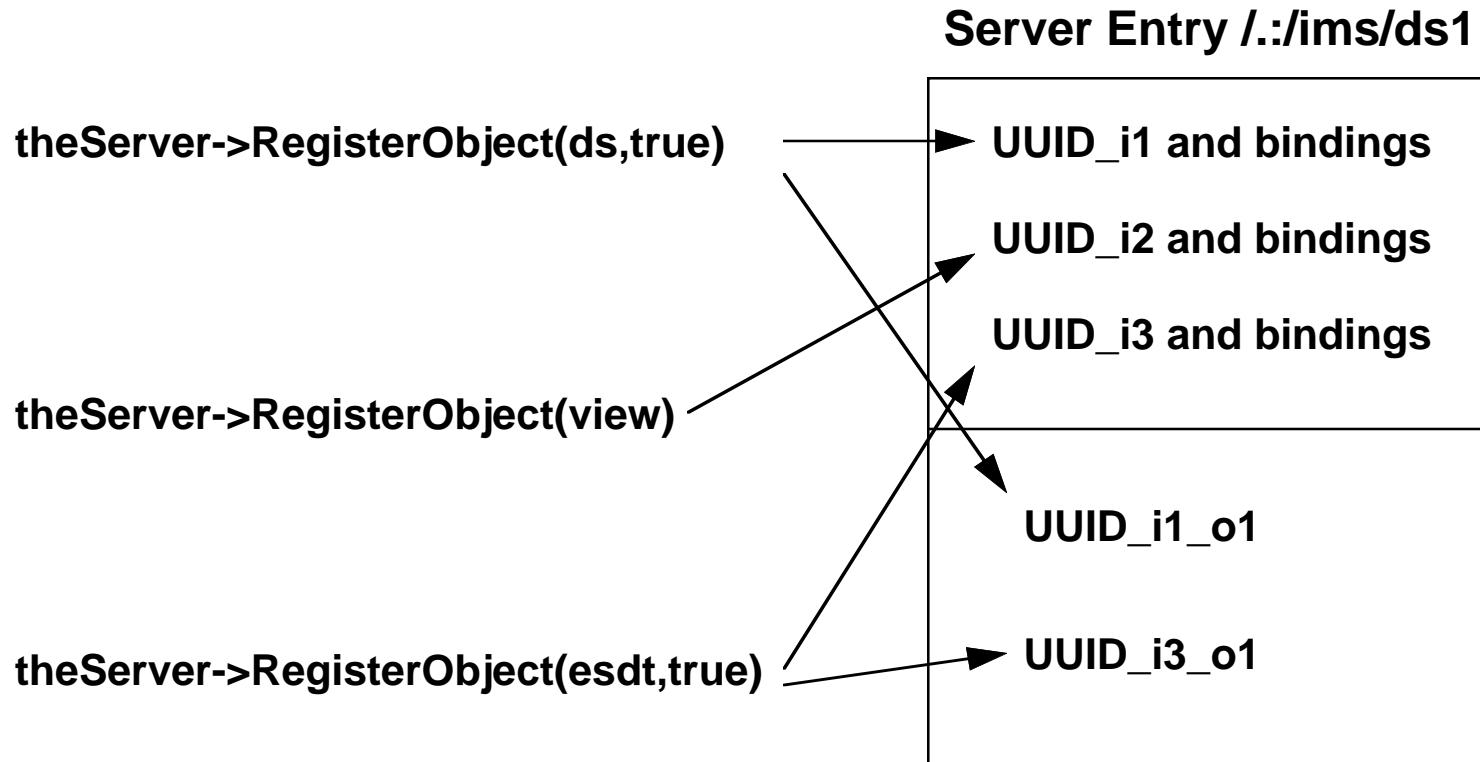
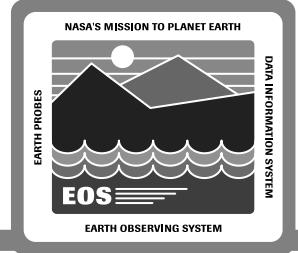
    DCEStdRefMon dsRefMon( rpc_c_protect_level_pkt_integ,
                          rpc_c_authn_dce_secret,
                          rpc_c_authz_dce);
    obj.SetRefMon(&dsRefMon);

    // register manager with GSO

    theServer->RegisterObject(obj,true);
    ...

