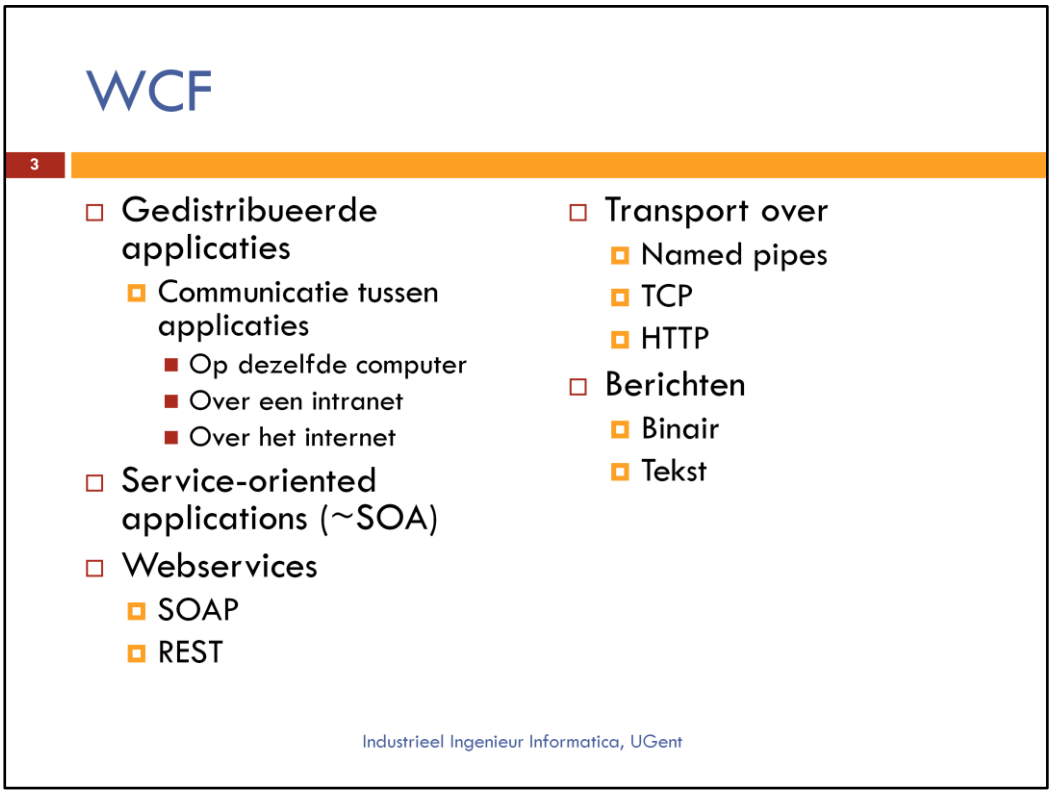
WINDOWS COMMUNICATION FOUNDATION (WCF)

Veerle Ongenae

WCF - overzicht

- Wat is WCF
- Architectuur WCF
  - Contracten, Gedrag, Berichten, Hosting
- Ontwikkeling WCF
  - Ontwerp servicecontract
  - Implementatie contract
  - Configuratie
  - Hosting
  - Ontwikkelen client



Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data. A few sample scenarios include:

A secure service to process business transactions.

A service that supplies current data to others, such as a traffic report or other monitoring service.

A chat service that allows two people to communicate or exchange data in real time.

A dashboard application that polls one or more services for data and presents it in a logical presentation.

Features WCF

Service Orientation

One consequence of using WS standards is that WCF enables you to create service oriented applications. Service-oriented architecture (SOA) is the reliance on Web services to send and receive data. The services have the general advantage of being loosely-coupled instead of hard-coded from one application to another. A loosely-coupled relationship implies that any client created on any platform can connect to any service as long as the essential contracts are met.

Interoperability

WCF implements modern industry standards for Web service interoperability.

Multiple Message Patterns

Messages are exchanged in one of several patterns. The most common pattern is the request/reply pattern, where one endpoint requests data from a second endpoint. The second endpoint replies. There are other patterns such as a one-way message in which a single endpoint sends a message without any expectation of a reply. A more complex pattern is the duplex exchange pattern where

two endpoints establish a connection and send data back and forth, similar to an instant messaging program.

Service Metadata

WCF supports publishing service metadata using formats specified in industry standards such as WSDL, XML Schema and WS-Policy. This metadata can be used to automatically generate and configure clients for accessing WCF services. Metadata can be published over HTTP and HTTPS or using the Web Service Metadata Exchange standard.

Data Contracts

Because WCF is built using the .NET Framework, it also includes code-friendly methods of supplying the contracts you want to enforce. One of the universal types of contracts is the data contract. In essence, as you code your service using Visual C# or Visual Basic, the easiest way to handle data is by creating classes that represent a data entity with properties that belong to the data entity. WCF includes a comprehensive system for working with data in this easy manner. Once you have created the classes that represent data, your service automatically generates the metadata that allows clients to comply with the data types you have designed.

Security

Messages can be encrypted to protect privacy and you can require users to authenticate themselves before being allowed to receive messages. Security can be implemented using well-known standards such as SSL or WS-SecureConversation.

Multiple Transports and Encodings

Messages can be sent on any of several built-in transport protocols and encodings. The most common protocol and encoding is to send text encoded SOAP messages using is the HyperText Transfer Protocol (HTTP) for use on the World Wide Web. Alternatively, WCF allows you to send messages over TCP, named pipes, or MSMQ. These messages can be encoded as text or using an optimized binary format. Binary data can be sent efficiently using the MTOM standard. If none of the provided transports or encodings suit your needs you can create your own custom transport or encoding.

Reliable and Queued Messages

WCF supports reliable message exchange using reliable sessions implemented over WS-Reliable Messaging and using MSMQ.

Durable Messages

A durable message is one that is never lost due to a disruption in the communication. The messages in a durable message pattern are always saved to a database. If a disruption occurs, the database allows you to resume the message exchange when the connection is restored. You can also create a durable message using the Windows Workflow Foundation (WF).

Transactions

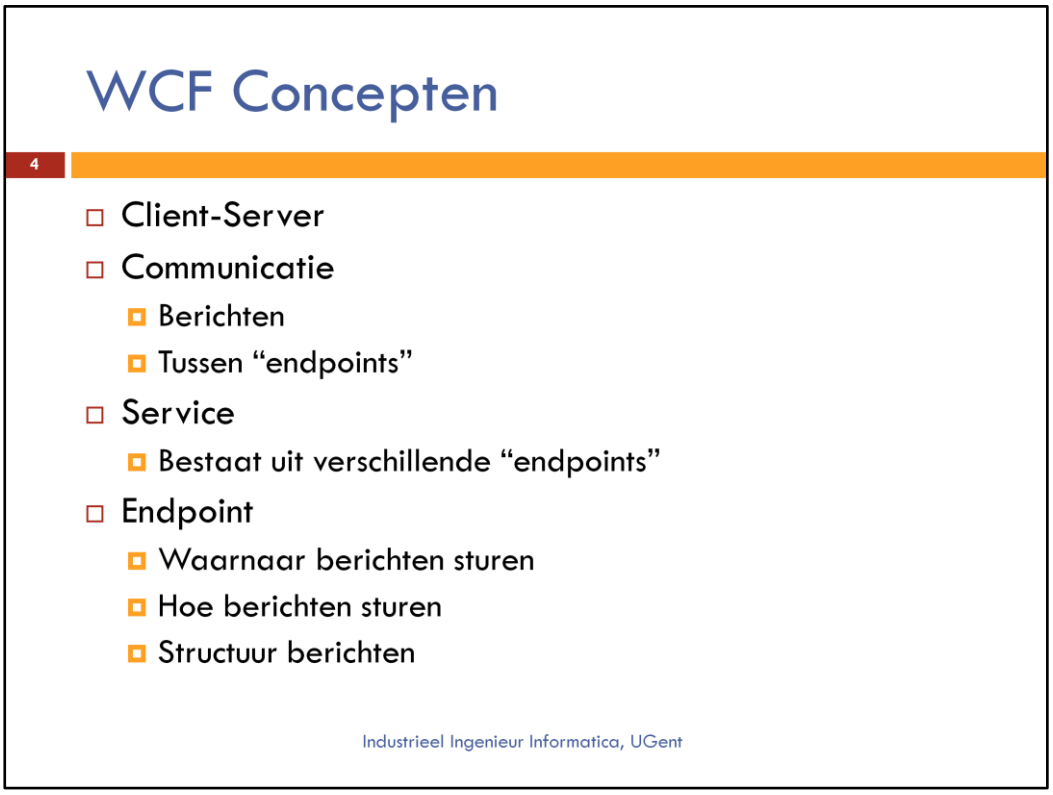
WCF also supports transactions using one of three transaction models: WS-AtomicTtransactions, the APIs in the System.Transactions namespace, and Microsoft Distributed Transaction Coordinator.

AJAX and REST Support

REST is an example of an evolving Web 2.0 technology. WCF can be configured to process "plain" XML data that is not wrapped in a SOAP envelope. WCF can also be extended to support specific XML formats, such as ATOM (a popular RSS standard), and even non-XML formats, such as JavaScript Object Notation (JSON).

Extensibility

The WCF architecture has a number of extensibility points. If extra capability is required, there are a number of entry points that allow you to customize the behavior of a service.



WCF is a runtime and a set of APIs for creating systems that send messages between services and clients. The same infrastructure and APIs are used to create applications that communicate with other applications on the same computer system or on a system that resides in another company and is accessed over the Internet.

Messaging and Endpoints
WCF is based on the notion of message-based communication, and anything that can be modeled as a message (for example, an HTTP request or a Message Queuing (also known as MSMQ) message) can be represented in a uniform way in the programming model. This enables a unified API across different transport mechanisms.

