The Microsoft way: COM, OLE/ActiveX, COM+, and .NET CLR (Chapter 15)

Prof. Dr. Wolfgang Pree

Department of Computer Science
University of Salzburg
cs.uni-salzburg.at

UNIVERSITÄT SALZBURG

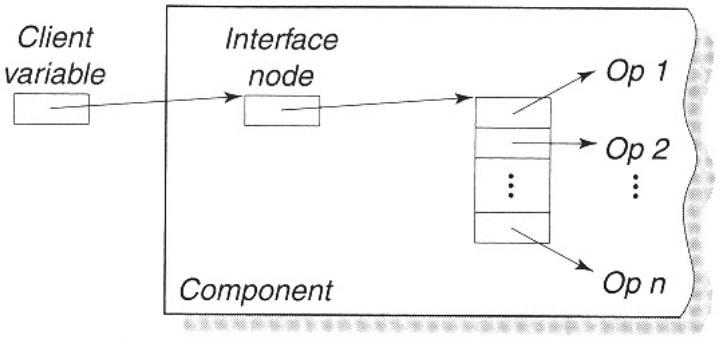
The Microsoft way: COM, OLE/ActiveX, COM+, and .NET CLR

- Microsoft is taking the easiest route.
- Component technology is introduced gradually, gaining leverage from previous success.
- Microsoft aims to establish a new language, C#.
- C# is positioned as CLR model language.
- Contextual composition - first sketched in COM's apartment model, ripened in the Microsoft Transaction Server (MTS), adopted and improved by Enterprise JavaBeans, independently matured in COM+, next adopted and refined in the CORBA Component Model (CCM), and finally taken to an extensible and open mechanism in CLR.
- COM is likely to be of continuing importance for years to come and CLR interoperability with COM is particularly strong.

The first fundamental wiring model – COM (1)

COM

- is Microsoft's foundation on which all component software on its platform is based.
- is a binary standard that does not specify what a component or an object is.
- neither requires nor prevents the use of objects to implement components.
- does define an interface (represented as a pointer to an interface node)



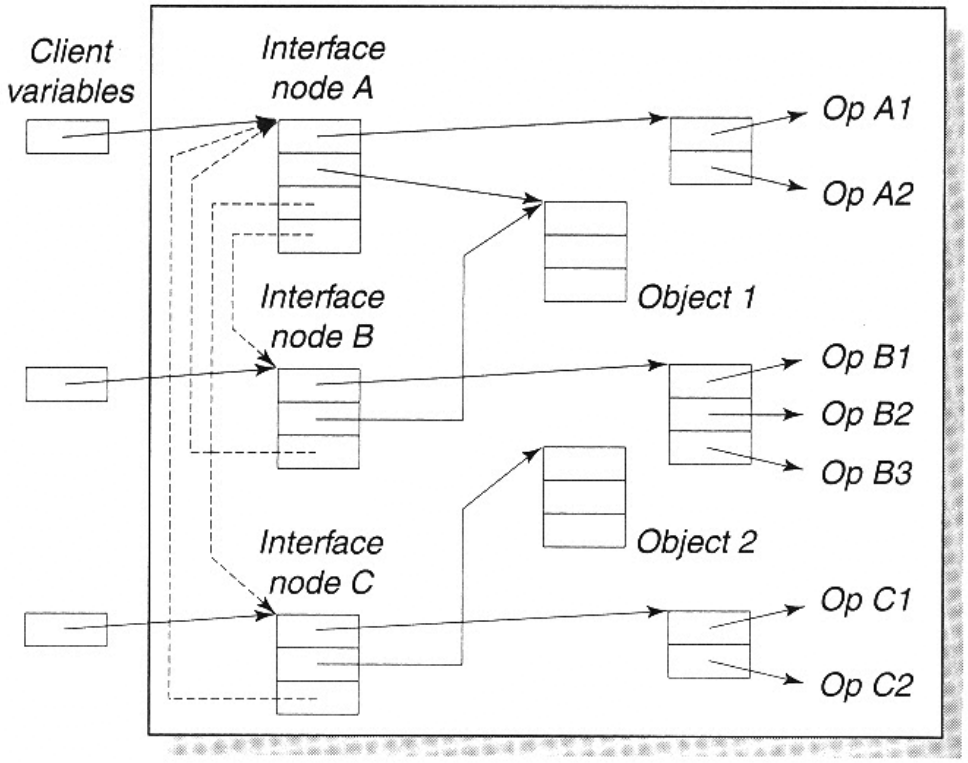
Methods of an object have one additional parameter – the object they belong to. Sometimes called self or this.

The interface pointer is passed as a self-parameter to any of the interface's operations.

UNIVERSITÄT SALZBURG

The first fundamental wiring model – COM (2)

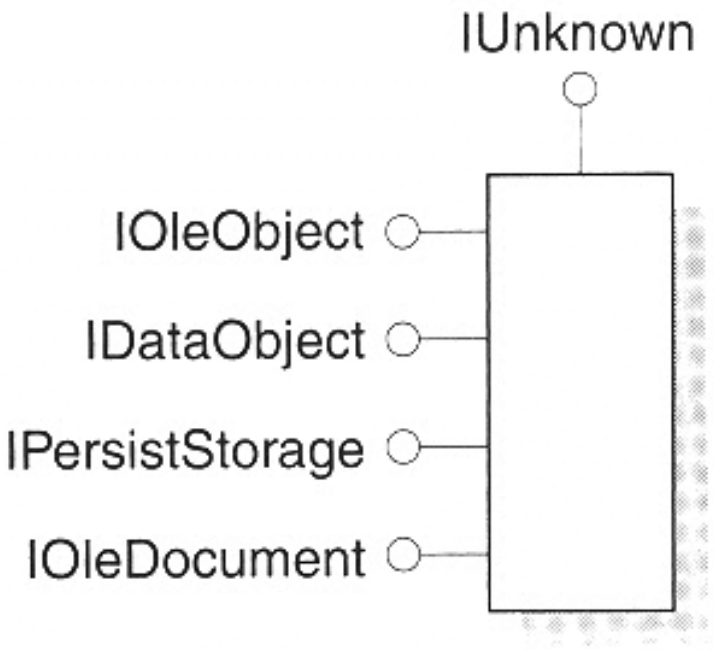
- A COM component
  - is free to contain implementations for any number of interfaces.
  - not necessarily a traditional class.