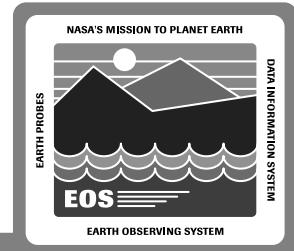
}
```

How Interface and Object UUIDs Are Stored In CDS

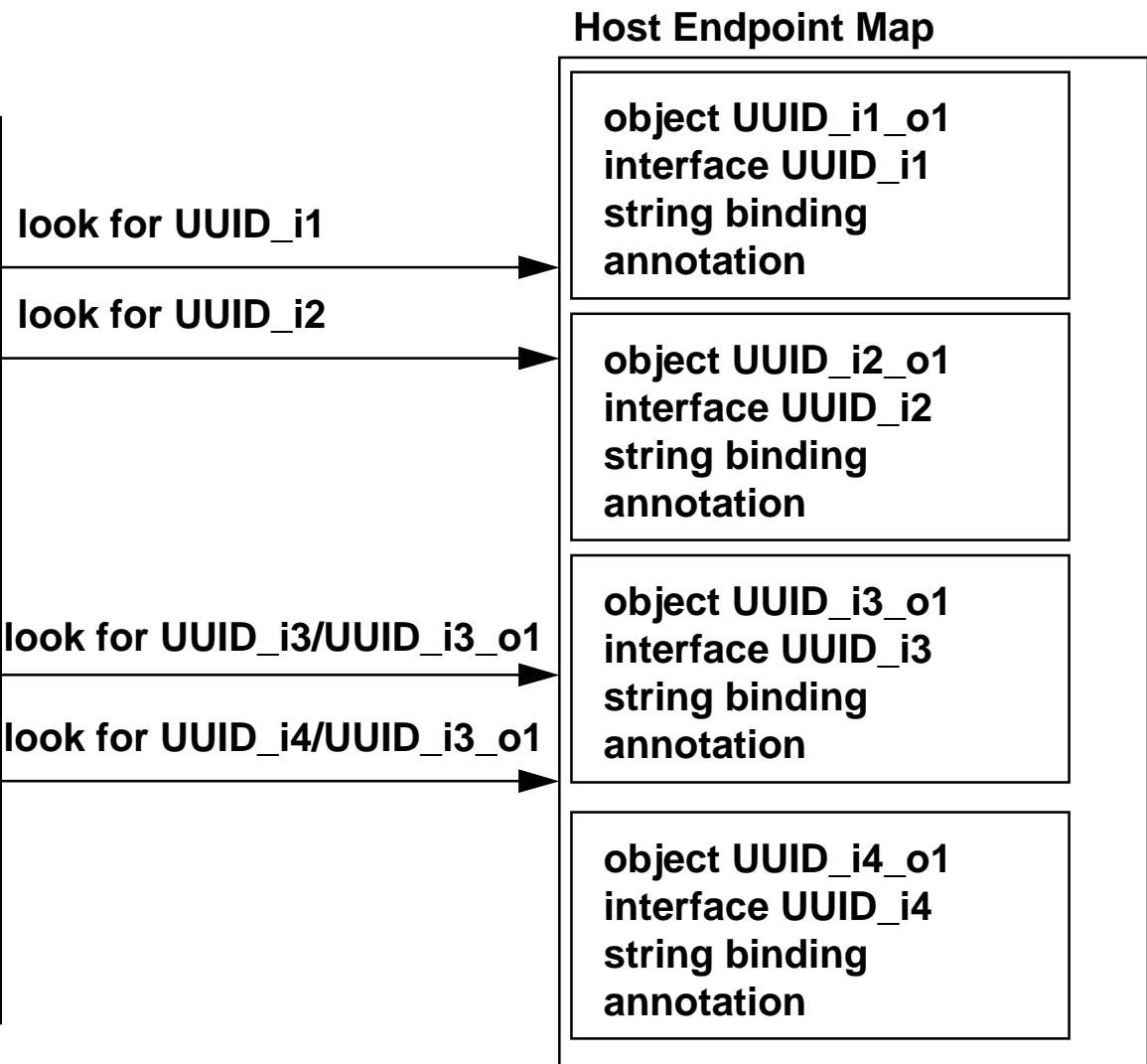
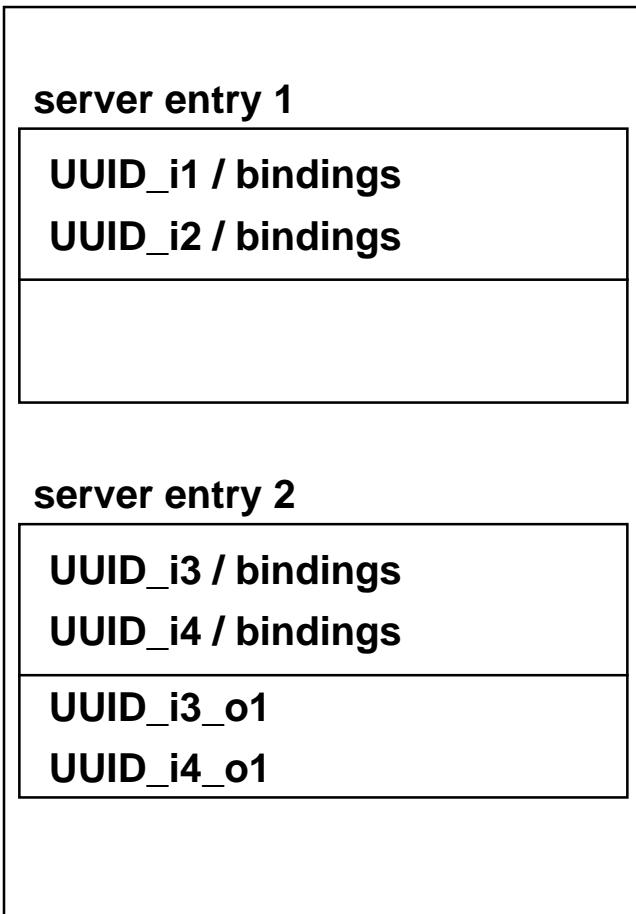


Note: The object UUIDs are associated with no particular binding in the CDS server entry.

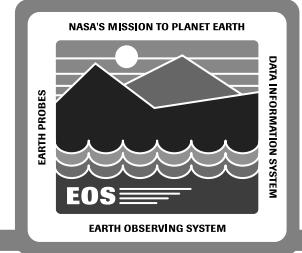
How CDS Is Used to Locate Objects



CDS

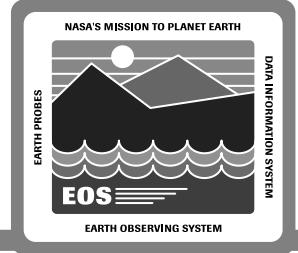


Object UUID / CDS Guidelines

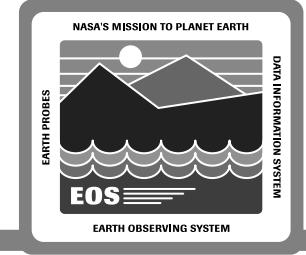


- **If one manager object exports its object UUID to CDS**
 - Make object UUIDs of all manager objects reached through CDS identical