The model distinguishes between clients, which are applications that initiate communication, and services, which are applications that wait for clients to communicate with them and respond to that communication. A single application can act as both a client and a service.

Messages are sent between endpoints. Endpoints are places where messages are sent or received (or both), and they define all the information required for the message exchange. A service exposes one or more application endpoints (as well as zero or more infrastructure endpoints), and the client generates an endpoint that is compatible with one of the service's endpoints.

An endpoint describes in a standard-based way where messages should be sent, how they should be sent, and what the messages should look like. A service can expose this information as metadata that clients can process to generate appropriate WCF clients and communication stacks.

Communication Protocols
One required element of the communication stack is the transport protocol. Messages can be sent over intranets and the Internet using common transports, such as HTTP and TCP. Other transports are included that support communication with Message Queuing applications and nodes on a Peer

Networking mesh.

Another required element in the communication stack is the encoding that specifies how any given message is formatted. WCF provides the following encodings:

Text encoding, an interoperable encoding.
Message Transmission Optimization Mechanism (MTOM) encoding, which is an interoperable way for efficiently sending unstructured binary data to and from a service.
Binary encoding for efficient transfer.

Message Patterns
WCF supports several messaging patterns, including request-reply, one-way, and duplex communication. Different transports support different messaging patterns, and thus affect the types of interactions that they support. The WCF APIs and runtime also help you to send messages securely and reliably.

message
A self-contained unit of data that can consist of several parts, including a body and headers.

service
A construct that exposes one or more endpoints, with each endpoint exposing one or more service operations.

endpoint
A construct at which messages are sent or received (or both). It comprises a location (an address) that defines where messages can be sent, a specification of the communication mechanism (a binding) that described how messages should be sent, and a definition for a set of messages that can be sent or received (or both) at that location (a service contract) that describes what message can be sent.

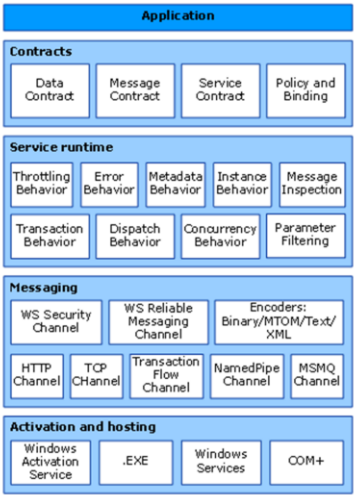
A WCF service is exposed to the world as a collection of endpoints.
application endpoint

An endpoint exposed by the application and that corresponds to a service contract implemented by the application.

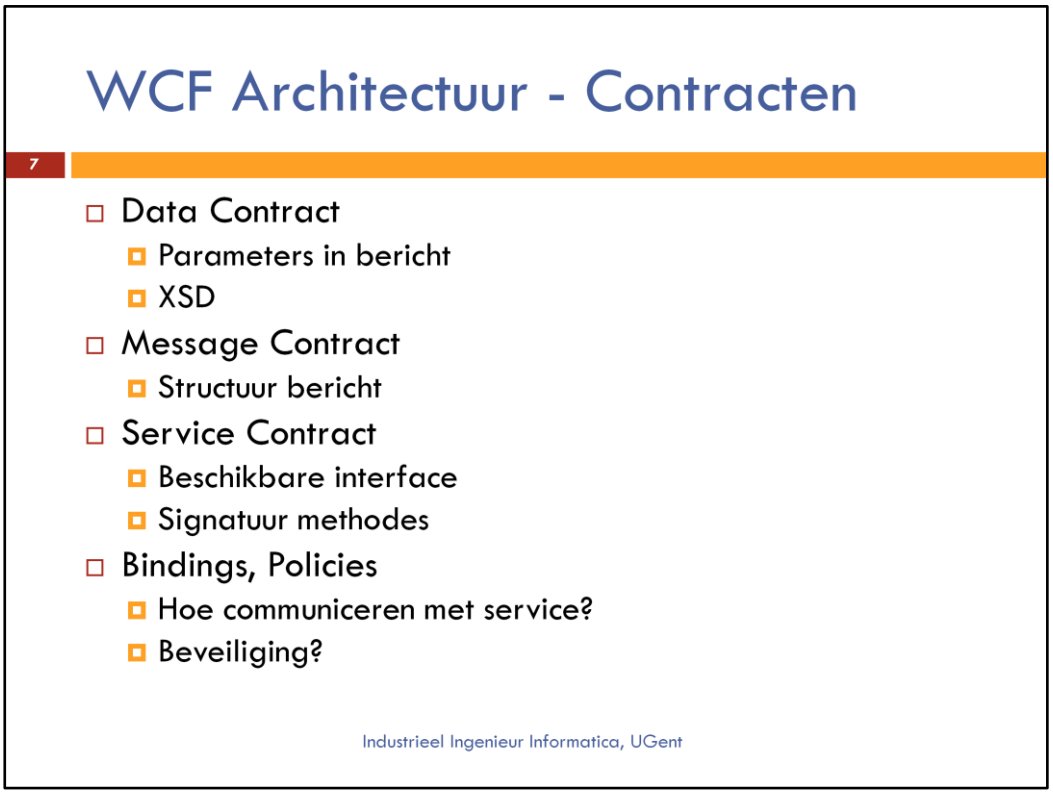
WCF - overzicht

- Wat is WCF
- Architectuur WCF
  - Contracten, Gedrag, Berichten, Hosting
- Ontwikkeling WCF
  - Ontwerp servicecontract
  - Implementatie contract
  - Configuratie
  - Hosting
  - Ontwikkelen client

WCF-architectuur



https://msdn.microsoft.com/en-us/library/ms733128%28v=vs.110%29.aspx



Contracts and Descriptions
Contracts define various aspects of the message system. The data contract describes every parameter that makes up every message that a service can create or consume. The message parameters are defined by XML Schema definition language (XSD) documents, enabling any system that understands XML to process the documents. The message contract defines specific message parts using SOAP protocols, and allows finer-grained control over parts of the message, when interoperability demands such precision. The service contract specifies the actual method signatures of the service, and is distributed as an interface in one of the supported programming languages, such as Visual Basic or Visual C#.
Policies and bindings stipulate the conditions required to communicate with a service. For example, the binding must (at a minimum) specify the transport used (for example, HTTP or TCP), and an encoding. Policies include security requirements and other conditions that must be met to communicate with a service.

service operation
A procedure defined in a service's code that implements the functionality for an operation. This operation is exposed to clients as methods on a WCF client. The method can return a value, and can take an optional number of arguments, or take no arguments, and return no response. For example, an operation that functions as a simple "Hello" can be used as a notification of a client's presence and to begin a series of operations.

service contract
Ties together multiple related operations into a single functional unit. The contract can define service-level settings, such as the namespace of the service, a corresponding callback contract, and other such settings. In most cases, the contract is defined by creating an interface in the programming language of your choice and applying the ServiceContractAttribute attribute to the interface. The actual service code results by implementing the interface.

operation contract

An operation contract defines the parameters and return type of an operation. When creating an interface that defines the service contract, you signify an operation contract by applying the OperationContractAttribute attribute to each method definition that is part of the contract. The operations can be modeled as taking a single message and returning a single message, or as taking a set of types and returning a type. In the latter case, the system will determine the format for the messages that need to be exchanged for that operation.

message contract

Describes the format of a message. For example, it declares whether message elements should go in headers versus the body, what level of security should be applied to what elements of the message, and so on.

fault contract

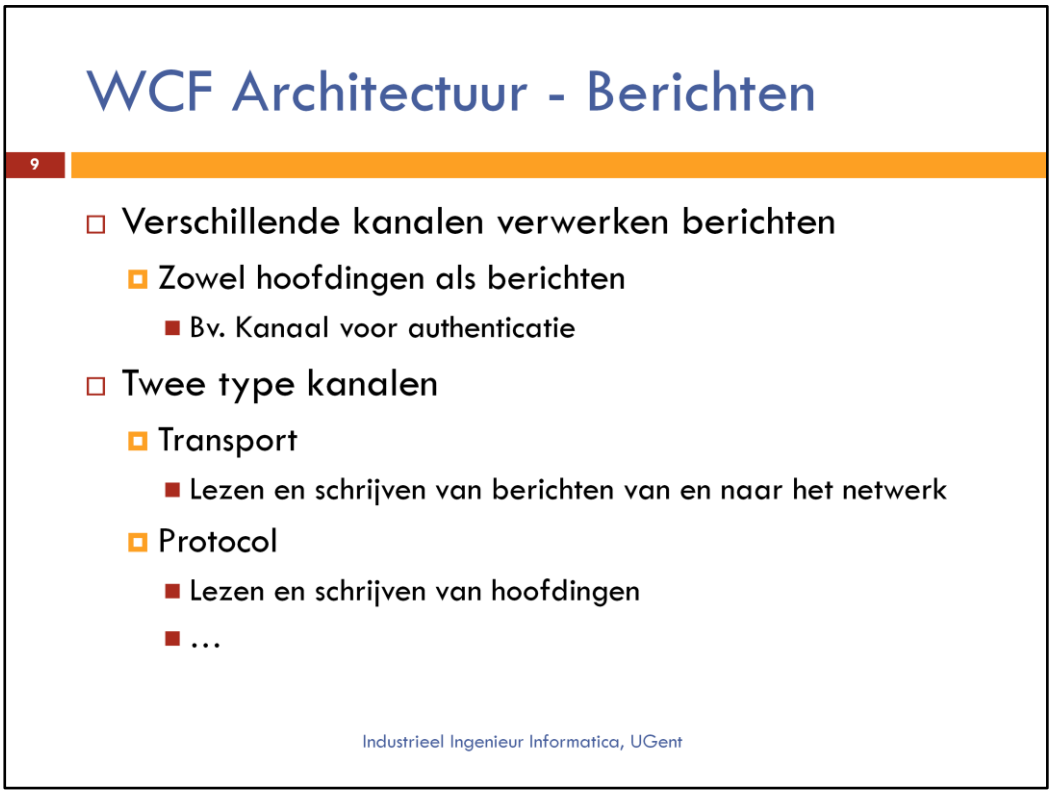
Can be associated with a service operation to denote errors that can be returned to the caller. An operation can have zero or more faults associated with it. These errors are SOAP faults that are modeled as exceptions in the programming model.

data contract

The descriptions in metadata of the data types that a service uses. This enables others to interoperate with the service. The data types can be used in any part of a message, for example, as parameters or return types. If the service is using only simple types, there is no need to explicitly use data contracts.

