Component Architectures

Ken Birman

A long term trend…

- In cs4410 we've seen that the modern O/S (and more and more applications) are networked
- Up to now we focused on a TCP based client/server style of application
  - The client runs on your machine
  - It does RPCs to a server on the network to fetch data or request that actions be taken on its behalf

2

But this is awkward

- When we link an application to a library, type-checking helps us avoid mistakes
  - Like trying to pass an integer argument to sin(), which expects a floating point argument
- If a client just uses TCP to talk to a server these sorts of benefits are lost
- Ideally we want
  - Type checking
  - Debugging support
  - Tools to help us tune and optimize our applications

3

A split is underway

- One path focuses on the Linux style of platform
  - Traditional applications on a traditional O/S
  - Common in data centers
- The other path leads towards more and more componentized styles of systems
  - O/S morphs into a common language runtime environment
  - Common on desktops and mobile devices like phones

4

A long history of object orientation

- Starting more than a decade ago, an "object oriented" community began to study this question
  - They adopted the approach of building fancy libraries that run on standard O/S platforms
  - These libraries link to the main programming languages in clean, standard ways
- Goal: instead of thinking of a "program" we think about "objects" that talk to other "objects" through remote method invocation

5

Major OO platforms

- The first of these platforms to succeed was the Distributed Computing Environment, DCE
- It gave way to CORBA, the Common Object Request Broker Architecture (remains widely used)
- CORBA in turn was supplanted by Java's J2EE environment and the Microsoft .NET/DCOM architecture

6

Genealogy

- DCOM split from J2EE, which evolved from CORBA

- Microsoft calls it the Distributed Component Architecture
  - DCOM is a Microsoft product but fairly open and widely supported outside of Microsoft platforms
  - In the context of .NET, probably the most powerful existing architecture of this kind

.NET

- Think of .NET as a standardized O/S interface
  - Microsoft designed it, too
  - A system called Mono lets us run it on Linux

- .NET provides
  - A virtual computer interface (has its own assembler language, which is "just in time" compiled on target)
  - Standard memory management
  - Standards for any .NET language to talk to any other .NET language
  - Compilers for about 40 languages, to date

8

.NET "versus" DCOM

- DCOM is used when applications (objects) constructed in .NET need to talk to other objects over a network
  - .NET supports direct calls when the objects are both on the same machine
  - The term "component" is a synonym for "object" in the context of .NET and DCOM

Next few slides from Jim Ries, UMD

9

DCOM Goals

- Encapsulation (separate code from interface)
- Versioning (important because software evolves)
- Execution context independence
- Language independence
- Object Creation / Lifetime Control
- Standard error code (HRESULT)
- Solve object discovery problem
- Scripting

- Overall: goal is to make component reuse feasible

Component Reuse

- Standard example: Windows "Contacts" database
  - Many applications potentially need data from it, such as email, web browser, phone auto-dialer, fax
  - Traditionally, solve by having a file in a standard format

- In DCOM, idea is that the contacts application is a component that other systems can talk to
  - It offers a standard, carefully implemented interface
  - They respect this interface, and needn't worry about the way the database is represented or where it was stored
  - The component is clean, correct, robust

11

Later and later binding

- "Editor inheritance" binds at compile time.
- Link libraries (.LIB) bind to "components" at link time.
- Dynamic link libraries (.DLL) bind at run time, but need a header at compile time, and path at runtime.
- COM components bind at runtime and may need neither a header, nor a path! (though typelib contains header-like "meta-data")

How does the O/S "help"?