 - Export them to the CDS
 - Note that the rdac1 interface exports its object UUID to the CDS
- **If you plan to run several instances of a server on the same machine**
 - Make object UUIDs of managers reached through CDS different for each server process
 - Export them to the CDS

Object UUID / CDS Guidelines (cont'd)

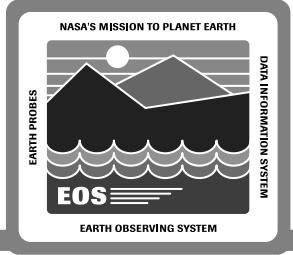


- **For manager objects that will not be reached through the CDS**
 - Make sure that different instances have different object UUIDs
 - Do not export these object UUIDs to CDS
 - To locate such an object from another process, use a CDS non-server entry to store the object UUID and binding information



The GSO Listen Member Function

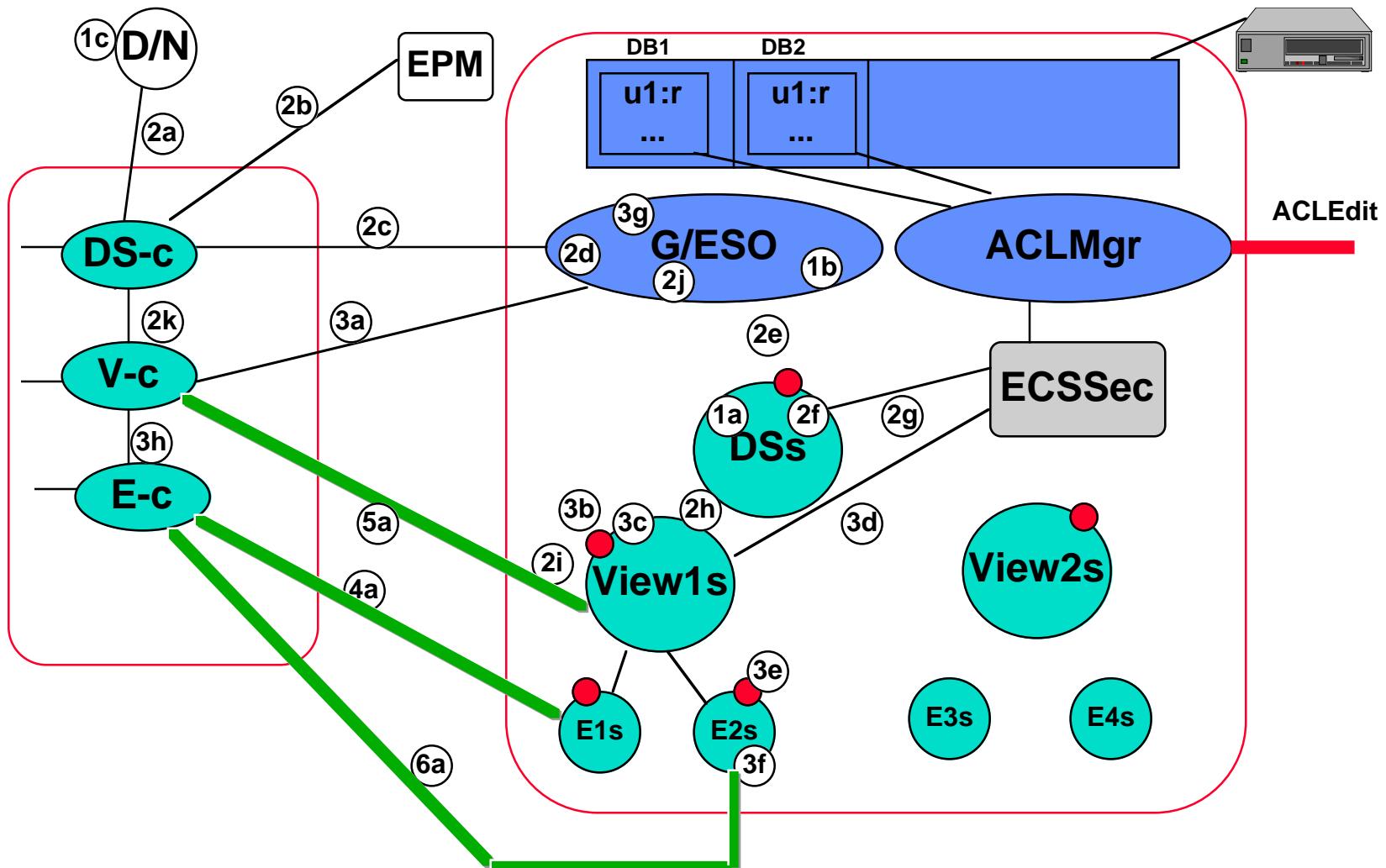
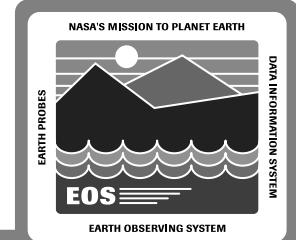
- **The GSO has five states**
 - initial
 - runtime registered
 - endpoint exported
 - CDS exported
 - listening
- **Listen() promotes GSO directly from initial to listening state**
 - GSO exports information for all objects currently registered
 - GSO exports information for all objects that register in the future immediately



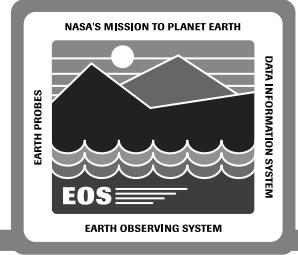
Server: Invoke GSO Listen

