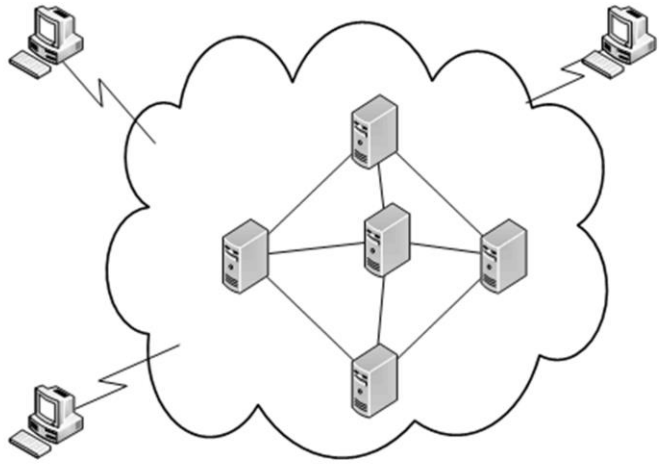
WEBSERVICES

Veerle Ongenae

Gedistribueerde applicaties

2



https://msdn.microsoft.com/en-us/library/cc239737.aspx

Industrieel Ingenieur Informatica, Universiteit Gent

Gedistribueerde applicaties/Gedistribueerd systeem

- Verschillende stukken software
  - Op computers in het netwerk
- Samenwerken
  - Via berichten
- Taken uitvoeren
  - Lijkt één applicatie voor de gebruiker

Verschillende spelers

- Verschillende computers
- Verschillende omgevingen

Gevolgen

Geen ruimtelijke beperking

- Moeilijk beheerbaar?

Aanvraag faalt. Oorzaak?

- Een van de softwarecomponenten?
- Netwerk?

Geen globale tijd

- Lokaal gegenereerde timestamps niet bruikbaar
- Geen globale toestand/status
- Softwarecomponenten kunnen parallel uitgevoerd worden
  - Inconsistenties mogelijk

Voordelen

- Bronnen delen
  - Informatie
  - Hardware
- Schaalbaar
- Kan makkelijker fouten opvangen

Gedistribueerde applicaties

3

- EDI

EDI

4

http://www.scaneurope.nl/wp/?page_id=396

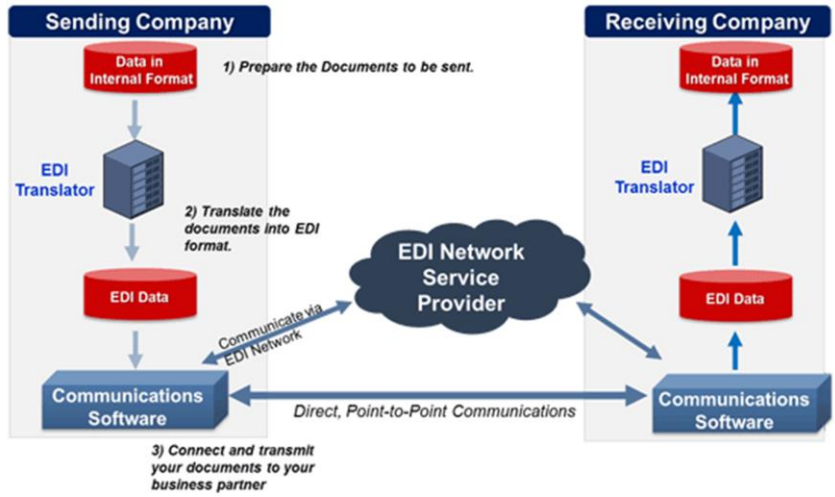
Industrieel Ingenieur Informatica, Universiteit Gent

Electronic Data Interchange (EDI) may be most easily understood as the replacement of paper-based purchase orders with electronic equivalents. It is actually much broader in its application than the procurement process, and its impacts are far greater than mere automation. EDI offers the prospect of easy and cheap communication of structured information throughout the corporate community, and is capable of facilitating much closer integration among hitherto remote organisations.

A more careful definition of EDI is 'the exchange of documents in standardised electronic form, between organisations, in an automated manner, directly from a computer application in one organisation to an application in another'.

EDI - architectuur

5



http://www.edibasics.com/what-is-edi/how-does-edi-work/

Industrieel Ingenieur Informatica, Universiteit Gent

EDI can be compared and contrasted with electronic mail (email). Email enables free-format, textual messages to be electronically transmitted from one person to another. EDI, on the other hand, supports structured business messages (those which are expressed in hard-copy, pre-printed forms or business documents), and transmits them electronically between computer applications, rather than between people.

The essential elements of EDI are:

the use of **an electronic transmission medium** (originally a value-added network, but increasingly the open, public Internet) rather than the despatch of physical storage media such as magnetic tapes and disks;

the use of **structured, formatted messages based on agreed standards** (such that messages can be translated, interpreted and checked for compliance with an explicit set of rules);

relatively fast delivery of electronic documents from sender to receiver (generally implying receipt within hours, or even minutes); and

direct communication between applications (rather than merely between computers).

EDI depends on a moderately sophisticated information technology infrastructure. This must include data processing, data management and networking capabilities, to enable the efficient capture of data into electronic form, the processing and retention of data, controlled access to it, and efficient and reliable data transmission between remote sites.

A common connection point is needed for all participants, together with a set of electronic mailboxes (so that the organisations' computers are not interrupted by one another), and security and communications management features. It is entirely feasible for organisations to implement EDI directly with one another, but it generally proves advantageous to use a third-party network services provider.

Advantages of EDI

EDI provides cost savings by reducing paper and eliminating paper processing.

Time savings and eliminating repetition are other benefits from the reduction in paper processing.

Documents can be transferred more quickly and processing errors can be decreased allowing business to be done more efficiently.

More efficient processing will likely lead to improved customer service which will ultimately expand the customer base.

Disadvantages of EDI

Contrasted to XML, which is not strictly standardized, many consider EDI to have too many standards.

There are various standards bodies who have developed 'standard document formats' for EDI which can cause problems with cross compatibility.

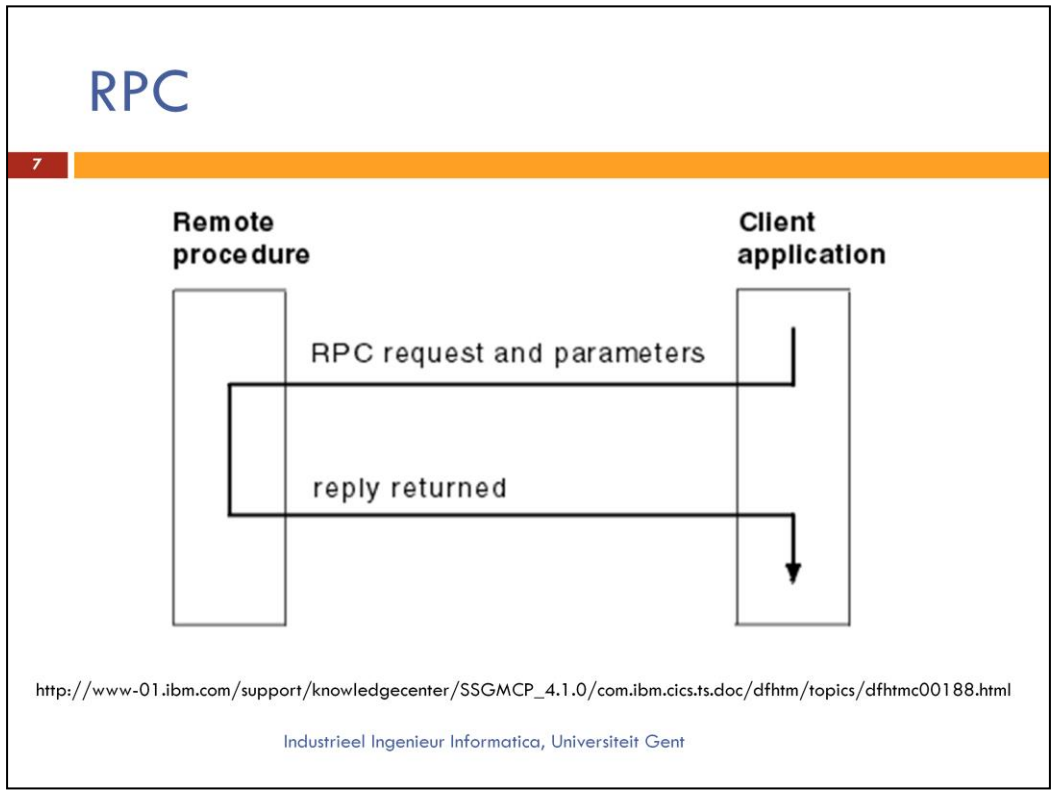
These standards bodies also push standards revisions annually which could cause problems if you have a more recent version of a document than a business partner.

EDI systems are extremely expensive making it difficult for small businesses to implement.

Many large organizations will only work with others who utilize EDI. This may limit the business small companies are able to do with such organizations and limit trading partners.

Gedistribueerde applicaties

- EDI
- RPC

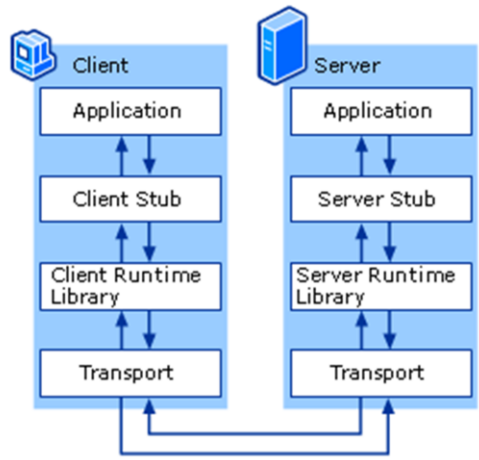


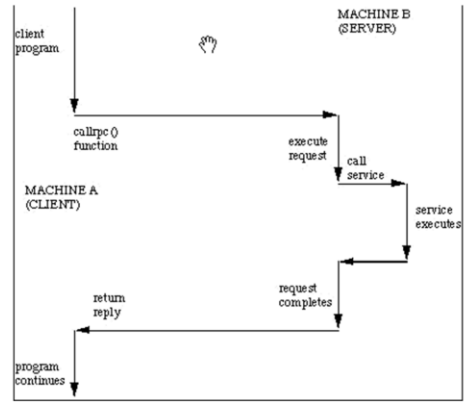
When a process invokes or calls a process on a remote system, that call is a *remote procedure call (RPC).*

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

Werking RPC

8





http://www.cs.cf.ac.uk/Dave/C/node33.html#SECTION00331000000000000000

https://technet.microsoft.com/en-us/library/cc738291%28v=ws.10%29.aspx

Industrieel Ingenieur Informatica, Universiteit Gent

The RPC process starts on the client side. The client application calls a local stub procedure instead of code implementing the procedure. Stubs are compiled and linked with the client application during development. Instead of containing code that implements the remote procedure, the client stub code retrieves the required parameters from the client address space and delivers them to the client runtime library. The client runtime library then translates the parameters as needed into a standard Network Data Representation (NDR) format for transmission to the server.

When the server receives the RPC, either locally or from a remote client, the server RPC runtime library functions accept the request and call the server stub procedure. The server stub retrieves the parameters from the network buffer and, using one of the NDR marshalling engines, converts them from the network transmission format to the format required by the server. The server stub calls the actual procedure on the server. The remote procedure then runs, possibly generating output parameters and a return value. When the remote procedure is complete, a similar sequence of steps returns the data to the client.

The remote procedure returns its data to the server stub which, using one of the NDR marshalling engines, converts output parameters to the format required for transmission back to the client and returns them to the RPC

runtime library functions. The server RPC runtime library functions transmit the data to the client computer using either LRPC or the network.

The client completes the process by accepting the data over the network and returning it to the calling function. The client RPC runtime library receives the remote-procedure return values, converts the data from its NDR to the format used by the client computer, and returns them to the client stub.

The server application contains calls to the server runtime library functions, which register the server's interface with the RPC runtime and allow the server to accept remote procedure calls. The server application also contains the application-specific remote procedures that are called by the client applications.

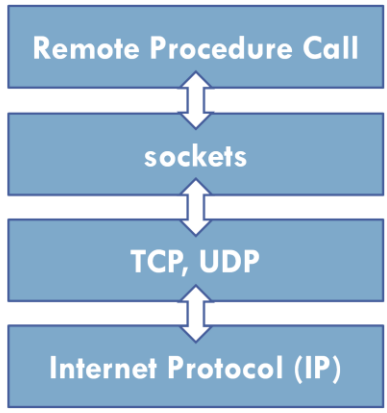
The client stub then calls functions in the RPC client runtime library to send the request and its parameters to the server. If the server is located on the same host as the client, the runtime library can use the Local RPC (LRPC) function and pass the RPC request to the Windows kernel for transport to the server. If the server is located on a remote host, the runtime library specifies an appropriate transport protocol engine and passes the RPC to the network stack for transport to the server.

An RPC is analogous to a function call. Like a function call, when an RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure. The second figure shows the flow of activity that takes place during an RPC call between two networked systems. The client makes a procedure call that sends a request to the server and waits. The thread is blocked from processing until either a reply is received, or it times out. When the request arrives, the server calls a dispatch routine that performs the requested service, and sends the reply to the client. After the RPC call is completed, the client program continues. RPC specifically supports network applications.