- In .NET / DCOM, the O/S has merged into the component runtime environment
  - In older systems, like Linux, programming language was distinct from the O/S and defined its own world
  - With Windows, there no longer is a separate O/S
    - Most features of the traditional programming language environment are now part of a shared runtime environment
    - Many languages use this same shared environment
    - As a result, components coded in different languages can easily and efficiently talk to one-another
  - .NET gets security and protection using type checking!
    - You can only see objects that are accessible to your code and can only access them in ways legal for their "type"

Interfaces

- COM enforces the concept of interfaces being separate from implementation.
  - Interface == Abstract Base Class
  - Objects support multiple interfaces through multiple inheritance.
- Multiple components can potentially implement the same interface
  - Very useful for plug-and-play behavior
  - The code changes... but the interfaces the code implements is a kind of standard and rarely evolves or changes

Example

- Many Windows applications talk to users through the standard Windows "forms" interfaces
  - Accessible from many languages
  - Defined in terms of components they call "Windows controls"
    - Like pull-down menus, buttons, file browsers
    - By using them, applications achieve a standard look and feel
- .NET standardization of type checking and memory management makes all of this "safe"

GUID's (or UUID's)

- Globally Unique Identifiers (Universally Unique Identifiers)
- Needed to avoid name collisions
- A class is associated with a GUID (CLSID).
- An interface is associated with a GUID (IID).
- The Windows Registry: a hierarchical database.

Demonstration - IDL

```
[
        object,
        uuid(75D873CD-7B63-11D3-9D43-00C0F031CDDE),
        helpstring("IServer Interface"),
        pointer_default(unique)
]
interface IServer : IUnknown
{
        HRESULT Hello([in, string] char * pszMessage);
};
```

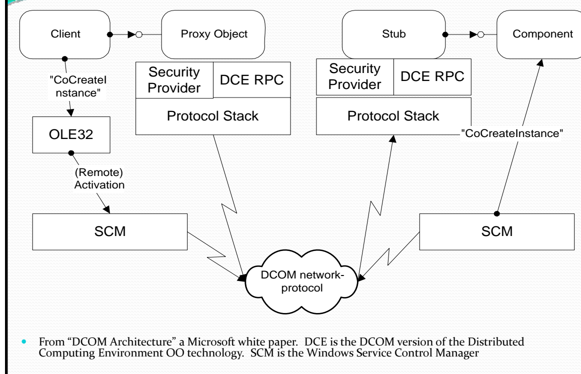
Demonstration - Server Code

```
// Prototype
class CServer :
    public IServer, public CComObjectRoot,     public CComCoClass<CServer,&CLSID_Server>
{
    // . . .   Some code omitted for brevity

    // IServer
    public:
    HRESULT STDMETHODCALLTYPE Hello(unsigned char * pszMessage);
};

// Code
HRESULT STDMETHODCALLTYPE CServer::Hello(unsigned char * pszMessage)
{
    char szBuf[256];
    wsprintf(szBuf,"%s",pszMessage);
    ::MessageBox(0,szBuf,"Server",MB_OK);
    return(S_OK);
}
```

Demonstration - Client Code

```
if (SUCCEEDED(
    hr=CoCreateInstance(CLSID_Server,NULL,
                        CLSCTX_LOCAL_SERVER,
                        IID_IServer,(void **)&pServer)))
{
    if (SUCCEEDED(hr=pServer->Hello((unsigned char *)"Hello from the client")))
            MessageBox("Client: Server printed the message");
    else
    {
            wsprintf(szBuffer,"Hello() method failed: ox%lX.\n",hr);
            MessageBox(szBuffer);
    }
    pServer->Release();
}
else
{
    wsprintf(szBuffer,"Unable to create a server: ox%lX.\n",hr);
    MessageBox(szBuffer);
}
```

Distributed Scenario



- From "DCOM Architecture" a Microsoft white paper. DCE is the DCOM version of the Distributed Computing Environment OO technology. SCM is the Windows Service Control Manager

Additional Technologies

- COM+
  - MTS - Microsoft Transaction Server
  - MSMQ - Microsoft Message Queue
  - Compiler supported IUnknown, etc.
- ADS - Active Directory Service
  - As "distributed registry"
  - As namespace abstraction
- All Microsoft products are COM based:
  - IIS - Internet Information Server
  - Exchange
  - Internet Explorer
  - Word, Excel, etc.