UNIVERSITÄT SALZBURG

The first fundamental wiring model – COM (3)

- A COM object is not necessarily a traditional single-bodied object.
- Every COM interface has a common first method named *QueryInterface*.
- Every interface has a *QueryInterface* operation.
- All COM objects must have an *IUnknown* interface.



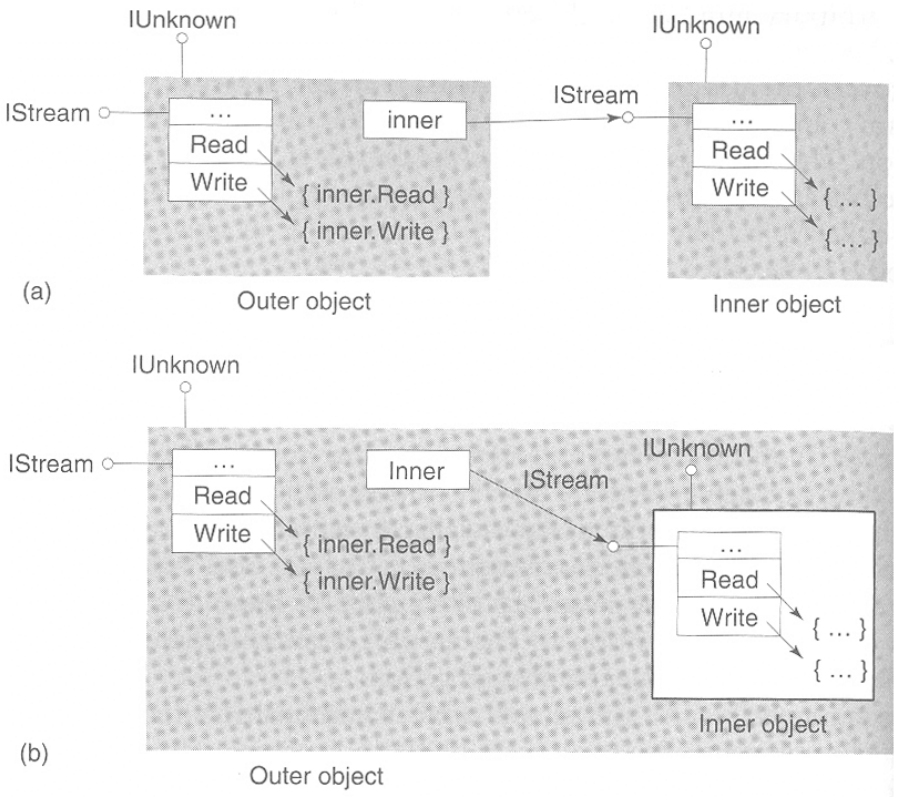
UNIVERSITÄT SALZBURG

COM object reuse

- COM does not support any from of implementation inheritance.
- Lack of implementation inheritance does not mean lack of support for reuse.
- COM does not define how and individual component is internally realized
- COM supports two forms of object composition to enable object reuse called containment and aggregation.

COM object reuse – Containment

- Containment is one object holds an exclusive reference to another.
- Containment is completely transparent to clients of an outer object.



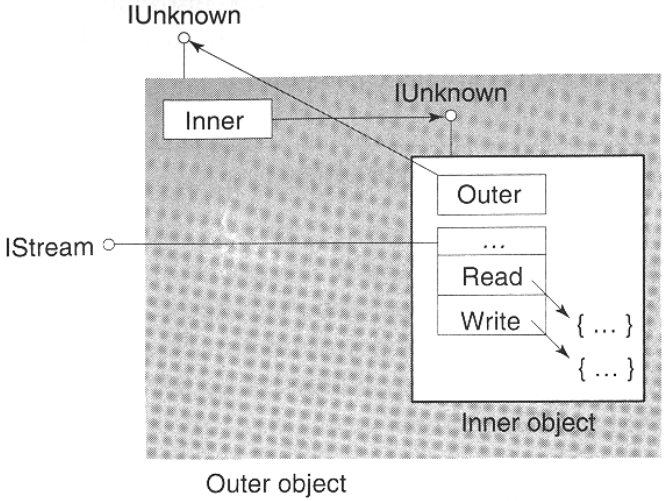
- With containment, the inner object is unaware of being contained.

UNIVERSITÄT SALZBURG

COM object reuse - Aggregation

Aggregation:

- Instead of forwarding requests, an inner object's interface reference could be handed out directly to an outer object's client,
- only useful where the outer object does not wish to intercept calls,
- it is important to retain transparency as a client of the outer object should have no way of telling that a particular interface has been aggregated from an inner object.



