Architecture of the CORBA Component Model

Overview over the .NET

Common Language Infrastructure

.NET

- Microsoft Branding Label
- .NET framework
- .NET framework SDK
- Common Language Infrastructure
- .NET class library
- Development environments
- Programming Languages
- Why here?
  - Component framework
  - Standardization
  - Distributed computing

.NET Framework

- Portable execution environment („a new computing platform“):
  - supports multiple programming languages
  - supports multiple operating systems
  - alternative implementations
- Common Language Infrastructure defines common type system, intermediate language
  - safe execution
- standard library
  - access to local OS features (including GUI – winforms)
  - access to various networking technologies (core networking, ..., web services)
- Standard package format (assembly) to support deployment
  - code signing and versioning to avoid „DLL hell“

.NET Framework SDK

- Development environment for developing .NET applications
- C# compiler (csc[.exe])
- Class library documentation
- Support tools

Common Language Infrastructure (CLI)

- International Standard (ECMA 335, ISO 23271)
- Specifies Common Language Runtime, Class Library
- Multiple implementations:
  - .NET Framework
  - Shared Source CLI (SSCLI) (code name ROTOR)
  - GNU Mono
  - DotGNU

Common Language Runtime (CLR)

- Runtime functions:
  - memory management (garbage collection)
  - thread execution
  - code safety verification (various degrees of trust: local or remote code)
  - compilation
  - ...
- Common Type System (CTS):
  - self-describing code (allows introspection)
  - Programming languages map to the CTS
  - Inheritance from System.Object
- Code is stored in Common Intermediate Language
  - typically executed through JIT compilation to native code (.NET, Rotor, (Mono))

.NET Framework Class Library

- Standard System libraries:
  - Collections, Configuration, Diagnostics, Globalization, IO, Net, Reflection, Resources Security, ServiceProcess, Text, Threading, Runtime, InteropServices, Remoting, (Serialization)
- Microsoft Extensions:
  - System.Data (ADO.NET): ADO, SQL, Design, Adapters
  - System.XML: XSLT, XPath, Serialization,
  - System.Drawing
  - System.Web (ASP.NET)

Development Environments / Implementations

- Visual Studio .NET:
  - GUI development
  - Managed C++ (C++ for .NET)
    - managed and unmanaged code
  - Visual Basic.NET
  - J#
  - Support tools
- Rotor
  - C#, J#
  - Windows, Mac OS X, FreeBSD/i386
- Mono
- DotGNU

Programming Languages

- C#
  - object-oriented language
  - type-safe
  - similar to C and C++
- Visual Basic .NET
- C++
- J#
- Eiffel, COBOL, Oberon, APL, Fortran
- Mondrian, Haskell, Mercury

Remoting

- Infrastructure for distributed computing in .NET
- Based on the notion of interfaces
- Bridges between „Application Domains“
  - in a single operating system process
  - across process boundaries
  - across machine boundaries
- Copies vs. References
  - Objects inheriting from MarshalByRefObj are remotely accessible
  - Parameters are marshalled either by reference or by value
- Channels
- Object Activation and Lifetime

Copies vs. References

- Call by reference: Inheritance from System.MarshalByRefObj
- Copying requires support for serialization
  - implementation of the System.Runtime.Serialization.ISerializable interface
  - decoration with the Serializable attribute
- Automatic serialization:
  - serialize all members that are not decorated with the NonSerialized attribute
  - raises SerializationException if non-serializable object is encountered

Channels

- objects that transport messages
- .NET supports two kinds of channels:
  - HttpChannel
  - TcpChannel
- Formatters determine on-the-wire representation of serialized objects:
  - BinaryFormatter
  - SoapFormatter
  - automatically chosen depending on the channel

Activation

- Process of instantiating implementation object
- can be initiated from either client or server side
- Server-side activation
  - client calls Activator.GetObject, activation is delayed until first invocation
  - two activation modes
    - Singleton: a single instance services all clients
    - SingleCall: a new instance is created for every call
- Client-side activation:
  - Client explicitly requests new instance through Activator.CreateInstance

Life-time

- SingleCall: implementation object only lives for the duration of the call
- Singleton, client-activated: distributed garbage collection through life time manager
- Clients register interest in object
- Life-time manager checks regularly whether clients are still interested
- If no client is interested, life-time of the implementation object ends
- Life time is configurable: implementation can define frequency of check etc.