```
// server main
int main(int argc, char **argv)
{
    ...
    // invoke GSO Listen
    theServer->Listen();
    ...
}
```

Client/Server Interaction



Dynamic Object Creation



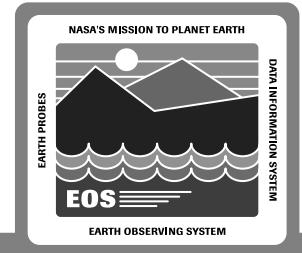
- The DataServer manager is a kind of object factory for creating View objects
- The View managers are kinds of object factories for creating Esdt objects

When the client application executes the statements ...

```
DataServer_cl ds( (unsigned char*)srvCdsName);
```

```
View_cl *view = ds.createView();
```

Dynamic Object Creation (Surrogate)



the derived Data Server
surrogate executes...

```
View_cl *DataServer_cl :: createView()
{
    View_cl    *view = 0;
    DCEObjRefT *ref = 0;

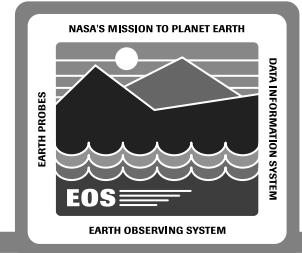
    try {
        ref = DataServer_1_0::createView();
    }
    catch (DCEErr &exc) {...}

    if( ref) {
        DCEUuid uuid(&(ref->objId));

        view = new View_cl( DCEGetBindingHandle(ref),
                            uuid);
        view->setSecurity( srvPrincName);

        free( ref);
    }
    return view;
}
```

Dynamic Object Creation (Manager)



```
DCEObjRefT *DataServerSrv :: createView()
{
    if( !AclServer::isAuth( "dataserverobj",
                           "x") ) {
        return 0;
    }

