OIOWSDL

Guideline for development and use of OIOWSDL

IT- og Telestyrelsen
Ministeriet for Videnskab
Teknologi og Udvikling

>

OIOWSDL
Guideline for development and use of OIOWSDL

Published by:
National IT & Telecom Agency in co-operation with Devoteam and Trifork

National IT & Telecom
Agency
Holsteinsgade 63
2100 København Ø

Phone: +45 3545 0000
Fax: +45 3545 0010

>

OIOWSDL

Guideline for development and use of OIOWSDL

National IT & Telecom Agency
May 2010

Content >

Introduction >

This document contains a description of OIOWSDL, including guidelines on how to build and use OIOWSDL. This structure is reflected in the way the document is built. OIOWSDL is an amalgamation of OIO (Open Information Online) and WSDL (Web Service Description Language).

The purpose of OIOWSDL is to promote Web Service interoperability in Digital Denmark, and streamline OIOWSDL document production. This guideline will assist in reaching that goal, by clearly presenting both the use of OIOWSDL documents for service consumers, and the production of these to service providers.

This guideline focuses exclusively on the development and use of WSDL documents in an OIO context. It is not an introduction to WSDL or other XML related technologies/standards. The guideline is based on the rules for WS-I Basic Profile Version 1.1, with a few adjustments. WS-I Basic Profile 1.1 is based on WSDL 1.1.

Target audience

The target audience for this guideline are companies that would like to:

1) Provide Web Services for public organizations and cooperation partners.
2) Utilize already developed and available Web Services.

In this connection, the interested parties are public organizations, as well as private companies, and their suppliers of IT solutions.

The OIOWSDL documents are available through the *service catalogue*. The catalogue includes facilities that enable the service provider to publish OIOWSDL documents, and the service consumer to find the OIOWSDL documents.

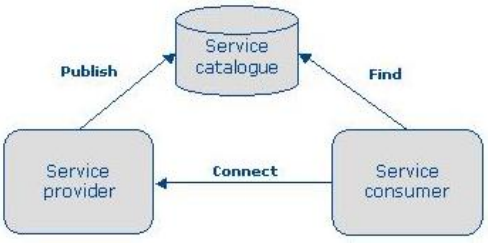


Figure 1 – Roles related to the use of WSDL

Goals

The goals for this guideline are:

- To promote the use of interoperable web services
- To standardize the use of WDSL in an OIO context
- To ease production and use of WSDL in an OIO context
- To ease the use of OIOXML in working with WSDL

>

As a prerequisite, it is expected that before the development of an OIOWSDL document, the service and its operations are identified, and relevant data structures for the interfaces are found.

What is OIOWSDL? >

OIOWSDL documents are WSDL documents that follow the guidelines described in this document. The guidelines for the OIOWSDL documents exist, as described in the introduction, to assist Web Service interoperability in Digital Denmark. Thus, the guidelines will help develop WSDL documents with uniform structures, naming policies, etc.

Architectural elements

The OIOWSDL documents are part of OIO Web Service Architecture (OWSA), outlined below. Figure 2 illustrates architectural elements that are part of OWSA only[1].

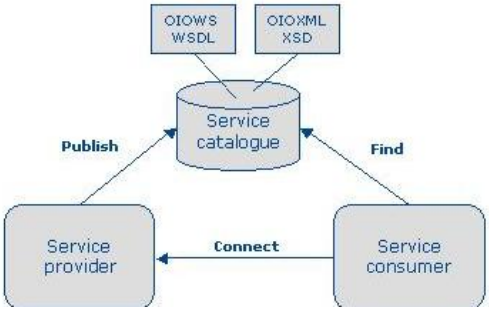


Figure 2 – Basic elements of OWSA

In this document, the focus lies on the three basic elements of OWSA:

- **A service provider**, who independently or through a supplier has developed one or more services, published in the Infostructure database (ISB)
- **A service consumer** who finds a relevant service in the service catalogue, and intends to make use of it
- **A service catalogue**, which contains XML data definitions (XSDs) and web service definitions (WSDL documents)

The OIOXML exchanged between service provider and consumer is defined in the OIOWSDL document, through one or more data definitions (XSD).

Contract-based services

A web service is described in a service contract, exchanged between the service provider and relevant service consumers. The service contract consists of two parts:

[1] A more detailed description of OWSA and the security related aspects are described in the ITST OWSA model transportbased security, ver. 1.0 published on Dec 5th, 2005.

- **The technical, a WSDL,** describing e.g. where a service is located, what operations it includes, as well as its input and output
- **The legal, a Service Level Agreement (SLA)**, used, e.g., to describe service accessibility, use agreement and service quality

Figure 3 below illustrates the two parts of the contract between service provider and consumers.

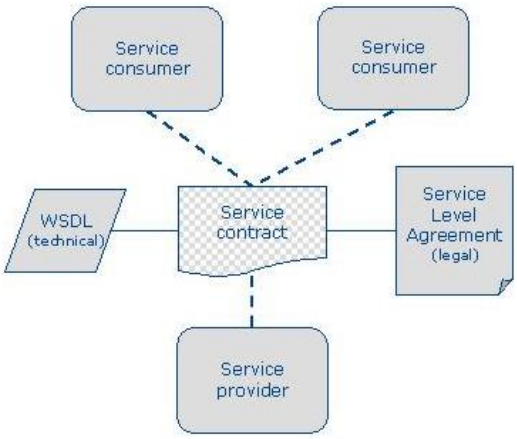


Figure 3 – Service contract for a Web Service

Utilization of a service happens based on the contract (both WSDL and SLA). The WSDL document is a precise description of service interfaces, containing a definition of both the information for which the service is responsible and manipulates, as well as the service behaviour. The WSDL document also contains a specification of how the service will physically be located and used, as well as any other infrastructure constraints.

The SLA describes the level of service quality the service provider must meet, and thus the quality level a service consumer can expect. Thus, all participants are familiar with their precise obligations, when a service is supplied or used. This principle can assist in improving the quality of IT solutions, as the risk of errors or misunderstandings is reduced. In practice, this poses relatively large demands for service contract description, maintenance and versioning[2].

This document only relates to the part of the service contract containing the WSDL document, i.e. the technical part of the contract.

[2] See OIO Standard agreement on web services: http://www.itst.dk/arkitektur-og-standarder/arkitektur/serviceorienteret-arkitektur/webservices/standardkontrakt-for-webservices

>

The Infostructure base

The service catalogue is an important element in the architecture outlined above. In order to use or re-use the services which are developed, locating and publication of services must be facilitated.

A well-functioning infrastructure for the containment of OIOXML schemas – the Infostructure base – already exists. Due to the close relationship to OIOXML schemas, the Infostructure base is an obvious choice for storage and display of OIOWSDL documents.

Guidelines for OIOWSDL

A web service consists of one or more *interfaces* and each may define a range of operations. The WSDL document describes these interfaces.
The service is accessed through *messages*, exchanged between service provider and consumer through a specific address (URI). The *binding* between the address and the interface defines which protocol is used. For OIOWSDL, HTTP and HTTPS bindings are used exclusively.

Graphical illustration of a WSDL document.

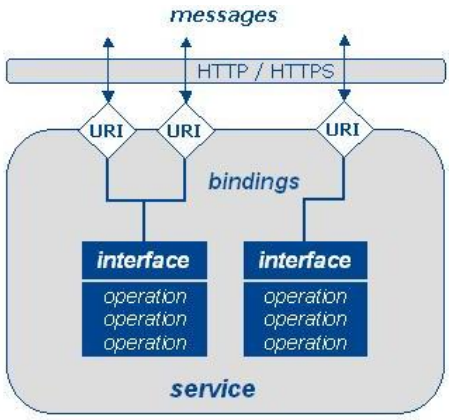


Figure 4 - Graphical illustration of a WSDL document

```
<!-- definition of WSDL structure -->
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/">

   <!-- abstract definitions -->
   <types> ...
   <message> ...
   <portType> ...

   <!-- specific definitions -->
   <binding> ...
   <service> ...

</definition>
```

Figure 5 – Elements of a WSDL document

>

Figure 5 outlines the elements of a WSDL document. The five elements are described in greater detail in Table 1.

Element name	***Description***
Types	A container for abstract data definitions, described through an XML schema
Message	A definition of an abstract message, which may consist of several different data definitions
portType	An abstract collection of operations, supported by one or more interfaces. Operations are defined by describing request and answer messages
Binding	A particular specification of protocol and data format for a particular portType
Service	A collection of related interfaces, where one interface is defined through a combination of binding and address (URI)

Table 1 – Elements in a WSDL document

The relation between OIOXML and OIOWSDL is built in the `<type>`-element. Here you `<include>` OIOXML schemas from the ISB - unlike the types from the service - as they may not be re-usable in other contexts.

OIOWSDL documents must generally follow the rules for WS-I Basic Profile Version 1.1[3]. However, to achieve the desired interoperability in Digital Denmark, the rules have been adapted to the OIO context – described in this chapter.

[DEF-1] Document literal binding

In order to create a message that can be validated through schema-validation only, R2705 from the WS-I Basic Profile is modified from *"MUST be a rpc-literal binding or a document-binding*" to "*MUST be a document-literal binding*".

[DEF-2] Data definition demands

Data definitions are XML schemas that define the data carrying elements, and their mutual relations. Data definitions must be based on the OIOXML standard. At present (early 2007), there are approx. 24.200 schemas in the ISB, which can be used for developing OIOWSDL documents.