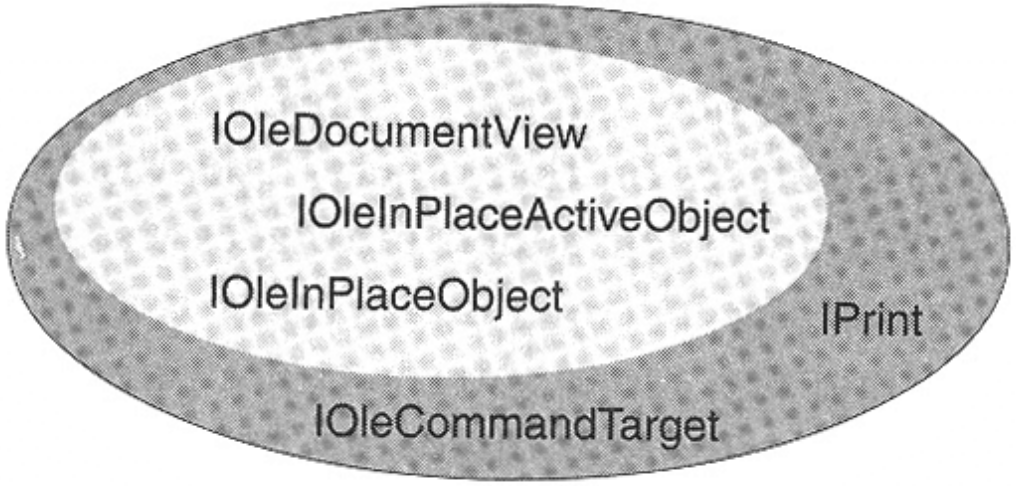
- needs the inner object to collaborate
- can go any number of levels deep.

- can be used to add support for new interfaces on otherwise unchanged objects. The interfaces must not interfere with any of the interfaces on that object.

UNIVERSITÄT SALZBURG

Interfaces and polymorphism

- COM interfaces can be derived from other COM interfaces using (single) interface inheritance.
- Interface inheritance in COM has nothing to do with the polymorphism COM supports.
- The true nature of polymorphism in COM is the support of sets of interfaces by COM objects.



Interfaces and polymorphism – Categories

- To support efficient handling o sets of interfaces, COM defines categories.
- Categories are roughly defined as sets of interface identifiers.
- A COM object can be a member of any number of categories, and categories among themselves are totally unrelated.
- Categories have a contractual nature.
- Currently definition of categories is largely left to Microsoft.

Interfaces and polymorphism – Interfaces and versioning

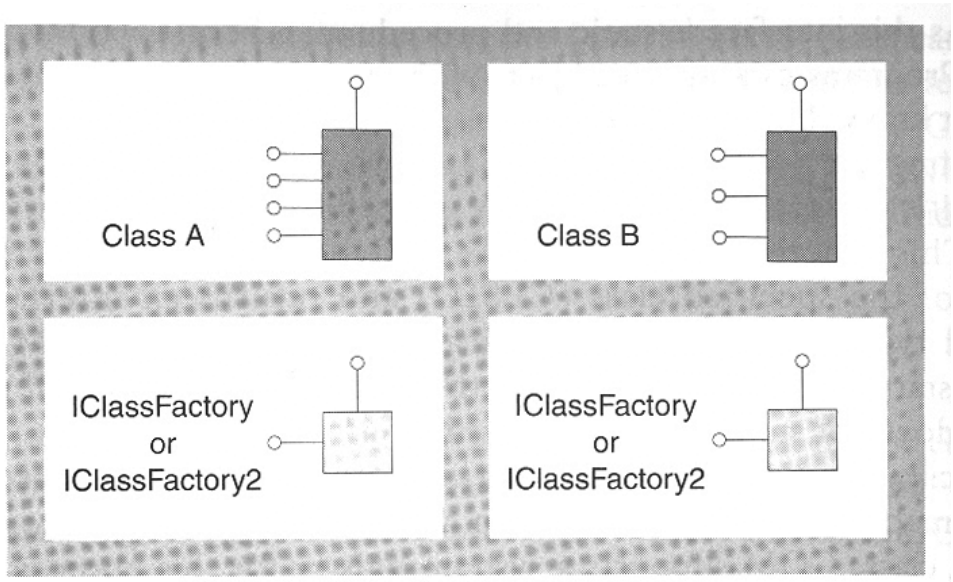
- Once published, a COM interface and its specification must not be changed in any way.
- This addresses both the syntactic and the semantic fragile base class problem by avoidance.
- A component may choose to implement several versions of an interface. These are handled like any other set of different interfaces.
- A COM-based system can concurrently support the old and the new while allowing for a gradual migration.

COM object creation and the COM library (1)

- To identify classes of COM objects, COM defines class identifiers (CLSIDs).
- A CLSID is also a globally unique identifier (GUID).
- To map the given CLSID to an actual component that contains the requested class.
- COM supports a system registry. It specifies which servers are available and which classes they support.
- In-process servers support objects that live in the client's process.
- Local servers support objects on the same machine, but in a separate process.

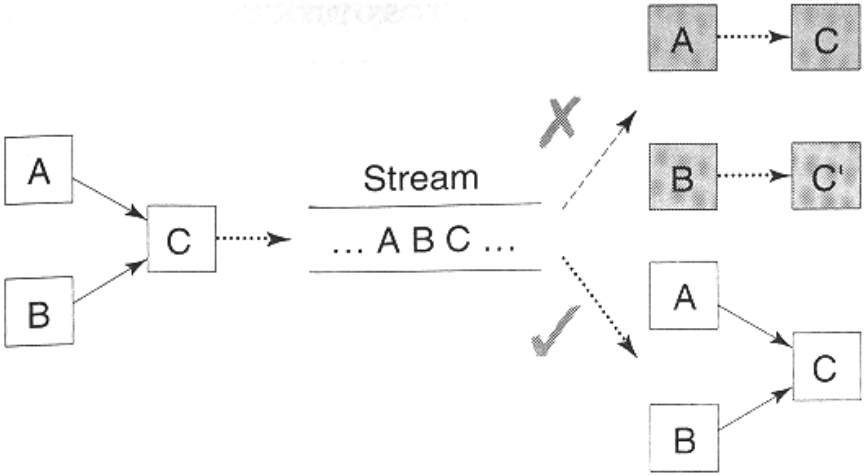
COM object creation and the COM library (2)

- Remote servers support objects on a different machine.
- Ex: Use the CLSID for „rich text.“
- To support generic CLSIDs and enable configuration, COM allows one class to emulate another.
- Emulation configurations are kept in the system registry.