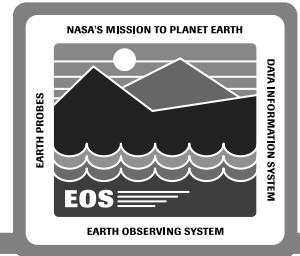
    DCEObjRefT *ref = 0;
    ViewSrv *newView = new ViewSrv;

    if( newView) {
        ...
        attachRefMon( newView);
        theServer->RegisterObject(*newView);

        ref = (DCEObjRefT*)newView->GetObjectReference();
    }
    return ref;
}
```

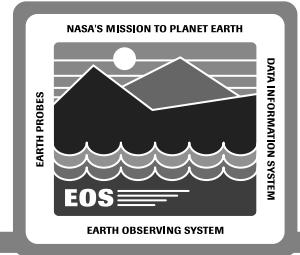
and the Data Server manager executes...

Iterators



- A View manager creates a set of Esdt manager objects from search criteria
- The client can “iterate” over these objects
 - The Esdt manager objects are not in the same address space as the client
 - On each iteration, the client obtains a pointer to an Esdt surrogate bound to the next manager object in the list

Iterators (cont'd)



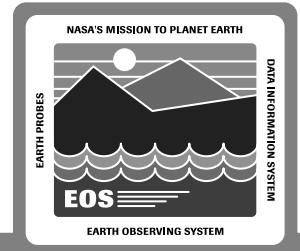
When the client application executes the statements ...

```
Esdt_cl *curEsdt
```

```
while( curEsdt = view->getNextEsdt())
{
    // call member functions of curEsdt

    ServiceList *list = curEsdt->getServiceList();
    curEsdt->executeCmd(...);
}
```

Iterators (Surrogate)

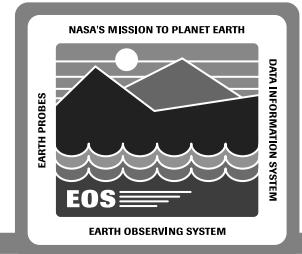


Esdt_cl *View_cl::getNextEsdt

```
Esdt_cl      *rtnPtr = 0;
DCEObjRefT *ref    = 0;
try {
    ref = View_1_0::getNextEsdt();
}
catch (DCEErr &exc) {...}
if( ref) {
    DCEUuid uuid(&(ref->objId));
    if( myCurEsdt) {
        myCurEsdt->SetBinding( DCEGetBindingHandle( ref));
        myCurEsdt->SetServerObject( uuid);
    }
    else {
        myCurEsdt = new Esdt_cl( DCEGetBindingHandle(ref),
                                  uuid);
    }
    free( ref);
    rtnPtr = myCurEsdt;
}
return rtnPtr;
```

the derived
View surrogate
executes ...

Iterators (Manager)



and the View manager
executes ...

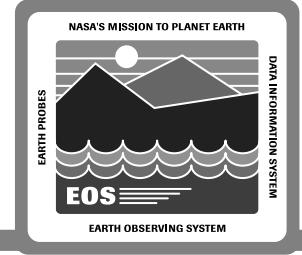
```
DCEObjRefT *ViewSrv :: getNextEsdt()
{
    if( !AclServer::isAuth( "viewobj",
                           "x")) {
        return 0;
    }
    DCEObjRefT *ref = 0;

    if( myCurNode) {
        EsdtSrv *obj = myCurNode->esdt();

        ref = (DCEObjRefT*)obj->GetObjectReference();

        myCurNode = myCurNode->next();
    }
    else {
        cout << "View: getNextEsdt no more elements in list"
        << endl << endl;
    }
    return ref;
}
```

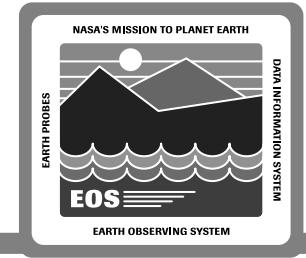
Using Threads for Asynchronous Communication



- An individual RPC is synchronous
- An RPC in a separate thread is asynchronous to a client process as a whole
- OODCE provides
 - DCEPthread
 - Other related classes
- CSS is providing
 - Abstract class AsyncCom
 - Non-member function EvalThread

**Note: Servers do not change
No restriction on number or format of arguments**

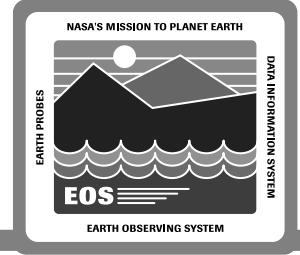
Using AsyncCom and EvalThread



- Derive a concrete class from AsyncCom
 - Attach any data you need
 - Make desired RPC from Invoke()
 - Allocate memory in PreInvoke(), free it in PostInvoke()

