What is in a Distributed Object System?
- Wire (transport) protocol
- Marshalling standard
- Language bindings
- Middle-ware (ORB)
- Interface specification formalism (IDL)
- Proxy/stub generation
- Object references
- Services

Examples
- Corba:
  - wire protocol: GIOP/IIOP
  - marshalling: some not well known binary encoding
- RMI:
  - wire protocol: JRMP (Java Remote Method Protocol)
  - marshalling: Java serialisation

Why not use standard protocols and encodings?
- e.g. HTTP, XML

HTTP protocol
- HTTP is a simple request/response protocol
  - Client sends a request to server
  - Server sends a response to client
- HTTP message structure:
  - A number of headers
  - Empty line
  - (optional) body

HTTP example
- without body:
  ```plaintext
  GET /docs/vakken/gob/index.html HTTP/1.1
  Connection: Keep-Alive
  Host: sunshine.cs.uu.nl:8000
  ```
- with body:
  ```plaintext
  POST /cgi-bin/form.py HTTP/1.1
  Host: sunshine.cs.uu.nl:8000
  Content-length: 11
  field=value
  ```

Example response
- 200 OK
- Content-type: text/plain
- Content-length: 12
  ```plaintext
  Hello, world
  ```
- Response can be any media type
- Most of them time HTML or images
- However, XML is better structured for data instead of visual information
XML-RPC

- XML-RPC is a RPC system with HTTP as transport protocol and XML as marshalling system.
- It has a fixed DTD.
- In principle not object-oriented (but could be used a bit OO if you fake object references)
- It is very lightweight
  - minimal number of features
  - easy to use
  - not extensible
  - fixed DTD

XML-RPC example

```
POST /RPC HTTP/1.0
User-Agent: Python
Host: xloo.cs.uu.nl
Content-Type: text/xml
Content-length: 172

<?xml version="1.0"?>
<methodCall>
  <methodName>Hstogram.addItem</methodName>
  <params>
    <param><value><int>7</int></value>
  </param>
</methodCall>
```

XML-RPC DTD

```xml
<!ELEMENT methodCall (methodName, params?)>
<!ELEMENT methodname (#pcdata)>
<!ELEMENT params (param*)>
<!ELEMENT param (value)>
<!ELEMENT value (
  i4 | int | boolean | string | double |
  | datetime.iso8601 | base64 |
  | struct | array) >

<!ELEMENT methodResponse (params|fault)>
<!ELEMENT fault (value)>
```

Datatypes

```xml
<!element i4 (#pcdata)>
<!element int (#pcdata)>
<!element boolean (#pcdata)>
<!element string (#pcdata)>
<!element double (#pcdata)>
<!element datetime.iso8601 (#pcdata)>
<!element base64 (#pcdata)>
<!element struct (member*)>
<!element member (name, value)>
<!element array (data)>
<!element data (value*)>
```

Example response

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 155
Content-Type: text/xml
Date: Fri Feb 25 09:57:17 CET 2005
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

Fault example

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>4</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>Too many parameters</string>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```
Python Usage example

```python
import xmlrpclib
server = xmlrpclib.ServerProxy("http://betty.userland.com")
try:
    print server.examples.getStateName(41)
except Error, v:
    print "ERROR", v
```

Question:
Where are the names 'example', 'getStateName' defined?
How does Python know what to do?

Python dynamic features

- Python has some unique features to accommodate dynamic behaviour:
  - Attributes can be defined at runtime
  - Attribute can 'magically' spring into existence
  - Any object can be made callable
  - Any callable attribute will be considered a method
- Missing attributes and methods:
  When `obj.attr` or `obj.attr(...)` are called and `obj` does not have 'attr' defined
  but `obj` has a method `__getattr__` defined
  then `obj.__getattr__('attr')` is called
  (The attribute name as a string is given as parameter)
- There is a similar `__setattr__` for assignments to attributes