Vaak in C.

Communicatie tussen applicaties

9



Remote Procedure Call:
Verbergt communicatie details als een aanroep van een procedure op een server ~ proxy pattern

sockets:
API voor onderliggende communicatieprotocollen

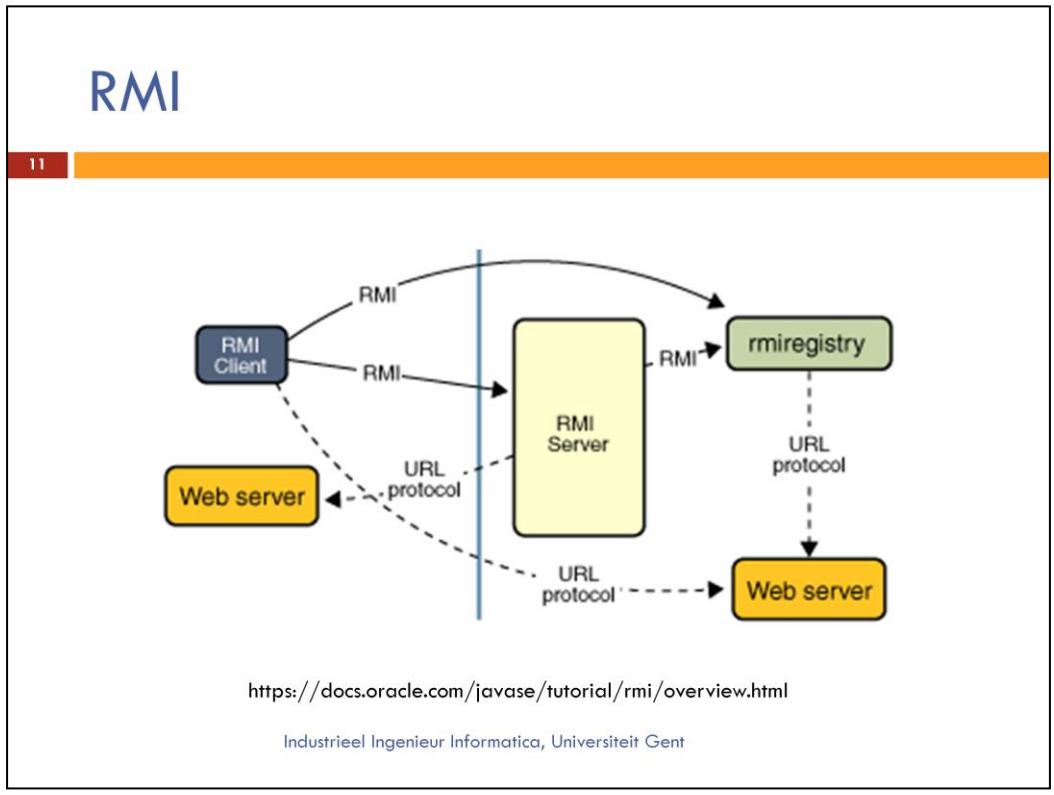
TCP, UDP:
UDP: transport datapakketen zonder garantie
TCP: verzekerde datastromen

Internet Protocol (IP):
Verstuurt datapakketten

Industrieel Ingenieur Informatica, Universiteit Gent

Gedistribueerde applicaties

- EDI
- RPC
- RMI, DCOM, CORBA



Remote Method Invocation - Java

RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a *distributed object application*.

Distributed object applications need to do the following:

Locate remote objects. Applications can use various mechanisms to obtain references to remote objects. For example, an application can register its remote objects with RMI's simple naming facility, the RMI registry. Alternatively, an application can pass and return remote object references as part of other remote invocations.

Communicate with remote objects. Details of communication between remote objects are handled by RMI. To the programmer, remote communication looks similar to regular Java method invocations.

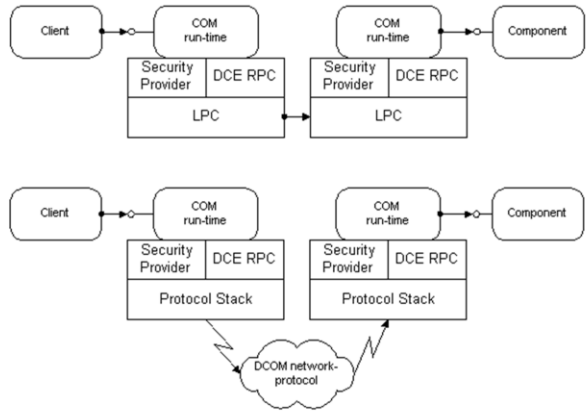
Load class definitions for objects that are passed around. Because RMI enables objects to be passed back and forth, it provides mechanisms for loading an object's class definitions as well as for transmitting an object's

data.

The following illustration depicts an RMI distributed application that uses the RMI registry to obtain a reference to a remote object. The server calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it. The illustration also shows that the RMI system uses an existing web server to load class definitions, from server to client and from client to server, for objects when needed.

DCOM

12



http://www.active-undelete.com/dcom-overview.htm

Industrieel Ingenieur Informatica, Universiteit Gent

The Distributed Component Object Model (DCOM) Remote Protocol is a protocol for exposing application objects by way of remote procedure calls (RPCs). The protocol consists of a set of extensions layered on Microsoft Remote Procedure Call Protocol Extensions as specified in [MS-RPCE].

The DCOM Remote Protocol is also referred to as Object RPC or ORPC.

Microsoft® Distributed COM (DCOM) extends the Component Object Model (COM) to support communication among objects on different computers-on a LAN, a WAN, or even the Internet. Because DCOM is a seamless evolution of COM you can take advantage of your existing investment in COM-based applications, components, tools, and knowledge to move into the world of standards-based distributed computing. DCOM handles low-level details of network protocols so you can focus on providing solutions to your customers.

DCOM is an extension of the Component Object Model (COM). COM defines how components and their clients interact. This interaction is defined such that the client and the component can connect without the need of any intermediary system component. The client calls methods in the component without any overhead whatsoever.

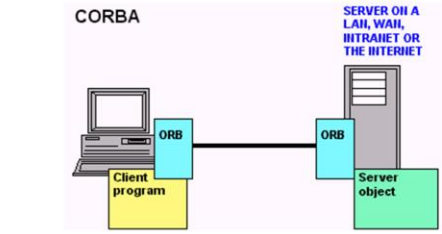
In today's operating systems, processes are shielded from each other. A client that needs to communicate with a component in another process cannot call the component directly, but has to use some form of interprocess communication provided by the operating system. COM provides this communication in a completely transparent fashion: it intercepts calls from the client and forwards them to the component in another process. The first figure illustrates how the COM/DCOM run-time libraries provide the link between client and component.

When client and component reside on different machines, DCOM simply replaces the local interprocess communication with a network protocol. Neither the client nor the component is aware that the wire that connects them has just become a little longer.

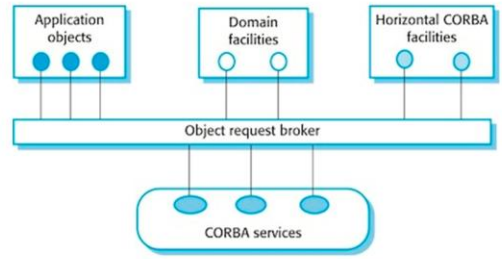
The second figure shows the overall DCOM architecture: The COM run-time provides object-oriented services to clients and components and uses RPC and the security provider to generate standard network packets that conform to the DCOM wire-protocol standard.

CORBA

13



http://www.pcmag.com/encyclopedia/term/40365/corba



https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/DistribSys/Corba.html

Industrieel Ingenieur Informatica, Universiteit Gent

CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate.

The Object Management Group (OMG) is responsible for defining CORBA. The OMG comprises over 700 companies and organizations, including almost all the major vendors and developers of distributed object technology, including platform, database, and application vendors as well as software tool and corporate developers.

CORBA defines an architecture for distributed objects. The basic CORBA paradigm is that of a request for services of a distributed object. Everything else defined by the OMG is in terms of this basic paradigm.

The services that an object provides are given by its interface. Interfaces are defined in OMG's Interface Definition Language (IDL). Distributed objects are identified by object references, which are typed by IDL interfaces.

Application objects that are designed and implemented for this application.

Standard objects that are defined by the OMG for a specific domain. These domain object standards cover finance/insurance, electronic commerce, healthcare, and a number of other areas.

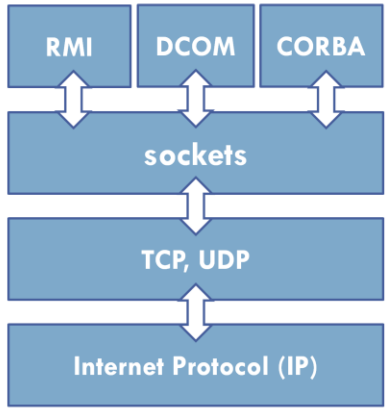
Fundamental CORBA services that provide basic distributed computing services such as directories, security management, etc.

Horizontal CORBA facilities such as user interface facilities, system management facilities, etc. The term horizontal facilities suggests that these facilities are common to many application domains and the facilities are therefore used in many different applications.

Communicatie tussen applicaties

14

- Gedistribueerd programmeren populairder → nood aan extra functionaliteit en flexibiliteit



Gespecialiseerde middleware, volgens RPC-model, met extra functionaliteit

sockets:
API voor onderliggende communicatieprotocollen

TCP, UDP:
UDP: transport datapakketen zonder garantie
TCP: verzekerde datastromen

Internet Protocol (IP):
Verstuurt datapakketten

Industrieel Ingenieur Informatica, Universiteit Gent

Gedistribueerde object-infrastructuur over het Internet

Transportprotocol bovenop TCP/IP (verschillend voor CORBA, RMI en DCOM) voor verschillende platformen

Afspraken over communicatie tussen objecten

Voordeel: de verschillende servers hoeven niet tot het zelfde netwerk te behoren

Nadeel: Kunnen enkel met dezelfde infrastructuur praten, tenzij extra lagen op een reeds ingewikkelde structuur

Gedistribueerde applicaties

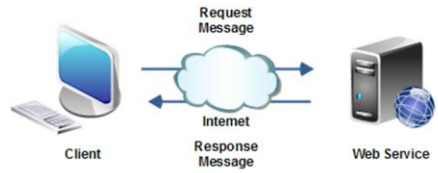
15

- EDI
- RPC
- RMI, DCOM, CORBA
- Webservices
  - Definitie

Webservice - definitie

16

- Software service (dienst)
- Toegankelijk via internet
  - Via bestaande protocollen bv. HTTP



http://tutorials.jenkov.com/web-services/message-formats.html

Industrieel Ingenieur Informatica, Universiteit Gent

Softwaresysteem

Communicatie tussen computers (applicaties)

Over een netwerk

Mogelijkheid tot het uitwisselen van data overal op het web

Geen koppeling tussen data en transport

Meestal niet binair

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Services versus Applications

Services

Perform a single or a few specialized operations.

Most often accessed by other programs.

Often (but not always) targets part of a larger problem domain.

Applications

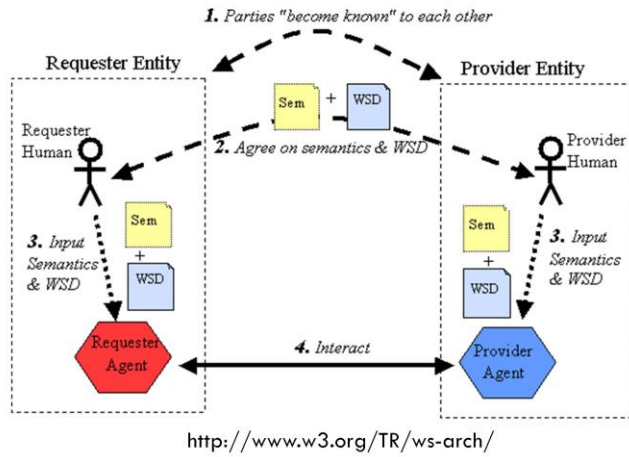
Perform a wide range of operations, and may even expose some of these operations as services.

Often (but not always) accessed by humans.

Often (but not always) targets a whole problem domain.

Webservice – entities

17



http://www.w3.org/TR/ws-arch/

Interface

- Beschreven in een formaat leesbaar door computers
- WSDL

Interactie met webservice

- SOAP-berichten
  - Meestal over HTTP
  - Inhoud XML

A Web service is an abstract notion that must be implemented by a concrete agent. The agent is the concrete piece of software or hardware that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided. To illustrate this distinction, you might implement a particular Web service using one agent one day (perhaps written in one programming language), and a different agent the next day (perhaps written in a different programming language) with the same functionality. Although the agent may have changed, the Web service remains the same.

The purpose of a Web service is to provide some functionality on behalf of its owner -- a person or organization, such as a business or an individual.

The provider entity is the person or organization that provides an appropriate agent to implement a particular service.

A requester entity is a person or organization that wishes to make use of a provider entity's Web service. It will use a requester agent to exchange messages with the provider entity's provider agent.

