Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Terms & Concepts

Seminar Component Programming & Middleware

Operating Systems & Middleware Group

Component vs. Object

Component

- unit of independent deployment
- unit of third-party composition
- has no (externally) observable state

Object

- unit of instantiation (unique identity)
- state (externally observable)
- encapsulate state & behavior

[Szyperski, Component Software, 1999]

Components

Implications:

- needs to be well-separated from its environment and from other components
- components are never partially deployed
- cannot be distinguished from copies of themselves
- at most one copy of a particular component per process

Objects

Implications:

- cannot be partially instantiated
- nothing but an object's abstract identity remains stable over time
- need a construction plan (state space, initial state, and behavior)
  - constructor, clone, prototype

Interfaces

define access points

- let clients access the component's services
- components may have multiple interfaces
- each access point may provide a different service

interface specifications have contractual nature

- mutual ignorance between component and clients
- standardized contract

Aspects of Scale & Granularity

Components are units of

- abstraction, analysis, extension
- locality, fault containment
- compilation, deployment, installation, loading
- delivery, accounting, dispute
- maintenance, system management

[Szyperski, Component Software, 1999]

Component Frameworks

similarities:

- component transfer format (jar, cab, dll)
- meta-information
- uniform data transfer, events, channels

differences:

- binary format & wire protocol
- memory management, life-cycle, GC
- exceptions/exception handling
- persistence

Summary

- components
- objects
- component framework

Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Microsoft .NET Framework

Seminar Component Programming & Middleware

Operating Systems & Middleware Group

Conceptual Overview

- object-oriented programming environment
- code-execution environment

NET Framework =

Common Language Runtime

+ .NET Framework Class Library

first published Feb 2002

Conceptual Overview

- Common Language Infrastructure (CLI)
  - developed by Microsoft
  - ECMA-335, ISO/IEC 23271
- Common Type System (CTS)
- Virtual Execution Environment (CES)
- Common Language Specification (CLS)
- Metadata
- Common Intermediate Language (CIL)

Conceptual Overview

- .NET Framework
  = implementation of CLI
- Common Language Runtime
  = implementation of VES
- .NET Framework Class Library
  = implementation of Standard Libraries

Conceptual Overview

Common Language Runtime (CLR)

- memory management, thread execution, code execution, code safety, compilation, …

Common Type System (CTS)

- defines how types are declared, used, and managed
- set of data types and operations
- important for cross-language integration

Common Language Specification (CLS)

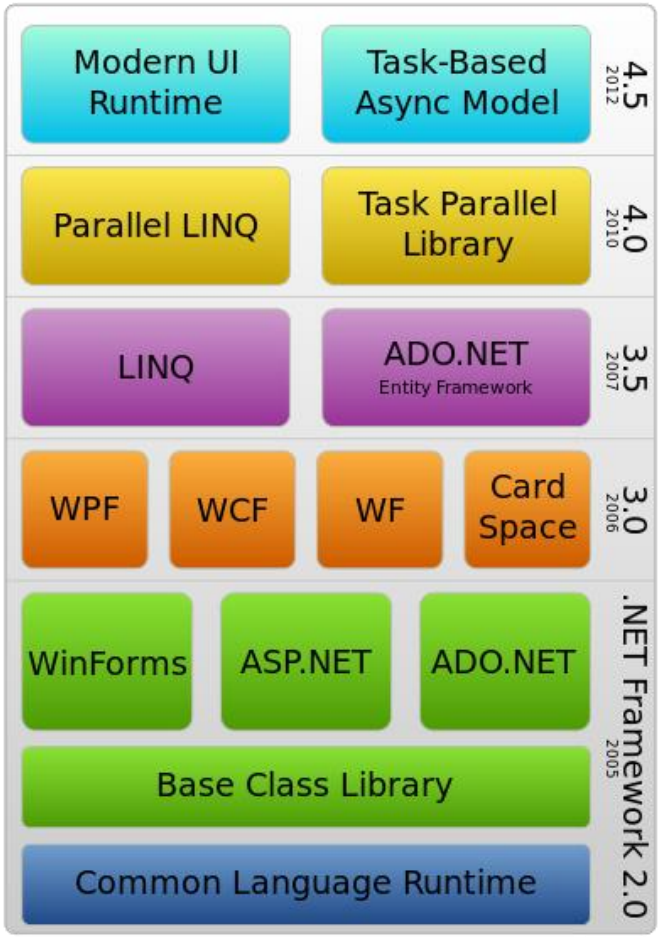
- CLR or Mono are implementations of CLS
- CLS compliance