Callable objects

- An object that has a method with name `__call__` can be called even when it is not a function object
- Then `obj(p1,p2,...)` is evaluated as `obj.__call__(p1,p2,...)`

Example resolved

Our example:
```python
server.examples.getStateName(41)
```
- The server proxy object does not have an attribute `examples` because it is defined without knowledge about the remote object
- Therefore `server.examples` calls
  - `server.__getattr__('examples')`
  - `server.__getattr__` creates an object
  - This object can either accept a call (with `__call__`) or new attributes (with `__getattr__`)
  - Recursively: `server.examples` and `server.examples.getStateName` are similar objects
- The `__call__` methods send the name of the method and the parameters to the server with the XML-RPC protocol and returns the response

XML-RPC issues

- What does the method name mean?
  - Not specified: it is up to the server
- How is it used?
  - At this moment usually the client just builds the request
  - With the help of a library
- How is the server used?
  - Existing HTTP server can be used
    - Request can be processed with CGI scripts, servlets etc.
  - Write an application-specific server
  - HTTP servers can be very simple
  - Programming language support e.g. for proxies (stubs) could be added

What is SOAP

- SOAP (originally Simple Object Access Protocol) is a protocol that uses Standard Internet protocols for communication and XML for marshalling
- It is not simple and not about objects (more like RPC)
- Usually HTTP, but also SMTP, FTP, etc. are possible.
- The body of the messages can be an XML document as defined in SOAP, but also other XML documents are possible.
- For other documents as parameters also MIME attachments like in email are possible.
- The marshalling can be user-defined.
- Additional headers for security, transaction processing etc. are possible.
Processing XML

- Parsing XML is usually done with SAX or DOM
  - SAX:
    - streaming parsing
    - During reading/parsing when something interesting is encountered, a method from the user program is called
    - Interesting things: element start, element end, text, processing instruction, etc.
    - The whole document doesn’t have to be in memory
  - DOM:
    - build a datastructure (tree-like) of the whole document in memory
    - user program can process the tree
  - Generating XML can be done with simple string operations
  - Usually easier to use a higher abstraction level
  - DOM is popular to build/manipulate XML trees

SOAP processing

- Do-it-yourself XML manipulation and HTTP protocol processing
  - Usually with standard HTTP (etc.) library and XML library
  - Gives greatest flexibility, but often is tedious
  - You have to write marshalling code yourself (can be simple read/print methods from the language)
- Higher abstraction that does the XML/HTTP manipulation for you
  - Use ‘request’, ‘parameter’ etc. as abstraction levels
  - Similar to Dynamic invocation in Corba and COM
  - Apache SOAP uses this model
  - RPC-like calling
    - Generally needs generated stubs and skeletons
    - Apache Axis, Python SOAPpy library
  - Built-in in programming language
    - .NET

SOAP characteristics

- The method name is specified in a separate header (1.1)

  SOAPAction: urn:stats-com:Stat#addItem

- In SOAP 1.2 Mime content type:
  - application/soap+xml; action=
- The value is a URI (URL or URN)
- Doesn’t have to be a real or existing URI
- Can be used by a firewall to filter requests
- How it is interpreted depends on the server
- However, there can also be a method name in the SOAP body:

```
<SOAP-ENV:Body>
  <m:GetLastTradePrice xmlns:m="Some-URI">
    <symbol>DIS</symbol>
  </m:GetLastTradePrice>
</SOAP-ENV:Body>
```

SOAP 1.1 specification

The SOAPAction HTTP request header field can be used to indicate the intent of the SOAP HTTP request. The value is a URI identifying the intent. SOAP places no restrictions on the format or specificity of the URI or that it is resolvable. An HTTP client MUST use this header field when issuing a SOAP HTTP Request. The presence and content of the SOAPAction header field can be used by servers such as firewalls to appropriately filter SOAP request messages in HTTP. The header field value of empty string (""") means that the intent of the SOAP message is provided by the HTTP Request-URI. No value means that there is no indication of the intent of the message.