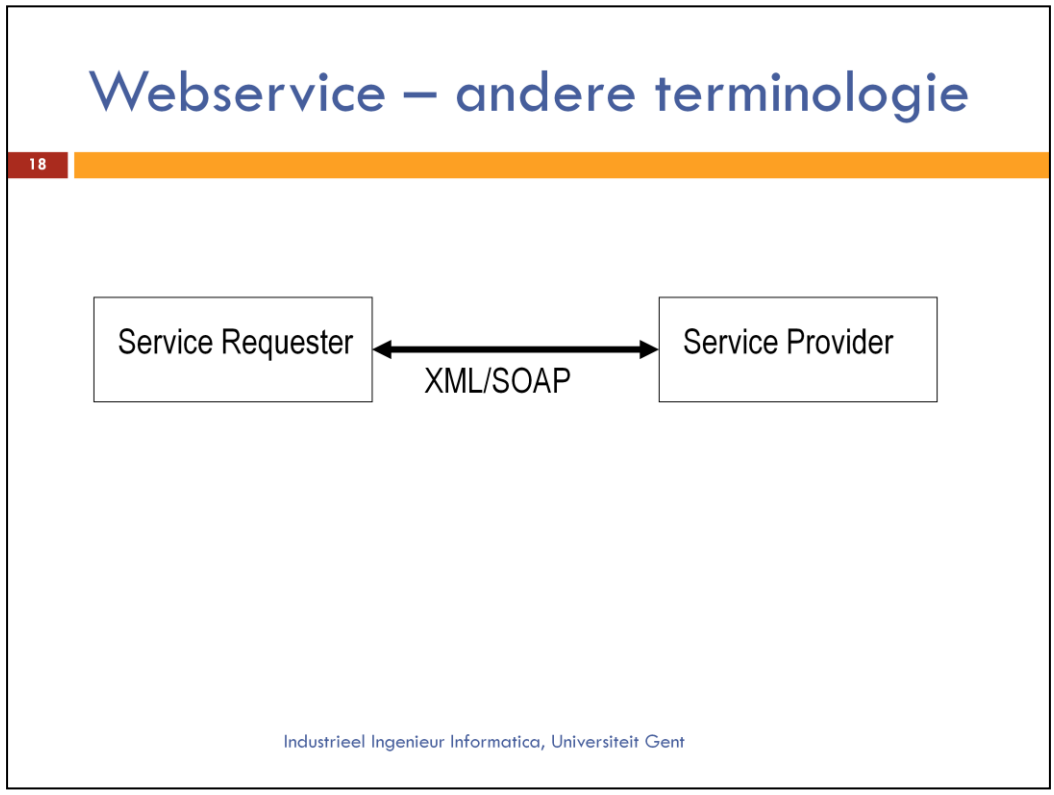
(In most cases, the requester agent is the one to initiate this message exchange, though not always. Nonetheless, for consistency we still use the term "requester agent" for the agent that interacts with the provider agent, even in cases when the provider agent actually initiates the exchange.)

In order for this message exchange to be successful, the requester entity and the provider entity must first agree on both the semantics and the mechanics of the message exchange.

The mechanics of the message exchange are documented in a Web service description (WSD). The WSD is a machine-processable specification of the Web service's interface, written in WSDL. It defines the message formats, datatypes, transport protocols, and transport serialization formats that should be used between the requester agent and the provider agent. It also specifies one or more network locations at which a provider agent can be invoked, and may provide some information about the message exchange pattern that is expected. In essence, the service description represents an agreement governing the mechanics of interacting with that service.

The semantics of a Web service is the shared expectation about the behavior of the service, in particular in response to messages that are sent to it. In effect, this is the "contract" between the requester entity and the provider entity regarding the purpose and consequences of the interaction. Although this contract represents the overall agreement between the requester entity and the provider entity on how and why their respective agents will interact, it is not necessarily written or explicitly negotiated. It may be explicit or implicit, oral or written, machine processable or human oriented, and it may be a legal agreement or an informal (non-legal) agreement.

While the service description represents a contract governing the mechanics of interacting with a particular service, the semantics represents a contract governing the meaning and purpose of that interaction. The dividing line between these two is not necessarily rigid. As more semantically rich languages are used to describe the mechanics of the interaction, more of the essential information may migrate from the informal semantics to the service description. As this migration occurs, more of the work required to achieve successful interaction can be automated.



Service provider

Biedt een webservice aan

Service requester

Maakt gebruik van een webservice

A word on terminology: Many documents use the term service provider to refer to the provider entity and/or provider agent. Similarly, they may use the term service requester to refer to the requester entity and/or requester agent. However, since these terms are ambiguous -- sometimes referring to the agent and sometimes to the person or organization that owns the agent -- this document prefers the terms requester entity, provider entity, requester agent and provider agent.

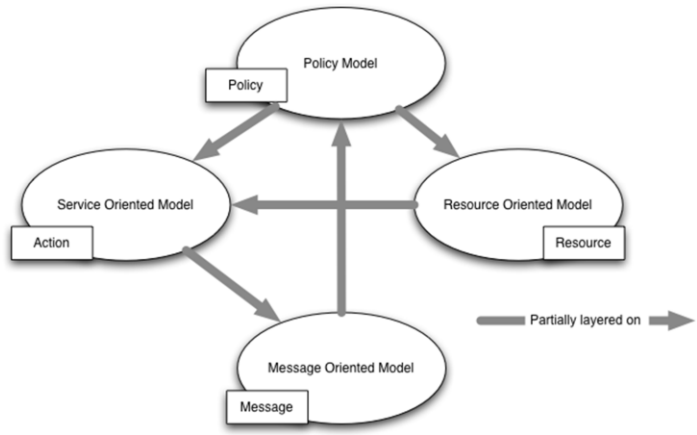
Gedistribueerde applicaties

19

- EDI
- RPC
- RMI, DCOM, CORBA
- Webservices
  - Definitie
  - Architectuur

Architecturale modellen

20

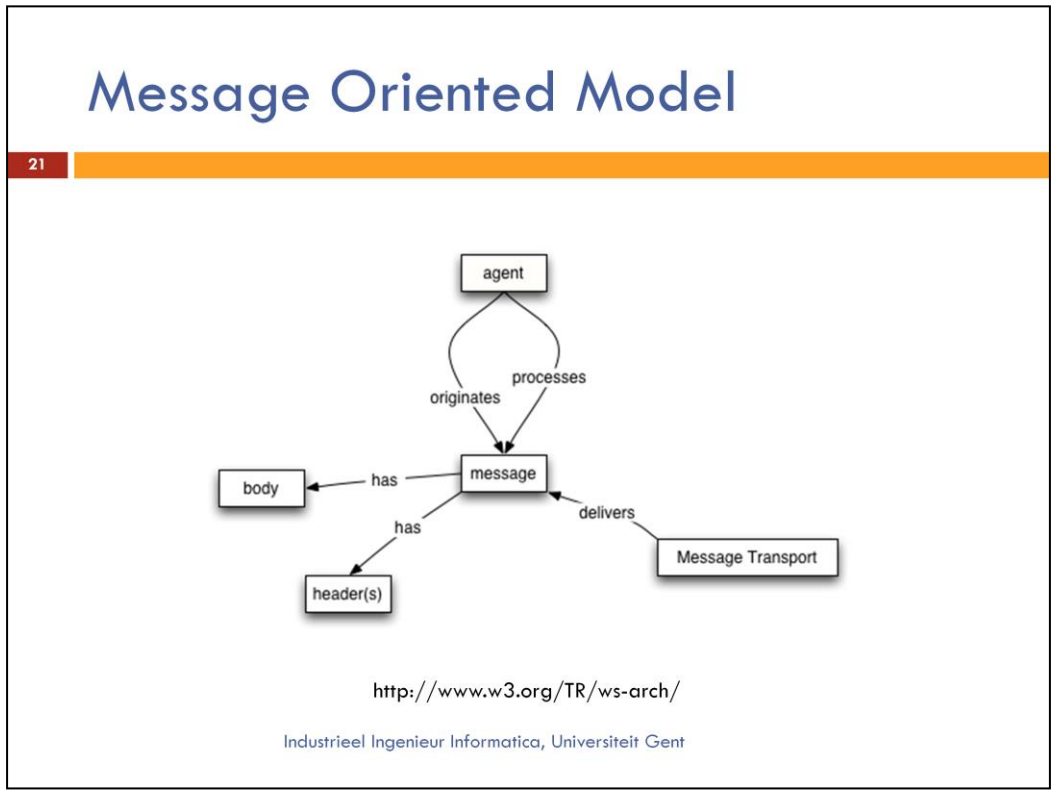


http://www.w3.org/TR/ws-arch/

Industrieel Ingenieur Informatica, Universiteit Gent

A model is a coherent subset of the architecture that typically revolves around a particular aspect of the overall architecture. Although different models share concepts, it is usually from different points of view; the major role of a model is to explain and encapsulate a significant theme within the overall Web services architecture.

For example, the Message Oriented Model focuses and explains Web services strictly from a message passing perspective. In particular, it does not attempt to relate messages to services provided. The Service Oriented Model, however, lays on top of and extends the Message Oriented Model in order to explain the fundamental concepts involved in service - in effect to explain the purpose of the messages in the Message Oriented Model.



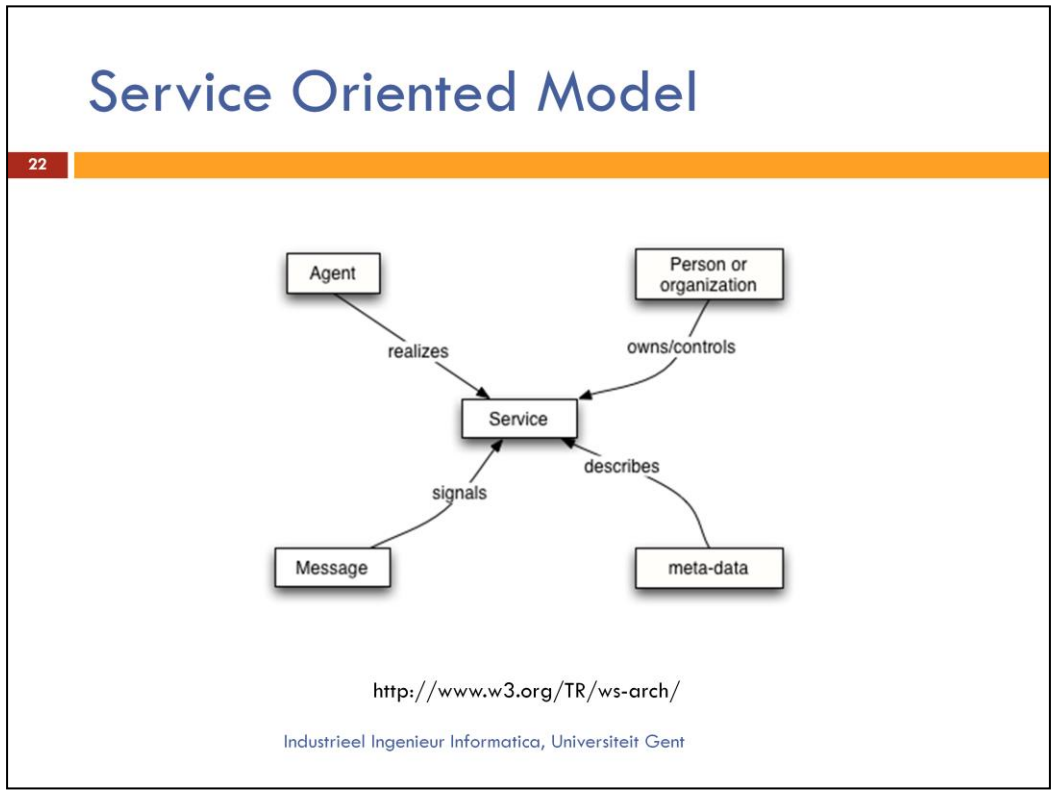
Simplified Message Oriented Model

The Message Oriented Model focuses on messages, message structure, message transport and so on — without particular reference as to the reasons for the messages, nor to their significance.

The essence of the message model revolves around a few key concepts illustrated above: the agent that sends and receives messages, the structure of the message in terms of message headers and bodies and the mechanisms used to deliver messages. Of course, there are additional details to consider: the role of policies and how they govern the message level model.

A message represents the data structure passed from its sender to its recipients. The structure of a message is defined in a service description.

Examples of delivery policies include quality of service assurances — such as reliable versus best effort message delivery — and security assurances — such as encrypted versus unencrypted message transport. Another kind of delivery policy could take the form of assertions about recording an audit of how the message was delivered.



Simplified Service Oriented Model

The Service Oriented Model focuses on aspects of service, action and so on. While clearly, in any distributed system, services cannot be adequately realized without some means of messaging, the converse is not the case: messages do not need to relate to services.