Service Runtime
The service runtime layer contains the behaviors that occur only during the actual operation of the service, that is, the runtime behaviors of the service. Throttling controls how many messages are processed, which can be varied if the demand for the service grows to a preset limit. An error behavior specifies what occurs when an internal error occurs on the service, for example, by controlling what information is communicated to the client. (Too much information can give a malicious user an advantage in mounting an attack.) Metadata behavior governs how and whether metadata is made available to the outside world. Instance behavior specifies how many instances of the service can be run (for example, a singleton specifies only one instance to process all messages). Transaction behavior enables the rollback of transacted operations if a failure occurs. Dispatch behavior is the control of how a message is processed by the WCF infrastructure.

Extensibility enables customization of runtime processes. For example, message inspection is the facility to inspect parts of a message, and parameter filtering enables preset actions to occur based on filters acting on message headers.



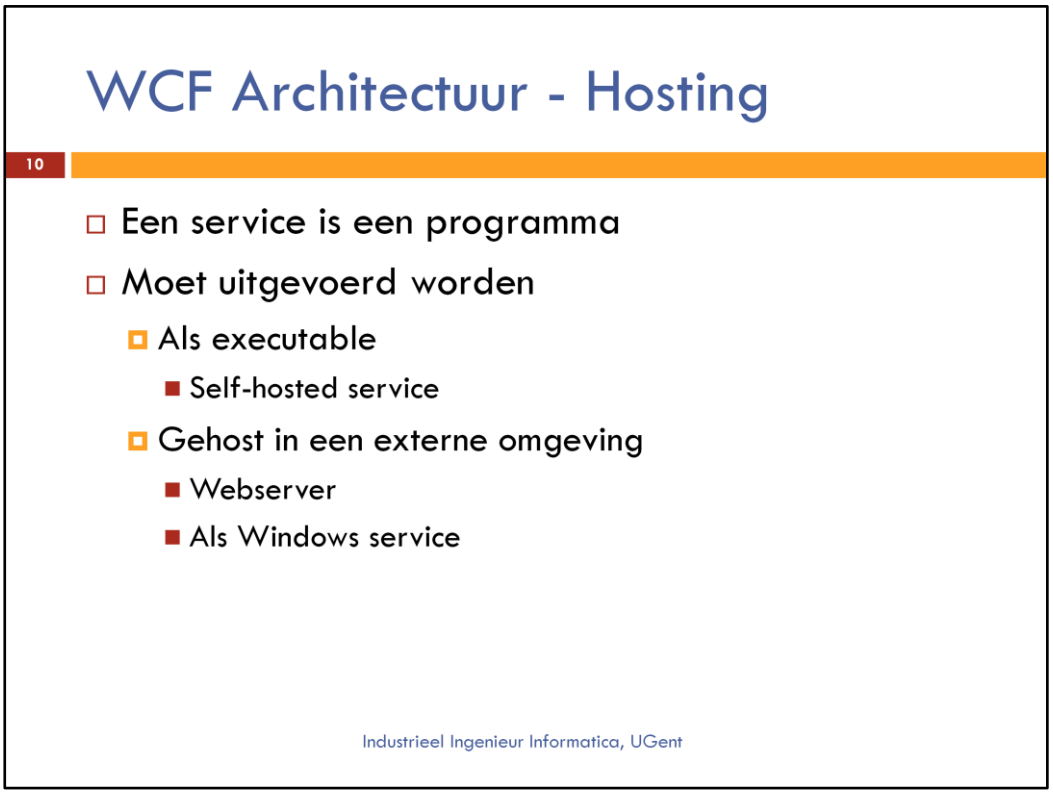
Messaging
The messaging layer is composed of channels. A channel is a component that processes a message in some way, for example, by authenticating a message. A set of channels is also known as a channel stack. Channels operate on messages and message headers. This is different from the service runtime layer, which is primarily concerned about processing the contents of message bodies.

There are two types of channels: transport channels and protocol channels.

Transport channels read and write messages from the network (or some other communication point with the outside world). Some transports use an encoder to convert messages (which are represented as XML Infosets) to and from the byte stream representation used by the network. Examples of transports are HTTP, named pipes, TCP, and MSMQ. Examples of encodings are XML and optimized binary.

Protocol channels implement message processing protocols, often by reading or writing additional headers to the message. Examples of such protocols include WS-Security and WS-Reliability.

The messaging layer illustrates the possible formats and exchange patterns of the data. WS-Security is an implementation of the WS-Security specification enabling security at the message layer. The WS-Reliable Messaging channel enables the guarantee of message delivery. The encoders present a variety of encodings that can be used to suit the needs of the message. The HTTP channel specifies that the HyperText Transport Protocol is used for message delivery. The TCP channel similarly specifies the TCP protocol. The Transaction Flow channel governs transacted message patterns. The Named Pipe channel enables interprocess communication. The MSMQ channel enables interoperation with MSMQ applications.



Hosting and Activation

In its final form, a service is a program. Like other programs, a service must be run in an executable. This is known as a *self-hosted* service.

Services can also be *hosted*, or run in an executable managed by an external agent, such as IIS or Windows Activation Service (WAS). WAS enables WCF applications to be activated automatically when deployed on a computer running WAS. Services can also be manually run as executables (.exe files). A service can also be run automatically as a Windows service. COM+ components can also be hosted as WCF services.

Voorbeeld

- Service op IIS
  - tijdService
  - tijdClient
  - tijdWebserviceClient
- Service als console-applicatie
  - tijdServer
  - toonTijdVanConsole (Client)

Windows Communication Foundation (WCF) enables applications to communicate whether they are on the same computer, across the Internet, or on different application platforms.

The Basic Tasks
The basic tasks to perform are, in order:

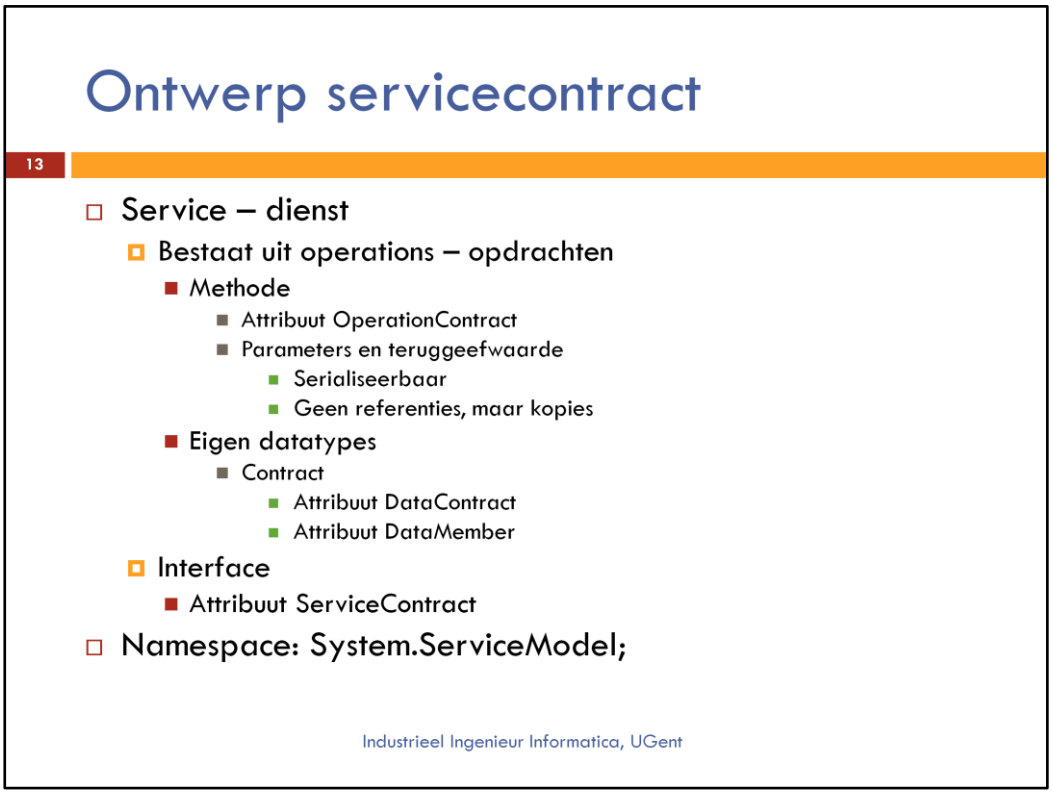
Define the service contract. A service contract specifies the signature of a service, the data it exchanges, and other contractually required data.

Implement the contract. To implement a service contract, create a class that implements the contract and specify custom behaviors that the runtime should have.

Configure the service by specifying endpoints and other behavior information.

Host the service.

Build a client application.



A service contract specifies what an endpoint communicates to the outside world. At a more concrete level, it is a statement about a set of specific messages organized into basic message exchange patterns (MEPs), such as request/reply, one-way, and duplex. If a service contract is a logically related set of message exchanges, a service operation is a single message exchange. For example, a Hello operation must obviously accept one message (so the caller can announce the greeting) and may or may not return a message (depending upon the courtesy of the operation).