[3] http://www.ws-i.org/Profiles/BasicProfile-1.1.html

>

While OIOXML schemas already exist, they do not cover all areas and segments in the public sector. Until that is the case, no requirement can be made that all data definitions must be OIOXML approved. Until then, non-OIOXML schemas may be used. These schemas must meet the "OIOXML Navngivnings- og Designregler 3.0"[4] (OIOXML Naming and Design Rules) (NDR), but need not be approved.

[DEF-3] Demands for message definitions
Message definitions describe the messages sent between service provider and consumer. A message definition is an XML schema, which collects the data definitions that are built on a specific message.

It is expected that messages that are part of an OIOWSDL document will not be re-used in other OIOWSDL documents. Therefore, there are no requirements that message definitions that are part of an OIOWSDL document must be either OIO approved, or cleared at a later date.

[NAV-1] Demands for naming of flies, namespaces and elements
Overall, NDR should be used as extensively as possible; e.g. names in singular form, abbreviations and acronyms, conjunctions, use of signs in names, use of Danish characters (æ, ø, å). Regarding choice of language, NDR will be observed, i.e. the chosen language (Danish or English) must be systematically applied; English is preferred. Generally, it is not recommended to use underscore (_), period (.) or hyphens (-) in building names; UpperCamelCase is recommended. This means that the name must begin with an upper-case letter. If an abbreviation consists of upper-case letters exclusively (e.g. DK), the following word should begin with a lower-case letter. The consequence of this is that the name must start in upper case and subsequently every new word must start in upper case.

[NAV-1a] Naming of OIOWSDL documents
As with NDR, it is recommended to name OIOWSDL documents as follows:

```
<namespace-prefix with upper-case letters>+"_"+<service
name>+".wsdl"
```

An OIOWSDL document containing a description of a service named Medicine Chest; prefixed DKMA will thus be named

```
DKMA_MedicineChest.wsdl
```

[4] http://www.oio.dk/files/OIO-6.OIOXML-NDR.v3.20041215.dk.pdf

>

[NAV-1b] Naming an OIOWSDL namespace
Naming of namespaces for the OIOWSDL document should follow the general OIO guidelines described in the NDR. However, there is an exception with regard to the NDR recommendations, namely the use of periods to separate substructures and dates.

```
”http://rep.oio.dk/” + <domain> + “/xml.wsdl/” + <year> + ”.” +
<month> + ”.” + <day> + ”/”
```

A namespace for OIOWSDL under the dkma.dk domain may for instance be given the namespace

```
http://rep.oio.dk/dkma.dk/homecare/xml.wsdl/2006.05.11
```

```
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.wsdl/2006.05.11"
    xmlns:dkma="http://rep.oio.dk/dkma.dk/homecare/xml.wsdl/2006.05.11"
xmlns:xsd="http://www.w3.org/2001/XMLschema"
    xmlns:tns="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
```

Figure 6 – Example of namespace naming

[NAV-1c] Naming of <message>
In OIOWSDL, two types of operations are used:

- *Request-response* – the operation can receive a request and return an answer.
- *One-way* – the operation can receive a message, but cannot return an answer.

The request-response type is the most widely used.

A message should be named after the operation in question. If the operation is a request or response operation, the name must be followed by Request or Response. If the operation is a one-way operation, the message is named using the name of the operations exclusively.

```
<name of operation> + “Request” | ”Response” | ””
```

Two messages, that request and return a cache of medicines information, respectively, may look like this:

Request: `MedicineChestStructureGetRequest`
Return: `MedicineChestStructureGetResponse`

Note that we have chosen to model the *Fault* message directly after the operation name, which means that any type of error will return the same Fault. For several reasons, this is not a general requirement; it gives a direct link

>

between error type and operation, and in many situations it would conflict with the idea of mapping Exceptions from the implementation directly to Faults.

```
<message name="MedicineChestStructureGetRequest"> ... </message>
<message name="MedicineChestStructureGetResponse"> ... </message>
<message name="MedicineChestStructureFault"> ... </message>
```

Figure 7 – Example of message element naming

No need has been identified for naming of message parts.

[NAV-1d] Naming of <operation>
An operation should be named after the *Object Handling* naming model.

There is no fixed definition for naming operations.

An operation that returns a medicine profile on request may be named e.g.: `MedicineChestStructureGet`

An operation that changes medicine profile content may be named, e.g.: `MedicineChestStructureUpdate`.

```
<operation name="MedicineChestStructureGet">
  <input name="MedicineChestStructureGetRequest"
         message="tns:MedicineChestStructureGetRequest"/>
  <output name="MedicineChestStructureGetResponse"
          message="tns:MedicineChestStructureGetResponse"/>
  <fault name="MedicineChestStructureFault"
          message="tns:MedicineChestStructureFault"/>
</operation>
```

Figure 8 – Example of operation element naming

[NAV-1e] Naming of <portType> *(interface)*
It is recommended that naming of the portType element corresponds to the service name. For WSDL documents with multiple interfaces, however, a unique naming of all interfaces is required.

`<service name> + <unique identification>`

A portType for the service MedicineChest, containing just one portType is thus simply called: `MedicineChest`.

```
<portType name="MedicineChest"> ... </portType>
```

Figure 9 – Example of portType element naming

[NAV-1f] Naming of <binding>
It is recommended that naming of a binding corresponds to the service name. If there is more than one binding element in the WSDL document, however, a unique naming of each binding element is required:

>

<service name> + <unique identification>

[NAV-1g] Naming of SOAP actions
SOAP actions are named using the following model:

<namespace> + ”#” + <operation name>

Where namespace refers to the target namespace specified in the WSDL document.

An example of a SOAP action might be:
http://rep.oio.dk/dkma.dk/xml.wsdl/2007.04.18/#MedicineChestGet

```
<binding name="MedicineChestBinding" type="tns:MedicineChestType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="getMedicineChestStructure">
    <soap:operation soapAtion=
    "http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11/#MedicineChestGet"
    style="document" />
    ...
  </operation>
</binding>
```

Figure 10 – Example of binding element naming

[NAV-1h] Naming of <service>
It is recommended that you use a covering term for naming the service element, which includes the service operations. Do not use operation types in the name (e.g. medicineGet), and also avoid using *service* in naming.

Example of recommended: <service name="MedicineChest">
Example of *non*-recommended: <service name="MedicineChestGetService">

```
<service name="MedicineChest"> ... </service>
```

Figure 11 – Example of service element naming

[DOK-1] Documentation
It is recommended that you add a <documentation> element to <service>, <operation> and <message> elements in the WSDL document, which overall give a full description of the functionality offered by the service.

How is OIOWSDL produced? >

Production of OIOWSDL files follows the general OIO approach, and should follow the Contract First method (KF method), which is described below.

The philosophy behind OIO includes promotion of reusability and sharing of data definitions, as well as the improvement of data quality. It should be possible to reuse already used elements, in order to reduce complexity and the number of elements. In order to utilize the reusability principle, a range of rules for naming and design of data definition elements (XSDs) [6] is created, to ease the integration of data definition elements when needed.

When you wish to display functionality as a web service, it is therefore the responsibility of the developer to combine message definitions based on available data definitions, displayed in the common public service registry. If further data definitions are needed, these will have to be developed[7].

The described approach is based on data and message definitions, before the service is fully defined with interfaces and operations. Thus, it is possible to reuse already developed OIOXML schemas. Subsequently, the code for the specific implementation can be generated based on the WSDL document.

Approach for developing OIOWSDL

The KF method can be described using the four steps below:

1. Find/model data definitions: This step defines the operational data to be used in the web service. Operational data is described in OIOXML schemas. Relevant OIOXML schemas are retrieved in the Infostructure database (the ISB).
2. Model message definitions: The messages to be exchanged between service provider and consumer is modelled. The messages are described in XML schemas and apply the data definitions from step 1. In connection with modelling of messages, include descriptions of how error messages are returned through the web service. Errors are returned using soap `fault`.
3. Model the interface: This step defines what operations will be offered by the web service. The operations are grouped in interfaces. If relevant, the step is concluded by testing the produced OIOWSDL document against WS-I Basic Profile Version 1.1 using WS-I tools8. Note that these tools cannot evaluate whether all profile guidelines have been met.

[6] These rules are found in the ITST Naming and Design Rules (NDR).
[7] Development of data definitions is outside the scope of this guideline. See ITST NDR and sector standardization activities.
[8] See http://www.ws-i.org/deliverables/workinggroup.aspx?wg=testingtools

>

4. Generate coding template: Based on the OIOWSDL document, coding templates are generated for both consumer and service provider.

The four steps are illustrated in Figure 12, where a number of data definitions are combined into message definitions, used in the interface description. Based on the interface description, the code for the web service implementation environment is generated.