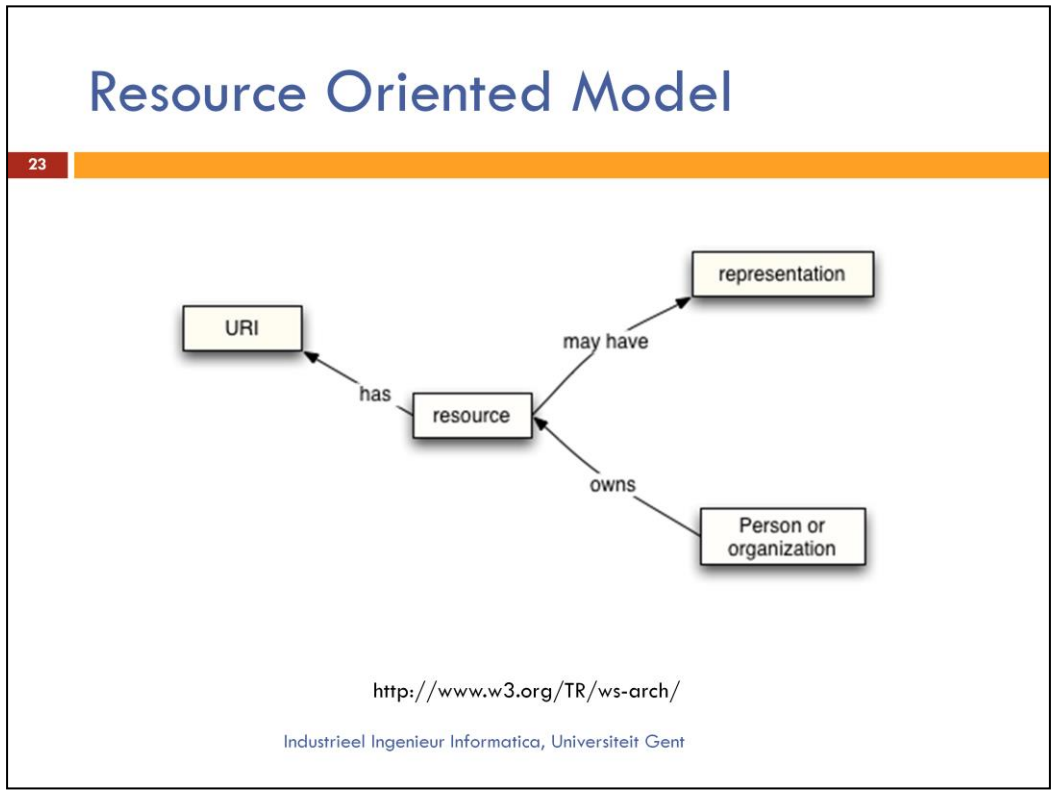
The Service Oriented Model is the most complex of all the models in the architecture. However, it too revolves around a few key ideas. A service is realized by an agent and used by another agent. Services are mediated by means of the messages exchanged between requester agents and provider agents.

A very important aspect of services is their relationship to the real world: services are mostly deployed to offer functionality in the real world. We model this by elaborating on the concept of a service's owner — which, whether it is a person or an organization, has a real world responsibility for the service.

Finally, the Service Oriented Model makes use of meta-data, which is a key property of Service Oriented Architectures. This meta-data is used to

document many aspects of services: from the details of the interface and transport binding to the semantics of the service and what policy restrictions there may be on the service. Providing rich descriptions is key to successful deployment and use of services across the Internet.

A service is an abstract resource that represents a person or organization in some collection of related tasks as having specific service roles. The service may be realized by one or more provider agents that act on behalf of the person or organization — the provider entity.

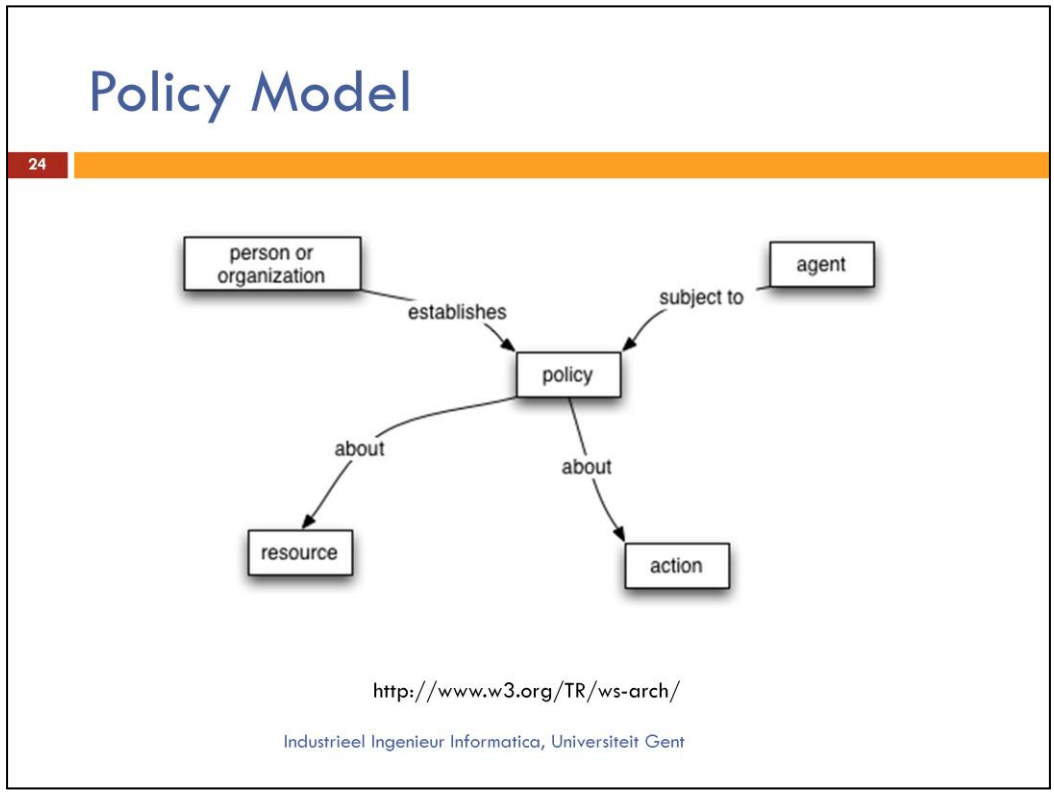


Simplified Resource Oriented Model

The Resource Oriented Model focuses on resources that exist and have owners.

The resource model is adopted from the Web Architecture concept of resource. We expand on this to incorporate the relationships between resources and owners.

Discovery is the act of locating a machine-processable description of a Web service-related resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions. The goal is to find an appropriate Web service-related resource.



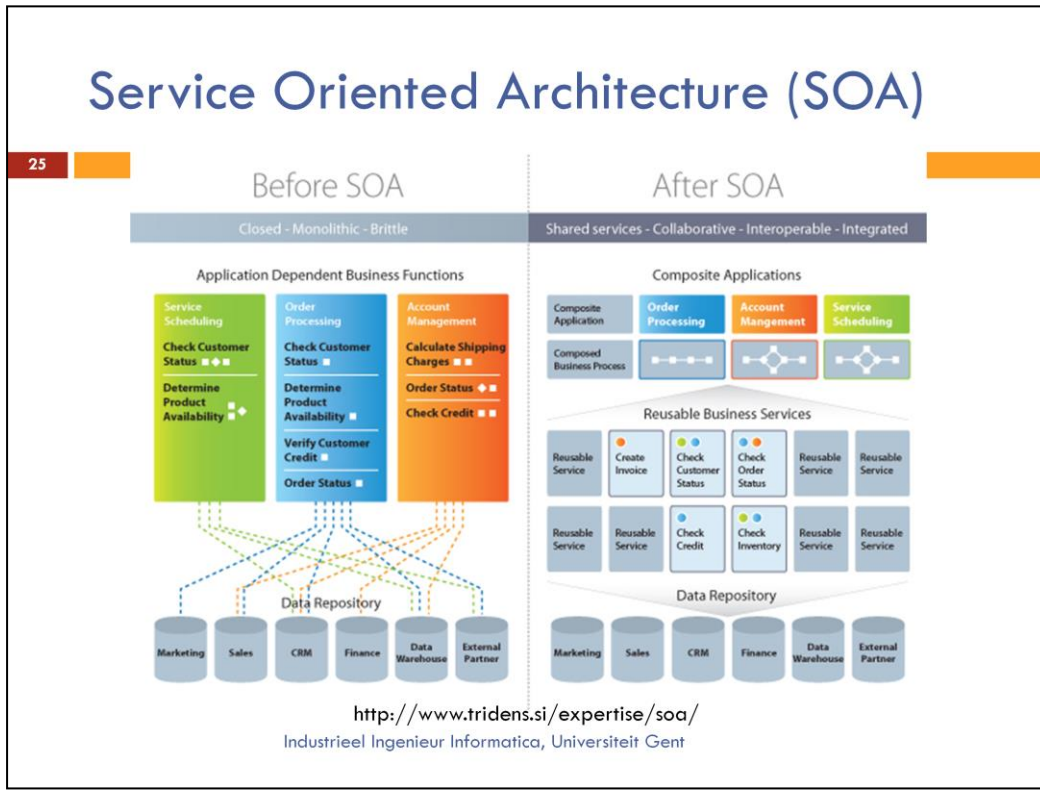
Simplified Policy Model

The Policy Model focuses on constraints on the behavior of agents and services. We generalize this to resources since policies can apply equally to documents (such as descriptions of services) as well as active computational resources.

Policies are about resources. They are applied to agents that may attempt to access those resources, and are put in place, or established, by people who have responsibility for the resource.

Policies may be enacted to represent security concerns, quality of service concerns, management concerns and application concerns.

Beperkingen en verplichtingen.



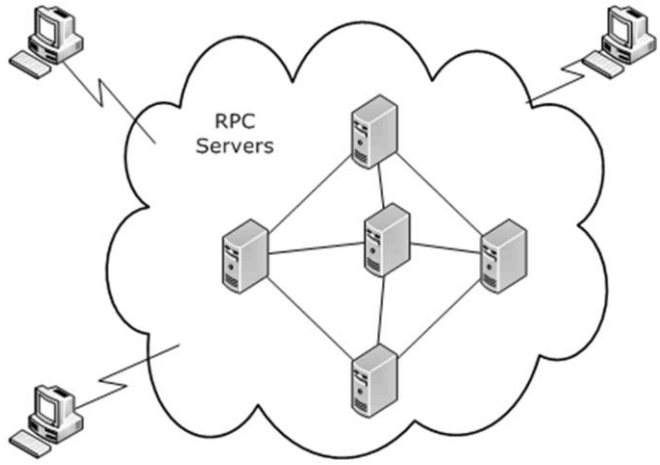
SOA is more than an IT solution; it is a paradigm shift, taking the focus off IT and putting it back where it belongs – on customers, service quality, and fundamental business objectives. The overall benefits are a greater return on investment (ROI) and a more flexible technology system, one better able to support the implementation of your strategic decisions.

Basically, SOA is about describing, building, using, and managing your IT environment, concentrating upon the services it supplies, rather than upon the technology it employs. When enterprises adopt a service-oriented architecture, they describe their IT investment in terms of modular services, each delivering specific value. A well architected, governed, and managed SOA environment enables enterprises to quickly create, combine, and deploy services so as to respond to rapidly changing business needs.

Architectuur voor gedistribueerde systemen

Gedistribueerde systemen

26



https://msdn.microsoft.com/en-us/library/cc239737.aspx

Industrieel Ingenieur Informatica, Universiteit Gent

A **distributed system** consists of diverse, discrete software agents that must work together to perform some tasks. Furthermore, the agents in a distributed system do not operate in the same processing environment, so they must communicate by hardware/software protocol stacks over a network. This means that communications with a distributed system are **intrinsically less fast and reliable** than those using direct code invocation and shared memory. This has important architectural implications because distributed systems require that developers (of infrastructure and applications) **consider the unpredictable latency of remote access**, and take into account **issues of concurrency and the possibility of partial failure**.

Distributed object systems are distributed systems in which the semantics of object initialization and method invocation are exposed to remote systems by means of a proprietary or standardized mechanism to broker requests across system boundaries, marshall and unmarshall method argument data, etc. Distributed objects systems typically (albeit not necessarily) are characterized by objects maintaining a fairly complex internal state required to support their methods, a fine grained or "chatty" interaction between an object and a program using it, and a focus on a shared implementation type system and interface hierarchy between the object and the program that uses it.

SOA - kenmerken

27

- Logical view
- Message orientation
- Description orientation
- Granularity
- Network orientation
- Platform neutral

Industrieel Ingenieur Informatica, Universiteit Gent

A Service Oriented Architecture (SOA) is a form of distributed systems architecture that is typically characterized by the following properties:

Logical view: The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.

Service is abstract gedefinieerd

Los van eigenlijke implementatie

Message orientation: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA: using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application that can be "wrapped" in message handling code that allows it to adhere to the formal service definition.

Definitie service aan de hand van berichten

Welke berichten verwacht service

Welke antwoordbericht stuurt service

Description orientation: A service is described by machine-processable meta data. The description supports the public nature of the SOA: only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.

Beschrijving service

Leesbaar voor computer

Enkel wat nodig is om de service te gebruiken

Granularity: Services tend to use a small number of operations with relatively large and complex messages.

Beperkt aantal opdrachten

Grote en complex berichten

Network orientation: Services tend to be oriented toward use over a network, though this is not an absolute requirement.

Service aangesproken over netwerk

Niet noodzakelijk

Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

Berichten in gestandariseerd formaat

Meestal XML

Onafhankelijk van een type besturingssysteem, programmeertaal, …

Uitdagingen en problemen SOA

28

- Traagheid en (on)betrouwbaarheid gebruikte transport
- Geen gedeeld geheugen voor requester en provider
- Wat indien een deel van de opdracht mislukt?
- Gelijktijdige toegang tot bronnen
- Wat als één partner incompatibel wordt

Industrieel Ingenieur Informatica, Universiteit Gent

Distributed object systems have a number of architectural challenges.