Service Contracts

A service contract specifies the following:
The operations a contract exposes.
The signature of the operations in terms of messages exchanged.
The data types of these messages.
The location of the operations.
The specific protocols and serialization formats that are used to support successful communication with the service.

For example, a purchase order contract might have a CreateOrder operation that accepts an input of order information types and returns success or failure information, including an order identifier. It might also have a GetOrderStatus operation that accepts an order identifier and returns order status information. A service contract of this sort would specify:
That the purchase order contract consisted of CreateOrder and GetOrderStatus operations.
That the operations have specified input messages and output messages.
The data that these messages can carry.
Categorical statements about the communication infrastructure necessary to successfully process the messages. For example, these details include whether and what forms of security are required to establish successful communication.

To convey this kind of information to other applications on many platforms (including non-Microsoft platforms), XML service contracts are publicly expressed in standard XML formats, such as Web Services Description Language (WSDL) and XML Schema (XSD), among others. Developers for many platforms can use this public contract information to create applications that can communicate with the service, both because they understand the language of the specification and because those languages are designed to enable interoperation by describing the public forms, formats, and protocols that the service supports. For more information about how WCF handles this kind of information, see Metadata.

Contracts can be expressed many ways, and while WSDL and XSD are excellent languages to describe services in an accessible way, they are difficult languages to use directly and are merely descriptions of a service, not service contract implementations. Therefore, WCF applications use managed attributes, interfaces, and classes both to define the structure of a service and to implement it.

The resulting contract defined in managed types can be exported as metadata—WSDL and XSD—when needed by clients or other service implementers. The result is a straightforward programming model that can be described (using public metadata) to any client application. The details of the underlying SOAP messages, the transportation and security-related information, and so on, can be left to WCF, which performs the necessary conversions to and from the service contract type system to the XML type system automatically.

A service contract groups the operations; specifies the message exchange pattern, message types, and data types those messages carry; and indicates categories of run-time behavior an implementation must have to support the contract (for example, it may require that messages be encrypted and signed). The service contract itself does not specify precisely how these requirements are met, only that they must be. The type of encryption or the manner in which a message is signed is up to the implementation and configuration of a compliant service.

Notice the way that the contract requires certain things of the service contract implementation and the run-time configuration to add behavior. The set of requirements that must be met to expose a service for use builds on the preceding set of requirements. If a contract makes requirements of the implementation, an implementation can require yet more of the configuration and bindings that enable the service to run. Finally, the host application must also support any requirements that the service configuration and bindings add.

Creating a Service Contract

Services expose a number of operations. In Windows Communication Foundation (WCF) applications, define the operations by creating a method and marking it with the OperationContractAttribute attribute. Then, to create a service contract, group together your operations, either by declaring them within an interface marked with the ServiceContractAttribute attribute, or by defining them in a class marked with the same attribute.

Any methods that do not have a OperationContractAttribute attribute are not service operations and are not exposed by WCF services.

Classes or Interfaces

Both classes and interfaces represent a grouping of functionality and, therefore, both can be used to define a WCF service contract. However, it is recommended that you use interfaces because they directly model service contracts. Without an implementation, interfaces do no more than define a grouping of methods with certain signatures. Implement a service contract interface and you have implemented a WCF service.

All the benefits of managed interfaces apply to service contract interfaces:

- Service contract interfaces can extend any number of other service contract interfaces.

A single class can implement any number of service contracts by implementing those service contract interfaces.

You can modify the implementation of a service contract by changing the interface implementation, while the service contract remains the same.

You can version your service by implementing the old interface and the new one. Old clients connect to the original version, while newer clients can connect to the newer version.

You can, however, use a class to define a service contract and implement that contract at the same time. The advantage of creating your services by applying ServiceContractAttribute and OperationContractAttribute directly to the class and the methods on the class, respectively, is speed and simplicity. The disadvantages are that managed classes do not support multiple inheritance, and as a result they can only implement one service contract at a time. In addition, any modification to the class or method signatures modifies the public contract for that service, which can prevent unmodified clients from using your service.

Parameters and Return Values

Each operation has a return value and a parameter, even if these are void. However, unlike a local method, in which you can pass references to objects from one object to another, service operations do not pass references to objects. Instead, they pass copies of the objects.

This is significant because each type used in a parameter or return value must be serializable; that is, it must be possible to convert an object of that type into a stream of bytes and from a stream of bytes into an object.

Primitive types are serializable by default, as are many types in the .NET Framework.

The value of the parameter names in the operation signature are part of the contract and are case sensitive. If you want to use the same parameter name locally but modify the name in the published metadata, see the System.ServiceModel.MessageParameterAttribute.

Data Contracts

Service-oriented applications like Windows Communication Foundation (WCF) applications are designed to interoperate with the widest possible number of client applications on both Microsoft and non-Microsoft platforms. For the widest possible interoperability, it is recommended that you mark your types with the DataContractAttribute and DataMemberAttribute attributes to create a data contract, which is the portion of the service contract that describes the data that your service operations exchange.

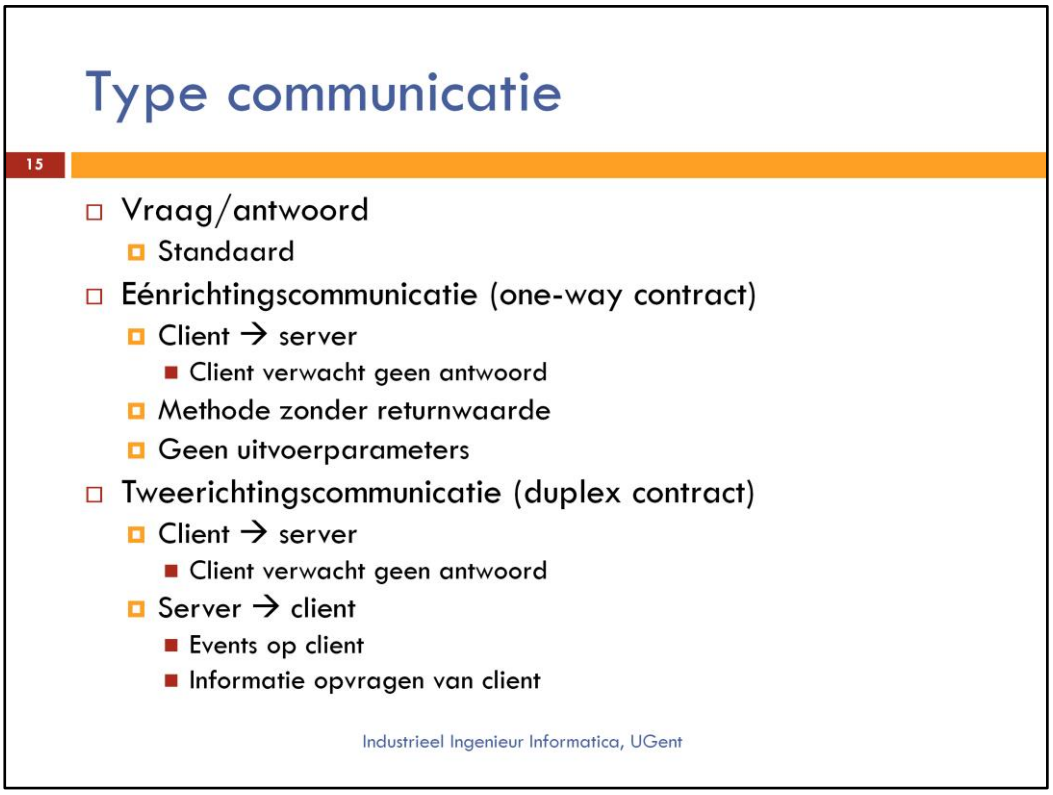
Data contracts are opt-in style contracts: No type or data member is serialized unless you explicitly apply the data contract attribute. Data contracts are unrelated to the access scope of the managed code: Private data members can be serialized and sent elsewhere to be accessed publicly. WCF handles the definition of the underlying SOAP messages that enable the operation's functionality as well as the serialization of your data types into and out of the body of the messages. As long as your data types are serializable, you do not need to think about the underlying message exchange infrastructure when designing your operations.

Although the typical WCF application uses the DataContractAttribute and DataMemberAttribute attributes to create data contracts for operations, you can use other serialization mechanisms. The standard ISerializable, SerializableAttribute and IXmlSerializable mechanisms all work to handle the serialization of your data types into the underlying SOAP messages that carry them from one application to another. You can employ more serialization strategies if your data types require special support.