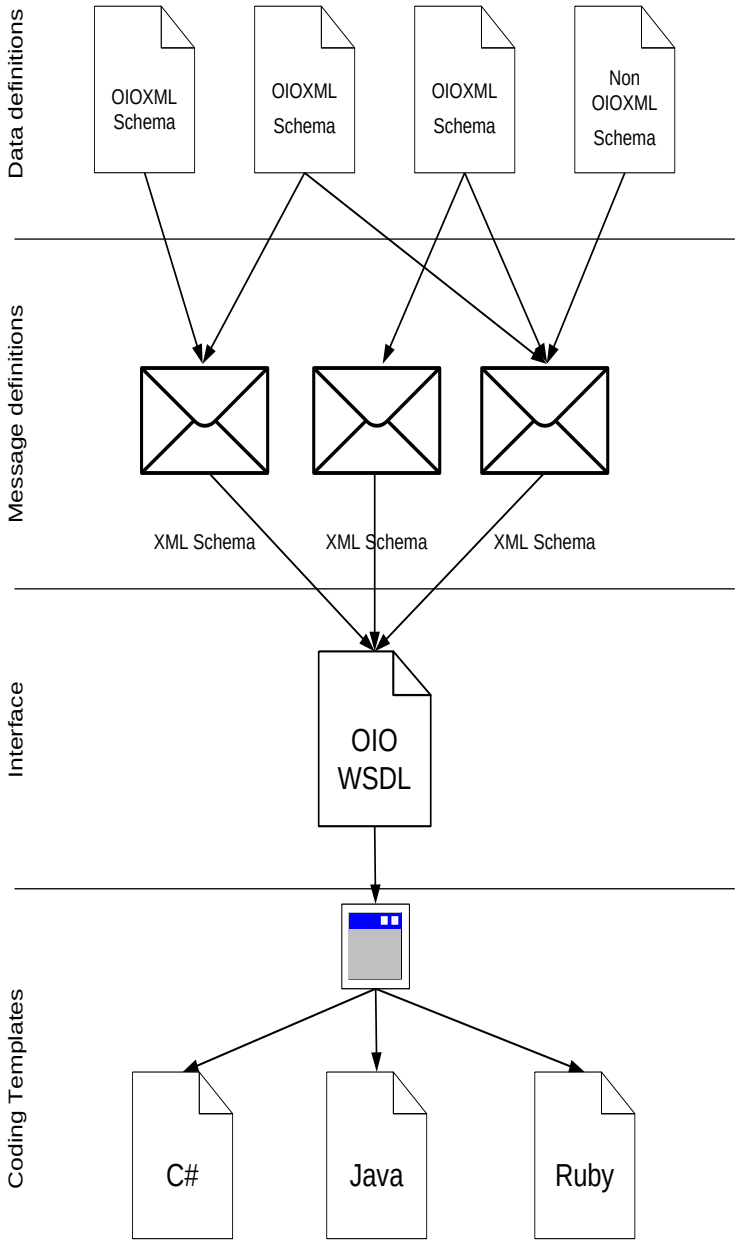


Figure 12 – schematic presentation of KF approach

>

Example

Let us review a realistic example. In connection with the project ”Hjemmesygeplejens adgang til Medicinprofilen” (Home Care access to the Medicines profile under Danish Medicines Agency (see http://medicinprofilen.dk/default.asp?id=806) a web service was developed permitting registration of over-the-counter medicine, observed in the medicine cabinet, in connection with Home Care visits. This is intended as a pre-emptive measure for preventing interaction with prescription medicine. The web service makes it possible to register observed over-the-counter medicine on the client’s CPR number, as well as retrieve previous registrations. Only the operation for retrieving registrations is shown below. The example is included in the appendix at the end of this document, and can also be downloaded from the page where this document is available. Note that the schema in the example and the WSDL documents are modified compared to the Medicine profile. Deviations from the present guidelines are due to the Medicine profile being completed prior to the formulation of the present guideline.

Step 1: Modelling of data definitions

XML schemas for the web service consist of a mix of new developed NDR compatible schemas, plus a lone schema from the core-class OIO (CPR no.).

The used schemas can generally be split into:

- Basic data carrying elements, which through either one or several types – possibly with restrictions – represent a single type, e.g. `DKMA_DosageFormText.xsd` (e.g. appendix: XSD schemas).
- Composite elements, combining the basic elements for combinations and sequences of element types, e.g. `DKMA_DrugStructure.xsd` (see appendix: XSD schemas for an example).

Generally, you could say that composite element types are more likely to be specific to a particular web service (e.g. as an outcome consisting of a unique composition of basic types), while the basic elements will be more stable across web services. It might therefore be beneficial to anchor the composite elements in the WSDL document, rather than creating independent XSD schemas on all levels.

Note that the schema definitions are not part of the WSDL document shown here (see the section below), but is referred to through an `include- import-` command. The schema definitions can be found in Appendix 9.

Step 2: Modelling message definitions

In this step, a request message, a response message and a fault detail element is defined. The request message is based on the element type `MedicineChestStructureGet` (see Appendix: XSD schemas for an example) and the response message is defined through the element type

>

`MedicineChestStructure` (see Appendix: XSD schemas for an example). The Fault message is defined through the element type `MedicineChestFault`, if the definition is part of the WSDL document.

The message definitions can be found in the WSDL document as the three message elements, using *name* = `"MedicineChestStructureGetRequest"`, `"MedicineChestStructureGetResponse"`, and `"MedicineChestStructureFault"`, respectively.

Step 3: Modelling of interface

In this step, an interface containing an operation is defined. The operation is defined through the above request and response messages, as well as the fault detail element, and looks as follows:

```
<operation name="MedicineChestStructureGet">
  <input name="MedicineChestStructureGetRequest"
     message="tns:MedicineChestStructureGetRequest" />
  <output name="MedicineChestStructureGetResponse"
     message="tns:MedicineChestStructureGetResponse" />
  <fault name="Fault" message="tns:MedicineChestStructureFault" />
</operation>
```

The resulting OIOWSDL – document appears as illustrated below:

>

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
  xmlns:xsd="http://www.w3.org/2001/XMLschema"
  xmlns:tns="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <types>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://
rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11">
      <xsd:include schemaLocation="../xsd/DKMA_MedicineChestStructureGet.xsd"/>
      <xsd:include schemaLocation="../xsd/DKMA_MedicineChestStructure.xsd"/>
      <xsd:element name="MedicineChestStructureFault" type="xsd:string"/>
    </xsd:schema>
  </types>

  <message name="MedicineChestStructureGetRequest">
    <part name="cpr" element="tns:MedicineChestStructureGet"/>
  </message>
  <message name="MedicineChestStructureGetResponse">
    <part name="arg2" element="tns:MedicineChestStructure"/>
  </message>
  <message name="MedicineChestStructureFault">
    <part name="fault" element="tns:MedicineChestStructureFault"/>
  </message>

  <portType name="MedicineChest">
    <operation name="MedicineChestStructureGet">
      <input name="MedicineChestStructureGetRequest"
        message="tns:MedicineChestStructureGetRequest"/>
      <output name="MedicineChestStructureGetResponse"
        message="tns:MedicineChestStructureGetResponse"/>
      <fault name="MedicineChestStructureFault"
        message="tns:MedicineChestStructureFault"/>
    </operation>
  </portType>

(continued on the following page)
```

>

```
  <binding name="MedicineChest" type="tns:MedicineChest">
    <soap:binding transport=http://schemas.xmlsoap.org/soap/http
      style="document"/>
    <operation name="MedicineChestStructureGet">
      <soap:operation
        soapAction=
          "http://
rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11/#MedicineChestGet"
        style="document"/>
      <input name="MedicineChestStructureGetRequest" >
        <soap:body use="literal"/>
      </input>
      <output name="MedicineChestStructureGetResponse">
        <soap:body use="literal"/>
      </output>
      <fault name="MedicineChestStructureFault">
        <soap:fault name="MedicineChestStructureFault" use="literal"/>
      </fault>
    </operation>
  </binding>