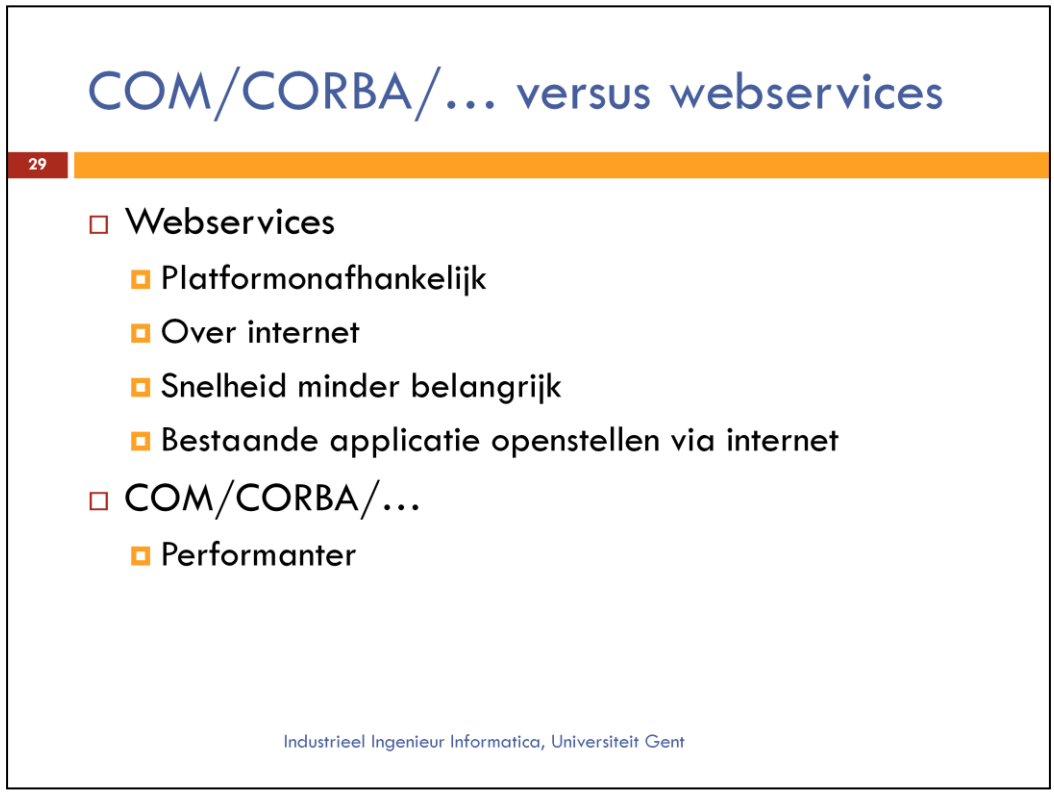
Problems introduced by latency and unreliability of the underlying transport.

The lack of shared memory between the caller and object.

The numerous problems introduced by partial failure scenarios.

The challenges of concurrent access to remote resources.

The fragility of distributed systems if incompatible updates are introduced to any participant.



These challenges apply irrespective of whether the distributed object system is implemented using COM/CORBA or Web services technologies. Web services are no less appropriate than the alternatives if the fundamental criteria for successful distributed object architectures are met. If these criteria are met, Web services technologies may be appropriate if the benefits they offer in terms of platform/vendor neutrality offset the performance and implementation immaturity issues they may introduce.

Conversely, using Web services technologies to implement a distributed system doesn't magically turn a distributed object architecture into an SOA. Nor are Web services technologies necessarily the best choice for implementing SOAs -- if the necessary infrastructure and expertise are in place to use COM or CORBA as the implementation technology and there is no requirement for platform neutrality, using SOAP/WSDL may not add enough benefits to justify their costs in performance, etc.

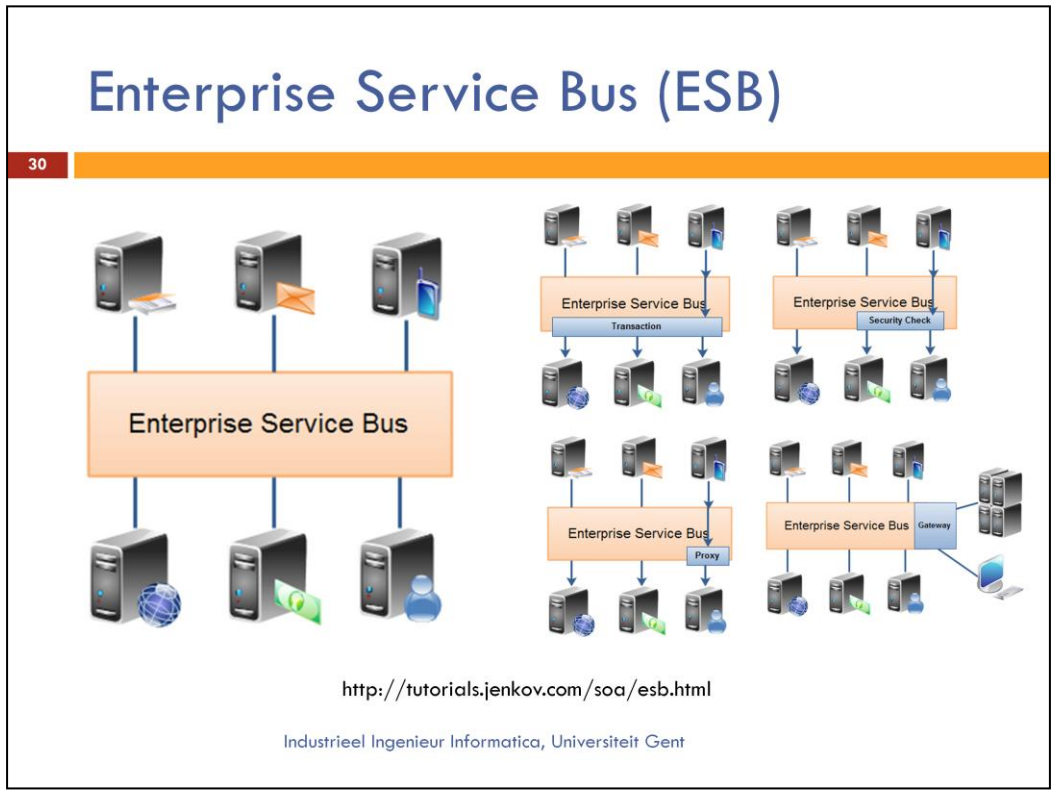
In general SOA and Web services are most appropriate for applications:

That must operate over the Internet where reliability and speed cannot be guaranteed;

Where there is no ability to manage deployment so that all requesters and providers are upgraded at once;

Where components of the distributed system run on different platforms and vendor products;

Where an existing application needs to be exposed for use over a network, and can be wrapped as a Web service.



An "Enterprise Service Bus" (ESB) is a system to which all services are connected. Through the enterprise service bus all connected services can also be accessed.

The term "bus" is an analogy to the internal bus of a computer onto which the CPU, RAM and other chips are connected. An enterprise service bus is typically implemented as a server or a set of servers, and is thus more than just a "network".

Clients and services connected to an enterprise service bus do not communicate directly. They communicate via the ESB. This is done by having the ESB essentially expose the same service interface to potential clients, that the connected services expose to the ESB. This indirection via the ESB has some advantages and disadvantages.

ESB as Single Point of Access

One advantage of connecting clients and services via an enterprise service bus is that clients need only look for services in a single location: The enterprise service bus. If a service is moved from one server to another, you only need to reconfigure the ESB. The clients still just access the service via

the ESB.

ESB as Transaction Manager

Another advantage is that the ESB can coordinate distributed transactions which multiple services participate in. When multiple distributed services need to participate in a transaction some entity typically has to coordinate the transaction. Rather than forcing the client to do this, the enterprise service bus can do so. The client may still have to demarcate the beginning and end of the transaction, even if the work of coordinating the participants is done by the ESB.

ESB as Security Manager

Security aspects like authentication and authorization can be centralized in the enterprise service bus. Even if a service in an application does not have authentication and authorization, the enterprise service bus can require this in the service interface it exposes to potential clients.

ESB as Service Proxy

An ESB may function as a gateway or proxy for applivations that do not expose a standardized service interface to the world. For instance, lets say an application exposes a Java RMI service. The rest of your network is running on .NET which cannot directly call the RMI service.

To solve this problem you can implement a service proxy in Java which can call the RMI service. The service proxy then exposes a web service interface (SOAP + WSDL) via the ESB to the .NET applications.

Such a proxy service does not have to be a built-in capability of an ESB. It can also just be deployed as a separate service, made available via the ESB.

ESB as Gateway to the World

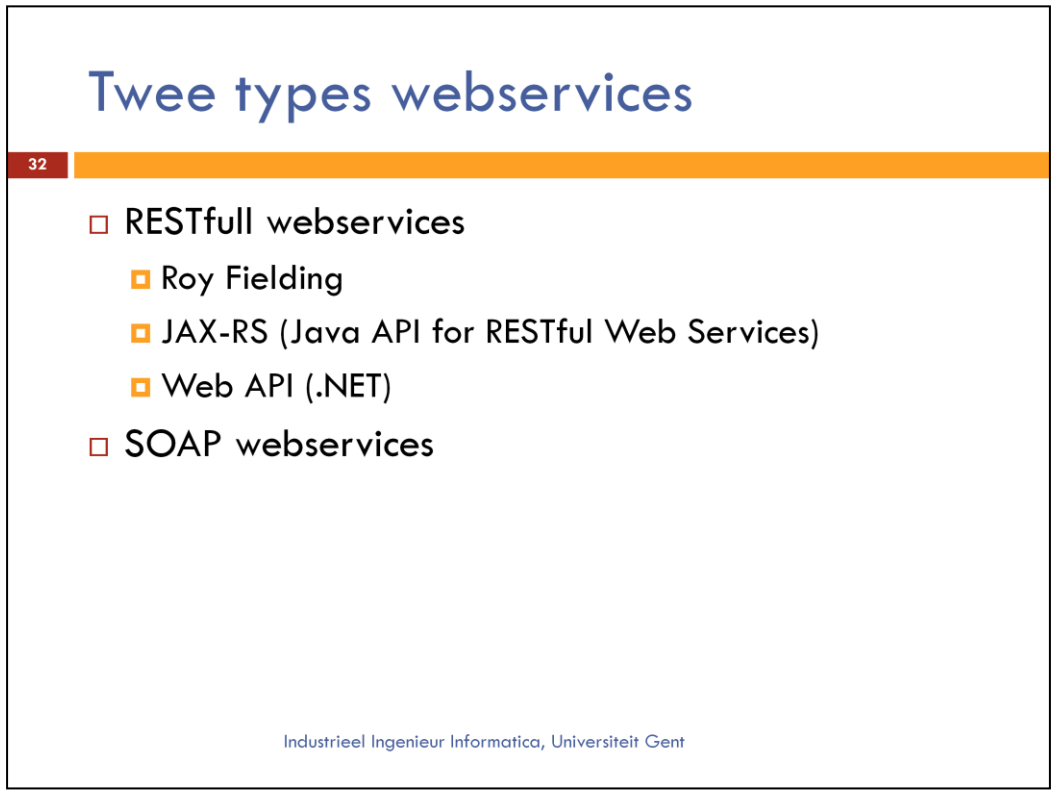
If some clients need to connect to services running in the outside world, the ESB can potentially function as a gateway to the world outside. Again, security aspects etc. can be added ontop of the external service. Furthermore, if the external service is capable of participating in distributed transactions, the ESB can coordinate this too.

ESB Disadvantages

Connecting your services via an ESB also has a few disadvantages. First of all the ESB may become a single point of failure. If the ESB is down, no communication between clients and services can take place. Second, the extra level of indirection may result in decreased performance of client-service communication.

Gedistribueerde applicaties

- EDI
- RPC
- RMI, DCOM, CORBA
- Webservices
  - Definitie
  - Architectuur
  - Technologieën
    - SOAP



The World Wide Web operates as a networked information system that imposes several constraints: Agents identify objects in the system, called resources, with Uniform Resource Identifiers (URIs). Agents represent, describe, and communicate resource state via representations of the resource in a variety of widely-understood data formats (e.g. XML, HTML, CSS, JPEG, PNG). Agents exchange representations via protocols that use URIs to identify and directly or indirectly address the agents and resources.