Initializing objects, persistence, structured storage, monikers

- COM uses a two-phase approach to object initialization.
- After creating a COM object, the object needs to be initialized.
- Direct way: ask it ot load its data from a file, a stream, or some other data store.
- COM defines a way to refer directly to a persistent object “by name” and initialize the new object from its source. Such object names are called monikers (nicknames).
- COM does not directly support persistent objects.



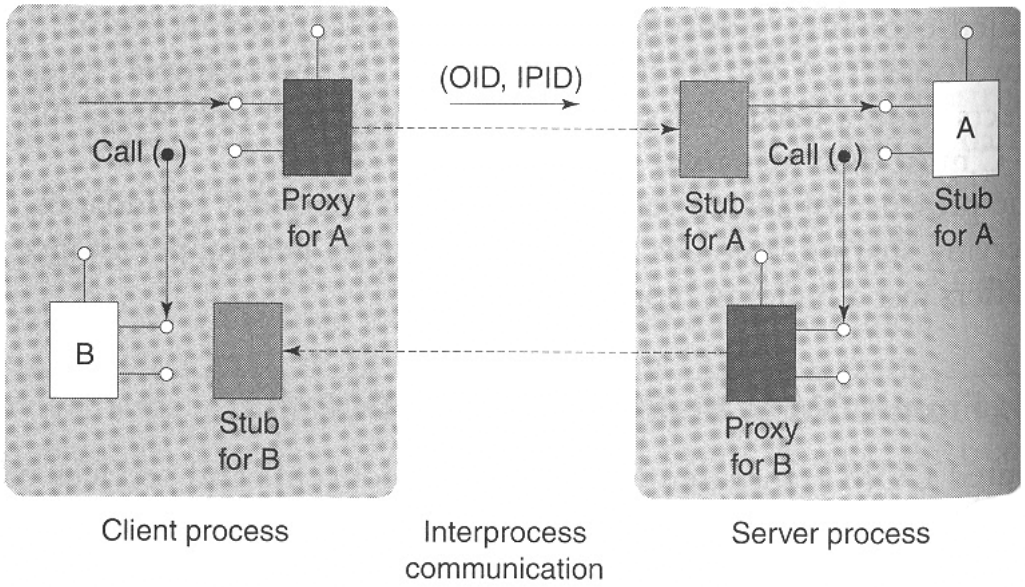
UNIVERSITÄT SALZBURG

From COM to distributed COM (DCOM) (1)

- Distributed COM transparently expands the concepts and services of COM.
- DCOM builds on the client-side proxy objects and the server-side stub objects already present in COM.
- To support transparent communication across process boundaries or across machine boundaries, COM creates proxy objects on the client's end and stub objects on the server's end.
- An interface reference sent across process boundaries needs to be mapped to an object reference that retains meaning across process boundaries.

From COM to distributed COM (DCOM) (2)

- With difference in data representations, DCOM marshals data into a representation called network data representation (NDR), a platform-independent format.
- In addition to low-level machinery to connect COM objects across machine boundaries, DCOM also provides higher-level mechanisms to speed up remote operations, provide security, and detect remote machine failures.



UNIVERSITÄT SALZBURG

Meta-information and automation

- COM does not require the use of a specific interface definition language, as it really is a binary standard.
- COM uses type libraries to provide runtime type information for interfaces and classes. Each type library can describe multiple interfaces and classes.
- Available, from the registry, are the categories to which a class belongs.

Uniform data transfer

- Uniform data transfer allows for the unified implementation of all sorts f data transfer mechanisms.
- Source and target need to agree on a number of things for such a transfer to work and make sense. This agreement is based on a mutually understood data format and a mutually agreed transfer medium.
- Uniform data transfer defines a number of standard data formats.

Dispatch interfaces (dispinterfaces) and dual interfaces

- Dispatch interfaces (dispinterfaces) have a fixed number of methods defined in the interface *IDispatch*.
- Dispinterfaces have one principal advantage – they always look the same – easy to implement services that generically forward or broadcast dispinterface calls.
- Several disadvantages:
  - performance penalty
  - restrict dispatch operations to parameters of a limited set of types, and to a most one return value
  - introduce considerable complexity per interface implementation.

Outgoing interfaces and connectable objects (1)

- An outgoing interface is an interface that a COM object would use (rather than provide) if it were “connected” to an object that provides this interface.
- Outgoing interfaces support the registration of other objects that wish to “listen” for notifications. Beyond this use, outgoing interfaces can also be used to realize configurable required interfaces as used in connection-oriented programming.
- To become a full connectable object, a COM object has to declare outgoing interfaces.

Outgoing interfaces and connectable objects (2)

- A connection is established by passing an interface reference of another object to the connection point.
- Connectable objects provide a uniform way to implement change propagation. As outgoing and incoming interfaces are matched the propagation can take the form of regular method invocations instead of requiring the creation of event objects.
- More lightweight alternatives are the event source and listener approaches in JavaBeans or the language-level support in C#.