Managed Execution Process

1. choose compiler (C#, F#, VB.NET, …)
2. compile code to MSIL
   - CPU independent set of instructions
   - +metadata
3. compile MSIL to native code
   - just-in-time or install-time compiler
   - code security verification
4. run code

Version History

- CLR: 1.0, 1.1, 2.0, 4
- .NET Framework:
  1.0, 1.1, 2.0, 3.0, 3.5, 4, 4.5



The .NET Framework Stack

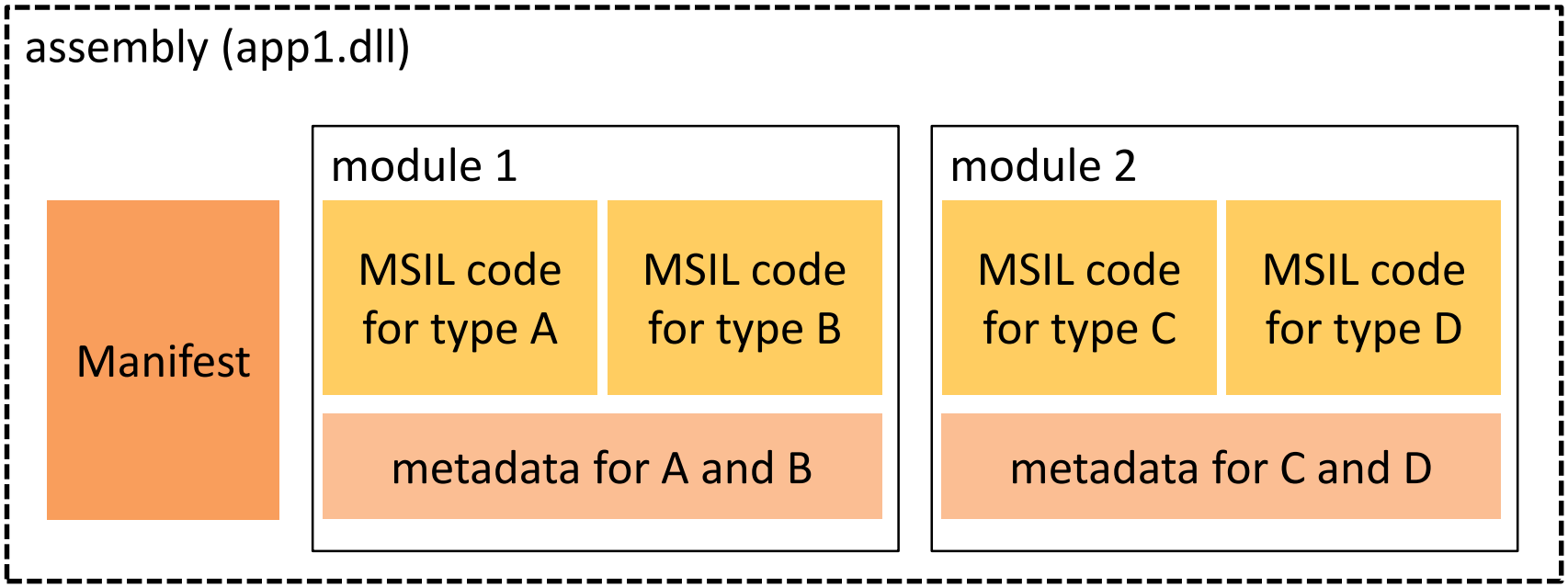
[CC BY-SA 3.0, Soumyasch, https://commons.wikimedia.org/wiki/File:DotNet.svg]

Types

- classes, structures, enumerations, interfaces, delegates
- type members:
  - fields, properties, methods, constructors, events, nested types
- value types & reference types
- assemblies vs. namespaces

Assemblies

- assembly = logical unit (not physical)
- manifest, type metadata, managed code, resources



Assemblies

function as:

- unit of deployment
- type boundary
- security boundary
- reference scope boundary
- version boundary
- unit of side-by-side execution

can be

- static (*.dll, *.exe)
- dynamic (CodeDOM, Reflection.Emit)

Binding to Assemblies

- assembly name, version, culture, public key, and digital signature
- strong name & signing
- Global Assembly Cache (GAC)
- metadata & policy