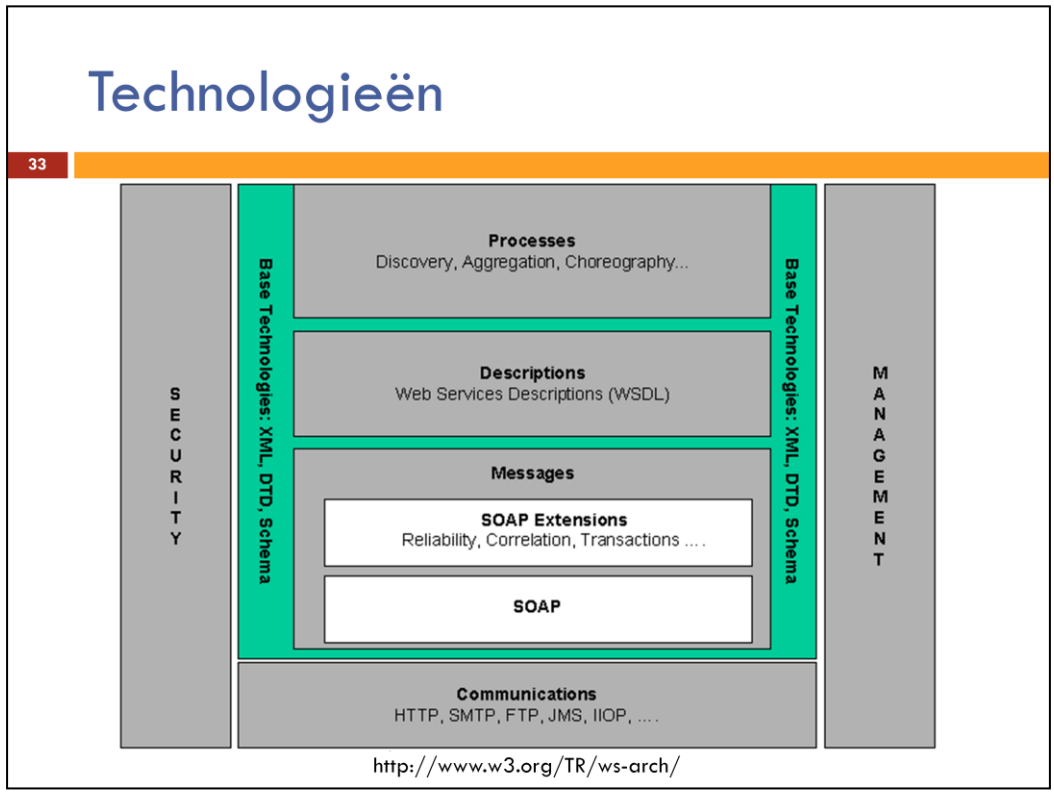
An even more constrained architectural style for reliable Web applications known as Representation State Transfer (REST) has been proposed by Roy Fielding and has inspired both the W3C Technical Architecture Group's architecture document and many who see it as a model for how to build Web services. The REST Web is the subset of the WWW (based on HTTP) in which agents provide uniform interface semantics -- essentially create, retrieve, update and delete -- rather than arbitrary or application-specific interfaces, and manipulate resources only by the exchange of representations. Furthermore, the REST interactions are "stateless" in the sense that the meaning of a message does not depend on the state of the conversation.

We can identify two major classes of Web services:

REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and

arbitrary Web services, in which the service may expose an arbitrary set of operations.

Both classes of Web services use URIs to identify resources and use Web protocols (such as HTTP and SOAP 1.2) and XML data formats for messaging. (It should be noted that SOAP 1.2 can be used in a manner consistent with REST. However, SOAP 1.2 can also be used in a manner that is not consistent with REST.)



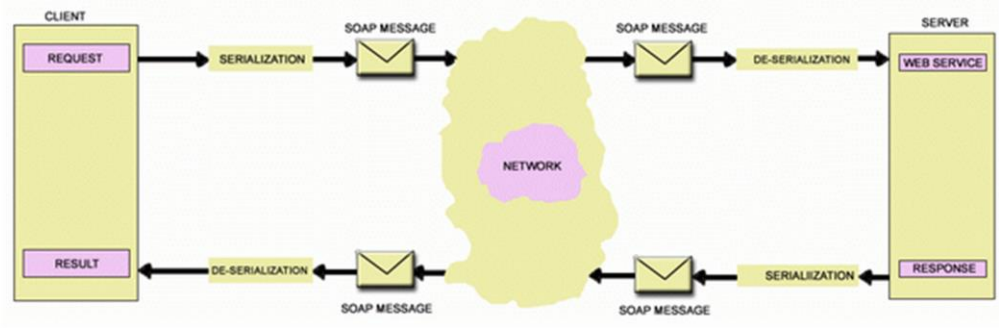
A choreography is a model of the sequence of operations, states, and conditions that control the interactions involved in the participating services. The interaction prescribed by a choreography results in the completion of some useful function. Examples include the placement of an order, information about its delivery and eventual payment, or putting the system into a well-defined error state.

A choreography can be distinguished from an orchestration. An orchestration defines the sequence and conditions in which *one* Web service invokes other Web services in order to realize some useful function.

A choreography may be described using a choreography description language. A choreography description language permits the description of how Web services can be composed, how service roles and associations in Web services can be established, and how the state, if any, of composed services is to be managed.

SOAP webservices

34



http://www.codeproject.com/Articles/6399/Securing-web-services-using-SOAP-extensions

Industrieel Ingenieur Informatica, Universiteit Gent

Framework

- Protocol gebaseerd op XML

Uitwisseling van informatie in een gedecentraliseerde, gedistribueerde omgeving

- Versturen van XML-berichten

Transport via het web (HTTP, …)

Betekenis

- Simple Object Acces Protocol
- Service Oriented Architecture Protocol

SOAP specificatie omvat

XML-envelop

- XML-tags die een SOAP-bericht beschrijven en die een framework geven voor de inhoud van het bericht
- Met XML-inhoud
  - Conventies voor het uitvoeren van methodes op een andere server en het voorstellen van het resultaat

Een aantal regels voor servers die een SOAP-boodschap ontvangen

Maakt gebruik van bestaande transportprotocols (bv. HTTP)

Oorspronkelijk Microsoft, ondertussen standaard van W3C

SOAP 1.2 provides a standard, extensible, composable framework for packaging and exchanging XML messages. In the context of this architecture, SOAP 1.2 also provides a convenient mechanism for referencing capabilities (typically by use of headers).

[SOAP 1.2 Part 1] defines an XML-based messaging framework: a processing model and an extensibility model. SOAP messages can be carried by a variety of network protocols; such as HTTP, SMTP, FTP, RMI/IIOP, or a proprietary messaging protocol.

[SOAP 1.2 Part 2] defines three optional components: a set of encoding rules for expressing instances of application-defined data types, a convention for representing remote procedure calls (RPC) and responses, and a set of rules for using SOAP with HTTP/1.1.

While SOAP Version 1.2 [SOAP 1.2 Part 1] doesn't define "SOAP" as an acronym anymore, there are two expansions of the term that reflect these different ways in which the technology can be interpreted:

Service Oriented Architecture Protocol: In the general case, a SOAP message represents the information needed to invoke a service or reflect the results of a service invocation, and contains the information specified in the service interface definition.

Simple Object Access Protocol: When using the optional SOAP RPC Representation, a SOAP message represents a method invocation on a remote object, and the serialization of in the argument list of that method that must be moved from the local environment to the remote environment.

A SOAP message is fundamentally a one-way transmission between SOAP nodes, from a SOAP sender to a SOAP receiver, but SOAP messages are expected to be combined by applications to implement more complex interaction patterns ranging from request/response to multiple, back-and-forth "conversational" exchanges.

Voorbeeld SOAP-aanvraag

35

```
POST /ZwiftBooks HTTP/1.1
Host: www.zwiftbooks.com
Content-Type:text/xml
Content-Length: 134
SOAP Action: “Some-URI”

<Envelope xmlns=“http://www.w3.org/2003/05/soap-envelope”
  encodingStyle=“http://www.w3.org/2003/05/soap-encoding”>
   <Body>
      <zwiftbooks:GetGuaranteedDeliveryTime
        xmlns:zwiftbooks=“www.zwiftbooks.com”>
          <zwiftbooks:isbn>0-13-18188-222-2</zwiftbooks:isbn>
          <zwiftbooks:zipcode>75240</zwiftbooks:zipcode>
      </zwifbooks:GetGuaranteedDeliveryTime>
   </Body>
</Envelope>
```


De eerste lijn en de headerlijnen blijven bestaan

Het corpus van een HTTP-aanvraag en HTTP-antwoord bevat XML ipv HTML, …

Analoog via FTP of SMTP

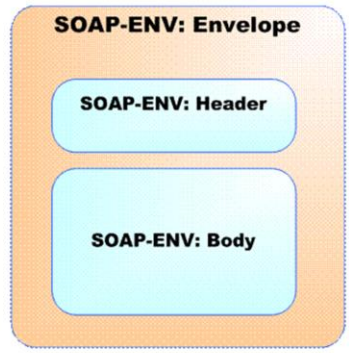
Voorbeeld SOAP-antwoord


```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 122

<Envelope xmlns="http://www.w3.org/2003/05/soap-
  envelope">
   <Body>
      <zwiftbooks:GetBestDeliveryTimeResponse
        xmlns:zwiftbooks="www.zwiftbooks.com">
          <zwiftbooks:Time>2 hours</zwiftbooks:Time>
      </zwiftbooks:GetBestDeliveryTimeResponse>
   </Body>
</Envelope>
```

Structuur SOAP Bericht

37



https://mytechprep.wordpress.com/tag/soap-actor/

Industrieel Ingenieur Informatica, Universiteit Gent

SOAP-envelop:

- Envelope-element
  - Attributen: encodingStyle: hoe geserialiseerd wordt
- Rootelement van het XML-document dat een SOAP-bericht voorstelt
- Verplicht

SOAP-hoofding:

- Header-element
- Extra richtlijnen en informatie voor SOAP-server

SOAP-corpus:

- Body-element
- Bevat de uit te wisselen XML
  - Onafhankelijk van SOAP-structuur
  - Domein-specifieke structuur
- Inhoud
  - Data/Resultaat
  - Aanroep van een methode
- Verplicht

Voorbeeld: SOAP-hoofding

38

```
<header>
   <n:alertcontrol
     xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
   </n:alertcontrol>
</header>
```

Industrieel Ingenieur Informatica, Universiteit Gent

SOAP-hoofding:

- Header-element
- Extra richtlijnen en informatie voor SOAP-server
  - Transacties
  - Beveiliging
  - Filters
  - …
- Optioneel

SOAP-berichtenpad

39

https://myshadesofgray.wordpress.com/category/programming/java/

Industrieel Ingenieur Informatica, Universiteit Gent

Bestaat uit SOAP-knopen:

- SOAP-zender
- Nul of meerdere tussenliggende SOAP-servers
- SOAP-ontvanger

Tussenliggende SOAP-servers

- Stuurt SOAP-bericht door
- Behandelt de voor hem bedoelde SOAP-hoofdingen

Elke SOAP-knoop heeft één of meerdere rollen

Mogelijke rollen

- Gespecifieerd door SOAP:
  - none (niet bedoeld voor een specifieke knoop)
  - next (ontvanger en alle tussenliggende)
  - ultimateReceiver (ontvanger)
- Gespecifieerd door de applicatie
  - Proxy
  - Beveiliging
  - Caching
  - …