Servicecontract

- Interface met attributen

```
[ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples")]
public interface ICalculator {
    [OperationContract]
    double Add(double n1, double n2);
    [OperationContract]
    double Subtract(double n1, double n2);
    [OperationContract]
    double Multiply(double n1, double n2);
    [OperationContract]
    double Divide(double n1, double n2);
}
```

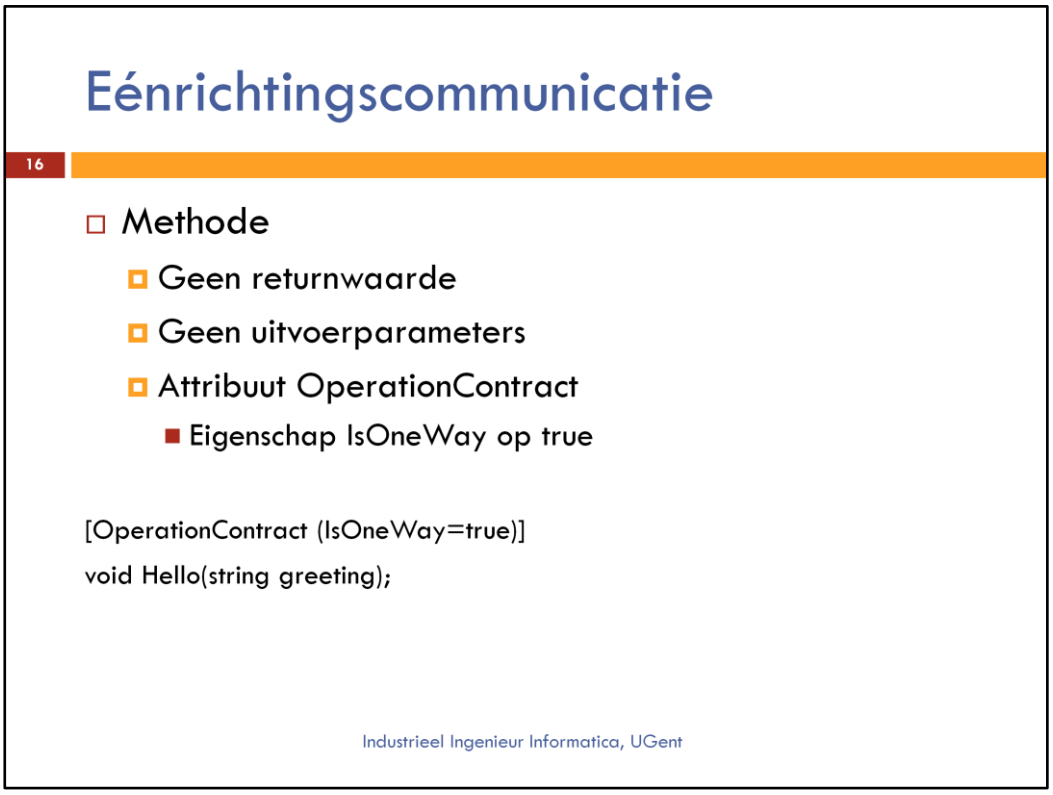


Mapping Parameters and Return Values to Message Exchanges
Service operations are supported by an underlying exchange of SOAP messages that transfer application data back and forth, in addition to the data required by the application to support certain standard security, transaction, and session-related features. Because this is the case, the signature of a service operation dictates a certain underlying message exchange pattern (MEP) that can support the data transfer and the features an operation requires. You can specify three patterns in the WCF programming model: request/reply, one-way, and duplex message patterns.

Request/Reply
A request/reply pattern is one in which a request sender (a client application) receives a reply with which the request is correlated. This is the default MEP because it supports an operation in which one or more parameters are passed to the operation and a return value is passed back to the caller.

Note that unless you specify a different underlying message pattern, even service operations that return void are request/reply message exchanges. The result for your operation is that unless a client invokes the operation asynchronously, the client stops processing until the return message is received, even though that message is empty in the normal case.

This can slow client performance and responsiveness if the operation takes a long time to perform, but there are advantages to request/reply operations even when they return void. The most obvious one is that SOAP faults can be returned in the response message, which indicates that some service-related error condition has occurred, whether in communication or processing. SOAP faults that are specified in a service contract are passed to the client application as a FaultException<TDetail> object, where the type parameter is the type specified in the service contract. This makes notifying clients about error conditions in WCF services easy.

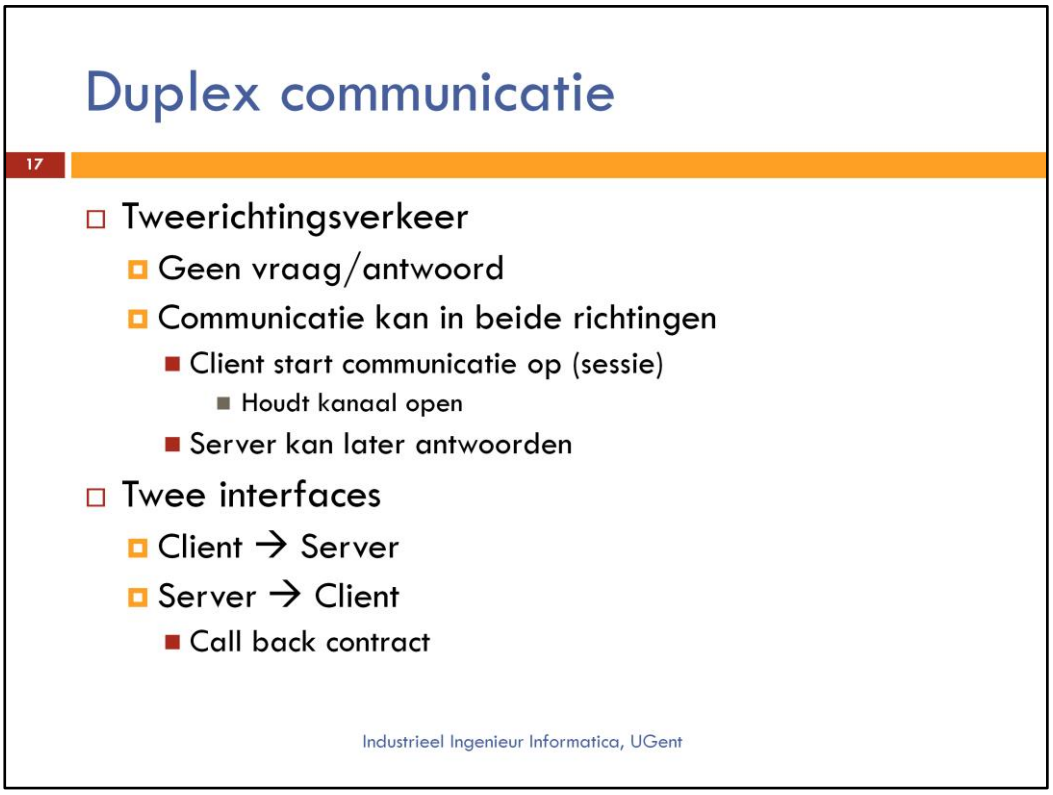


One-way
If the client of a WCF service application should not wait for the operation to complete and does not process SOAP faults, the operation can specify a one-way message pattern. A one-way operation is one in which a client invokes an operation and continues processing after WCF writes the message to the network. Typically this means that unless the data being sent in the outbound message is extremely large the client continues running almost immediately (unless there is an error sending the data). This type of message exchange pattern supports event-like behavior from a client to a service application.

A message exchange in which one message is sent and none are received cannot support a service operation that specifies a return value other than void; in this case an InvalidOperationException exception is thrown.

No return message also means that there can be no SOAP fault returned to indicate any errors in processing or communication. (Communicating error information when operations are one-way operations requires a duplex message exchange pattern.)

To specify a one-way message exchange for an operation that returns void, set the IsOneWay property to true.



Duplex
A duplex pattern is characterized by the ability of both the service and the client to send messages to each other independently whether using one-way or request/reply messaging. This form of two-way communication is useful for services that must communicate directly to the client or for providing an asynchronous experience to either side of a message exchange, including event-like behavior.

The duplex pattern is slightly more complex than the request/reply or one-way patterns because of the additional mechanism for communicating with the client.

To design a duplex contract, you must also design a callback contract and assign the type of that callback contract to the CallbackContract property of the ServiceContractAttribute attribute that marks your service contract.

To implement a duplex pattern, you must create a second interface that contains the method declarations that are called on the client.

When a service receives a duplex message, it looks at the ReplyTo element in that incoming message to determine where to send the reply. If the channel that is used to receive the message is not secured, then an untrusted client could send a malicious message with a target machine's ReplyTo, leading to a denial of service (DOS) of that target machine.

Duplex communicatie - syntax


```
[ServiceContract(Namespace =
    "http://Microsoft.ServiceModel.Samples",
    SessionMode=SessionMode.Required,
    CallbackContract=typeof(ICalculatorDuplex
    Callback))]
public interface ICalculatorDuplex {
    [OperationContract(IsOneWay = true)]
    void Clear();
    [OperationContract(IsOneWay = true)]
    void AddTo(double n);
    [OperationContract(IsOneWay = true)]
    void SubtractFrom(double n);
    [OperationContract(IsOneWay = true)]
    void MultiplyBy(double n);
    [OperationContract(IsOneWay = true)]
    void DivideBy(double n);
}
```

Implementatie op server

```
public interface ICalculatorDuplexCallback
    {
    [OperationContract(IsOneWay = true)]
    void Result(double result);
    [OperationContract(IsOneWay = true)]
    void Equation(string eqn);
}
```

Implementatie op client

Overzicht

- Ontwerp servicecontract
- Implementatie contract
- Configuratie
- Hosting
- Ontwikkelen client

Implementatie servicecontract

- Klasse die interface implementeert

```
[ServiceContract]
public interface IMath {
    [OperationContract]
    double Add(double A, double B);
    [OperationContract]
    double Multiply (double A, double B);
}
public class MathService : IMath {
    public double Add (double A, double B) { return A + B; }
    public double Multiply (double A, double B) { return A * B; }
}
```


A service is a class that exposes functionality available to clients at one or more endpoints. To create a service, write a class that implements a Windows Communication Foundation (WCF) contract. You can do this in one of two ways. You can define the contract separately as an interface and then create a class that implements that interface. Alternatively, you can create the class and contract directly by placing the ServiceContractAttribute attribute on the class itself and the OperationContractAttribute attribute on the methods available to the clients of the service.

