Asynchronous Calls

• Standard COM+ model is completely synchronous
  – Emulates standard procedure calls
  – Problematic in distributed scenarios with high network latencies

• Use threads to handle multiple calls asynchronously
  – Efficiency might be limited
  – Synchronization of multiple threads may be difficult

• Idea: support asynchronous calls by infrastructure
  – COM+ starts call and returns to client immediately
  – Call object can be used to obtain results later
  – Language constructs like futures support this approach
  – Works currently under Windows 2000 only
Defining Asynchronous Interfaces

• New IDL attribute [async_uuid]
  – MIDL generates both, synchronous and asynchronous version of IF

  [object, uuid(10000001-AAAA-0000-0000-A00000000001),
   async_uuid(10000001-AAAA-0000-0000-B00000000001)]

Interface IPrime : IUnknown
{
    HRESULT IsPrime(int num, [out, retval] int * v);
}

• Methods are split into two for asynch. Interface
  • Begin_method accepts all [in], [in, out] parameters
  • Finish_method accepts all [out], [in, out] parameters
Asynchronous Interface generated by MIDL

MIDL_INTERFACE(“10000001-AAAA-0000-0000-B00000000001“)
AsynchPrime : public IUnknown {

public:

    virtual HRESULT STDMETHODCALLTYPE Begin_IsPrime(
        int testnumber ) = 0;
    virtual HRESULT STDMETHODCALLTYPE Finish_IsPrime(
        /* out, retval */ int __RPC_FAR *v) = 0;

};

• New registry entries for asynchronous interfaces:
  – AsynchronousInterface subkey under IID of synchronous interface
Calling Asynchronous Interfaces

To begin an asynchronous call

1. Query the server object for the ICallFactory interface. If QueryInterface returns E_NOINTERFACE, the server object does not support asynchronous calling.

2. Call ICallFactory::CreateCall to create a call object corresponding to the interface you want, and then release the pointer to ICallFactory.

3. If you did not request a pointer to the asynchronous interface from the call to CreateCall, query the call object for the asynchronous interface.

4. Call the appropriate Begin_method.

interface ICallFactory : IUnknown {
    HRESULT CreateCall([in] REFIID riid, [in] IUnknown *pCtrlUnk,
                        [out, iid_is(riid2)] IUnknown **ppv);
}

Client makes asynchronous call

IPrime * pPrime = 0;
CoCreateInstance( CLSID_Prime, 0,
    CLSCTX_LOCAL_SERVER,
    IID_IPrime, (void **) &pPrime;

ICallFactory* pCallFactory = 0;
pPrime->QueryInterface(IID_ICallFactory,
    (void **) &pCallFactory);

AsyncIPrime* pAsyncPrime = 0;
pCallFactory->CreateCall(IID_AsyncIPrime,
    0, IID_AsyncIPrime, (IUnknown**) &pAsyncPrime);

pAsyncPrime->Begin_IsPrime(number);

int result = 0;

// do other work here
pAsyncPrime->Finish_IsPrime(&result);
if (result)
    printf("%d is prime\n", number);

pAsyncPrime->Release();
pCallFactory->Release();
pPrime->Release();
Asynchronous Calls (contd.)

• A call object can process only one asynchronous call at a time.
  – If the same or a second client calls a Begin_ method before a pending asynchronous call is finished, the Begin_ method will return RPC_E_CALL_PENDING.

• If the client does not need the results of the Begin_ method, it can release the call object at the end of this procedure.
  – COM detects this condition and cleans up the call. The Finish_ method is not called, and the client does not get any out parameters or a return value.

• When the server object is ready to return from the Begin_ method, it signals the call object that it is done.
  – When the client is ready, it checks to see if the call object has been signaled.
  – If so, the client can complete the asynchronous call.
Finishing an asynchronous call

- The mechanism for this signaling and checking between client and server is the ISynchronize interface on the call object.
  - The call object normally implements this interface by aggregating a system-supplied synchronization object.
  - The synchronization object wraps a Win32 event handle, which the server signals just before returning from the Begin_ method by calling ISynchronize::Signal.

- To complete an asynchronous call
  1. Query the call object for the ISynchronize interface.
  2. Call ISynchronize::Wait.
  3. If Wait returns RPC_E_TIMEOUT, the Begin_ method is not finished processing. The client can continue with other work and call Wait again later. It cannot call the Finish_ method until Wait returns S_OK.
  4. If Wait returns S_OK, the Begin_ method has returned. Call the appropriate Finish_ method.
interface ISynchronize : IUnknown {
    // waits for the synchronization object to be signaled
    // or for a specified timeout period to elapse, whichever comes first
    HRESULT Wait([in] DWORD dwFlags, [in] DWORD dwMillisec);

    // sets synchronization object's state to signaled
    HRESULT Signal();

    // resets synchronization object to non-signaled state
    HRESULT Reset();
}
Interoperability

• Asynchronous and synchronous IF are considered as two parts of the same interface
  – Although they have different unique IIDs
  – What happens if component implements synchronous IF only?

• If component implements synch IF only...
  – COM+ infrastructure automatically supports ICallFactory interface in the proxy (standard marshaling)
  – Maps async calls to synch interface (Begin_ ...)
  – Proxy holds values of synch call until client calls Finish_ ...

• If component implements both versions of IF...
  – Duplication of code -> Components need only support async IF...
  – COM+ infrastructure maps synchronous calls to async version of IF