Voorbeeld: SOAP-hoofding

40

```
<Header role="http://www.w3.org/2003/05/soap-
  envelope/role/next">
  <n:alertcontrol
    xmlns:n="http://example.org/alertcontrol"
    mustUnderstand="true">
    <n:priority>1</n:priority>
    <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
</Header>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Een SOAP-hoofding kan aangeven voor welke SOAP-knoop de hoofding bedoeld is door gebruik te maken van rollen

- Attribuut role
- Waarde attribuut
  - Rol gespecifieerd door SOAP
  - URI SOAP-knoop

Regels voor SOAP-server

Identificatie van de stukken van het SOAP-bericht die voor de huidige server bedoeld zijn

Nagaan of deze server alle delen van de headers die bedoeld zijn voor deze server en die het attribuut mustUnderstand hebben, ondersteunt. Indien niet fout genereren

Uitvoeren van de stukken van de headers bedoeld voor deze server

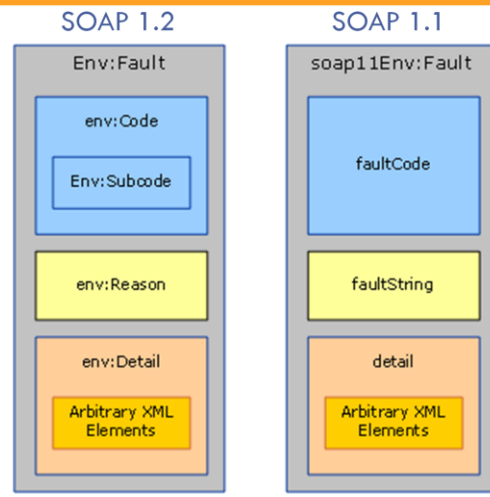
Indien niet de SOAP-ontvanger: alle headers bedoeld voor deze server verwijderen en bericht doorsturen

SOAP headers have been designed in anticipation of various uses for SOAP, many of which will involve the participation of other SOAP processing nodes - called SOAP intermediaries - along a message's path from an initial

SOAP sender to an ultimate SOAP receiver. This allows SOAP intermediaries to provide value-added services. Headers may be inspected, inserted, deleted or forwarded by SOAP nodes encountered along a SOAP message path. (It should be kept in mind, though, that the SOAP specifications do not deal with what the contents of header elements are, or how SOAP messages are routed between nodes, or the manner by which the route is determined and so forth. These are a part of the overall application, and could be the subject of other specifications.)

SOAP-fouten

41



https://msdn.microsoft.com/en-us/library/ms789039%28v=vs.110%29.aspx

Wanneer

- Niet begrijpen bericht
- Fout bij behandelen van bericht

fault-element

- Kind van body-element
- Kinderen (SOAP 1.1)
  - faultcode: foutcode
  - faultstring: leesbare verklaring
  - detail: informatie over het probleem

SOAP 1.2 provides a model for handling situations when faults arise in the processing of a message. SOAP distinguishes between the conditions that result in a fault, and the ability to signal that fault to the originator of the faulty message or another node. The ability to signal the fault depends on the message transfer mechanism used, and one aspect of the binding specification of SOAP onto an underlying protocol is to specify how faults are signalled, if at all. The remainder of this section assumes that a transfer mechanism is available for signalling faults generated while processing received messages, and concentrates on the structure of the SOAP fault message.

The SOAP env:Body element has another distinguished role in that it is the place where such fault information is placed. The SOAP fault model requires that all SOAP-specific and application-specific faults be reported using a single distinguished element, env:Fault, carried within the env:Body element. The env:Fault element contains two mandatory sub-elements, env:Code and env:Reason, and (optionally) application-specific information in the env:Detail sub-element. Another optional sub-element, env:Node, identifies via a URI the SOAP node which generated the fault, its absence implying that it was the ultimate recipient of the message which did so. There is yet another optional sub-element, env:Role, which identifies the role being played by the node which generated the fault.

The env:Code sub-element of env:Fault is itself made up of a mandatory env:Value sub-element, whose content is specified in the SOAP specification (see SOAP Part 1 section 5.4.6) as well as an optional env:Subcode sub-element.

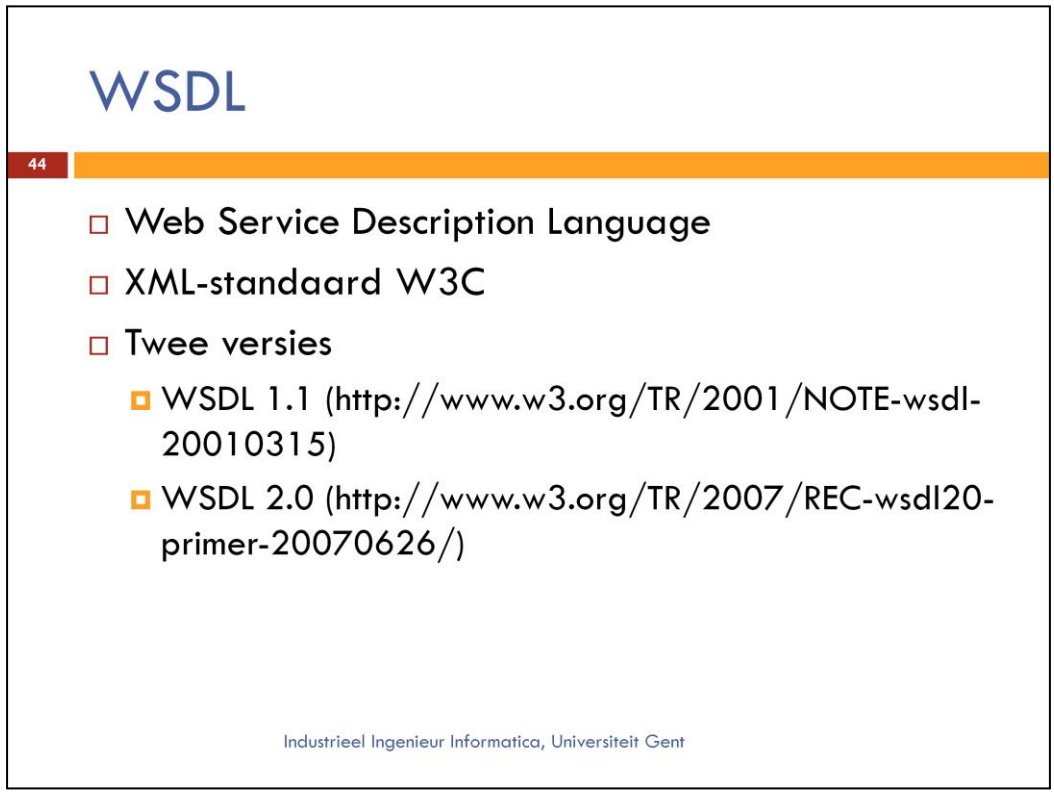
Voorbeeld SOAP-fouten

42

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env= “” xmlns:rpc='http://www.w3.org/2003/05/soap-rpc'>
  <env:Body> <env:Fault>
     <env:Code>
       <env:Value>env:Sender</env:Value>
       <env:Subcode><env:Value>rpc:BadArguments</env:Value></env:Subcode></env:Co
     <env:Reason>
      <env:Text xml:lang="en-US">Processing error</env:Text></env:Reason>
     <env:Detail>
      <e:myFaultDetails xmlns:e="http://travelcompany.example.org/faults">
        <e:message>Name does not match card number</e:message><e:errorcode>999</e
      </e:myFaultDetails>
     </env:Detail>
   </env:Fault> </env:Body>
</env:Envelope>
```

Gedistribueerde applicaties

- EDI
- RPC
- RMI, DCOM, CORBA
- Webservices
  - Definitie
  - Architectuur
  - Technologieën
    - SOAP
    - WSDL



Beschrijven webservice

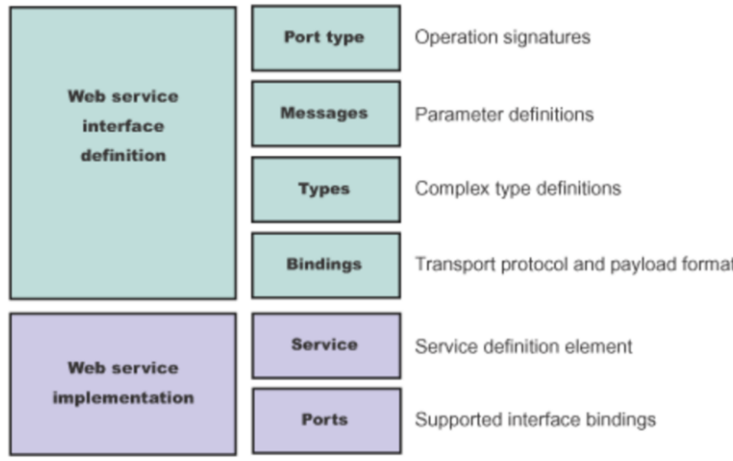
- Interface
- Implementatiedetails

Bepaalt hoe de service requester de aangeboden dienst (service provider) moet aanspreken

Web Services Description Language (WSDL) provides a model and an XML format for describing Web services. WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered.

WSDL 1.1

45



https://www-01.ibm.com/support/knowledgecenter/SSAW57_8.0.0/com.ibm.websphere.nd.doc/info/ae/ae/cwbs_wsdl.html

Industrieel Ingenieur Informatica, Universiteit Gent

Specificatie interface webservice

- Leesbaar door computers

Definieert

- Formaat berichten
- Datatypes
- Transportprotocollen
- Serialisatieformaten voor transport
- Netwerklocaties
- Patroon om berichten uit te wisselen

Types: Type-declaraties van argumenten, methodes, teruggeefwaardes, … (XML Schema)

Message: Beschrijving bericht (aanvraag/antwoord)

PortType: Beschrijving mogelijke opdracht (naam, input, output)

Binding: Verbinden portType met een locatie en een protocol (SOAP, HTTP GET, …)

Service: Uit welke portTypes bestaat de webservice

As communications protocols and message formats are standardized in the

web community, it becomes increasingly possible and important to be able to describe the communications in some structured way. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication.

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

Types– a container for data type definitions using some type system (such as XSD).

Message– an abstract, typed definition of the data being communicated.

Operation– an abstract description of an action supported by the service.

Port Type–an abstract set of operations supported by one or more endpoints.

Binding– a concrete protocol and data format specification for a particular port type.

Port– a single endpoint defined as a combination of a binding and a network address.

Service– a collection of related endpoints.

It is important to observe that WSDL does not introduce a new type definition language. WSDL recognizes the need for rich type systems for describing message formats, and supports the XML Schemas specification (XSD) as its canonical type system. However, since it is unreasonable to expect a single type system grammar to be used to describe all message formats present and future, WSDL allows using other type definition languages via extensibility.

In addition, WSDL defines a common binding mechanism. This is used to attach a specific protocol or data format or structure to an abstract message, operation, or endpoint. It allows the reuse of abstract definitions.

In addition to the core service definition framework, this specification introduces specific binding extensions for the following protocols and message formats:

SOAP

HTTP GET / POST

MIME

Voorbeeld WSDL 1.1


```
<?xml version="1.0"?>
<definitions name="StockQuote“
   targetNamespace=http://example.com/stockquote.wsdl
   xmlns:tns=http://example.com/stockquote.wsdl
   xmlns:xsd1=http://example.com/stockquote.xsd
   xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
   xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
       <schema targetNamespace=http://example.com/stockquote.xsd
        xmlns="http://www.w3.org/2000/10/XMLSchema">
          <element name="TradePriceRequest">
             <complexType>…</complexType>
          </element>
          <element name="TradePrice">
             <complexType> …</complexType>
          </element>
       </schema>
    </types>
```

Voorbeeld WSDL 1.1

47

```
<message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
</message>

<portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
       <input message="tns:GetLastTradePriceInput"/>
       <output message="tns:GetLastTradePriceOutput"/>
    </operation>
</portType>
```

Industrieel Ingenieur Informatica, Universiteit Gent

WSDL 1.1 has four transmission primitives that an endpoint can support:

One-way. The endpoint receives a message.

```
<wsdl:operation name="nmtoken">
  <wsdl:input name="nmtoken"? message="qname"/>
</wsdl:operation>
```

Request-response. The endpoint receives a message, and sends a correlated message.

```
<wsdl:operation name="nmtoken" parameterOrder="nmtokens">
  <wsdl:input name="nmtoken"? message="qname"/>
  <wsdl:output name="nmtoken"? message="qname"/>
  <wsdl:fault name="nmtoken" message="qname"/>*
</wsdl:operation>
```

Solicit-response. The endpoint sends a message, and receives a correlated message.

```
<wsdl:operation name="nmtoken" parameterOrder="nmtokens">
  <wsdl:output name="nmtoken"? message="qname"/>
  <wsdl:input name="nmtoken"? message="qname"/>
  <wsdl:fault name="nmtoken" message="qname"/>*
```

Notification. The endpoint sends a message.

```
<wsdl:operation name="nmtoken">
  <wsdl:output name="nmtoken"? message="qname"/>
</wsdl:operation>
```

Although the base WSDL 1.1 structure supports bindings for these four transmission primitives, WSDL only defines bindings for the One-way and Request-response primitives. It is expected that specifications that define the protocols for Solicit-response or Notification would also include WSDL binding extensions that allow use of these primitives.

Voorbeeld WSDL 1.1

48

```
<binding name="StockQuoteSoapBinding"
   type="tns:StockQuotePortType">
    <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
       <soap:operation
   soapAction="http://example.com/GetLastTradePrice"/>
       <input>
           <soap:body use="literal"/>
       </input>
       <output>
           <soap:body use="literal"/>
       </output>
    </operation>
</binding>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Style: document, rpc

The style attribute indicates whether the operation is RPC-oriented (messages containing parameters and return values) or document-oriented (message containing document(s)). This information may be used to select an appropriate programming model. The value of this attribute also affects the way in which the Body of the SOAP message is constructed. If the attribute is not specified, it defaults to the value specified in the soap:binding element. If the soap:binding element does not specify a style, it is assumed to be "document".

Alternatieve bindings: HTTP GET & POST Binding

The soapAction attribute specifies the value of the SOAPAction header for this operation. This URI value should be used directly as the value for the SOAPAction header; no attempt should be made to make a relative URI value absolute when making the request. For the HTTP protocol binding of SOAP, this is value required (it has no default value). For other SOAP protocol bindings, it MUST NOT be specified, and the soap:operation element MAY be omitted.

The soap:body element specifies how the message parts appear inside the SOAP Body element.

The parts of a message may either be abstract type definitions, or concrete schema definitions. If abstract definitions, the types are serialized according to some set of rules defined by an encoding style. Each encoding style is identified using a list of URIs, as in the SOAP specification. Since some encoding styles such as the SOAP Encoding (http://schemas.xmlsoap.org/soap/encoding/) allow variation in the message format for a given set of abstract types, it is up to the reader of the message to understand all the format variations: "reader makes right". To avoid having to support all variations, a message may be defined concretely and then indicate it's original encoding style (if any) as a hint. In this case, the writer of the message must conform exactly to the specified schema: "writer makes right".

The required **use** attribute indicates whether the message parts are encoded using some encoding rules, or whether the parts define the concrete schema of the message.

If use is **encoded**, then each message part references an abstract type using the **type** attribute. These abstract types are used to produce a concrete message by applying an encoding specified by the **encodingStyle** attribute. The part **names**, **types** and value of the **namespace** attribute are all inputs to the encoding, although the namespace attribute only applies to content not explicitly defined by the abstract types. If the referenced encoding style allows variations in it's format (such as the SOAP encoding does), then all variations MUST be supported ("reader makes right").

If use is **literal**, then each part references a concrete schema definition using either the **element** or **type** attribute. In the first case, the element referenced by the part will appear directly under the Body element (for document style bindings) or under an accessor element named after the message part (in rpc style). In the second, the type referenced by the part becomes the schema type of the enclosing element (Body for document style or part accessor element for rpc style).

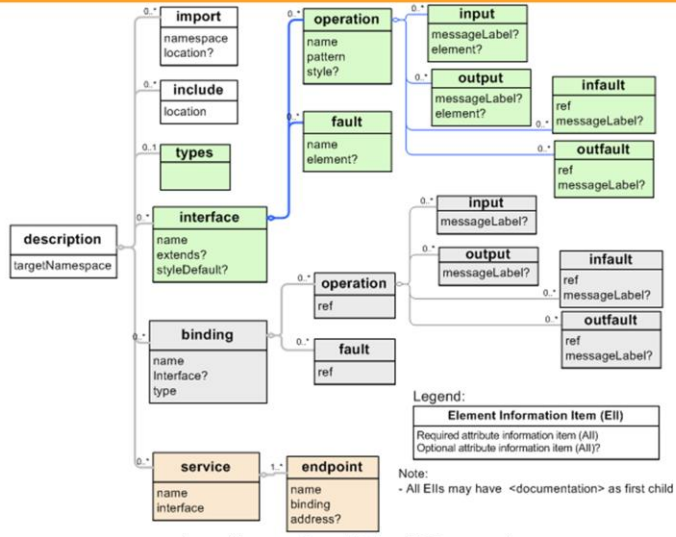
Voorbeeld WSDL 1.1


