Components

David Talby

This Lecture

- What Components Are
  - With Demonstrations in Delphi
- Common Object Model (COM)
  - Creating and calling objects
  - Distributed COM
- Component-Oriented Software

Components

- The Holy Grail of Software Engineering
  Build software by connecting existing components in simple ways
- "Beyond Object-Oriented Software"
  Small-scale success, large-scale failure
- The three parts of industry-scale reuse
  - A widely used object-oriented framework
  - Developers building components for it
  - Other developers using them to build apps

Using Components in Delphi

- Visual Components
  - Visual editors, containers, dialogs
  - Use of properties, events, methods
- Database Connectivity
  - Non-visual components
  - Delegation & Inheritance between components
- Office Automation
  - Visual & Non-Visual use
  - Whole programs as components

Writing Components in Delphi

- Writing Components
  - Inheritance
  - Use of properties, events, methods
- 3rd Party Components
  - Tens of thousands of components
  - Many are freeware or shareware
  - www.torry.net and others
- Delphi's Framework
  - Based on Object Pascal
  - Can import components from COM, CORBA, COM+, EJB, Web Services

TObject
TPersistent
TComponent
TControl
TWinControl
TCustomListControl
TCustomCombo
TCustomComboBox
TComboBox

A Component's Interface

- A component's interface has:
  - Methods - like draw() and disconnect()
  - Properties – like color and name
    (wrappers to getter and setter methods)
  - Events – like *onClick* and *onNetworkError*
    (pointer to function or list of such pointers)
- What about implementation?
  - Completely unavailable to clients
  - Can be one or many classes
  - Can be in another language

Defining Components

- Definition: a software component is a module with these properties:
  - It can be used by other software modules, called clients (not just humans)
  - Only its specification (interface) is required for clients to use it
  - The clients and their authors do not need to be known to the component's authors

Defining Components II

- Usually there's another property:
  - It can be used within a visual designer tool
  - To do so, it must be derived from the framework the tool supports
- Commercial component frameworks
  - COM (Visual Basic, Visual C++)
  - JavaBeans (JBuilder, Café, WebSphere)
  - Delphi (Delphi)
  - .NET (Visual Studio.NET)

Comparison of Reuse Techniques

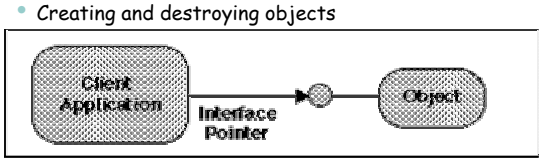
- Components are less abstract than frameworks
  - Frameworks are incomplete applications: They compile, but they don't run
  - Components are usually framework-specific
  - Frameworks make components possible
- Frameworks are less abstract than patterns
  - Include actual code, not just essays
  - Specific to one programming language
  - Specific to one application domain
  - Many patterns came from successful FWs

COM: Common Object Model

- Most widely used component FW on earth
- Microsoft's goals in the early 90's
  - Attracting developers to Windows
  - Combining the speed and power of C++ with the ease of use of Visual Basic
  - Versioning of Windows UI & Services
  - OLE (Object Linking & Embedding) and Clipboard features in Office
- COM introduced in 1995 as OLE 2.0

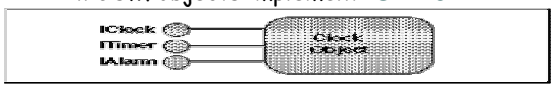
A Binary Standard

- COM objects are used via interface pointers
- Use by normal methods calls
- COM is a binary standard for doing this:
  - calling methods: calling convention, argument passing
  - Common data types, marshalling
  - Creating and destroying objects



Supporting Interfaces

- An object can implement many interfaces
  - All COM objects implement IUnknown



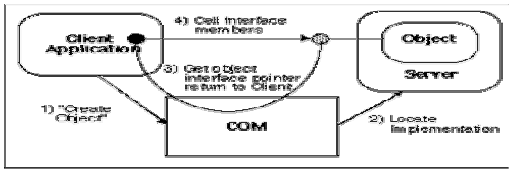
- IUnknown defines QueryInterface()
  - Cast to another interface by giving its ID
  - Implementations can be changed dynamically
- IUnknown also defines AddRef() and Release()
  - COM manages memory by reference counting