SOAP request body

- The SOAP request body contains an XML document
- The document has a SOAP envelope, SOAP headers and a SOAP body
- These are all XML elements
- It uses namespaces
- The namespace determines the SOAP version
  - HTTP request
    - HTTP headers
    - HTTP body
      - SOAP envelope
        - SOAP header
        - SOAP body

SOAP processing

- SOAP requests can be processed by more than one processor
- Each processor picks up its own part from the request body and processes it
- It can remove that part or replace it
- It sends the rest of the request to the next processor
- Parts can be mandatory or optional
- Can be used e.g. for security, transactions etc.
SOAP example – HTTP headers

POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope

Example SOAP header

The SOAP header is optional.

<SOAP-ENV:Envelope

SOAP datatypes

- int : 12345
- float : 2718.28E-3
- negativeInteger : -98765, positiveInteger: 12345
- also nonnegativeInteger, nonpositiveInteger
- string : “How are you?”
- Enumerations
- Structs
- Arrays
- Byte arrays (binary, BASE64 encoded)
- Polymorphic accessor, must contain the type

These are defined using XML schema datatypes (XSD)

Enumerations

<element name="EyeColor" type="tns:EyeColor"/>
<simpleType name="EyeColor" base="xsd:string">
    <enumeration value="Green"/>
    <enumeration value="Blue"/>
    <enumeration value="Brown"/>
</simpleType>

Structs

<e:Book>
<title>Paradise Lost</title>
<firstauthor href="http://www.dartmouth.edu/~milton/">
</e:Book>
Arrays

- Arrays can be multidimensional `int[2,3]`
- Or ‘ragged’ `int[1][3]`
- In this case it contains 3 arrays which can each have a different length

Web Services

Some Definitions

- “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.” (W3C)

- “Web Services are Web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data with calling clients.” (Sun)

Web Applications

Web Application:

- Programs and resources embedded in, or accessed through, a Web server

Two kinds:

- Presentation oriented, built around (dynamic) HTML pages intended for people
- Service oriented, a Web Service made available to other applications through XML and HTTP protocols intended for computers/programs

⇒ A Web Service is software on local or remote machine with a set of well defined interfaces (ports) specified in XML (WSDL)

Web Services/SOA

Used for

- EAI (Enterprise Application Integration): integrating the various applications in an enterprise with each other.
- B2Bi (Business-to-Business Integration): the business interactions between different enterprises.

- “Service-Oriented Architecture (SOA) is a component model that inter-relates an application’s different functional units, called services, through well-defined interfaces and contracts between these services. The interface is defined in a neutral manner that should be independent of the hardware platform, the operating system, and the programming language in which the service is implemented. This allows services, built on a variety of such systems, to interact with each other in a uniform and universal manner.” (IBM)
Web services vs Distributed Objects
▶ Web Services (WS) and Distributed Objects (DO) are similar but there are differences
▶ Both involve distributed entities but Web Services are more loosely coupled
▶ Both are (more or less) based on the RPC paradigm
▶ WS interact with messages; DO with RPC
▶ DO have “factories”; WS manage instances internally (not really OO)
▶ DO have explicit state (stateful services)
▶ WS use context in the messages to link interactions (stateful interactions)

Web Services structure

<table>
<thead>
<tr>
<th>UDDI</th>
<th>Universal Description, Discovery and Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>XSD</td>
<td>language neutral type descriptions</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>

Current Java standard support
▶ Java Platform, Enterprise Edition (Java EE) provides the APIs and tools
▶ Java API for XML-based RPC (JAX-RPC) uses SOAP and HTTP
▶ client programs can make XML-based remote procedure calls.
▶ supports WSDL
▶ with JAX-RPC and a WSDL, can interoperate with clients and services running on Java-based or non-Java-based platforms such as .NET.
▶ The SOAP with Attachments API for Java (SAAJ) enables to produce and consume messages conforming to the SOAP 1.1 specification and SOAP with Attachments.
▶ JAX-RPC is now called JAX-WS (it is not only RPC).
▶ Some people think it is quite bad