Compound documents and OLE (1)

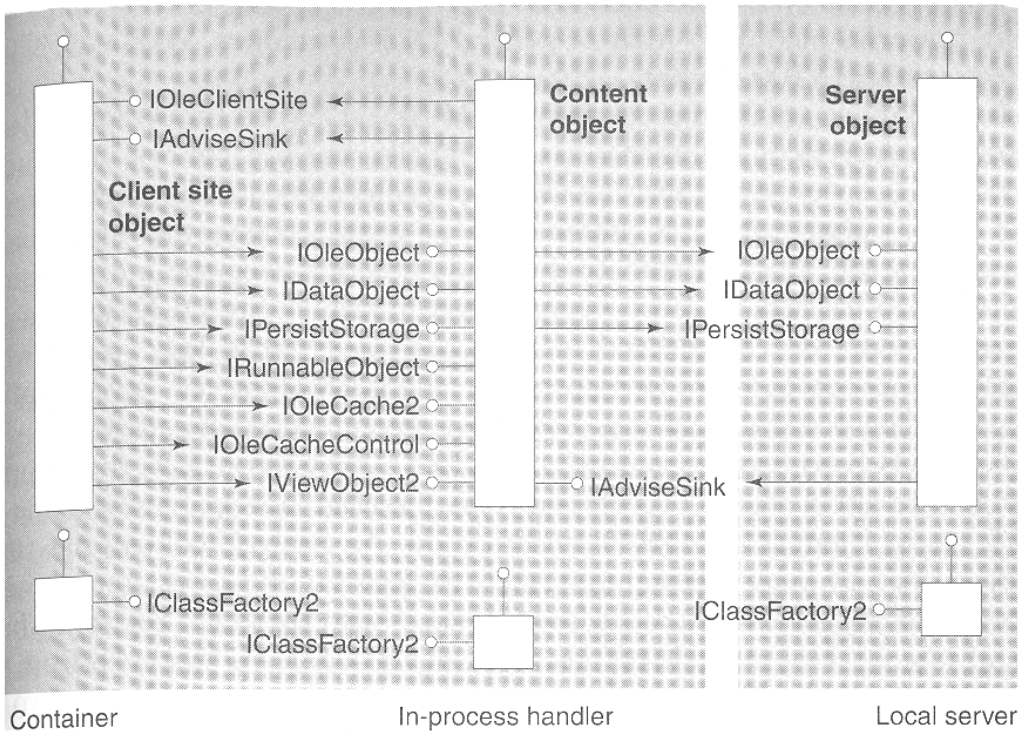
- Object linking and embedding (OLE) is Microsoft's compound document standard. Created to blend legacy applications, with their own application-centric view of the world, into a single document-centric paradigm.
- OLE can be summarized as a (large) collection of predefined COM interfaces.
- The OLE compound document's approach distinguishes between document containers and documents servers.
- Besides embedding, OLE supports linking of document parts.
- Linking rests on monikers, a container storing a moniker to the linked object. The linked object advises the container of changes.

Compound documents and OLE (2)

- A document server provides some content model and the capabilities to display and manipulate that content.
- A document container has no native content, but can accept parts provided by arbitrary document servers.
- Many document containers are also document servers – that is, they support foreign parts but also have their native content.

OLE containers and servers

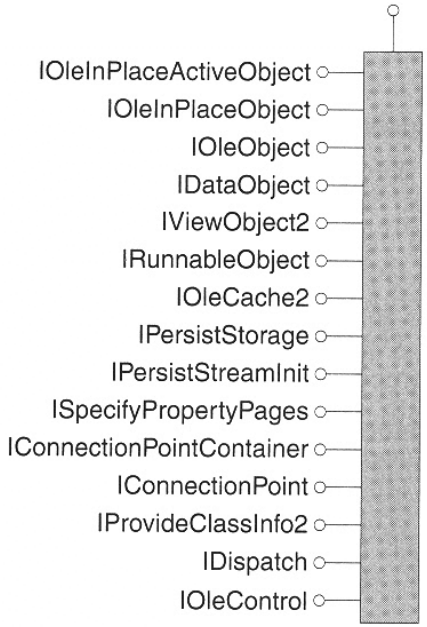
- The interaction of containers and servers is complex by nature.
- The interaction between document containers and document servers is governed by two interfaces provided by a containers client site object and seven interfaces provided by a server's content



UNIVERSITÄT SALZBURG

Controls – from Visual Basic via OLE ActiveX

- Visual Basic controls (VBXs) - the first successful component technology released by Microsoft.
- Visual Basic uses a simple and fixed model in which controls are embedded into forms.
- A form binds the embedded controls together and allows the attachment of scripts that enable the controls to interact.
- To qualify as an OLE control, a COM object has to implement a large number of interfaces.
- ActiveX controls
  - are COM objects supported by a special server.
  - have regular COM interfaces + outgoing interfaces.
  - have properties (settings that a user or application assembler can use to fine-tune looks and behavior) Containers can also have properties
- ActiveX defines a number of interfaces that can be used to handle properties.
- An ActiveX container is an OLE container with a few additional properties.



UNIVERSITÄT SALZBURG

Contextual composition and services

- Microsoft offers a number of key services that build on DCOM directory service.
- The Windows Registry.
- Common property of the services is their contextual binding.

COM apartments – threading and synchronization (1)

- Apartments are, in many ways, the most surprising and most commonly misunderstood feature of COM.
- A “hack” to make things work in a world that started out with single threading (the original Windows API).
- The idea is to not associate synchronization with individual objects.
- Synchronization is associated with synchronization domains, apartments.
- A process is partitioned by apartments and each apartment can have its own synchronization regime.
- When instantiating new objects, a program can either just rely on the constraints on the instantiated class or take some control of how objects are inserted into apartments.