Overzicht

- Ontwerp servicecontract
- Implementatie contract
- Configuratie
- Hosting
- Ontwikkelen client

Configuratie

22

- Specifiëren endpoint
  - Welke service?
    - Interface
    - Implementatie
  - Waar?
    - URL
  - Hoe communiceren? (binding)
    - TCP? HTTP?
    - Tekst? Binair?
    - Beveiligd?
- Configuratie = belangrijk deel WCF-programmatie

Once you have designed and implemented your service contract, you are ready to configure your service. This is where you define and customize how your service is exposed to clients, including specifying the address where it can be found, the transport and message encoding it uses to send and receive messages, and the type of security it requires.

configuration versus coding

Control of an application can be done either through coding, through configuration, or through a combination of both. Configuration has the advantage of allowing someone other than the developer (for example, a network administrator) to set client and service parameters after the code is written and without having to recompile. Configuration not only enables you to set values like endpoint addresses, but also allows further control by enabling you to add endpoints, bindings, and behaviors. Coding allows the developer to retain strict control over all components of the service or client, and any settings done through the configuration can be inspected and if needed overridden by the code.

Specifiëren enpoint in code

Adres

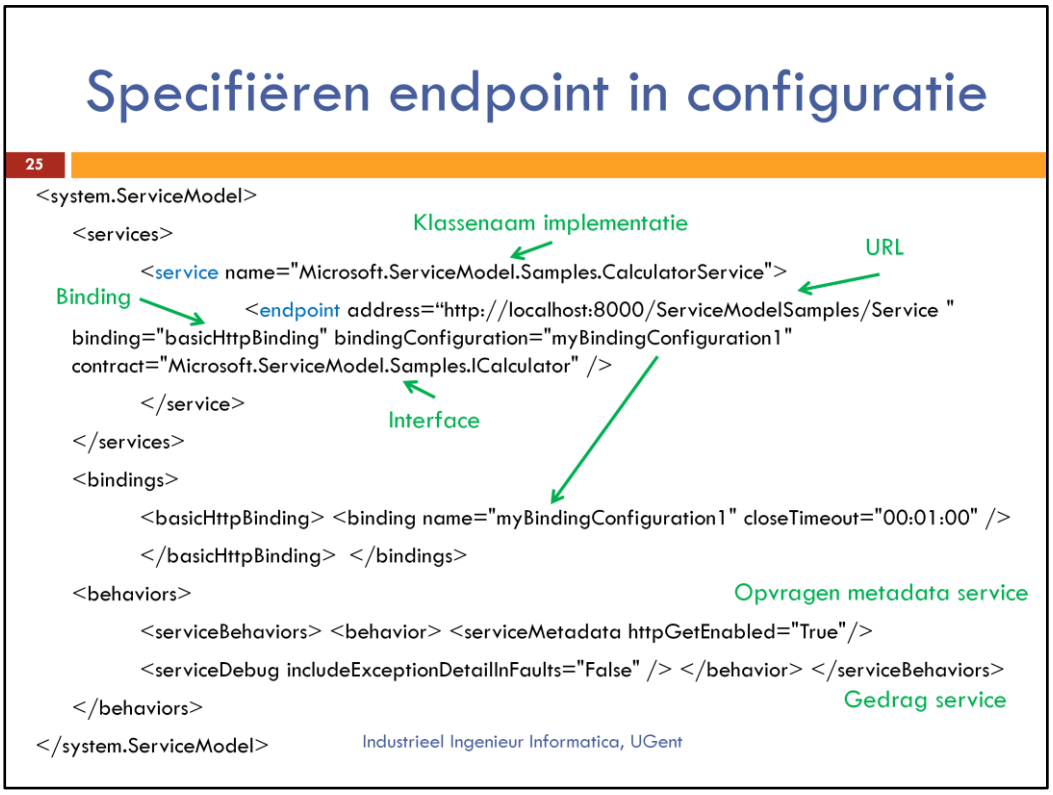
```
Uri baseAddress = new Uri("http://localhost:8000/ServiceModelSamples/Service");
```

Implementatie service

```
ServiceHost ServiceHost selfHost = new ServiceHost(typeof(CalculatorService),
    baseAddress);
```

Welke service? Binding

```
try {
    selfHost.AddServiceEndpoint( typeof(ICalculator), new WSHttpBinding(),
    "CalculatorService");
    … // gedrag instellen, service runnen
} catch (CommunicationException ce) { Console.WriteLine("An exception occurred:
    {0}", ce.Message); selfHost.Abort();
}
```



Configuring a Windows Communication Foundation (WCF) service with a configuration file gives you the flexibility of providing endpoint and service behavior data at the point of deployment instead of at design time.

A WCF service is configurable using the .NET Framework configuration technology. Most commonly, XML elements are added to the Web.config file for an Internet Information Services (IIS) site that hosts a WCF service. The elements allow you to change details such as the endpoint addresses (the actual addresses used to communicate with the service) on a machine-by-machine basis. In addition, WCF includes several system-provided elements that allow you to quickly select the most basic features for a service. Starting with .NET Framework version 4, WCF comes with a new default configuration model that simplifies WCF configuration requirements. If you do not provide any WCF configuration for a particular service, the runtime automatically configures your service with some standard endpoints and default binding/behavior. In practice, writing configuration is a major part of programming WCF applications.

When configuring a service in Visual Studio, use either a Web.config file or an App.config file to specify the settings. The choice of the configuration file name is determined by the hosting environment you choose for the service. If you are using IIS to host your service, use a Web.config file. If you are using any other hosting environment, use an App.config file.

In WCF service configuration files, the <**system.serviceModel**> section contains a <**service**> element for each service hosted. The <**service**> element contains a collection of <**endpoint**> elements that specify the endpoints exposed for each service and optionally a set of service behaviors.

Each service has these attributes:
name. Specifies the type that provides an implementation of a service contract. This is a fully qualified name which consists of the namespace, a period, and then the type name.
behaviorConfiguration. Specifies the name of one of the **behavior** elements found in the **behaviors**

element. The specified behavior governs actions such as whether the service allows impersonation. If its value is the empty name or no **behaviorConfiguration** is provided then the default set of service behaviors is added to the service.

The <**endpoint**> elements specify the address, binding, and contract exposed by the endpoint, and optionally binding configuration and endpoint behaviors. They are represented by the following attributes:

address. Specifies the service's Uniform Resource Identifier (URI), which can be an absolute address or one that is given relative to the base address of the service.
binding. Typically specifies a system-provided binding, but can also specify a user-defined binding. The binding specified determines the type of transport, security and encoding used, and whether reliable sessions, transactions, or streaming is supported or enabled.
bindingConfiguration. If the default values of a binding must be modified, this can be done by configuring the appropriate **binding** element in the **bindings** element. This attribute should be given the same value as the **name** attribute of the **binding** element that is used to change the defaults. If no name is given, or no **bindingConfiguration** is specified in the binding, then the default binding of the binding type is used in the endpoint.
contract. Specifies the interface that defines the contract. This is the interface implemented in the common language runtime (CLR) type specified by the **name** attribute of the **service** element.

The <**system.serviceModel**> section also contains a <**behaviors**> element that allows you to specify service or endpoint behaviors.

infrastructure endpoint
An endpoint that is exposed by the infrastructure to facilitate functionality that is needed or provided by the service that does not relate to a service contract. For example, a service might have an infrastructure endpoint that provides metadata information.

address
Specifies the location where messages are received. It is specified as a Uniform Resource Identifier (URI). The URI schema part names the transport mechanism to use to reach the address, such as HTTP and TCP. The hierarchical part of the URI contains a unique location whose format is dependent on the transport mechanism.

The endpoint address enables you to create unique endpoint addresses for each endpoint in a service or, under certain conditions, to share an address across endpoints. The following example shows an address using the HTTPS protocol with a non-default port:
HTTPS://cohowinery:8005/ServiceModelSamples/CalculatorService

binding
Defines how an endpoint communicates to the world. It is constructed of a set of components called binding elements that "stack" one on top of the other to create the communication infrastructure. At the very least, a binding defines the transport (such as HTTP or TCP) and the encoding being used (such as text or binary). A binding can contain binding elements that specify details like the security mechanisms used to secure messages, or the message pattern used by an endpoint.

binding element
Represents a particular piece of the binding, such as a transport, an encoding, an implementation of an infrastructure-level protocol (such as WS-ReliableMessaging), or any other component of the communication stack.

behaviors
A component that controls various run-time aspects of a service, an endpoint, a particular

operation, or a client. Behaviors are grouped according to scope: common behaviors affect all endpoints globally, service behaviors affect only service-related aspects, endpoint behaviors affect only endpoint-related properties, and operation-level behaviors affect particular operations. For example, one service behavior is throttling, which specifies how a service reacts when an excess of messages threaten to overwhelm its handling capabilities. An endpoint behavior, on the other hand, controls only aspects that are relevant to endpoints, such as how and where to find a security credential.

system-provided bindings

WCF includes a number of system-provided bindings. These are collections of binding elements that are optimized for specific scenarios. For example, the WSHttpBinding is designed for interoperability with services that implement various WS-* specifications. These predefined bindings save time by presenting only those options that can be correctly applied to the specific scenario. If a predefined binding does not meet your requirements, you can create your own custom binding.

channel

A concrete implementation of a binding element. The binding represents the configuration, and the channel is the implementation associated with that configuration. Therefore, there is a channel associated with each binding element. Channels stack on top of each other to create the concrete implementation of the binding: the channel stack.

security

In WCF, includes confidentiality (encryption of messages to prevent eavesdropping), integrity (the means for detection of tampering with the message), authentication (the means for validation of servers and clients), and authorization (the control of access to resources). These functions are provided by either leveraging existing security mechanisms, such as TLS over HTTP (also known as HTTPS), or by implementing one or more of the various WS-* security specifications.

transport security mode

Specifies that confidentiality, integrity, and authentication are provided by the transport layer mechanisms (such as HTTPS). When using a transport like HTTPS, this mode has the advantage of being efficient in its performance, and well understood because of its prevalence on the Internet. The disadvantage is that this kind of security is applied separately on each hop in the communication path, making the communication susceptible to a "man in the middle" attack.