```
class AsyncCom {  
    ...  
    virtual void SetData( void *data);  
    virtual void *GetData();  
    virtual int PreInvoke() = 0;  
    virtual int Invoke() = 0;  
    virtual int PostInvoke() = 0;  
    ...  
}
```

Using AsyncCom and EvalThread (cont'd)



- Start an independent thread of execution
 - EvalThread is the procedure to execute
 - A pointer to a derived AsyncCom instance is the parameter

```
YourAsyncCom *yourCom = new YourAsyncCom;
```

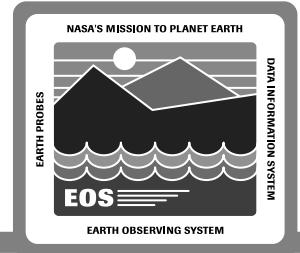
... set up your async com...

```
DCEPthread aThread;
```

... set up thread attributes as desired...

```
aThread.Start( EvalThread, (DCEPthreadParam)yourCom);
```

Using AsyncCom and EvalThread (cont'd)

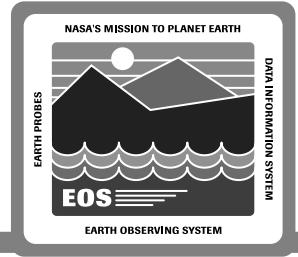


- **EvalThread calls overridden member functions**
 - RPC call made in independent thread, while main thread continues to execute

```
DCEPthreadResult EvalThread( DCEPthreadParam p)
```

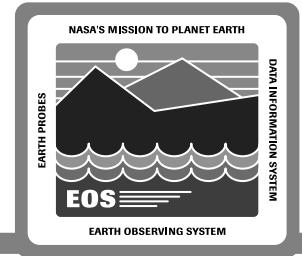
```
{  
    AsyncCom *acom = (AsyncCom*)p;  
    acom->PreInvoke();  
    acom->Invoke(); → int YourAsyncCom :: Invoke()  
    acom->PostInvoke();  
    ... make some RPC ...  
    delete acom;  
    return 0;  
}
```

Example - Executing All Commands Simultaneously



- Motif-based client presents “All” option for executing all commands in service list
- Client uses EvalThread and a derived AsyncCom for simultaneous RPCs to Esdt

Motif Client: Executing All Commands Simultaneously



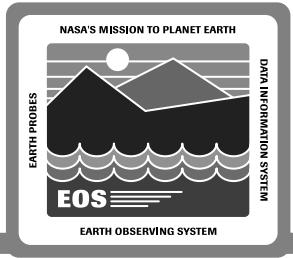
```
void MotifCliAppl :: executeAllCmds()
{
    for each cmd in dialog {
        ExecCmdParams *params = new ExecCmdParams;
        params->appl( this );
        params->command( cmd );

        CliAsyncCom *com = new CliAsyncCom;
        com->SetData((void*)params);

        DCEPthread aThread;
        aThread.Termination(Pthread_detach_on_delete);
        aThread.Start( EvalThread, (DCEPthreadParam)com );
    }
}

int CliAsyncCom :: Invoke()
{
    ExecCmdParams *params = (ExecCmdParams*)myData;
    MotifCliAppl *appl = params->appl();
    appl->executeSpecificCmd( params->command() ); // rpc call made in here
}
```

Makefile