Versioning

- COM interfaces are logically immutable
  - Change = New version of interface
  - Not enforced, but required from developers
- For example, each Office version is backward compatible to old interfaces:
  - "Word.Application.7", "Word.Application.8", …
  - Current (last) version is "Word.Application"
- Windows UI is updated in the same way
  - Buttons, Windows, Standard dialogs, …

Creating Objects

- Create objects using the Windows API
- The Registry names a "server" – EXE or DLL – which contains the implementation
- Object is created in server, and the "client" receives a pointer to it



Creating Objects II

- An interface must be uniquely identified
  - CoCreateInstance() receives a CLSID
  - CLSID is a 128-bit globally unique identifier
  - PROGID is easier: "Word.Application.10"
- The Windows Registry

```
\HKEY_CLASSES_ROOT\
  CLSID\
    {000209FF-0000-0000-C000-000000000046}\
        LocalServer32 = C:\Program Files\...\WINWORD.EXE
  Word.Application.10\
    CLSID = {000209FF-0000-0000-C000-000000000046}
```

Creating Objects III

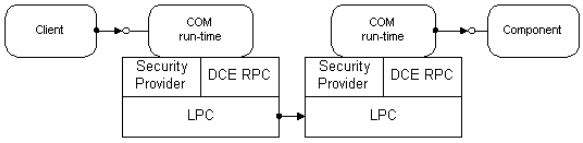
- COM "Servers"
  - EXE or DLL file, with a server entry function
  - DLL can be in-process or out-of-process
  - Server can enquire client & context
  - Request cannot specify an implementation
  - OS manages DLLs and object pools
- Consequences
  - Requires operating system support
  - A dynamic, very expensive operation
  - Works on all versions of Windows since '95

Calling Methods

- Simplest case: In-process DLL
  - "Client" and "Server" in same process
- Calls are dispatched using virtual tables
  - Same as standard C++ (multiple inheritance)
  - Calling convention is also same as in C/C++
  - Should be as efficient as virtual methods
  - In reality slower, due to less optimization

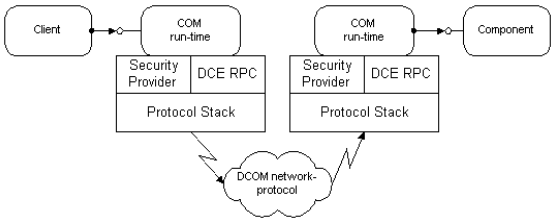
Calling Methods II

- The COM object can be in a different process
  - If it's implemented in an EXE or out-of-process DLL
  - Less efficient calls
  - Shared data between clients
  - Failure of either process won't terminate the other



Calling Methods III

- The COM object can be in a different machine
  - This is DCOM: Distributed COM
  - Transparent: Add network location in Registry
  - Each method call is very expensive



Language Support

- Any language can use COM objects
  - And can be used for writing such objects
- Each language has a syntax for interfaces
  - class in C++, interface in Delphi and Java
- Compiler Support Required
  - Handle COM method calls transparently
  - Replace creation operator by CoCreateInstance()
  - Replace casting with QueryInterface()
  - Call AddRef() and Release() in = operator code
  - Wrap returned error codes with exceptions