  <service name="MedicineChest">
    <documentation>WSDL file for MedicineChest</documentation>
    <port name="MedicineChest" binding="tns:MedicineChest">
      <soap:address location=
        "http://localhost:8080/wstestservice/services/TestPort"/>
    </port>
  </service>
```

Step 4: Generate coding template

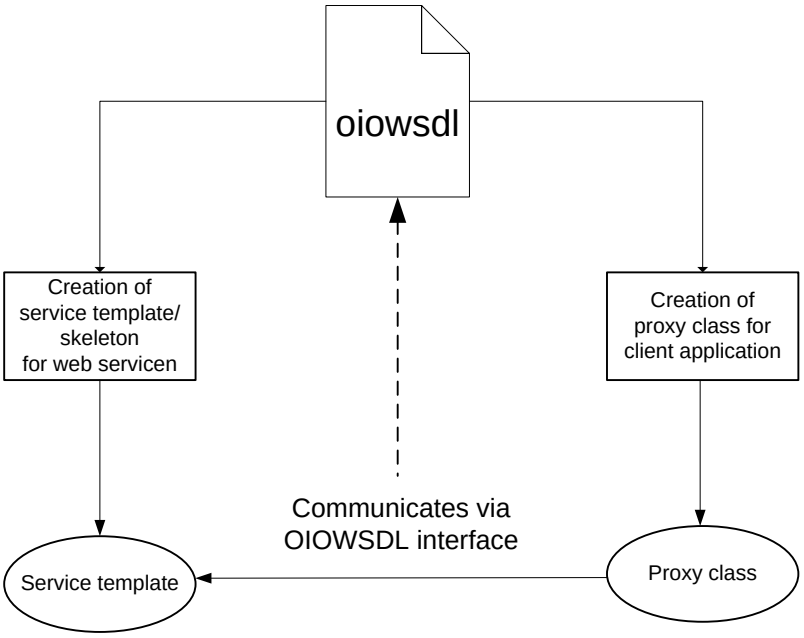
Service provider
The service provider implements the web service specified by the WSDL document. The general pattern for server side implementation of a WSDL based web service is that the selected tool, using the WSDL document as input, generates a skeleton and data binding classes/code in the used programming language. Subsequently, it is the responsibility of the developer to write server side code that can be activated by the generated skeletons. The data binding classes reflect the XSD types that the WSDL document is based on, and are used to represent the transferred messages.

Service consumer
A service consumer will generally, through the selected client side tool, generate stubs and data binding classes/code. The developer can also apply these stubs for calling the web service.

The following chapter *How is OIOWSDL used*? contains references to a number of the most widely used development tools.

How is OIOWSDL used? (Tool support) >

This section describes how you, based on an OIOWSDL document, can generate code for client and server, i.e. both implementing a web service (service skeleton) in compliance with the OIOWSDL description, and also generate code to call the web service (client stubs).



Recently (early 2007), the most developed development tools have been surveyed, to see how they handle OIOWSDL documents based on the contract-first approach. The surveyed tools are:

- BEA WebLogic 9.2
- Axis 1.3 and 2
- Eclipse 3.2
- IBM Rational Application Developer 6.0, 6.0.1 and 7.0
- .Net 1.1, 2.0 and 3.0
- Ruby

No extensive descriptions have been made for creating web services using the individual tools. Instead, the basis was to follow the general approach of the tool for producing web services. Additionally, for each tool, an event review has been made for working with OIOWSDL documents.

Thus, there will be a large number of links to the reference documentation for each tool. In some cases, the approach will directly follow the tool documentation.

For each reviewed tool, there is a general introduction, and a section for client stubs and service skeletons, respectively. As an example, the above example

about the medicine cabinet (MedicineChest) will continue to be used. It is assumed that the WSDL and XSD examples from the medicine cabinet were copied to files on the local disk. In practice, XSD files for exchange with WSDL documents must be retrieved from the common service catalogue. However, during development, it will often be advantageous to have copies available locally – in particular if you combine XSDs from the ISB with own/newly developed XSDs.

General problems with imports

Before we commence, it is important to identify a common implementation detail in the tools. Certain XML parsers interpret the XSD specification very hard, and therefore return errors, when the same namespace is imported several times, in different schemaLocations.

The specification reads as follows: "*Note: The above is carefully worded so that multiple <import>s of the same schema document will not constitute a violation of clause 2 of schema Properties Correct (§3.15.6), but applications are allowed, indeed encouraged, to avoid <import>ing the same schema document more than once to forestall the necessity of establishing identity component by component. Given that the schemaLocation [attribute] is only a hint, it is open to applications to ignore all but the first <import> for a given namespace, regardless of the actual value of schemaLocation, but such a strategy risks missing useful information when new schemaLocations are offered.*"

In other words, this means that parsers interpreting the specification along the lines of the above will only retrieve the first identified XSD file, and then assume that it contains the complete definition of the given namespace. However, this parsing is problematic when compared to the OIO NDR standard, which indicates that a namespace must be split, putting each type in its separate file (section 3.5 in NDR). In cases where several types are imported from the same namespace, some parsers can return errors to the effect that all types other than the first one are undefined.

So far, the problem has been seen with Xerces-2[9] / JAXP 1.3 and .Net 2.0 (.Net 1.1 also has a general problem with includes, but this is described separately.) Xerces-2/JAXP 1.3 is standard under Java5 and used in e.g. WebLogic 9 and RAD 7.0.

[9] In Xerces, you have the following option for circumventing the problem with duplicated namespaces across several schema locations: http://xerces.apache.org/xerces2-j/features.html#honour-all-schemaLocations

>

Solutions

Two solutions are proposed:

1. Inlining of types. By pulling all definitions into the WSDL file, includes and imports are completely avoided, and the problem disappears.
2. Generate a new XSD file, collecting the necessary types to be imported, rather than the individual types. In our example, it might look like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLschema"
xmlns:common="http://
rep.oio.dk/dkma.dk/common/xml/xsd/2006/05/11"
targetNamespace="http://rep.oio.dk/dkma.dk/common/xml/xsd/2006/
05/11" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <include schemaLocation="DKMA_DrugIdentifier.xsd" />
  <include schemaLocation="DKMA_DrugName.xsd" />
  <include schemaLocation="DKMA_DosageFormText.xsd" />
  <include schemaLocation="DKMA_DrugStructure.xsd" />
</schema>
```

We recommend solution no. 2, as it maintains the split, but does not require changing existing types. In this regard, it is recommended that you create an XSD file that includes all types in a namespace, and which must be made accessible along with the other types.

Note that the reason for this solution is exclusively to be able to use OIOWSDL in code generation in certain tools. Thus, it is the original/general WSDL document that must be published, not the two identified solutions.

BEA WebLogic 9.2

Introduction

Generally, client stubs and service skeletons are generated through *Ant*. In Workshop, there is a possibility for generating the same features, but if you use those options, utilize the Eclipse WTP functionality, which applies Axis.

The import restrictions mentioned WebLogic 9.2 at the beginning of this chapter. The easiest way around this is to create a new XSD file, `DKMA_Common.xsd`, containing the XML shown above.

- Correct `DKMA_MedicineChestItemChoiceStructure.xsd` and add `<import namespace="http://rep.oio.dk/dkma.dk/common/xml/xsd/2006/05/11" schemaLocation="common.xsd" />` in place of the existing import
- Repeat with `DKMA_MedicineChestItemStructure.xsd`

>

Another approach is to configure Xerces-2 to read all schemaLocations by adding a feature to the parser. If `http://apache.org/xml/features/honour-all-schemaLocations` is added to `Boolean.TRUE`, the parser will use *all* import statements. Generally, the problem with this is that the parser is not accessible, when new stubs are generated, as it is protected. It is also possible to implement a new schemaFactory, which may be indicated using a system property, but this, too, is a rather inflexible solution.

Below, we focus on generating client stubs and service skeletons using *Ant*, plus subsequent import in Workshop.

- Start by creating a new server under ”`File->New->Server`”. Use default settings on the first page, and click ”`next`”. Use the Configuration Wizard for creating a new domain with support for Workshop. Click ”`next`”, select “`Workshop for WebLogic platform`”, “`next`”, enter new password, “`next`”, “`next`”, “`next`”, “`create`”, “`done`”. Now, the new domain can be selected under “`Domain home`”. The domain is placed in `$BEA_HOME/user_projects/domains/base_domain`. Click ”`finish`” to create the server.
- To class path, add `WEBLOGIC_HOME/server/lib/weblogic.jar` and `WEBLOGIC_HOME/server/lib/xbean.jar`, whose source is imported into Workspace.
- From runtime, there will be warnings concerning exception types, but these do not affect production, not even in cases where an exception actually occurs.

(Reference documentation for web services:
`http://edocs.bea.com/wls/docs92/webserv/` and
`http://edocs.bea.com/wls/docs92/webserv/use_cases.html#wp220705`)

Now we are ready to create client and service.

Client stubs

Create a new Java Project in Workspace using the settings described above. First create a `build.xml` containing the following:

>

```
<project>
  <taskdef name="clientgen"
    classname="weblogic.wsee.tools.anttasks.ClientGenTask" />

  <target name="build-client">
    <clientgen
      wsdl="wsdl/DKMA_MedicineChest.wsdl"
      destDir="gen"
      packageName="com.acme.test"/>
  </target>
</project>
```

Create a new dir, `WSDL`, and copy the WSDL file here. Copy any new XSD files, to place them properly in relaion to the WSDL file. Correct both `build.XML` so that the WSDL attribute indicates the right file and the packageName attribute, so it indicates the package where the client stub will be placed.

Then run

```
$BEA_HOME/user_projects/domains/base_domain/bin/setDomainEnv.sh
ant build-client
```

The generated code is now placed in the gen-directory, and can be compressed to a jar file, using

```
jar cf client.jar -C gen .
```

Subsequently, the generated stubs can be used in Workshop:

1. Refresh the project
2. Right-click `gen` and select ”`Build path->Use as source folder`”
3. Right-click the project and select ”`Properties->Java Build Path->Libraries`”
   a. “`Add Library->Server Runtime->Next->Finish`”
   b. “`Add Variable->WEBLOGIC_HOME->Extend->server/lib/weblogic.jar`”
   c. “`Add Variable->WEBLOGIC_HOME->Extend->server/lib/xbean.jar`”
4. Create a new class, e.g. ”`com.acme.test.Client`” adding the following in the ”`main`”- method:

>

```
MedicineChest_Impl service =
         new MedicineChest_Impl();
MedicineChestType port = service.getMedicineChestPort();
MedicineChestStructureType type =
         new MedicineChestStructureType();
type.setPersonCivilRegistrationIdentifier("1212121212");
MedicineChestStructureType medicineChestStructure =
         port.getMedicineChestStructure(type);
```

Service skeletons

Create a new project, "`File->New->Project->Web Services->Web Service Project`". Use default settings.

The approach is the same as in client stubs: First create a `build.xml` (or add a new target to an existing target):

```
<project>
  <taskdef name="wsdlc"
     classname="weblogic.wsee.tools.anttasks.WsdlcTask"/>
  <target name="generate">
    <wsdlc
       srcWsdl="wsdl/DKMA_MedicineChest.wsdl"
       destJwsDir="gen/jws"
       destImplDir="gen/impl"
       packageName="com.acme.wsdl"
       explode="true"
    />
  </target>
</project>
```

Create a new dir, `WSDL`, and copy the WSDL file here. Copy any new XSD files, to place them properly in relaion tothe WSDL file. Correct both `build.XML` so that the WSDL attribute indicates the right file, and the packageName attribute, so it indicates the package where the client stub will be placed.

