



University of Dublin  
Trinity College



---

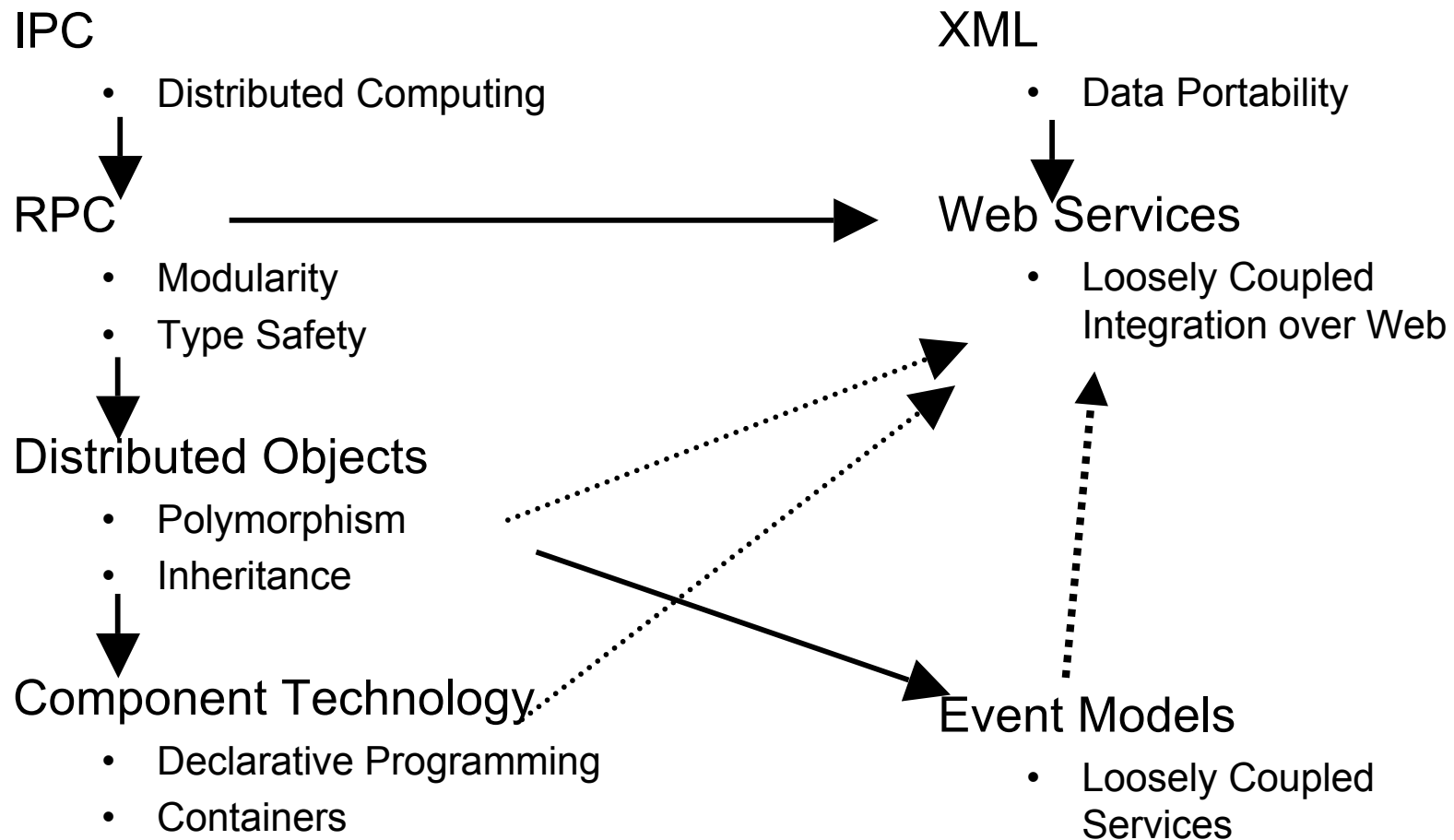
# Web Services

## Component Technology

---

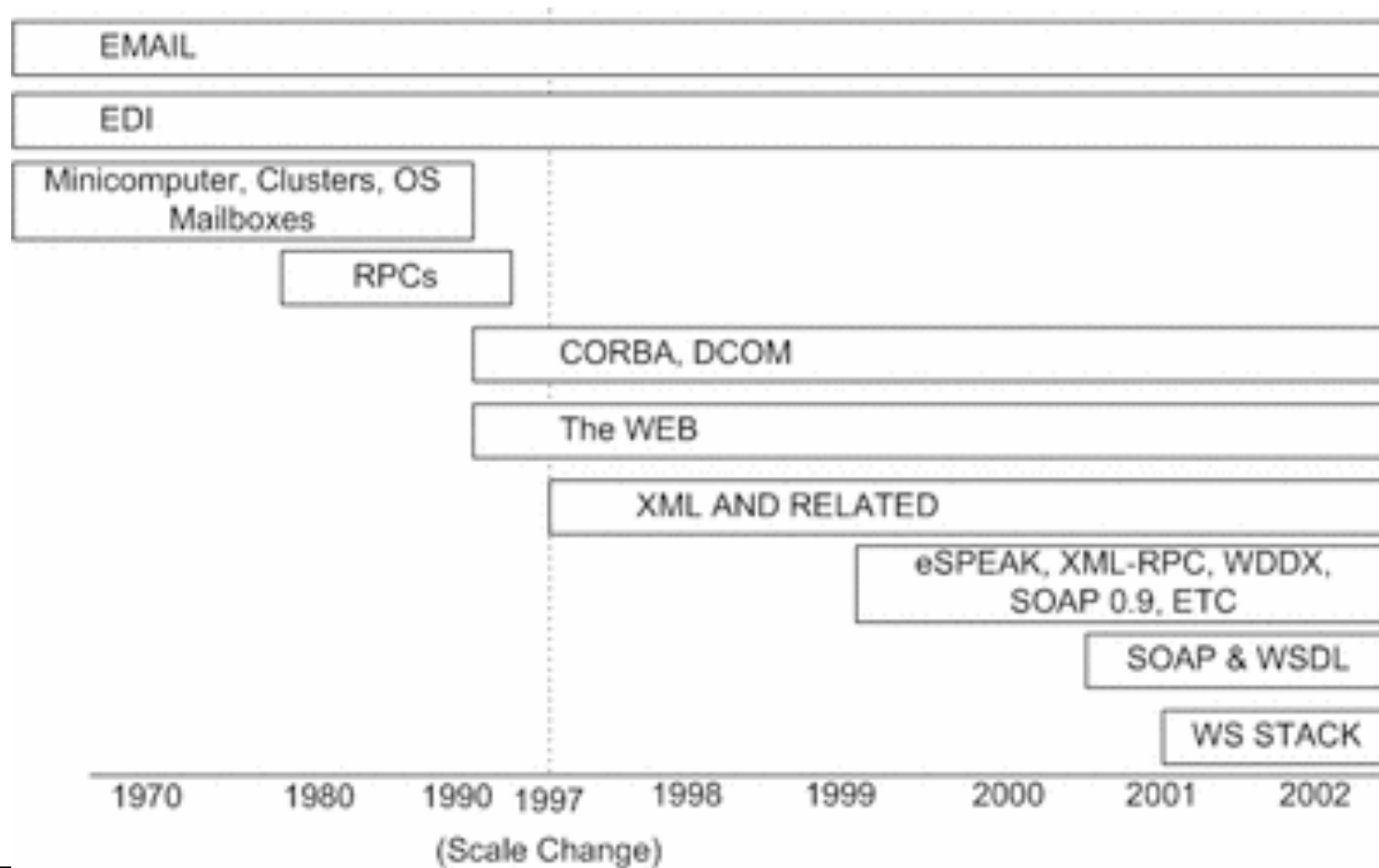
# Evolution of Web Services

---



# Web Services & Dist. Comp.

---



# The Three Revolutions

---

## The Data Revolution

- Open Data Specifications
  - *XML Schemas*

## The Architectural Revolution

- Loosely coupled systems over HTTP & TCP/IP
  - *XML protocol languages, e.g. SOAP, XML/RPC*