Metadata & Self-Description

description of assembly
- identity (name, version, culture, public key)
- exported types
- other assemblies this assembly depends on
- security permissions

description of types
- name, visibility, base class, implemented interfaces
- members (methods, properties, fields, …)

attributes
- additional descriptive elements for types/members

Metadata & Self-Description

used for

- compiler
- serialization
- type browser
- schema generator
- reflection
- designers
- debugger
- profiler
- proxy generators
- …

demo

metadata & self-description

demo

- typeof(), GetType,
- MemberInfo, MethodInfo *Info
- IsValueType, IsInterface, …
- MethodInfo.Invoke
- Activator. CreateInstance

- ILSpy

Application Domains

- isolation boundary for security, versioning, reliability, and unloading of managed code
- several application domains in a single process
- isolated from each other
  - separate data and resources
  - separate security permissions
  - no direct calls, cross-domain calls via Remoting
- unload allocation domains

Code Access Security

Code Access Security (CAS)

- check sensitive actions,
- assure each assembly in the stack-walk hast necessary permissions

scenarios

- script engines
- plugins
- multi-user environments

predefined security zones

- Local, Intranet, Internet, Restricted

Code Access Security

security checks

- declarative (annotate type/member with attribute)
- imperative (call method)

examples:

- FileIOPermission
- FileDialogPermission
- UIPermission
- PrintingPermission
- WebPermission
- SocketPermission

demo

programming with application domains

demo

- AppDomain.CreateDomain
- PermissionSet
- Activator.CreateInstanceFrom
- AppDomain.Unload

Summary

- conceptual overview
- assemblies & types
- metadata & self-description
- application domains
- code access security

Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Windows Communication Foundation

Seminar Component Programming & Middleware

Operating Systems & Middleware Group

Building Distributed Applications

Component

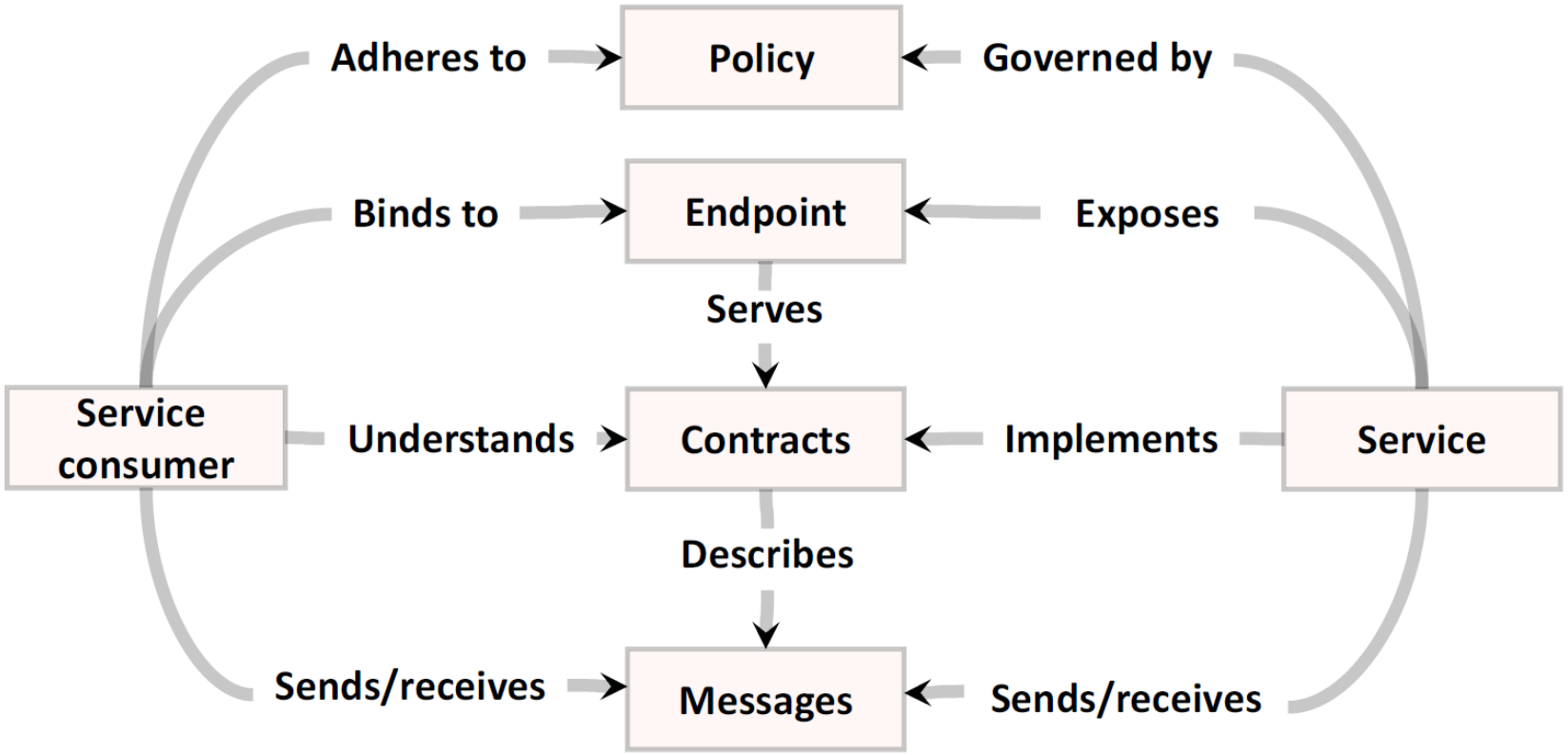
- unit of independent deployment
- unit of third-party composition
- has no (externally) observable state