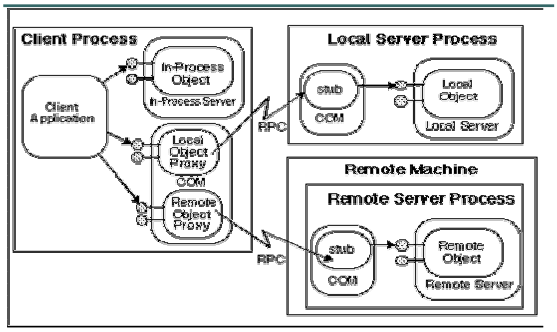
Defining COM Interfaces

- Microsoft Interface Definition Language
  - Interface names, CLSID and properties
  - Method names, argument names and types
  - Properties & events have a special syntax
  - Data Types: Primitives, arrays, COM objects
  - 'in' and 'out' parameters, and other settings
- MIDL Compiler
  - Produces "Type Libraries" (*.TLB Files)
  - Used by compilers & tools that support COM

Writing a COM Server

- Start a new "COM Server" project
  - Development tool generates code for server entry, register and unregister functions
  - Development tool generates empty IDL file
- Define interfaces and implement them
  - Define interface in IDL (or visual tool)
  - Implement in your favorite language
- Build the DLL or EXE
  - Then use *regsvr32.exe* to register it

Calling Methods Summary



Distributed COM

- DCOM is COM's extension to networks
- Location Transparency
- Language Neutrality
- Simple, two-way connection model
  - Easier than a custom socket-based protocol
- Connection Management
  - Shared connection pool between clients
- Distributed garbage collection
  - Efficient ping: per machine, groups all remote object IDs, and piggybacked on messages

Distributed COM II

- Scalability
  - Thread pool, multi-processing
- Protocol Neutrality: UDP, TCP, others
- Flexible deployment & redeployment
  - No server downtime during redeployment
  - Exploits COM's support for versioning
- Referral: passing remote references
  - A directory of waiting remote chess players
  - A load-balancing broker component
  - DCOM automatically short-circuits middleman

Distributed COM III

- Security
  - Transparent: NT's basis & online admin tool
  - Can also be programmatic
  - Any NT security provider – supports encryption and many authentication protocols
- Distributed Components
  - Injecting server code into the client side
  - An infrastructure for load balancing and fault tolerance (hot failover, recovery cache)
  - Used by MTS for distributed transactions

DCOM Design Issues

- Each method call is very expensive
  - Violate command-query separation to minimize network round-trips (=method calls)
  - Also useful for ordinary COM objects
- Stateless components are more scalable
- Transparent proxy & stub model is also useful when objects are a different:
  - Thread
  - Security Context
  - Transaction Context

COM Based Technologies

- All '90s MS technologies for software developers are based on COM
  - Tied to the Windows platform
  - Efficient, scalable, language neutral
- A few examples
  - ActiveX: "applications" inside web browsers
  - MTS and COM+: Transaction services
  - VBA and Windows Scripting
  - DirectX, Internet Explorer, ODBC & ADO, …

The Competition: JavaBeans

- The Java Component Model
  - Component = Can be used in a visual designer
  - A component is a Java class
  - With a default constructor
- Relies heavily on reflection
  - Property = defined by getAbc() and setAbc()
  - Event = same as Swing's event model
- With added information in *java.beans.**
  - Names, version, company, …

The Competition: RMI, EJB

- Remote Method Invocation
  - Java-to-Java transparent remote objects
  - Generates proxy and stub, like DCOM
  - Supports exceptions, unlike (D)COM
  - Not protocol neutral, less admin tools
  - Easier to learn: Java only
- Enterprise JavaBeans
  - Adds distributed transaction services
  - An open standard, with multiple vendors
  - All non-MS big names support it

Making a Choice

- The COM+/EJB issue is a huge economic struggle between giant companies
- The basic criteria:
  - COM based: One OS, language independent
  - Java based: One language, OS independent
- But there's much more to it than that
- Microsoft is now upgrading to .NET
  - Web services: finding components on the net
  - Much improved language inter-operability

Summary: Back to Basics

- What is a component?
  - A software module that can be used by other modules, knowing only its interface
  - Written "inside" a framework
  - Can usually be inserted into a visual designer
- Components are key to large-scale reuse
  - Delphi, COM, JavaBeans reuse is a success
  - Object-oriented methods alone aren't
  - A few leading frameworks lead the industry
  - Write your software as components!