message security mode

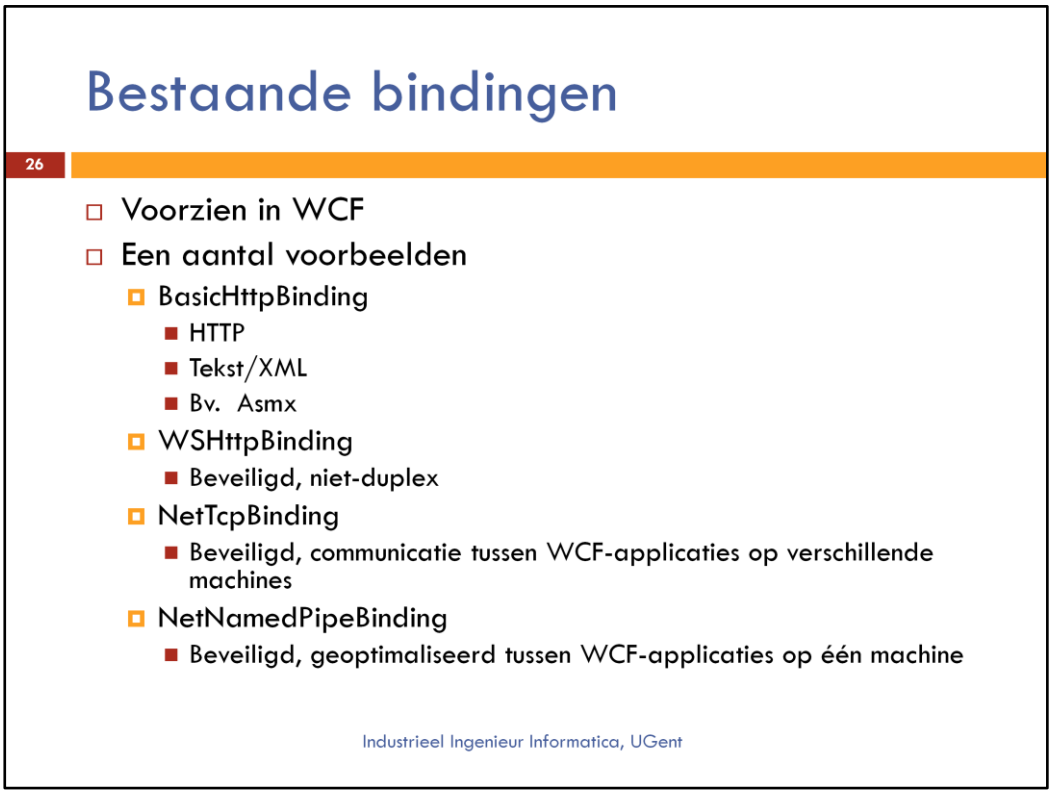
Specifies that security is provided by implementing one or more of the security specifications, such as the specification named Web Services Security: SOAP Message Security. Each message contains the necessary mechanisms to provide security during its transit, and to enable the receivers to detect tampering and to decrypt the messages. In this sense, the security is encapsulated within every message, providing end-to-end security across multiple hops. Because security information becomes part of the message, it is also possible to include multiple kinds of credentials with the message (these are referred to as claims). This approach also has the advantage of enabling the message to travel securely over any transport, including multiple transports between its origin and destination. The disadvantage of this approach is the complexity of the cryptographic mechanisms employed, resulting in performance implications.

transport with message credential security mode

Specifies the use of the transport layer to provide confidentiality, authentication, and integrity of the messages, while each of the messages can contain multiple credentials (claims) required by the receivers of the message.

WS-*

Shorthand for the growing set of Web Service (WS) specifications, such as WS-Security, WS-ReliableMessaging, and so on, that are implemented in WCF.



Bindings are objects that are used to specify the communication details that are required to connect to the endpoint of a Windows Communication Foundation (WCF) service. Each endpoint in a WCF service requires a binding to be well-specified. This topic outlines the types of communication details that the bindings define, the elements of a binding, what bindings are included in WCF, and how a binding can be specified for an endpoint.

Bindings are objects that are used to specify the communication details that are required to connect to the endpoint of a Windows Communication Foundation (WCF) service. Each endpoint in a WCF service requires a binding to be well-specified. This topic outlines the types of communication details that the bindings define, the elements of a binding, what bindings are included in WCF, and how a binding can be specified for an endpoint.

What a Binding Defines
The information in a binding can be very basic, or very complex. The most basic binding specifies only the transport protocol (such as HTTP) that must be used to connect to the endpoint. More generally, the information a binding contains about how to connect to an endpoint falls into one of the following categories.

Protocols Determines the security mechanism being used: either reliable messaging capability or transaction context flow settings.
Encoding Determines the message encoding (for example, text or binary).
Transport Determines the underlying transport protocol to use (for example, TCP or HTTP).

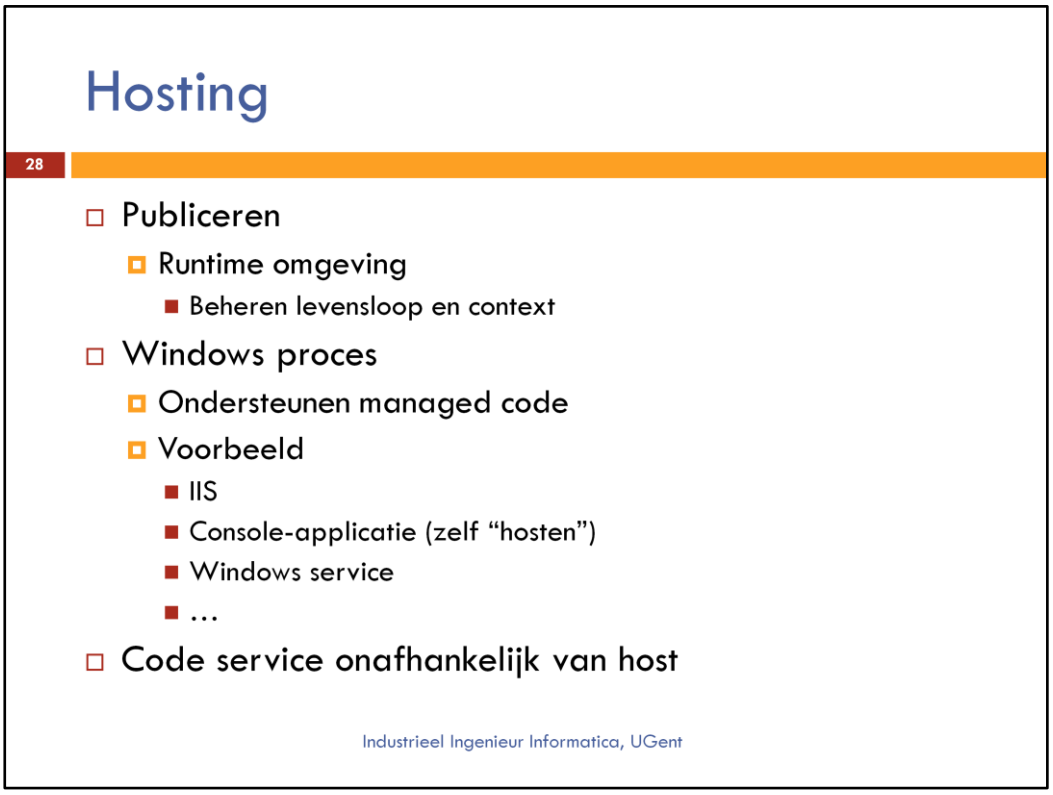
System-Provided Bindings
The information in a binding can be complex, and some settings may not be compatible with others. For this reason, WCF includes a set of system-provided bindings. These bindings are designed to cover most application requirements. The following classes represent some examples of system-provided bindings:

BasicHttpBinding: An HTTP protocol binding suitable for connecting to Web services that conforms to the WS-I Basic Profile specification (for example, ASP.NET Web services-based services).
WSHttpBinding: An interoperable binding suitable for connecting to endpoints that conform to the WS-* protocols.
NetNamedPipeBinding: Uses the .NET Framework to connect to other WCF endpoints on the same machine.
NetMsmqBinding: Uses the .NET Framework to create queued message connections with other WCF endpoints.

basicHttpBinding → adres start met “http”
netTcpBinding → adres start met “net.tcp”
netNamedPipeBinding → adres start met “net.pipe”
netMSMQBinding → adres start met “net.msmq”

Overzicht

- Ontwerp servicecontract
- Implementatie contract
- Configuratie
- Hosting
- Ontwikkelen client



To become active, a service must be hosted within a run-time environment that creates it and controls its context and lifetime. Windows Communication Foundation (WCF) services are designed to run in any Windows process that supports managed code.

WCF provides a unified programming model for building service-oriented applications. This programming model remains consistent and is independent of the run-time environment in which the service is deployed. In practice, this means that the code for your services looks much the same whatever the hosting option.

These hosting options range from running inside a console application to server environments such as a Windows service running within a worker process managed by Internet Information Services (IIS) or by Windows Process Activation Service (WAS). Developers choose the hosting environment that satisfies the service's deployment requirements. These requirements might derive from the platform on which the application is deployed, the transport on which it must send and receive messages, or on the type of process recycling and other process management required to ensure adequate availability, or on some other management or reliability requirements.

hosting

A service must be hosted in some process. A host is an application that controls the lifetime of the service. Services can be self-hosted or managed by an existing hosting process.

self-hosted service

A service that runs within a process application that the developer created. The developer controls its lifetime, sets the properties of the service, opens the service (which sets it into a listening mode), and closes the service.

hosting process

An application that is designed to host services. These include Internet Information Services (IIS), Windows Activation Services (WAS), and Windows Services. In these hosted scenarios, the host controls the lifetime of the service. For example, using IIS you can set up a virtual directory that contains the service assembly and configuration file. When a message is received, IIS starts the service and controls its lifetime.

instancing

A service has an instancing model. There are three instancing models: "single," in which a single CLR object services all the clients; "per call," in which a new CLR object is created to handle each client call; and "per session," in which a set of CLR objects is created, one for each separate session. The choice of an instancing model depends on the application requirements and the expected usage pattern of the service.