## The Software Revolution

- Modular Design & Systems Integration
  - *Web Services*

# Web Services

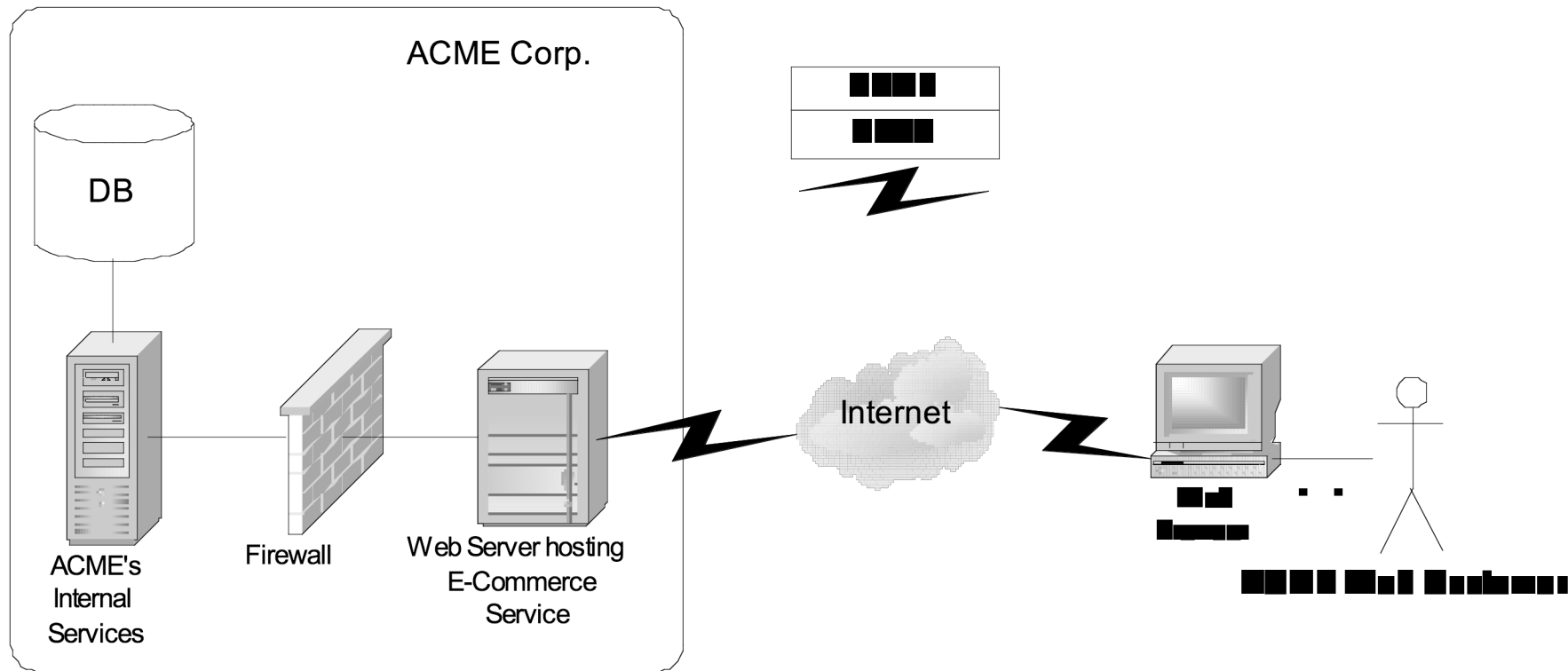
---

What technologies can businesses use to integrate their services via the Internet?

What possibilities does this open up?

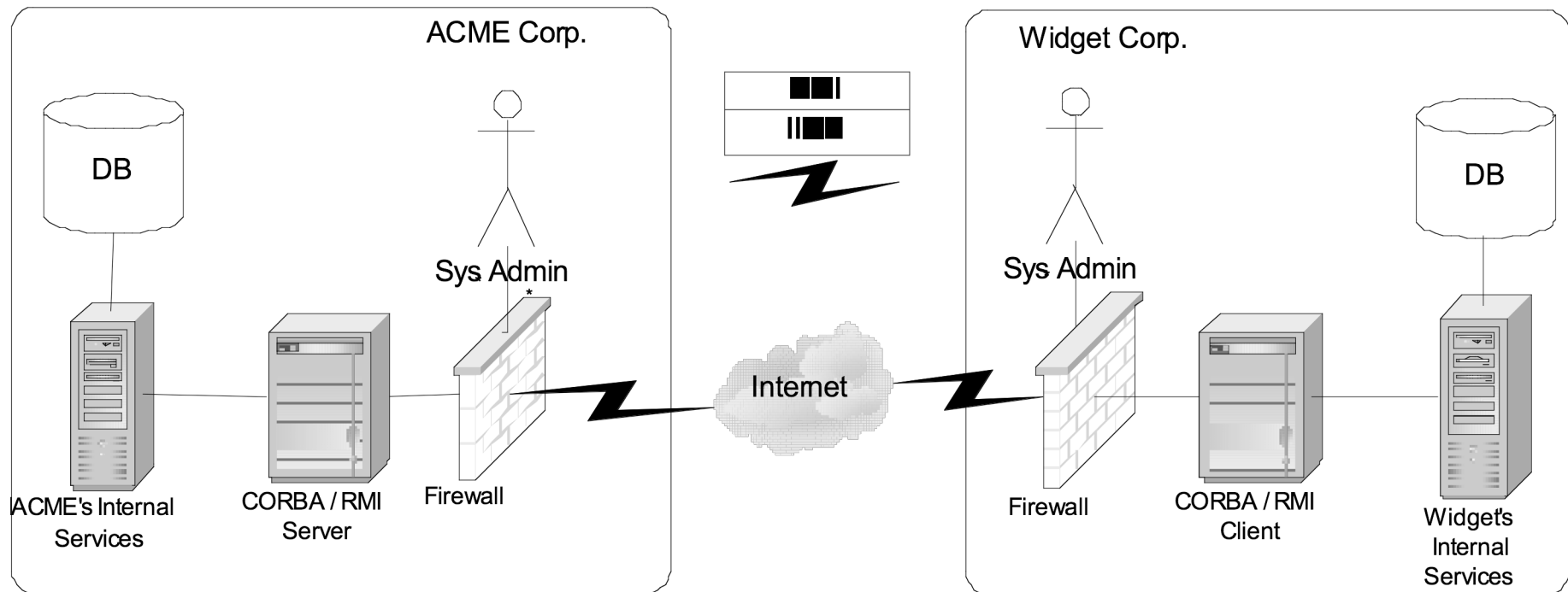
# Loosely Coupled "Traditional" Web Services