Pros and Cons of Components

- The Windows focus on components confuses many developers
  - In Linux we think mostly in terms of standalone programs that link against standardized libraries
  - Some programs are "servers" and others (clients) talk to those servers, but the model is rather "flat"
- With Windows
  - Makes more sense to think about graphs of components
  - "Events" flow between these components

22

Events

- Rather than focusing on a request/reply style of method invocation, Windows prefers asynchronous streams of events
- Examples
  - If an application is "watching" a Windows folder and it changes, a "refresh" event is delivered
    - Tells the application to reread the folder
  - Similarly if a control should redraw itself because something changed on the screen

23

Events

- In .NET, an event is a standard kind of object
  - Like a small, strongly typed, message
- Components expose endpoints (queues) on which events can be delivered
  - Internally, applications attach handler functions to these queues
  - When an event arrives, the .NET runtime will issue upcalls to these handlers, passing the event as an argument

24

Example: Asynchronous I/O

- In Linux, most applications read files or network messages one by one
- In this Windows model, many applications "post" a vector of requests all at once
  - Perhaps, 100 network message receive requests
  - Or multiple file reads
- Later as each request completes, .NET posts an asynchronous completion event to the associated queue

Example: Asynchronous I/O

- This asynchronous style of I/O is extremely fast
  - Can easily support tens of thousands of I/Os per sec!
  - Event handling is as cheap as a procedure call
- In fact, all of .NET runs in a single address space!
  - The platform is using strong type checking to eliminate the need for memory protection
  - Calls from component to component are done with direct procedure calls and context switching is incredibly cheap.

Pros and Cons of Components

Pros

- Promotes reuse of "large" abstract data types
  - Like contacts database, Windows "tables"
- Managed type system, memory
- Very fast inter-component calls
- Works really well with asynchronous event-oriented applications
- Forces standardization at the component level: each component is a standard of a sort

Cons

- Can seem unnatural to people who don't program in OO languages
- Can't support unsafe C, asm code
- Strongly encourages ("forces") designer to think in terms of event interfaces, asynchronous interactions
- Linux only had ~100 interfaces. Windows .NET seems to have tens of thousands!
- Components may end up "too interdependent": spaghetti

Future of the O/S

- In ten years, will we still teach an O/S course?
  - Could argue that more and more, we should teach about components and distributed computing
  - How many people will ever write a device driver?
- On the other hand, understanding the O/S seems to be key to understanding things like .NET and DCOM
  - They got this way via evolution
  - Good reason to doubt that they can be comprehended without retracing the steps by which they evolved!

The situation today

- More and more of a split
  - Like it or not, Windows is by far the dominant desktop platform
  - Linux is mostly the dominant enterprise data center platform
  - Mobile devices like telephones: competing options
- Meanwhile, web browsers and cloud computing are vying to displace Windows
  - With some success, but many "gotcha's"
  - Platform security is improving… Browser and hence Cloud security seems to be eroding…

Some major O/S research topics

- Making effective use of multicore
  - Many applications literally slow down if you give them more cores – even multithreaded ones
  - This is because of lock, cache contention, poor pipeline performance, "false sharing"
  - Can the O/S somehow promote better use of cores?
- Virtualization tricks
  - Can we fix some of the consolidation issues we saw?
  - Can virtualization protect against viruses?

Some major O/S research topics

- Exploiting TPM hardware
  - We haven't even begun to scratch the surface
  - Could invent entirely new ways of using cryptographic tools and components
- Component architectures
  - Incredibly powerful... but they seem to promote spaghetti style dependencies
  - Can we have components and yet maintain clean abstraction boundaries and flexibility?

31

Some major O/S research topics

- Operating systems and tools for cloud computing
  - Help building applications to run in data centers
  - Tools for replicating data and building scalable services
  - Better ways of monitoring and managing, especially for purpose of reducing power footprint
- Components gave us drag-and-drop solution development on the desktop (like these slides)
  - Can we do drag-and-drop application development?
  - Are there ways to comprehend/debug/manage the resulting constructs?

32