```
    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```

WSDL 2.0 -structuur

50



http://www.w3.org/TR/wsdl20-primer/

Industrieel Ingenieur Informatica, Universiteit Gent

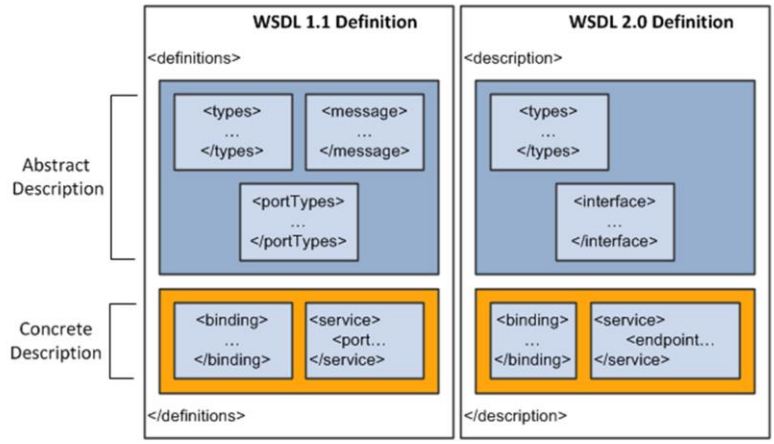
types: beschrijving berichten in XML Schema

interface: abstracte funcionaliteit

binding: hoe toegang tot diensten

service: waar toegang tot diensten

WSDL 1.0 versus 2.0



http://docs.oracle.com/cd/E41633_01/pt853pbh1/eng/pt/tibr/concept_UnderstandingProvidingWSDLDocuments-076201.htr

Voorbeeld WSDL 2.0

52

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/ns/wsdl"
   targetNamespace=http://greath.example.com/2004/wsdl/resSvc
   xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc" …>
   …
</description>
```

Voorbeeld WSDL 2.0

53

```
<description  …> …
 <types>
  <xs:schema …>
    <xs:element name="checkAvailability" type="tCheckAvailability"/>

      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
         <xs:element name="checkInDate" type="xs:date"/>
         <xs:element name="checkOutDate" type="xs:date"/>
         <xs:element name="roomType" type="xs:string"/>
        </xs:sequence> </xs:complexType>
    <xs:element name="checkAvailabilityResponse" type="xs:double"/>
    <xs:element name="invalidDataError" type="xs:string"/>
 </xs:schema>
 </types> …
</description>
```

Voorbeeld WSDL 2.0

54

```
<description …> … <types> … </types>
  <interface name="reservationInterface" >
    <fault name="invalidDataFault" element="ghns:invalidDataError"/>
    <operation name="opCheckAvailability"
      pattern="http://www.w3.org/ns/wsdl/in-out"
      style="http://www.w3.org/ns/wsdl/style/iri" wsdlx:safe = "true">
        <input messageLabel="In" element="ghns:checkAvailability" />
        <output messageLabel="Out"
           element="ghns:checkAvailabilityResponse" />
        <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
    </operation>
  </interface> … </description>
```

Industrieel Ingenieur Informatica, Universiteit Gent

WSDL 2.0 enables one to separate the description of a Web service's abstract functionality from the concrete details of how and where that functionality is offered. This separation facilitates different levels of reusability and distribution of work in the lifecycle of a Web service and the WSDL 2.0 document that describes it.

A WSDL 2.0 interface defines the abstract interface of a Web service as a set of abstract operations, each operation representing a simple interaction between the client and the service. Each operation specifies the types of messages that the service can send or receive as part of that operation. Each operation also specifies a message exchange pattern that indicates the sequence in which the associated messages are to be transmitted between the parties. For example, the in-out pattern indicates that if the client sends a message in to the service, the service will either send a reply message back out to the client (in the normal case) or it will send a fault message back to the client (in the case of an error).

style="http://www.w3.org/ns/wsdl/style/iri“ This line indicates that the XML schema defining the input message of this operation follows a set of rules as specified in IRI Style that ensures the message can be serialized as an IRI. When using this style, the value of the {message content model} property of the Interface Message Reference component corresponding to

the initial message of the message exchange pattern MUST be "#element".†

Use of this value indicates that XML Schema [XML Schema Structures] was used to define the schema of the {element declaration} property of the Interface Message Reference component of the Interface Operation component corresponding to the initial message of the message exchange pattern.

wsdlx:safe="true" > This line indicates that this operation will not obligate the client in any way, i.e., the client can safely invoke this operation without fear that it may be incurring an obligation (such as agreeing to buy something).

<input messageLabel="In" The input element specifies an input message. Even though we have already specified which message exchange pattern the operation will use, a message exchange pattern represents a template for a message sequence, and in theory could consist of multiple input and/or output messages. Thus we must also indicate which potential input message in the pattern this particular input message represents. This is the purpose of the messageLabel attribute. Since the in-out pattern that we've chosen to use only has one input message, it is trivial in this case: we simply fill in the message label "In" that was defined in WSDL 2.0. However, if a new pattern is defined that involve multiple input messages, then the different input messages in the pattern could then be distinguished by using different labels.

<output messageLabel="Out" . . . This is similar to defining an input message.

Voorbeeld WSDL 2.0

55

```
<description … > … <types> … </types>
<interface name = "reservationInterface" > … </interface>
<binding name="reservationSOAPBinding"
   interface="tns:reservationInterface"
   type="http://www.w3.org/ns/wsdl/soap"
   wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
   <operation ref="tns:opCheckAvailability"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
   <fault ref="tns:invalidDataFault" wsoap:code="soap:Sender"/>
</binding>
. . .
</description>
```

Industrieel Ingenieur Informatica, Universiteit Gent

Although we have specified what abstract messages can be exchanged with the GreatH Web service, we have not yet specified how those messages can be exchanged. This is the purpose of a binding. A binding specifies concrete message format and transmission protocol details for an interface, and must supply such details for every operation and fault in the interface.

In the general case, binding details for each operation and fault are specified using operation and fault elements inside a binding element.

interface="tns:reservationInterface“ This is the name of the interface whose message format and transmission protocols we are specifying

type="http://www.w3.org/ns/wsdl/soap" This specifies what kind of concrete message format to use, in this case SOAP 1.2

wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/“ This attribute is specific to WSDL 2.0's SOAP binding extension (thus it uses the wsoap: prefix). It specifies the underlying transmission protocol that should be used, in this case HTTP.

<operation ref="tns:opCheckAvailability“ This is not defining a new operation; rather, it is referencing the previously defined opCheckAvailability operation in order to specify binding details for it.

wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"> This attribute is also specific to WSDL 2.0's SOAP binding extension. It specifies the SOAP message exchange pattern (MEP) that will be used to implement the abstract WSDL 2.0 message exchange pattern (in-out) that was specified when the opCheckAvailability operation was defined.

One-way.

Request-response

Duplex

When HTTP is used as the underlying transport protocol (as in this example) the wsoap:mep attribute also controls whether GET or POST will be used as the underlying HTTP method. In this case, the use of wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response" causes GET to be used by default.

<fault ref="tns:invalidDataFault“ As with a binding operation, this is not declaring a new fault; rather, it is referencing a fault (invalidDataFault) that was previously defined in the opCheckAvailability interface, in order to specify binding details for it.

wsoap:code="soap:Sender"/> This attribute is also specific to WSDL 2.0's SOAP binding extension. This specifies the SOAP 1.2 fault code that will cause this fault message to be sent. If desired, a list of subcodes can also be specified using the optional wsoap:subcodes attribute.

Voorbeeld WSDL 2.0


```
<?xml version="1.0" encoding="utf-8" ?>
<description …> … <types> … </types>
 <interface name = "reservationInterface" > … </interface>
 <binding name="reservationSOAPBinding"
   interface="tns:reservationInterface" …> ... </binding>
 <service name="reservationService"
   interface="tns:reservationInterface">
   <endpoint name="reservationEndpoint"
      binding="tns:reservationSOAPBinding"
      address ="http://greath.example.com/2004/reservation"/>
 </service>
</description>
```

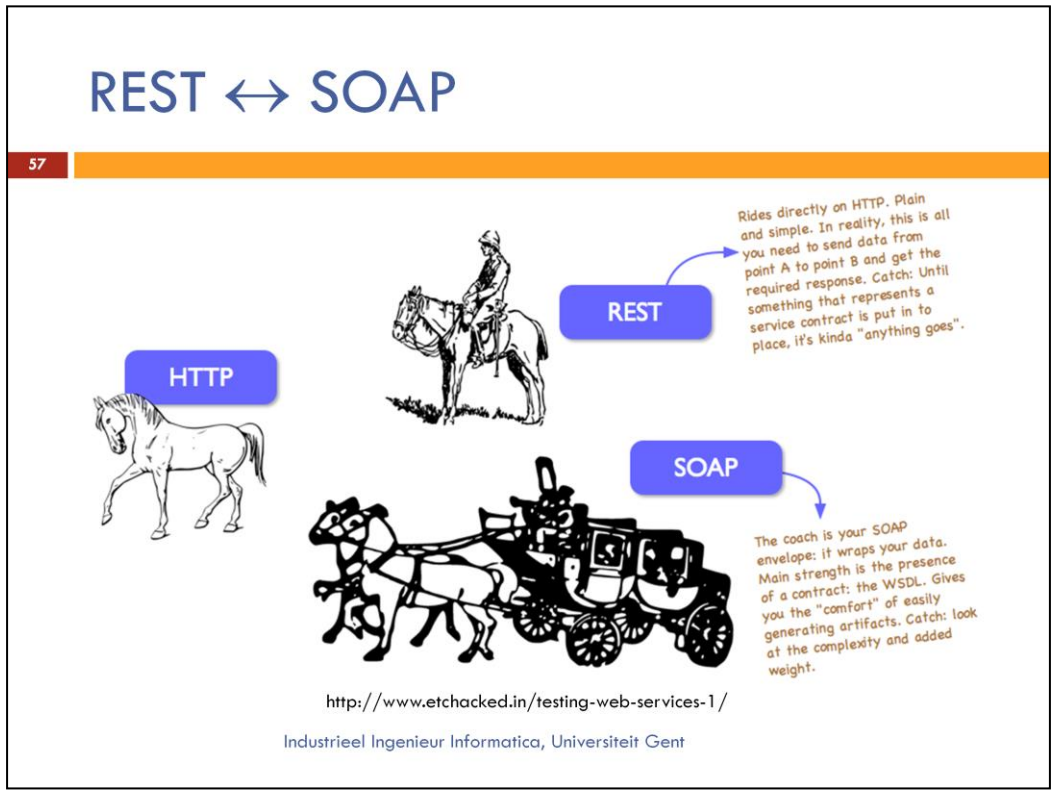

<service name="reservationService“ This defines a name for this service, which must be unique among service names in the WSDL 2.0 target namespace. The name attribute is required.

interface="tns:reservationInterface"> This specifies the name of the previously defined interface that these service endpoints will support.

<endpoint name="reservationEndpoint“ This defines an endpoint for the service, and a name for this endpoint, which must be unique within this service.

binding="tns:reservationSOAPBinding“ This specifies the name of the previously defined binding to be used by this endpoint.

address ="http://greath.example.com/2004/reservation"/> This specifies the physical address at which this service can be accessed using the binding specified by the binding attribute.



REST

- Volledig statusloos
- Caching verhoogt performantie
- Service en gebruiker service kennen de context (structuur berichten, …)
  - Geen formele beschrijving zoals WSDL
- Beperkte bandbreedte (bv. Mobiele toestellen)
- Invoegen in bestaande website met bv. Ajax

SOAP

- Formeel contract tussen gebruiker en service (WSDL)
- Complexere functionaliteit
  - Meer dan CRUD
  - Beveiliging, transacties, …
- Asynchrone afhandeling