COM apartments – threading and synchronization (2)

- A single-threaded apartment services all contained objects with a single thread.
- A rental-threaded apartment restricts execution to a single thread at a time, but as one thread leaves another can enter.
- Free-threaded apartment allows for any number of concurrent threads to coexist.
- Synchronization actions are inserted automatically by the COM infrastructure at apartment boundaries

Microsoft transaction server – contexts and activation

- The transaction server supports online transaction processing of COM-based applications.
- Currently does not address fault tolerance issues beyond the properties of transactions.
- Transparent addition of transactional capabilities to existing COM components.
- The server automatically detects references to other COM objects by a transactional component and extends transactional handling to these as well.
- Component developers need to be aware of transactions to keep exclusive locking of resources to a minimum.

COM+ - generalized contexts and data-driven composition (1)

- COM+, an extension of COM.
- COM+ 2.0 was used as the target name for the .NET Framework.
- COM+ combines MTS and MSMQ declarative attributes with several new ones, leading to the following list of application-level attributes.
  - Activation type: library (in process) or server (separate process).
  - Authentication level: none, connect, call, packet, integrity, or privacy.
  - Authorization checks: application-only or application-and-component.
  - Debugger: command line to launch debugger.
  - Enable compensating resource manager: on or off.
  - Enable 3GB support: on or off.
  - Impersonation level: identify, impersonate, or delegate.
  - Process shutdown: never or *n* minutes after idle.
  - Queueing: queued or queued with listener.
  - Security identity: interactive user or hard-coded user/password.

COM+ - generalized contexts and data-driven composition (2)

- At the component level, the following attributes are supported.
  - Activation-time load balancing: on or off.
  - Auto-deactivation: on or off.
  - Declarative authorization: zero or more role names.
  - Declarative construction: class-specific string.
  - Instrumentation events: on or off.
  - JIT activation: on or off.
  - Must activate in activator's context: on or off.
  - Object pooling: on (min. and max. instances, timeout) or off.
  - Synchronization: not supported, supported, required, reqires new.
  - Transaction: not supported, supported, required, requires new.
- The component-level attributes apply to classes, except for declarative authorization, which also applies to interfaces and methods, and auto-deactivation, which only aööies to methods.
- New with COM+, an attribution model is provided that allows for the automatic mapping between procedural invocations and message queuing.

Take two – the .NET Framework (1)

- The .NET Framework is part of the larger .NET space. It comprises the common language runtime (CLR), a large number of partially interfaced, partially class-based frameworks, packaged into assemblies, and a number of tools.
- CLR is an implementation of the common language infrastructure (CLI) specification, adding COM+ interoperation and Windows platform access services.
- CLR offers dynamic loading and unloading, garbage collection, context interception, metadata reflection, remoting, persistence, and other runtime services that are fully language independent.

Take two – the .NET Framework (2)

- Assemblies are the units of deployment, versioning, and management in .NET – that is, they are the .NET software components.
- Side-by-side use of the same assembly in multiple versions is fully supported. Assemblies contain metadata, modules, and resources, all of which are expressed in a platform-independent way.
- CLR either compiles at install- or at load-time, always executing native code.
- CLR reflection and other type-based con At the components cover a large type system space called CTS (common type system).

The .NET big picture

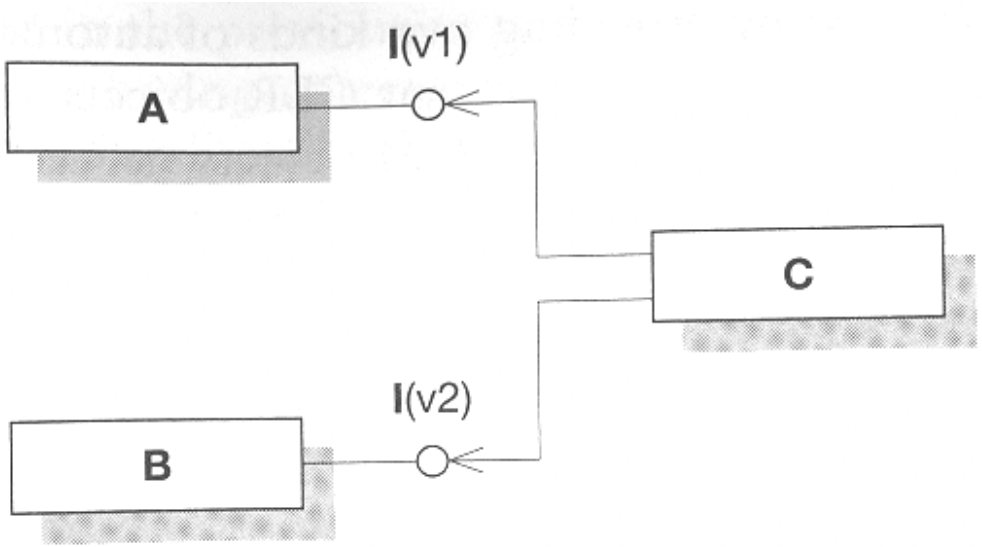
- .NET targets three levels:
  - Web services;
  - Deployment platforms (servers and clients);
  - Developer platform.
- Web services aim for transitive programmability of the internet.
- Microsoft plans to make available a number of foundational core services. .NET Passport, .NET Alerts.
- Like JVM, CLR defines a virtual instruction set to isolate from particular processors.

Common language infrastructure (1)

- CLI defines an intermediate language (IL) and deployment file format (assemblies), such as Java bytecode, class, and JAR files.
- CLI includes support for extensible metadata.
- The common language runtime (CLR) is the Microsoft implementation of the CLI specification.
- CLR goes beyond CLI compliance and includes support for COM and platform interoperation.
- CLO comprises the specification of execution engine services (such as loader, JIT compiler, and garbage-collecting memory manager), the common type system (CTS), and the common language specification (CLS).
- CTS and CLS play two complementary roles. The CTS scope is the superset of many languages' core concepts in the type space.
- CLS is a strict CTS subset that is constructed in such a way that a wide variety of languages can cover it completely. If a definition is CLS-compliant, then any language classified as a CLS consumer will be able to make use of that definition.