Then run:

```
$BEA_HOME/user_projects/domains/base_domain/bin/setDomainEnv.sh
ant generate
```

The generated code is split into two parts: In `gen/jws` you will find auto-generated types, which match the ones defined in the XSD files, while the skeleton class will be in `gen/impl`. In the MedicineChest case, there will be a class named `MedicineChestImpl`, with an empty method to be implemented.

Building and deployment of a service is demonstrated at http://edocs.bea.com/wls/docs92/webserv/setenv.html#wp209796 and http://edocs.bea.com/wls/docs92/webserv/setenv.html#wp213597.

In Workspace, the generated classes may now be used:

>

1. Refresh the project
2. Add `gen/impl` and `gen/jws` to build path as source folders
5. Right-click the project and select ”`Properties->Java Build Path -> Libraries`” and ”`Add Variable->WEBLOGIC_HOME->Extend->server/lib/xbean.jar`”
6. Open the type `MedicineChestImpl` and implement operational logic for the service
7. Open `OwnerTypeTextType`. This is an `enumeration`-class, and deploy will fail if an empty constructor is not added
8. `DKMA_MedicineChest.wsdl` in `gen/jws/wsdls` contains an empty name attribute. Delete this, or there will be validation errors.
9. Add the project to the WebLogic server:
   a. Open the Servers view
   b. Right-click ”`BEA WebLogic v9.2 server`” and select ”`Add and remove projects`”
   c. Add the service project
   d. Start the server
   e. Test the service. WebLogics admin interface has a testfeature at `http://host:7001/console -> Deployments->service->MedicineChest->Testing->Expand MedicineChest->Test client`

RAD 6.0.0, RAD 6.0.1, RAD 7.0

Introduction

Reference documentation for web services:
`http://publib.boulder.ibm.com/infocenter/radhelp/v6r0m1/topic/com.ibm.etools.webservice.doc/concepts/cwstopdown.html`
and
`http://publib.boulder.ibm.com/infocenter/radhelp/v6r0m1/topic/com.ibm.etools.webservice.was.creation.ui.doc/tasks/tsampappw.html`

RAD 6.0.0 and RAD 7.0 both have the import restriction mentioned earlier in this chapter. The easiest way around this is to create a new XSD file, `DKMA_Common.xsd`, containing the XML shown earlier in this chapter.

- Correct `DKMA_MedicineChestItemChoiceStructure.xsd` and add `<import namespace="http://www.dkma.dk/common/xml/xsd/2006/05/11" schemaLocation="common.xsd" />` in place of the existing import
- Repeat for `DKMA_MedicineChestItemStructure.xsd`

RAD 6.0.1 does not have the import restriction.

As JAX-RPC only defines optional support for `<xsd:choice>`, RAD cannot generate related Java classes. This generates a `WSWS2029W warning`. (See `http://www-1.ibm.com/support/docview.wss?uid=swg21207642`) - requiring a *custom data binder*, or that `<xsd:choice>` is re-written to `<xsd:sequence>` with `<xsd:element maxOccurs="1" minOccurs="0">`. The latter, however, does not have quite the same semantic significance. Section 5.3.1 in NDR notes that *choice* can be an ambiguous choice.

>

Start by configuring a server: Right-click the Servers view and select ”New -> Server” and select ”WebSphere 6.1 Server” and ”finish”. Correct the XSD files as described above, putting all types from a namespace in the same file.

Client stubs

In order to create the client stub, perform the following stubs:

1. Select “New->Other->Web Services->Web Service Client”
2. Select the WSDL file under ”Service definition”
3. ”Finish”
4. The client can now be used as follows:

```
URL url =
   new URL("http://localhost:8080/services/MedicineChest");
MedicineChest service =
   new MedicineChestLocator().getMedicineChestPort(url);
MedicineChestStructureType arg = new MedicineChestStructureType();
arg.setPersonCivilRegistrationIdentifier("121212121212");
service.medicineChestStructureGet(arg);
```

Service skeleton

1. Create a new ”Dynamic Web Project” using default settings
2. Copy WSDL and XSD files into the new project
3. Select“New->Other->Web Services->Web Service”
4. Select “Top down Java bean Web Service” under “Web service”
5. Select the WSDL file under ”Service definition”
6. ”Finish”

RAD will automatically open the Web Services Explorer, which you can use to test services.

- Unfold ”MedicineChest” (The highlighted element) and select ”medicineChestStructureGet”
- Add a 10-key text in the text field and click ”Go”
- The Web Services Explorer can be opened by finding the local WSDL file (WebContent/WEB-INF/wsdl/DKMA_MedicineChest.wsdl), right clicking it, and selecting ”Web Services->Test with Web Services Explorer”.

Open the ”MedicineChestImpl” type, and implement the operational code.

Deploy to the server (if necessary) by right-clicking the server in the Servers view and selecting ”Publish”.

>

Axis 1.3

Introduction

Generation of both client stubs and service skeletons follow the reference documentation for WSDL driven development:
http://ws.apache.org/axis/java/user-guide.html#WSDL2JavaBuildingStubsSkeletonsAndDataTypesFromWSDL

Note that the arrays contain the generated xml `<Item>` tags, rather than, e.g. `<MedicineChestItemStructure>`.
For example, if you declare the following type:

```
<complexType name="PharmacyCollectionType">
    <sequence>
        <element ref="PharmacyStructure" minOccurs="0"
              maxOccurs="unbounded"/>
    </sequence>
</complexType>

<element name="PharmacyStructure"
    type="common:PharmacyStructureType"/>
<complexType name="PharmacyStructureType">
  <sequence>
    ...
  </sequence>
</complexType>
```

Axis will serialise the array elements using the element name *item* in place of *PharmacyStructure*, as prescribed in the XSD. This only happens with arrays, i.e. if "`maxOccurs > 1`". This is not a problem if both service and client are developed in Axis, though it would if, e.g., the client is .NET.
In order to obtain the requisite interoperability, you may use the *inline* definition shown below:

```
<complexType name="PharmacyCollectionType">
    <sequence>
        <element name="PharmacyStructure"
           type="common:PharmacyStructureType"
           minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

This, however, means that OIO standards are not met!
The problem was fixed in Axis2 1.1.1.

Client stubs

Use the WSDL-to-Java tool in the class `org.apache.axis.wsdl.WSDL2Java`:

>

```
java org.apache.axis.wsdl.WSDL2Java DKMA_MedicineChest.wsdl
```

(Note that there are also Ant targets, and a number of shell scripts performing the command).
This creates all the client stubs, which are placed in a directory reflecting the target namespace from the WSDL file; i.e., in this case, they are put under `dk/dkma/homecare`.

Service skeleton

As is the case when generating client stubs, `WSDL2Java` is used to generate service skeletons; although with a number of server side options:

```
% java org.apache.axis.wsdl.WSDL2Java --server-side
    --skeletonDeploy true DKMA_MedicineChest.wsdl
```

While this in fact generates the client stubs, it also generates the skeleton classes for the server side implementation.

Axis2

Introduction

Both client stubs and service skeletons are generated as described in the reference documentation:
Client stubs: http://ws.apache.org/axis2/1_1/userguide-creatingclients.html#createclients
Service skeletons: http://ws.apache.org/axis2/1_1/userguide-buildingservices.html#deployrun

Axis2 suffers from the same problems with `<item>` as Axis1. The problem was fixed in Axis2.1.1.1.

Client

The stubs are simply generated using the `wsdl2java` command:

```
%AXIS2_HOME%\bin\wsdl2java -uri DKMA_MedicineChest.wsdl
  -p <package> -s
```

where

- `-p` denotes package for the generated stubs
- `-s` denotes synchro/blocking calls.

Service skeleton

The skeleton classes are generated using the `wsdl2java` command, with a few extra options:

>

```
%AXIS2_HOME%\bin\wsdl2java -uri DKMA_MedicineVhest.wsdl -p
<package> -o <directory> -s -ss –sd
```

where
- `-p` denotes package for the generated classes
- `-o` denotes directory for the generated packages
- `-s` denotes synchro/blocking behavior
- `-ss` denotes server side generation
- `-sd` denotes that the services.xml service descriptoren must be created.

Eclipse

Introduction

Support for web services in Eclipse from the Web Tools Project (WTP), found at http://www.eclipse.org/webtools/.

WTP uses Axis 1.3 for generating code, see section 4.4.1 concerning array problems.

Client stubs

Generation of client stubs follows this guideline:

http://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/WebServiceClient/WebServiceClient.html.

Service skeleton

On the server side, follow this guideline/tutorial for implementing a service skeleton:

http://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/TopDownWebService/TopDownWebService.html

.NET 1.1

Introduction

Client stubs and service skeletons are built using `wsdl.exe`.

Reference documentation for wsdl.exe: `http://msdn2.microsoft.com/en-us/library/7h3ystb6(vs.71).aspx`
Consider using WSCF from
http://www.thinktecture.com/Resources/Software/WSContractFirst/default.html

In .Net 1.1, XSD files generally do not work with `include`. Instead, all types must be embedded. This problem is described at length here:
http://support.microsoft.com/kb/820122

>

Another workaround is to copy `wsdl.exe` from a Mono installation under e.g. Linux, which does not suffer from the same problem.

The WSDL file with the embedded definitions is shown in appendix 11.

Note that using the discover tool, `disco.exe`, all types can be downloaded to the local hard drive. `disco.exe` uses the complete wsdl file URL as argument, possibly indicated as a UNC path (`\\<server>\<share>`) . Consequently, in our case the command could look as follows:

```
disco.exe c:\dkma-eksempel\wsdl\DKMA_MedicineChest.wsdl
```

Client stubs

In order to use the local copies of the XSDs instead of retrieving them in the ISB (which is often an advantage during development), the completed list of *imported* (not the included, which have become embedded) XSDs must be indicated on the command line.

In our example, we have embedded the range of XSDs, leaving the completed command line as follows:

```
wsdl.exe DKMA_MedicineChest.wsdl
```

This will create the file `MedicineChest.cs`, using the method:

```
public MedicineChestItemStructureType[]
   MedicineChestStructureGet(...)
```

Service skeleton

Generation of service skeletons is identical to generation of client stubs, though the "`/server`" option must be added to the command:

```
wsdl.exe /server DKMA_MedicineChest.wsdl
```

- Which generates the file with service skeleton, the abstract class, `MedicineChest.cs`

.Net 2.0

Introduction

As with .Net 1.1, client stubs and service skeletons are built using `wsdl.exe`.

The reference documentation for wsdl.exe: http://msdn2.microsoft.com/en-us/library/7h3ystb6(VS.80).aspx
Consider using WSCF from
http://www.thinktecture.com/Resources/Software/WSContractFirst/default.html

The include problem from .Net 1.1 was fixed in .Net 2.0, though .Net 2.0 retains a residual of the import problem covered early in this chapter; there is an error message, but no real error. The message also claims that "*Warning: Schema could not be validated. Class generation may fail or may produce incorrect results.*"
However, the generated class files have no errors.

Note that using the discover tool, `disco.exe`, all types can be downloaded to the local hard drive. `disco.exe` uses the complete wsdl file URL as argument, possibly indicated as a UNC path (`\\<server>\<share>`) . Consequently, in our case the command could look as follows:

```
disco.exe c:\dkma-eksempel\wsdl\DKMA_MedicineChest.wsdl
```

Client stubs

In order to use the local copies of the XSDs instead of retrieving them in the ISB, the entire list of *both included and imported XSDs* must be indicated on the command line.
Any embedded types do not need to be indicated on the command line.

As the XSD files in the example are referred to in relative terms, such as "`./DKMA_xxx.xsd`", it is expected that there are two directories, called `xsd and wsdl`, and that the XSD and WSDL files are placed respectively in those directories.

The following command is executed from the `xsd` directory, and will generate the client source file:

>

```
wsdl.exe ..\wsdl\DKMA_MedicineChest.wsdl
DKMA_DosageFormText.xsd DKMA_DrugIdentifier.xsd
DKMA_DrugName.xsd DKMA_DrugStrengthText.xsd
DKMA_DrugStructure.xsd DKMA_MedicineChestStructureGet.xsd
DKMA_MedicineChestItemChoiceStructure.xsd
DKMA_MedicineChestItemDate.xsd
DKMA_MedicineChestItemIdentifier.xsd
DKMA_MedicineChestItemStructure.xsd
DKMA_MedicineChestItemVersionIdentifier.xsd
DKMA_MedicineChestStructure.xsd DKMA_OwnerTypeText.xsd
CPR_PersonCivilRegistrationIdentifier.xsd
```

This will create `MedicineChest.cs`

(Note that `CPR_PersonCivilRegistrationIdentifier.xsd` was downloaded to the local hard drive and therefore appears on the command line).

Service skeleton

Service skeleton generation is completely similar to client stub generation, save the addition of the "`/serverinterface`" option to the command.
Note that in .Net 1.1 this option is called "`/server`". This was *depricated* on .Net 2.0, where it is called "`/serverinterface`".
If you indicate "`/server`" on .Net 2.0, it will be ignored, and you will create the client part!

Therefore, the command is:

```
wsdl.exe /serverinterface ..\wsdl\DKMA_MedicineChest.wsdl
DKMA_DosageFormText.xsd DKMA_DrugIdentifier.xsd
DKMA_DrugName.xsd DKMA_DrugStrengthText.xsd
DKMA_DrugStructure.xsd DKMA_MedicineChestStructureGet.xsd
DKMA_MedicineChestItemChoiceStructure.xsd
DKMA_MedicineChestItemDate.xsd
DKMA_MedicineChestItemIdentifier.xsd
DKMA_MedicineChestItemStructure.xsd
DKMA_MedicineChestItemVersionIdentifier.xsd
DKMA_MedicineChestStructure.xsd DKMA_OwnerTypeText.xsd
CPR_PersonCivilRegistrationIdentifier.xsd
```

This creates the abstract class for web service implementation;
`MedicineChest.cs`

.NET 3.0

Introduction

.NET Framework version 3.0 contains the Windows Communication Framework (WCF). WCF contains its own code generation program, based on

a WSDL named `svcutil.exe`. Service and client code generation must be done in the same way. The result is both an interface and classes implementing the various messages communicated. These elements can be used for both client and service code development.

The reference documentation for svcutil.exe:
http://msdn2.microsoft.com/en-us/library/aa347733.aspx

`svcutil.exe` is placed in the Windows SDK pack, downloadable from `microsoft.com`.

As opposed to `wsdl.exe` from .Net 1.1 and .Net 2.0, `svcutil.exe` can resolve wildcards, meaning that the complete list of xsd files can be referred to as `*.xsd`.

If needed, run

```
disco.exe c:\dkma-eksempel\wsdl\DKMA_MedicineChest.wsdl
```

- Which will download all xsd files.

Client stubs

The client stubs can be created in two ways; either through `svcutil.exe` or from Visual Studio.
In the example, the XSD files are referred to relatively, e.g. "`../DKMA_xxx.xsd`". Therefore, it is expected that two directories exist, called `xsd` and `wsdl`, and that all XSD and WSDL files are placed respectively in these directories.

If they are created using svcutil.exe, the approach is completely similar to that of .Net 2.0. From the xsd directory, run:

```
svcutil.exe DKMA_MedicineChest.wsdl *.xsd
```

The other option is to create the client stubs directly from Visual Studio:

1. Place the WSDL file and the XSD files on a web server
2. In Solution Explorer, right-click the project and select "`Add Service Reference`"
3. Input URL for the WSDL file
4. Input desired namespace in "`Web Reference Name`"
5. "`Add Reference`"

>

The client can now be used as follows:

```
MedicineChest service =
    new MedicineChest();
MedicineChestStructureType type =
    new MedicineChestStructurType();
type.PersonCivilRegistrationIdentifier = "1212121212";
Console.WriteLine(service.medicineChestStructureGet(type));
```

Instantiation of a client with a different endpoint than the one indicated in the WSDL file happens as follows (note that in this instance, the client is run against an axis2 web service):

```
MedicineChestTypeClient client =
    new MedicineChestTypeClient(new BasicHttpBinding(),
    new EndpointAddress
("http://10.0.0.206:8080/axis2/services/MedicineChest"));
```

Service skeleton

There is no service interface option for `svcutil.exe`. Instead, write a class that implements the `MedicinChest` interface, generated under the client part.

Ruby

Introduction

Generally, `soap4r` is used to generate both client and service. See reference documentation: `http://dev.ctor.org/soap4r`
Also, `soap4r` is very poorly documented, and experience shows that `rdebug` (`http://www.datanoise.com/articles/2006/07/12/tutorial-on-ruby-debug`) can be of great help along the way.

Client stubs

Run:

```
wsdl2ruby.rb --wsdl
   http://localhost:8080/wsdl/wsdl/DKMA_MedicineChest.wsdl
   --type client
```

This creates a simple client program, in this case: `MedicineChestClient.rb`.

The client works well, but the generated Ruby classes are not used. Instead, new elements can be accessed as arrays:

>

```
obj = MedicineChestType.new(endpointUrl)
str = MedicineChestStructureType.new
str.PersonCivilRegistrationIdentifier = "1212121212"
res = obj.medicineChestStructureGet(str)
id = res[MedicineChestItemStructure'][0]['MedicineChestItemIdentifier']
```

Service skeleton

Use `soap4r` (`http://dev.ctor.org/soap4r`) for generating client and service:

```
wsdl2ruby.rb --wsdl
    http://localhost:8080/wsdl/wsdl/DKMA_MedicineChest.wsdl
    --type server
```

The service is generated in e.g. `MedicineChestServiceGet.rb`, which can be run using Ruby, for starting a simple server.
`defaultServant.rb` contains the implementation.

Appendix: FAQ

>

What is an OIOWSDL?
OIOWSDL is a WSDL document, built on the guidelines set forth in this document, on how to build WSDL. OIOWSDL is based on *Web Service Description Language (WSDL) 1.1*, WS-I Basic Profile Version 1.1 and OIOXML NDR.

Why OIOWSDL?
A standardised approach to WSDL documents in Denmark, meaning that organisations only need to relate to one way to produce and apply WSDL documents, yielding the advantage that all OIO Web Services can be handled in the same way.

Relationship with WSDL Wizard (Web Service Workshop)
WSDL Wizard is a web-based tool placed on the ISB for a contract-first development of WSDL documents. Note that as the tool was developed before the guidelines for OIOWSDL, it does not follow this guideline completely.

Relationship to OWSA Model T
OIOWSDL changes the rules for WSDL, compared to the OWSA Model T. OWSA Model T ranks the WSDL files equally, with imported and embedded types respectively.
According to the OIOWSDL guidelines, only the imported version is published.

Is OIOWSDL meant only to function in the OWSA-T scenario?
There are no bindings to OWSA-T.

Why only support for HTTP and HTTPS? OIOSI also uses e.g. SMTP.
OIOWSDL is based on the WS-I Basic Profile version 1.1, and therefore only indicates HTTP and HTTPS. OIOSI exceeds the WS-I framework, and describes this in its profile.

What to do with data that is not yet standardized?
Before all parts of the public sector are covered, there can be no demands that all data definitions must be OIOXML approved. Until then, non-OIOXML schemas may be used. These schemas must as a minimum meet the "OIOXML Naming and Design Rules 3.0" (NDR). The exception is internationally standardised schemas adopted by OIOXML.

In WS-I Basic Profile 1.1 second edition rule R1127, "*A RECEIVER MUST NOT rely on the value of the SOAPAction HTTP header to correctly process the message.*" Why is this not reflected in the document?

>

We base OIOWSDL on WS-I Basic Profile version 1.1, which again is based on SOAP version 1.1. This is done to ensure the widest possible tool support.

Are WS Policy and other WS standards considered?
The focus has been on WSDL, but the guidelines can be used in connection with other WS* standards.

Is the WSDL document registered under the path indicated by the namespace?
That is not a requirement, but you may choose to follow the mechanism from the OIOXML schemas; i.e. a namespace indicates the location of the document.

Why is the example not in the ISB?
In order to make the example as realistic as possible, we have taken our point of departure in an existing on-going project, where we cannot control domain, WSDL and schemas.

Appendix: References

>

Definition of WSDL
http://en.wikipedia.org/wiki/WSDL

InfoStrukturBase
http://isb.oio.dk/Info/About/Introduktion+til+Infostrukturbasen.htm?wbc_purpose=Basic&WBCMODE=PresentationUnpublished

Web Service Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

WS-I Basic Profile Version 1.1
http://www.ws-i.org/Profiles/BasicProfile-1.1.html

OIOXML Naming and Design Rules (NDR) version 3
http://www.oio.dk/index.php?o=0f3120e8e51a0bf4c8247743abb6659c

OWSA Model T (OIO Web Service Architecture model Transport-based security) http://www.oio.dk/arkitektur/soa/webservices/owsa/model-t

Appendix: Check list >

The following checklist can be used to ensure that a WSDL document meets the OIO guidelines set forth in this document.

The WSDL document:

- Must be able to be validated with the WS-I Basic Profile Version 1.1, and thus WSDL 1.1
- May only include document literal bindings, i.e. no rpc literal bindings
- Must follow NDR as much as possible
- Has a name indication prefix, e.g. *DKMA*, followed by an underscore (_) and the service name
- Has a target-namespace, built by domain followed by *xml.wsdl* and date
- Exclusively applies request-response or one way operations
- Has messages named after the operation the given message is used for and indicating whether it is a request, a response or a fault message
- Has operations named after the *ObjectHandling* naming model
- Has portTypes named after the service, but unique in case there are more than one portType
- Has SOAP actions named by namespace followed by # and operation name
- Has a service element named by service name

Appendix: Definitions of terms

>

Inter operational, the ability for two systems to exchange and use exchanged information.

ISB, the InfoStrukturBase is the common public registry where all OIOXML schemas must be stored.

OIO, is a brand and is used as a prefix for public sector standards and guidelines approved by the National IT- and Telecom Agency. “OIO” is an acronym for “Open Information Online” or in Danish: “Offentlig Information Online” which translates to: “Public Information Online.

NITA, is the abbreviation for the Danish National IT and Telecom Agency.

OIOWS, are Web Services in an OIO context, as described in OWSA.

OIOXML, is XML that abides by the rules for naming and designing of XML for use in the public sphere, described in the ITST OIOXML collection of schema rules (NDR).

OWSA, the OIO Web Service Architecture is the common public architecture structure, created in a project called WS-pilot, focusing on OIO Web Service Architecture. In this connection, reference models are in the process of being described. Built on WS-I profiles, these models are specific to a Danish context. At the same time, reference implementation is being built, showing that the reference model can be built and operated.

Service Consumer, is the actor who makes use/utilises a displayed Web Service.

Service Provider, is the actor that displays a Web Service for use.

Service Developer, is the actor that develops a Web Service. This need not be the actor that becomes the service provider. It could an IT supplier that develops, and an agency that becomes the provider of the service.

WS-I Basic Profile, a specification from the *Web Services Interoperability* consortium, providing guidance on the interplay between Web Service specifications such as SOAP, WSDL and UDDI.

WSDL, Web Service Description Language is XML, which describes Web Services through their operations and addresses.

Appendix: XSD schemas as example >

DKMA_DosageFormText.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns=http://www.w3.org/2001/XMLSchema
xmlns:common="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <element name="DosageFormText" type="common:DosageFormTextType">
      <annotation>
         <documentation xml:lang="en-uk">Dosage form (DMA01 field 7)
         </documentation>
         <documentation xml:lang="da-dk">LÃ¦gemiddelform (LMS01 felt 7)
         </documentation>
    </annotation>
  </element>
  <simpleType name="DosageFormTextType">
    <restriction base="string">
      <maxLength value="100"/>
    </restriction>
  </simpleType>
</schema>
```

>

DKMA_DrugIdentifier.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:common="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="DrugIdentifier" type="common:DrugIdentifierType">
    <annotation>
      <documentation xml:lang="en-uk">Unique drug identification (DMA01 field
1)</documentation>
      <documentation xml:lang="da-dk">Unik identifikation af lÃ¦gemiddel (LMS01
felt 1)</documentation>
    </annotation>
  </element>
  <simpleType name="DrugIdentifierType">
    <restriction base="long"/>
  </simpleType>
</schema>
```

>

DKMA_DrugName.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:common="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="DrugName" type="common:DrugNameType">
    <annotation>
      <documentation xml:lang="en-uk">Drug name (DMA01 field 6)</documentation>
      <documentation xml:lang="da-dk">LÃ¦gemiddelnavn (LMS01 felt
6)</documentation>
    </annotation>
  </element>
  <simpleType name="DrugNameType">
    <restriction base="string">
      <maxLength value="30"/>
    </restriction>
  </simpleType>
</schema>
```

>

DKMA_DrugStrengthText.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:common="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="DrugStrengthText" type="common:DrugStrengthTextType">
    <annotation>
      <documentation xml:lang="en-uk">Drug strength (DMA01 field
10)</documentation>
      <documentation xml:lang="da-dk">LÃ¦gemiddelstyrke (LMS01 felt
10)</documentation>
    </annotation>
  </element>
  <simpleType name="DrugStrengthTextType">
    <restriction base="string">
      <maxLength value="20"/>
    </restriction>
  </simpleType>
</schema>
```

>

DKMA_DrugStructure.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:common="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <include schemaLocation="DKMA_DrugIdentifier.xsd"/>
  <include schemaLocation="DKMA_DrugName.xsd"/>
  <include schemaLocation="DKMA_DosageFormText.xsd"/>
  <include schemaLocation="DKMA_DrugStrengthText.xsd"/>
  <element name="DrugStructure" type="common:DrugStructureType">
    <annotation>
      <documentation xml:lang="en-uk">Drug</documentation>
      <documentation xml:lang="da-dk">LÃ¦gemiddel</documentation>
    </annotation>
  </element>
  <complexType name="DrugStructureType">
    <sequence>
      <element ref="common:DrugIdentifier" minOccurs="0"/>
      <element ref="common:DrugName" minOccurs="0"/>
      <element ref="common:DosageFormText" minOccurs="0"/>
      <element ref="common:DrugStrengthText" minOccurs="0"/>
    </sequence>
  </complexType>
</schema>
```

>

DKMA_MedicineChestItemChoiceStructure.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:homecare="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
xmlns:common="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <import namespace="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
schemaLocation="DKMA_DrugStructure.xsd"/>
  <element name="MedicineChestItemChoiceStructure"
type="homecare:MedicineChestItemChoiceStructureType">
  </element>
  <complexType name="MedicineChestItemChoiceStructureType">
    <choice>
      <element ref="common:DrugStructure" minOccurs="0"/>
      <element ref="homecare:MedicineChestDrugStructure" minOccurs="0"/>
    </choice>
  </complexType>
  <element name="MedicineChestDrugStructure"
type="homecare:MedicineChestDrugStructureType">
  </element>
  <complexType name="MedicineChestDrugStructureType">
    <sequence>
      <element ref="common:DrugName" minOccurs="0"/>
      <element ref="common:DosageFormText" minOccurs="0"/>
      <element ref="common:DrugStrengthText" minOccurs="0"/>
    </sequence>
  </complexType>
</schema>
```

>

DKMA_MedicineChestItemDate.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:homecare="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="MedicineChestItemDate" type="homecare:MedicineChestItemDateType">
  </element>
  <simpleType name="MedicineChestItemDateType">
    <restriction base="date"/>
  </simpleType>
</schema>
```