Hosten via console-applicatie

29

```
Uri baseAddress = new Uri("http://localhost:8000/ServiceModelSamples/Service");
using (ServiceHost host = new ServiceHost(typeof(CalculatorService), baseAddress)) {

    host.AddServiceEndpoint( typeof(ICalculator), new WSHttpBinding(), "CalculatorService");
    ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
    smb.HttpGetEnabled = true;
    host.Description.Behaviors.Add(smb);

    host.Open();
    Console.WriteLine("The service is ready at {0}", baseAddress);
    Console.WriteLine("Press <Enter> to stop the service.");
    Console.ReadLine();

    host.Close();
}
```

Industrieel Ingenieur Informatica, UGent

Self-Hosting in a Managed Application

WCF services can be hosted in any managed application. This is the most flexible option because it requires the least infrastructure to deploy. You embed the code for the service inside the managed application code and then create and open an instance of the ServiceHost to make the service available.

This option enables two common scenarios: WCF services running inside console applications and rich client applications such as those based on Windows Presentation Foundation (WPF) or Windows Forms (WinForms). Hosting a WCF service inside a console application is typically useful during the application's development phase. This makes them easy to debug, easy to get trace information from to find out what is happening inside of the application, and easy to move around by copying them to new locations. This hosting option also makes it easy for rich client applications, such as WPF and WinForms applications, to communicate with the outside world. For example, a peer-to-peer collaboration client that uses WPF for its user interface and also hosts a WCF service that allows other clients to connect to it and share information.

To host a service inside a managed application, embed the code for the service inside the managed application code, define an endpoint for the service either imperatively in code, declaratively through configuration, or using default endpoints, and then create an instance of ServiceHost.

To start receiving messages, call Open on ServiceHost. This creates and opens the listener for the service. Hosting a service in this way is often referred to as "self-hosting" because the managed application is doing the hosting work itself. To close the service, call CommunicationObject.Close on ServiceHost.

Hosten als Windowsservice

30

```
public class CalculatorWindowsService : ServiceBase {
    public ServiceHost serviceHost = null;
    public CalculatorWindowsService() {
        ServiceName = "WCFWindowsServiceSample"; }

    public static void Main() {
        ServiceBase.Run(new CalculatorWindowsService()); }

    protected override void OnStart(string[] args) {
        if (serviceHost != null) { serviceHost.Close(); }
        serviceHost = new   ServiceHost(typeof(CalculatorService));
        serviceHost.Open(); }

    protected override void OnStop() {
        if (serviceHost != null) {
            serviceHost.Close();
            serviceHost = null; } } }
```

```
[RunInstaller(true)]
public class ProjectInstaller : Installer {
    private ServiceProcessInstaller process;
    private ServiceInstaller service;
    public ProjectInstaller() {
        process = new ServiceProcessInstaller();
        process.Account = ServiceAccount.LocalSystem;
        service = new ServiceInstaller();
        service.ServiceName =
     "WCFWindowsServiceSample";
        Installers.Add(process);
        Installers.Add(service); }
    }
}
```

Industrieel Ingenieur Informatica, UGent

Managed Windows Services

This hosting option consists of registering the application domain (AppDomain) that hosts an WCF service as a managed Windows Service (formerly known as NT service) so that the process lifetime of the service is controlled by the service control manager (SCM) for Windows services. Like the self-hosting option, this type of hosting environment requires that some hosting code is written as part of the application. The service is implemented as both a Windows Service and as an WCF service by causing it to inherit from the ServiceBase class as well as from an WCF service contract interface. The ServiceHost is then created and opened within an overridden OnStart(String[]) method and closed within an overridden OnStop() method. An installer class that inherits from Installer must also be implemented to allow the program to be installed as a Windows Service by the Installutil.exe tool

The scenario enabled by the managed Windows Service hosting option is that of a long-running WCF service hosted outside of IIS in a secure environment that is not message-activated. The lifetime of the service is controlled instead by the operating system. This hosting option is available in all versions of Windows.

Internet Information Services (IIS)

The IIS hosting option is integrated with ASP.NET and uses the features these technologies offer, such as process recycling, idle shutdown, process health monitoring, and message-based activation. On the Windows XP and Windows Server 2003 operating systems, this is the preferred solution for hosting Web service applications that must be highly available and highly scalable. IIS also offers the integrated manageability that customers expect from an enterprise-class server product. This hosting option requires that IIS be properly configured, but it does not require that any hosting code be written as part of the application.

Note that IIS-hosted services can only use the HTTP transport.

Windows Process Activation Service (WAS)

Windows Process Activation Service (WAS) is the new process activation mechanism for the Windows Server 2008 that is also available on Windows Vista. It retains the familiar IIS 6.0 process model (application pools and message-based process activation) and hosting features (such as rapid failure protection, health monitoring, and recycling), but it removes the dependency on HTTP from the activation architecture. IIS 7.0 uses WAS to accomplish message-based activation over HTTP. Additional WCF components also plug into WAS to provide message-based activation over the other protocols that WCF supports, such as TCP, MSMQ, and named pipes. This allows applications that use communication protocols to use the IIS features such as process recycling, rapid fail protection, and the common configuration system that were only available to HTTP-based applications.

This hosting option requires that WAS be properly configured, but it does not require you to write any hosting code as part of the application.

Overzicht

- Ontwerp servicecontract
- Implementatie contract
- Configuratie
- Hosting
- Ontwikkelen client

A client application is a managed application that uses a WCF client to communicate with another application. To create a client application for a WCF service requires the following steps:

Obtain the service contract, bindings, and address information for a service endpoint.
Create a WCF client using that information.
Call operations.
Close the WCF client object.

In WCF, services and clients model contracts using managed attributes, interfaces, and methods. To connect to a service in a client application, you need to obtain the type information for the service contract. Typically, you do this by using the ServiceModel Metadata Utility Tool (Svcutil.exe), which downloads metadata from the service, converts it to a managed source code file in the language of your choice, and creates a client application configuration file that you can use to configure your WCF client object.

client application

A program that exchanges messages with one or more endpoints. The client application begins by creating an instance of a WCF client and calling methods of the WCF client. It is important to note that a single application can be both a client and a service.

metadata

In a service, describes the characteristics of the service that an external entity needs to understand to communicate with the service. Metadata can be consumed by the ServiceModel Metadata Utility Tool (Svcutil.exe) to generate a WCF client and accompanying configuration that a client application can use to interact with the service.

The metadata exposed by the service includes XML schema documents, which define the data contract of the service, and WSDL documents, which describe the methods of the service.

When enabled, metadata for the service is automatically generated by WCF by inspecting the service and its endpoints. To publish metadata from a service, you must explicitly enable the metadata behavior.

WCF client

A client-application construct that exposes the service operations as methods (in the .NET Framework programming language of your choice, such as Visual Basic or Visual C#). Any application can host a WCF client, including an application that hosts a service. Therefore, it is possible to create a service that includes WCF clients of other services.

A WCF client can be automatically generated by using the ServiceModel Metadata Utility Tool (Svcutil.exe) and pointing it at a running service that publishes metadata.

Proxy gebruiken

34

- Proxy aanmaken
- Methodes proxy oproepen
- Proxy afsluiten

A WCF client is a local object that represents a WCF service in a form that the client can use to communicate with the remote service. WCF client types implement the target service contract, so when you create one and configure it, you can then use the client object directly to invoke service operations. The WCF run time converts the method calls into messages, sends them to the service, listens for the reply, and returns those values to the WCF client object as return values or out or ref parameters.

Proxy gebruiken


```
class Client {
   static void Main() {
        CalculatorClient client = new CalculatorClient();
        double value1 = 100.00D;
        double value2 = 15.99D;
        double result = client.Add(value1, value2);

        client.Close();
   }
}
```

Voorbeeld REST

- ComponistService.svc
- http://localhost:50633/ComponistService.svc/GetComponisten?letters=b


```
{"d":[{"__type":"Componist:#Componisten","Category":"Baroque","F
irstName":"Johann Sebastian","Id":"1","LastName":"Bach"},
{"__type":"Componist:#Componisten","Category":"Classical","First
Name":"Ludwig van","Id":"8","LastName":"Beethoven"},
{"__type":"Componist:#Componisten","Category":"Classical","First
Name":"Johannes","Id":"9","LastName":"Brahms"},
{"__type":"Componist:#Componisten","Category":"Romantic","FirstN
ame":"Louis-Hector","Id":"20","LastName":"Berlioz"},
{"__type":"Componist:#Componisten","Category":"Romantic","FirstN
ame":"Georges","Id":"21","LastName":"Bizet"},
{"__type":"Componist:#Componisten","Category":"Romantic","FirstN
ame":"Bedrich","Id":"37","LastName":"Smetana"},
{"__type":"Componist:#Componisten","Category":"Post-
Romantic","FirstName":"Bela","Id":"41","LastName":"Bartok"},
{"__type":"Componist:#Componisten","Category":"Post-
Romantic","FirstName":"Leonard","Id":"42","LastName":"Bernstein"
},{"__type":"Componist:#Componisten","Category":"Post-
Romantic","FirstName":"Benjamin","Id":"43","LastName":"Britten"}
]}
```

Informatie

- WCF (http://msdn.microsoft.com/en-us/library/dd456779.aspx)