Common language infrastructure (2)

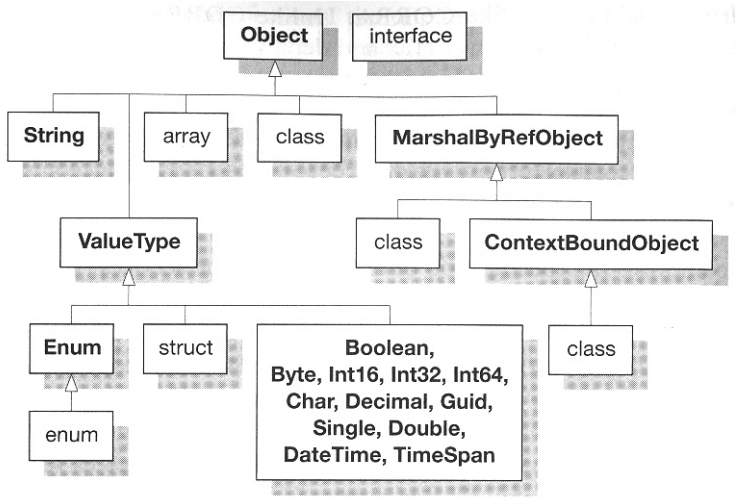
- A language that can also introduce new definitions in the CLS space is called CLS producer. Finally, a language that can extend existing definitions in the CLS space is called a CLS extender.
- CLS extenders are always also CLS producers and CLS producers are always also CLS consumers.



UNIVERSITÄT SALZBURG

Common language infrastructure (3)

- There are no primitive types, so types such as integer or floating point are merely predefined value types.
- Multiple interface and single class inheritance relations are supported.
- Even value types can inherit (implement) multiple interfaces.
- Accessibility is controlled by the definition and use points
  - in the same location
  - related by class inheritance.



UNIVERSITÄT SALZBURG

Common language infrastructure (4)

- Methods can be static, instance-bound, or virtual (which implies instance-bound).
- Overloading is supported on the basis of method names and signatures, but not return types.
- The overload resolution policies vary from language to language.
- A class can implement multiple interfaces and can qualify method names which the name of the introducing interface.
- CTS anchors all names of definitions in the names of their containing assemblies.
- Various method naming conventions, such as property and indexer get-set methods are part of CTS.
- Throwable exceptions are not part of CTS method signatures. CTS has no provisions for static checking or throwable exceptions when calling a method.

COM and platform interoperation

- CLR includes substantial support
  - for COM interoperation via wrappers
    - COM callable wrappers (present CLR objects via COM interfaces)
    - Runtime callable wrappers (present COM objects via CLR interfaces)
  - direct access to the underlying platform with minimal overhead (e.g. JIT compiler) => almost optimal performance

Exemplary .NET language – C# (1)

- C# is an object-oriented language that sits somewhere between Java and C++.
- C# provides direct support for most – but not all – CLR features, some of which are unique for CLR.
- The C# language specification is an ECMA standard, in parallel with the CLI specification (ECMA 2001a/b).
- C# distinguishes five kinds of top-level definitions – interfaces, classes (CTS reference classes), structs (CTS value classes), enums (CTS enumeration classes), and delegates (CTS reference classes).
- C# organizes names into namespaces. Namespaces are orthogonal to assemblies. A namespace can span multiple assemblies. An Assembly can add definitions to multiple namespaces. Mapping from names in namespaces to names in assemblies is established at compile-time.

Exemplary .NET language – C# (2)

- Unlike Java reliance on naming patterns, C# includes properties.
- Property
  - abstract field that has a get method, set method, or both.
  - can be defined on interfaces, classes, and structs.

```
interface IColored {
  System.Drawing.Color Color { get; set; }
}
class SampleGadget : IColored {
  System.Drawing.Color color;
  IColored.Color {
     get { return this.color; }
     set { this.color = value; }
  }
}
class UseGadget {
  SampleGadget gadget = new SampleGadget();
  gadget.Color = new System.Drawing.Color.RoyalBlue;
  ...
}
```

Exemplary .NET language – C# (3)

- By not providing a getter or a setter, a property can be made write- or read-only.
- By abstracting a field as a property, proper action can be taken whenever the property value is accessed.
- When accessing a property, most field operations work as if a property were a field.
- Properties do not have an address and thus cannot be passed by reference.

Exemplary .NET language – C# (4)

- C# supports the abstraction of arrays by indexers.
- Indexer
  - can be defined on interfaces, classes and structs.
  - has not name, there can be at most one per interface, class or struct.
  - can have a getter, setter, or both.
  - takes any number of additional arguments that are meant to correspond to the indices of array access expressions.
  - can be used to abstract semantic arrays

```
interface IMapStrings {
  string this [string key] { get; set; }
}
using System;
class StringMapper : IMapStrings {
  Hashtable h = new Hashtable();
  string IMapStrings.this [string key] {
    get { return (string) h[key]; }
    set { h.Add(key, value); }
  }
}
```

Exemplary .NET language – C# (5)

- C# includes direct support for a simple form of connection-oriented programming – the “wiring” of method-level handlers for *events*.
- Every event source is multicast enabled and accepts the registration of any number of listeners.
- A listener is a delegate on a method of matching signature.
- If more than one delegate is registered with an event source, events are multicast. The ordering of event delivery is not specified.
- Unlike delegates, which are special CTS types, events are mere syntactic sugar introduced by C#.
- C# fully supports CLI custom attributes – user-defined meta-attributes attached to interfaces, classes structs, fields, methods, parameters, and so on.