Object

- unit of instantiation (unique identity)
- state (externally observable)
- encapsulate state & behavior

[Szyperski, Component Software, 1999]

Service-oriented Architectures



[Arnon Rotem-Gal-Oz, SOA Patterns, Manning Publications 2012]

Service, Contract & Endpoint

Service

- implement all functionality promised by exposed contracts
- service autonomy, self-sufficient

Contract

- collection of all messages supported by service
- ~"interface" in object-oriented design

Endpoint

- universal resource identifier (URI)
- address or specific place

Message & Policy

Message

- unit of communication
- different forms: Remoting, HTTP, SOAP, SMTP, …
- (generic) header + payload

Policy

- security (encryption, authentication, authorization), auditing, service-level agreements, …

What is WCF?

- unified communication framework for building service-oriented applications
- introduced in .NET Framework 3.0

goals:

- one API for all protocols
- interoperable

- prior: DCOM, ASMX Web Services, Remoting, socket-based communication

Communication Model

- address (where?)
  - location of the service
- binding (how?)
  - connectivity requirements
- contract (what?)
  - service contract
  - data contract
  - message contract & fault contract

Responsibilities of a Service Host

- resource management
  - threads, memory, isolation, throttling, load balancing
- configuration management
  - identity management, authentication, authorization
- monitoring
  - availability, health, performance, state
- error handling
  - fail-over, failed message management, exception handling

Hosting WCF Services

a host brings a service into life

- endpoint = address + binding + contract
- multiple endpoints per service possible

can be any Windows process; or IIS, WAS, AppFabric

Binding Elements

transport elements
- TCP, HTTP, IPC, MSMQ, custom

encoder elements
- text, binary, MTOM, custom

protocol elements
- security, reliability, transactions, …
- implementation of various WS-* protocols

Predefined Bindings

- basicHttpBinding
- wsHttpBinding
- netTcpBinding
- netMsmqBinding
- netNamedPipesBinding

choosing a Binding:

- technologies
- network topology
- performance
- functional requirements

Behaviours

- instancing
  - single, per call, per session
- concurrency
  - single- vs. multi-threaded instances
- throttling
  - how many concurrent calls
- security
  - authorization mechanisms
- serialization
- metadata

Messaging Patterns

request-response
- client call is blocked until service completes operation

one-way
- client call is blocked until service receives message

duplex (callback)
- both ends act as service

streaming

demo

WCF in action

demo

- IService1
  - ServiceContract OperationContract
  - DataContract DataMember
- ServiceClient : ClientBase<IService1>
- binding / binding configuration
- ServiceBehavior(InstanceContextMode = …)
- WSDL

Multiple Endpoints & Interoperability

security reasons
- different security policies for different endpoints

business reasons
- different capabilities in different locations

technology reasons
- firewalls, performance

interoperability reasons
- text encoding + WS*, JSON, ATOM, …

Advanced Topics

WCF Service Discovery

- Where is the service?
- service location is dynamic
- message categories
  - announcement (hello/bye)
  - discovery (probe/resolve)
- ad-hoc discovery vs. managed discovery

transactions

serialization (big objects)

Summary

- "ABC": address, binding, contract
- behaviors
- service hosts & service clients
- multiple endpoints & interoperability