>

DKMA_MedicineChestItemIdentifier.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:homecare="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="MedicineChestItemIdentifier"
type="homecare:MedicineChestItemIdentifierType">
  </element>
  <simpleType name="MedicineChestItemIdentifierType">
    <restriction base="long"/>
  </simpleType>
</schema>
```

>

DKMA_MedicineChestItemStructure.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:homecare="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
xmlns:common="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <include schemaLocation="DKMA_MedicineChestItemIdentifier.xsd"/>
  <include schemaLocation="DKMA_MedicineChestItemVersionIdentifier.xsd"/>
  <include schemaLocation="DKMA_MedicineChestItemDate.xsd"/>
  <include schemaLocation="DKMA_MedicineChestItemChoiceStructure.xsd"/>
  <include schemaLocation="DKMA_OwnerTypeText.xsd"/>
  <element name="MedicineChestItemStructure"
type="homecare:MedicineChestItemStructureType">
  </element>
  <complexType name="MedicineChestItemStructureType">
    <sequence>
      <element ref="homecare:MedicineChestItemIdentifier" minOccurs="0"/>
      <element ref="homecare:MedicineChestItemVersionIdentifier" minOccurs="0"/>
      <element ref="homecare:MedicineChestItemDate" minOccurs="0"/>
      <element ref="homecare:MedicineChestItemChoiceStructure" minOccurs="0"/>
      <element ref="homecare:OwnerTypeText" minOccurs="0"/>
    </sequence>
  </complexType>
</schema>
```

>

DKMA_MedicineChestItemVersionIdentifier.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:homecare="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="MedicineChestItemVersionIdentifier"
type="homecare:MedicineChestItemVersionIdentifierType">
  </element>
  <simpleType name="MedicineChestItemVersionIdentifierType">
    <restriction base="long"/>
  </simpleType>
</schema>
```

>

DKMA_MedicineChestRemarkText.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:common="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="MedicineChestRemarkText"
type="common:MedicineChestRemarkTextType">
  </element>
  <simpleType name="MedicineChestRemarkTextType">
    <restriction base="string">
      <maxLength value="256"/>
    </restriction>
  </simpleType>
</schema>
```

>

DKMA_MedicineChestStructure.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:homecare="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <include schemaLocation="DKMA_MedicineChestItemStructure.xsd"/>
  <element name="MedicineChestStructure" type="homecare:MedicineChestStructureType">
  </element>
  <complexType name="MedicineChestStructureType">
    <sequence>
      <element ref="homecare:MedicineChestItemStructure" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</schema>
```

>

DKMA_MedicineChestStructureGet.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:homecare="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
xmlns:cpr="http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/03/18/"
targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <import namespace="http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/03/18/"
schemaLocation="http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/03/18/CPR_PersonCivilRe
gistrationIdentifier.xsd"/>
  <element name="MedicineChestStructureGet"
type="homecare:MedicineChestStructureGetType">
  </element>
  <complexType name="MedicineChestStructureGetType">
    <sequence>
      <element ref="cpr:PersonCivilRegistrationIdentifier"/>
    </sequence>
  </complexType>
</schema>
```

>

DKMA_OwnerTypeText.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:homecare="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="OwnerTypeText" type="homecare:OwnerTypeTextType">
  </element>
  <simpleType name="OwnerTypeTextType">
    <restriction base="string">
      <enumeration value="hjemmesygepleje"/>
      <enumeration value="borger"/>
    </restriction>
  </simpleType>
</schema>
```

Appendix: WSDL as an example >

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
  xmlns:xsd="http://www.w3.org/2001/XMLschema"
  xmlns:tns="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <types>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11">
      <xsd:include schemaLocation="../xsd/DKMA_MedicineChestStructureGet.xsd"/>
      <xsd:include schemaLocation="../xsd/DKMA_MedicineChestStructure.xsd"/>
      <xsd:element name="MedicineChestStructureFault" type="xsd:string"/>
    </xsd:schema>
  </types>

  <message name="MedicineChestStructureGetRequest">
    <part name="arg1" element="tns:MedicineChestStructureGet"/>
  </message>
  <message name="MedicineChestStructureGetResponse">
    <part name="arg2" element="tns:MedicineChestStructure"/>
  </message>
  <message name="MedicineChestStructureFault">
    <part name="fault" element="tns:MedicineChestStructureFault"/>
  </message>

  <portType name="MedicineChest">
    <operation name="MedicineChestStructureGet" parameterOrder="arg1">
      <input name="MedicineChestStructureGetRequest"
        message="tns:MedicineChestStructureGetRequest"/>
      <output name="MedicineChestStructureGetResponse"
        message="tns:MedicineChestStructureGetResponse"/>
      <fault name="MedicineChestStructureFault"
        message="tns:MedicineChestStructureFault"/>
    </operation>
  </portType>

  <binding name="MedicineChest" type="tns:MedicineChest">
    <soap:binding transport=http://schemas.xmlsoap.org/soap/http
      style="document"/>
    <operation name="MedicineChestStructureGet">
      <soap:operation

soapAction="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11/#MedicineChestGet"
        style="document"/>
      <input name="MedicineChestStructureGetRequest" >
        <soap:body use="literal"/>
      </input>
      <output name="MedicineChestStructureGetResponse">
        <soap:body use="literal"/>
      </output>
      <fault name="MedicineChestStructureFault">
        <soap:fault name="MedicineChestStructureFault" use="literal"/>
      </fault>
    </operation>
  </binding>

  <service name="MedicineChest">
    <documentation>WSDL file for MedicineChest</documentation>
    <port name="MedicineChest" binding="tns:MedicineChest">
      <soap:address location="http://localhost:8080/wstestservice/services/TestPort"/>
    </port>
  </service>

</definitions>
```

Appendix: WSDL with embedded types

>

```
<?xml version="1.0" encoding="UTF-8" ?>
  <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
    xmlns:xsd="http://www.w3.org/2001/XMLschema"
    xmlns:tns="http://rep.oio.dk/dkma.dk/homecare/xml.schema/2006.05.11"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <types>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLschema"
      xmlns:cpr="http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/03/18/"
      targetNamespace="http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/03/18/"
      elementFormDefault="qualified" attributeFormDefault="unqualified">
      <xsd:element name="PersonCivilRegistrationIdentifier"
        type="cpr:PersonCivilRegistrationIdentifierType">
      </xsd:element>
      <xsd:simpleType name="PersonCivilRegistrationIdentifierType">
        <xsd:restriction base="xsd:string">
          <xsd:pattern
            value="((((0[1-9]|1[0-9]|2[0-9]|3[0-1])(01|03|05|07|08|10|12))|((0[1
            -9]|1[0-9]|2[0-9]|30)(04|06|09|11))|((0[1-9]|1[0-9]|2[0-9])(02)))[0-
            9]{6})|0000000000" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:schema>

    <schema xmlns="http://www.w3.org/2001/XMLschema"
      xmlns:common="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
      targetNamespace="http://rep.oio.dk/dkma.dk/common/xml.schema/2006.05.11"
      elementFormDefault="qualified" attributeFormDefault="unqualified">
      <element name="DrugIdentifier" type="common:DrugIdentifierType">
        <annotation>
          <documentation xml:lang="en-uk">Unique drug identification
            (DMA01 field 1)
          </documentation>
          <documentation xml:lang="da-dk">Unik identifikation af
            lægemiddel (LMS01 felt 1)
          </documentation>
        </annotation>
      </element>
      <simpleType name="DrugIdentifierType">
        <restriction base="long" />
      </simpleType>
      <element name="DrugName" type="common:DrugNameType">
        <annotation>
          <documentation xml:lang="en-uk">Drug name (DMA01 field 6)</documentation>
          <documentation xml:lang="da-dk">Lægemiddelnavn
            (LMS01 felt 6)
          </documentation>
        </annotation>
      </element>
      <simpleType name="DrugNameType">
        <restriction base="string">
          <maxLength value="30" />
        </restriction>
      </simpleType>
```

>

```
      <element name="DosageFormText" type="common:DosageFormTextType">
        <annotation>
          <documentation xml:lang="en-uk">Dosage form (DMA01
            field 7)
          </documentation>
          <documentation xml:lang="da-dk">Lægemiddelform
            (LMS01 felt 7)
          </documentation>
        </annotation>
      </element>
      <simpleType name="DosageFormTextType">
        <restriction base="string">
          <maxLength value="100" />
        </restriction>
      </simpleType>
      <element name="DrugStructure" type="common:DrugStructureType">
        <annotation>
          <documentation xml:lang="en-uk">Drug</documentation>
          <documentation xml:lang="da-dk">Lægemiddel</documentation>
        </annotation>
      </element>
      <complexType name="DrugStructureType">
        <sequence>
          <element ref="common:DrugIdentifier" minOccurs="0" />
          <element ref="common:DrugName" minOccurs="0" />
          <element ref="common:DosageFormText" minOccurs="0" />
          <element ref="common:DrugStrengthText" minOccurs="0" />
        </sequence>
      </complexType>
      <element name="MedicineChestRemarkText"
        type="common:MedicineChestRemarkTextType">
        <annotation>
          <documentation xml:lang="en-uk">Remark text on Medicine Chest
            Item
          </documentation>
          <documentation xml:lang="da-dk">Bemærkninger til Præparat
            i Medicinskab
          </documentation>
        </annotation>
      </element>
      <element name="DrugStrengthText" type="common:DrugStrengthTextType">
        <annotation>
          <documentation xml:lang="en-uk">Drug strength
           (DMA01 field 10)
          </documentation>
          <documentation xml:lang="da-dk">Lægemiddelstyrke
            (LMS01 felt 10)
          </documentation>
        </annotation>
      </element>
      <simpleType name="DrugStrengthTextType">
        <restriction base="string">
          <maxLength value="20" />
        </restriction>
      </simpleType>
    </schema>

    (below follows the remaining embedded schemas)

  <message name="MedicineChestStructureGetRequest">
    <part name="arg1" element="tns:MedicineChestStructureGet"/>
  </message>

  (remainder of wsdl document)
```

<

Overskrift
Bagside kursiv tekst

Bagside brødtekst