Visual Studio .NET

- Visual Studio .NET (VS.NET) integrates most tools for the .NET Framework. Can support editing, building, and debugging of applications – all across multiple languages.

Assemblies – the .NET software components (1)

Assembly

- CLI deployment units
- set of files in a directory hierarchy (similar with a JAR file) that may be split into module and resource files
- contains a manifest file (table of contents)
- may have satellite assemblies for localization that contains everything required for a specific culture (typically translated text material, but may also contain forms and other user-interface elements that require adjusting) and otherwise defaults to its main assembly.

Assemblies – the .NET software components (2)

- either private to a single application or shared among multiple applications.
- shared assemblies equipped with policies are stored in the Global Assembly Cache
- policies can be used to provide versioning support (backwards compatibility)

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="myAssembly"
                          publicKeyToken="32ab4ba45e0a69a1"
                          culture="en-us" />
          <bindingRedirect oldVersion="1.0.0.0" newVersion="2.0.0.0"/>
        </dependentAssembly>
      </assemblyBinding>
    </runtime>
</configuration>
```

Common language frameworks

- On top of CLR, the .NET Framework provides a large collection of frameworks.
- Some frameworks emphasize customization by subclassing; others provide interfaces for that purpose. The frameworks make use of sealed classes to close the design in cases where the intricate nature of the implementation would make valid overriding very difficult.

AppDomains, contexts, reflection, remoting (1)

- The CLR execution engine partitions a process into one or more AppDomains (application domains). An AppDomain isolates sets of objects from all objects in other AppDomains, but is more lightweight and thus cheaper than typical operating system processes.
- The unit of execution is a logical thread that is mapped to a physical thread whenever entering an AppDomain.
- AppDomains are also the scope of loading and unloading.
- Finer structuring of execution spaces beyond AppDomains is provided by contexts. A context is a partition of an AppDomanin the member objects of which share the properties of their context.

AppDomains, contexts, reflection, remoting (2)

- C# supports a lock statement to acquire a lock for the duration of a block (similar with Java's synchronized statement)
- Synchronized blocks are a very fine-grained construct.
- Synchronized methods are in the middle - they lock an entire object for the duration of a method call.
- Synchronizing contexts lock all objects that share the same synchronizing context

AppDomains, contexts, reflection, remoting (3)

- A CLI object is pre-classified as either context agile or context bound.
- Context programming use CLI's custom attributes on the class of a context-bound object.
- The interaction semantics of context-bound and context-agile objects forms an unique CLI property.

```
using System;
public delegate void UpdateEventHandler(object sender);
[Synchronization(IsReEntrant=true)]
public class SynchSample : ContextBoundObject {
  private int sampleVar;
  public event ValueChange;
  public int SampleProperty {
    get { return sampleVar; }
    set {
      sampleVar = value;
      if (ValueChange != null) ValueChange(this);
    }
  }
}
```

AppDomains, contexts, reflection, remoting (4)

- Context-agile objects can be used by context-bound ones as if the agile objects resided in the same context.
- CLO reflection
  - supports grants full access to the type structure of assemblies, including all attributes and custom attributes defined on these types.
  - enables creation of instances of these types and invocation of methods on these types.
- The CLI remoting support combines context and reflection infrastructure with flexible support for proxies, channels, and messages to provide building blocks from a wide variety of communication styles and patterns.

Windows Forms, data, management

- Windows Forms is the .NET Framework family that enables the construction of Windows applications using CLR-hosted managed code.
- Basic model of Windows Forms: component classes.
- Component classes support having a site, which is a helper object supplied by a container object.
- Sites are used to attach properties to component objects and enable a contained object to access its container.
- Windows Forms uses properties, events, and delegates to support connection-oriented programming.

Web Forms, Active Server Pages (ASP) .NET (1)

- Web Forms
  - a framework to build dialogs within ASP.NET.
  - Works by “rendering” to DHTML and relying on a remote browser to display forms and interact with users.
  - uses the same connection-oriented programming model as Windows Forms.
  - the framework fully abstracts the capture of an event of the client and its transmission to the server. Registered handlers are invoked as if the event was originated locally.
  - Programming model presents a form as a logical unit, despite the split into client versus server side.
- Web Forms and Windows Forms follow the same programming model and abstraction.

Web Forms, Active Server Pages (ASP) .NET (2)

- The ASP.NET approach is a departure from the older ASP model.
- With ASP.NET page classes the entire page is turned into an object the output of which is HTML. The page object goes through a series of stages – initialize, process, and dispose.
- A final stage after regular processing and before disposal, a page object renders itself as an HTML stream. ASP.NET pages and page classes are similar to Java ServerPages (JSP) pages and Java servlets, respectively.

XML and data

- The world of data access and processing is now predominantly split into two halves – normalized, structured data in relational tables and semi-structured, self-describing data in XML trees. Second dimension is the distinction between online and off-line data access.
- Access to and manipulation of relational data is handled by a part of the framework called ADO.NET.
- ADO.NET supports ODBC and OLEDB to access relational data sources.
- Access to and manipulation of XML data is handled by another part of the framework. The framework supports validation against XML Schema, handling of XPath, XSL, and XSLT.

Web services with .NET

- Web services with their WSDL-defined ports are similar to interfaces with methods. The .NET Framework allows lightweight implementations of (simple) web services.
- There is support for a service implementer to explicitly deal with asynchronous calls.
  - File IO, Stream IO, Socket IO;
  - Networking – HTTP, TCP;
  - Remoting channels (HTTP, TCP) and proxies;
  - XML web services created using ASP.NET;
  - ASP.NET Web Forms;
  - Messaging message queues over MSMQ;
  - Asynchronous delegates (which can be requested on any delegate type-compilers and CLR cooperate to synthesize the asynchronous *Begin…* and *End…* methods with strongly typed signature.)