```
DEBUG           = -g
INCENV          = -I. -I/usr/include/CC -I/usr/include/oodce -I/usr/include/reentrant
ANSI_FLAGS      = -Aa
HP_FLAGS         = -Dunix -DNIDL_PROTOTYPES -D__STDC__ -D_HPUX_SOURCE\
                   -D_CMA_PROTO_ -D_REENTRANT

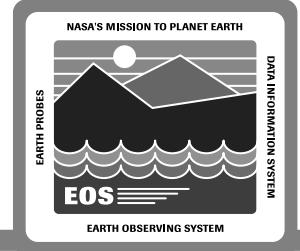
CFLAGS           = ${DEBUG} ${ANSI_FLAGS} ${HP_FLAGS} ${INCENV}
CXXFLAGS          = +eh ${DEBUG} ${ANSI_FLAGS} ${HP_FLAGS} ${INCENV}
CXXLDFLAGS        = +eh ${DEBUG} -WI,-a,archive
CXXLIBS           = -loodce -lbb -ldce -lm -lc_r

PROGRAMS          = string_conv_server string_conv_client
server_OFILES       = string_conv_sstub.o string_conv.o string_convE.o server.o
client_OFILES        = string_conv_cstub.o string_convC.o client.o

IDLXXFLAGS         = -keep source ${INCENV}
IDLXXFILES          = string_conv.idl
IDLXXGEN            = string_conv.h string_conv_cstub.c string_conv_sstub.c\
                     string_convC.H string_convS.H\
                     string_convC.C string_convE.C

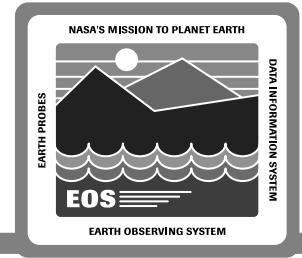
IDLXX             = /usr/bin/idl++
```

Makefile (contd)



```
# Build a regular object file from a C source file.  
.c.o:  
        [ -f $@ ] && rm -f $@ ; $(CC) ${CFLAGS} -c -o $@ $<  
# Build a regular object file from a C++ source file.  
.C.o:  
        [ -f $@ ] && rm -f $@ ; $(CXX) ${CXXFLAGS} -c -o $@ $<  
  
all:          objects ${PROGRAMS}  
objects:      ${server_OFILES} ${client_OFILES}  
fresh:        clean all  
  
clean::;  
        rm -f ${server_OFILES} ${client_OFILES} ${IDLXXGEN}  
        rm -rf ./ptrepository  
  
clobber:  
        clean  
        rm -f ${PROGRAMS} a.out core ERRS make.out *~  
  
string_conv_server:  
        ${server_OFILES} ${OODCE_LIB}  
        [ -f $@ ] && rm -f $@ ; $(CXX) ${CXXLDFLAGS} ${server_OFILES} ${CXXLIBS} -o $@  
  
string_conv_client:  
        ${client_OFILES} ${OODCE_LIB}  
        [ -f $@ ] && rm -f $@ ; $(CXX) ${CXXLDFLAGS} ${client_OFILES} ${CXXLIBS} -o $@  
  
${IDLXXGEN}:  
        ${IDLXXFILES}  
        rm -f ${IDLXXGEN}  
        ${IDLXX} ${IDLXXFLAGS} ${IDLXXFILES}
```

Makefile (contd)



```
client.o:\n        string_conv.h\\n        string_convC.H\\n        client.C\n\nserver.o:\\n        string_conv.h\\n        string_convS.H\\n        server.C\n\nstring_conv.o:\\n        string_conv.h\\n        string_convS.H\\n        string_conv.C\n\nstring_convC.o:\\n        string_conv.h\\n        string_convC.H\\n        string_convC.C\n\nstring_convE.o:\\n        string_conv.h\\n        string_convS.H\\n        string_convE.C\n\nstring_conv_cstub.o:\\n        string_conv.h\\n        string_conv_cstub.c\n\nstring_conv_sstub.o:\\n        string_conv.h\\n        string_conv_sstub.c
```