SOAP request body
▶ The SOAP request body contains an XML document
▶ The document has a SOAP envelope, SOAP headers and a SOAP body
▶ These are all XML elements
▶ It uses namespaces
▶ The namespace of the SOAP envelope determines the SOAP version
▶ Namespaces used:
  ▶ SOAP message (envelope, header, body) namespace
  ▶ SOAP Encoding namespace
  ▶ namespace for the methods
  ▶ XML Schema namespace (for schema tags)
  ▶ XML Schema types (for standard types)
  ▶ maybe namespaces for transactions, own types and marshalling etc.

SOAP use
▶ SOAP is not yet a mature system like Corba or COM
▶ No (general) support for object creation, object references etc.
▶ How to use:
  ▶ Write your own support
  ▶ Microsoft .NET uses SOAP internally
  ▶ Microsoft SOAP SDK
  ▶ SOAP for Java: Apache or Axis
  ▶ Apache SOAP:
    ▶ originally developed by IBM
    ▶ uses special objects and methods, e.g.
      org.apache.soap.rpc.Call object
  ▶ Python support is available (but not standard)
SOAP styles

There are different ways in which SOAP can be used

▶ RPC - like remote procedure call:
  - request and response
▶ Message oriented:
  - Only one message is sent
  - Response message may be sent asynchronously and processed independently

Corresponding to this there are different styles of encoding the input and/or output:

▶ RPC style:
  - data is encoded (in XML) as methodname + list of parameters
▶ document style:
  - data is encoded as an XML document

RPC style encoding

```java
public void myMethod(int x, float y);
```

WSDL example:

```xml
<message name="myMethodRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>
<portType name="MyPort">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>
</binding>
```

Port Type / Service / Binding

▶ A port type component describes a set of messages that a service sends and/or receives. It does this by grouping related messages into operations. An operation is a set of input and output messages, a port type is a set of operations.

▶ A port type can optionally extend one or more other port types. In such cases the port type contains the operations of the port types it extends, along with any operations it defines. (So port types are similar to interfaces in RMI and Corba)

▶ A service component describes the set of port types that a service provides and the ports they are provided over.

▶ A binding component described a concrete binding of a port type component and associated operations to a particular concrete message format and transmission protocol.

SOAP message

Namespaces used:

```xml
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

SOAP message:

```xml
<soap:envelope>
  <soap:body>
    <myMethod>
      <x xsi:type="xsd:int">5</x>
      <y xsi:type="xsd:float">5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

Notes

▶ This is called the RPC/encoded style, because it contains type info
▶ The operation name appears in the message (easy for receiver to dispatch)
▶ The type encoding info (such as xsi:type="xsd:int") is maybe just overhead
▶ But it may be useful for overloading
▶ You cannot easily validate this message since there is no schema defining the message (only the x and y part)
▶ RPC/encoded is not WS-I compliant.
RPC/literal style

WSDL is almost the same, except there is a `use="literal"` instead of `use="encoded"` in the binding part.

SOAP message:

```xml
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
      <y>5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

The type info has disappeared.

Notes

▷ Document/encoded style is not used.
We use only Document/literal style.
▷ There is a schema definition in the WSDL
▷ The parameters have an element name

```xml
<schema>
  <element name="xElement" type="xsd:int"/>
  <element name="yElement" type="xsd:float"/>
</schema>
...
<part name="x" element="xElement"/>
```

Document/literal wrapped

This style wraps a document root with the method name around the parameters

```xml
<message name="myMethodRequest">
  <part name="parameters" element="myMethod"/>
</message>
<message name="empty">
  <part name="parameters" element="myMethodResponse"/>
</message>
```

SOAP message

```xml
<soap:envelope>
  <soap:body>
    <xElement>5</xElement>
    <yElement>5.0</yElement>
  </soap:body>
</soap:envelope>
```

Notes:

▷ There is no type encoding info.
▷ You can validate this message
   Everything within the soap:body is defined in a schema.
▷ The operation name in the SOAP message is lost.
▷ WS-I only allows one child of the soap:body in a SOAP message.
   This example’s soap:body has two children.

Document/literal wrapped

```xml
<types>
  <schema>
    <element name="myMethod">
      <complexType>
        <sequence>
          <element name="x" type="xsd:int"/>
          <element name="y" type="xsd:float"/>
        </sequence>
      </complexType>
    </element>
    <element name="myMethodResponse">
      <complexType/>
    </element>
  </schema>
</types>
```

```xml
<message name="myMethodRequest">
  <part name="parameters" element="myMethod"/>
</message>
```

```xml
<message name="empty">
  <part name="parameters" element="myMethodResponse"/>
</message>
```
SOAP message

```xml
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
      <y>5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

- Looks like the RPC/literal style
  - in RPC it is the method name
  - here it is the root element of the document
- Document style can be used to pass any type of XML document, not only a parameter list
- The message can be validated (there is a schema)
- No type info in the message (difficult in case of overloading)
- There is a method name

Advantages of document style

- You can use any XML document as input or output
- You are not bound to a fixed interface (in the IDL sense)
- Gives more flexibility
- Good for asynchronous processing

Python example

- Python can use `__getattr__` and `__call__` like XMLRPC
- Class can override the `__call__()` operator
- Undefined method names and attributes can be intercepted
- SOAPpy and ZSI (Zolera SOAP Infrastructure) are two open-source SOAP libraries for Python
- These are not yet complete
- (http://www.xmethods.com/ contains a list of free SOAP services)
- E.g. currency exchange (how many dollars per euro?)

```python
from SOAPpy import SOAPProxy
url = "http://xurrency.com/servidor_soap.php"
namespace = "http://xurrency.com/api.wsdl"
server = SOAPProxy(url, namespace)
server.config.dumpSOAPOut = True
server.config.dumpSOAPIn = True
print server.getValue(100, "eur", "usd")
```

Using WSDL:

```python
from SOAPpy import SOAPProxy
url = "http://xurrency.com/servidor_soap.php"
namespace = "http://xocurrency.com/api.wsdl"
service = SOAPProxy(url, namespace)
service.config.dumpSOAPOut = True
service.config.dumpSOAPIn = True
print service.getValue(100, "eur", "usd")
```

WSDL extract

```xml
<message name="getValueRequest">
  <part name="amount" type="xsd:float" />
  <part name="base" type="xsd:string" />
  <part name="target" type="xsd:string" />
</message>

<message name="getValueResponse">
  <part name="return" type="xsd:float" />
</message>

<portType name="xurrencyPortType">
  ...
  <operation name="getValue">
    <input message="tns:getValueRequest" />
    <output message="tns:getValueResponse" />
  </operation>
</portType>
```

WSDL extract

```xml
<binding name="xurrencyBinding" type="tns:xocurrencyPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/transport/uri"/>
  ...
  <operation name="getValue">
    <input message="tns:getValueRequest" />
    <output message="tns:getValueResponse" />
  </operation>
</binding>

<service name="xocurrency">
  <port name="xocurrencyPort" binding="tns:xocurrencyBinding">
    <soap:address location="http://xocurrency.com/servidor_soap.php"/>
  </port>
</service>
```
SOAP servers

- SOAP services run in an HTTP server (when protocol is HTTP)
- Could be standard HTTP server, like Apache, Tomcat
- Often run in Application server like IBM WebSphere or JBoss (together with EJB)
- Or run server with its own HTTP request processing
- Some of these frameworks have a HTTP server built in

Example CGI scripts (using Python ZSI)

```python
def hello():
    return "Hello, world"

def echo(*args):
    return args

def average(*args):
    sum = 0
    for i in args: sum += i
    return sum / len(args)

from ZSI import dispatch
dispatch.AsCGI()
```

This makes all the methods (functions) in the module available as Web Services.