---



# Integrating Distributed Object Systems over IP

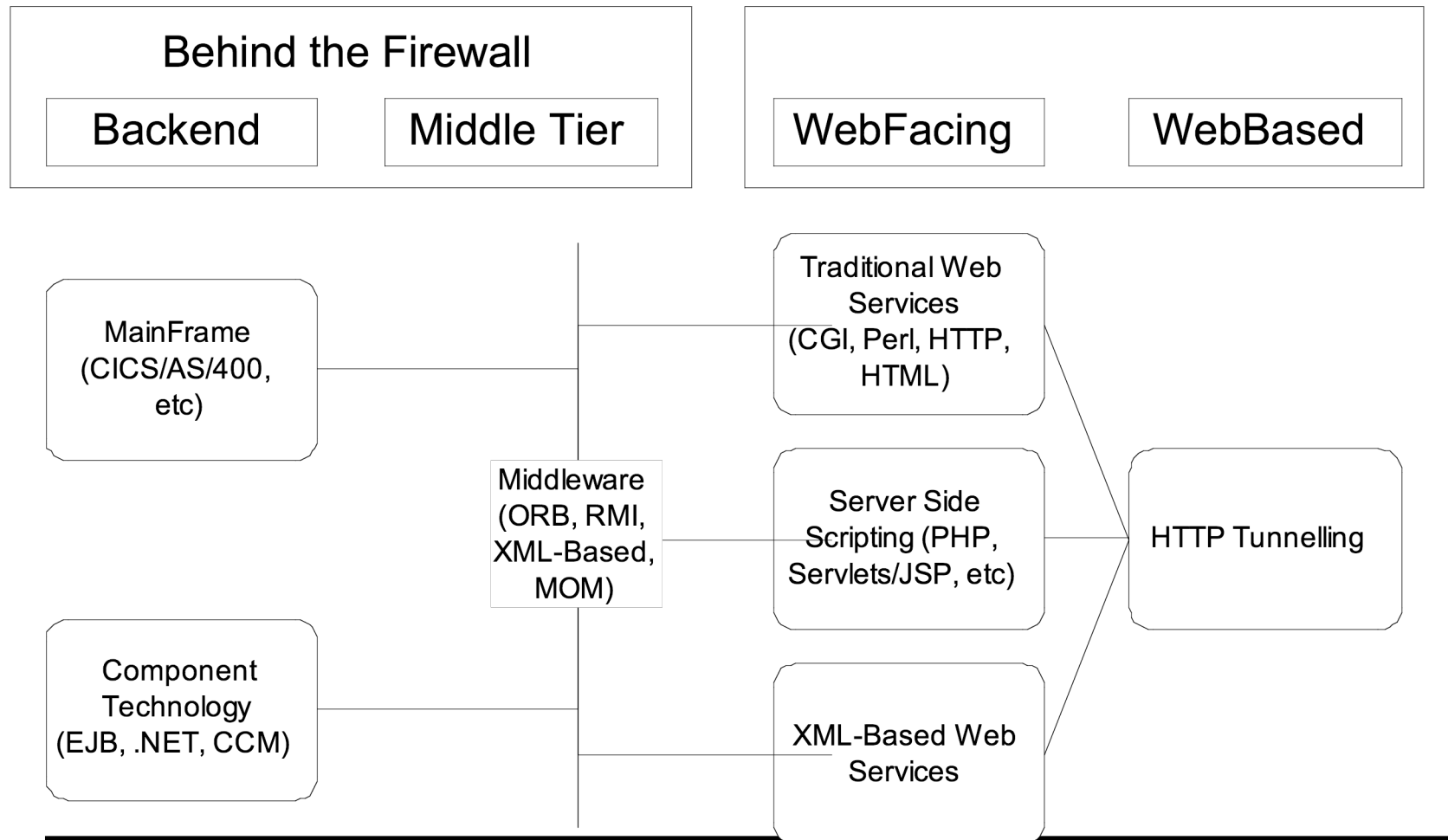
---



System Administrators have to reconfigure firewalls to allow Distributed Object-Based Application work over the Internet ☹

---

# Enterprise Computing and Firewalls





# Firewall Traversal

---

Firewalls block traffic on different TCP/IP ports.

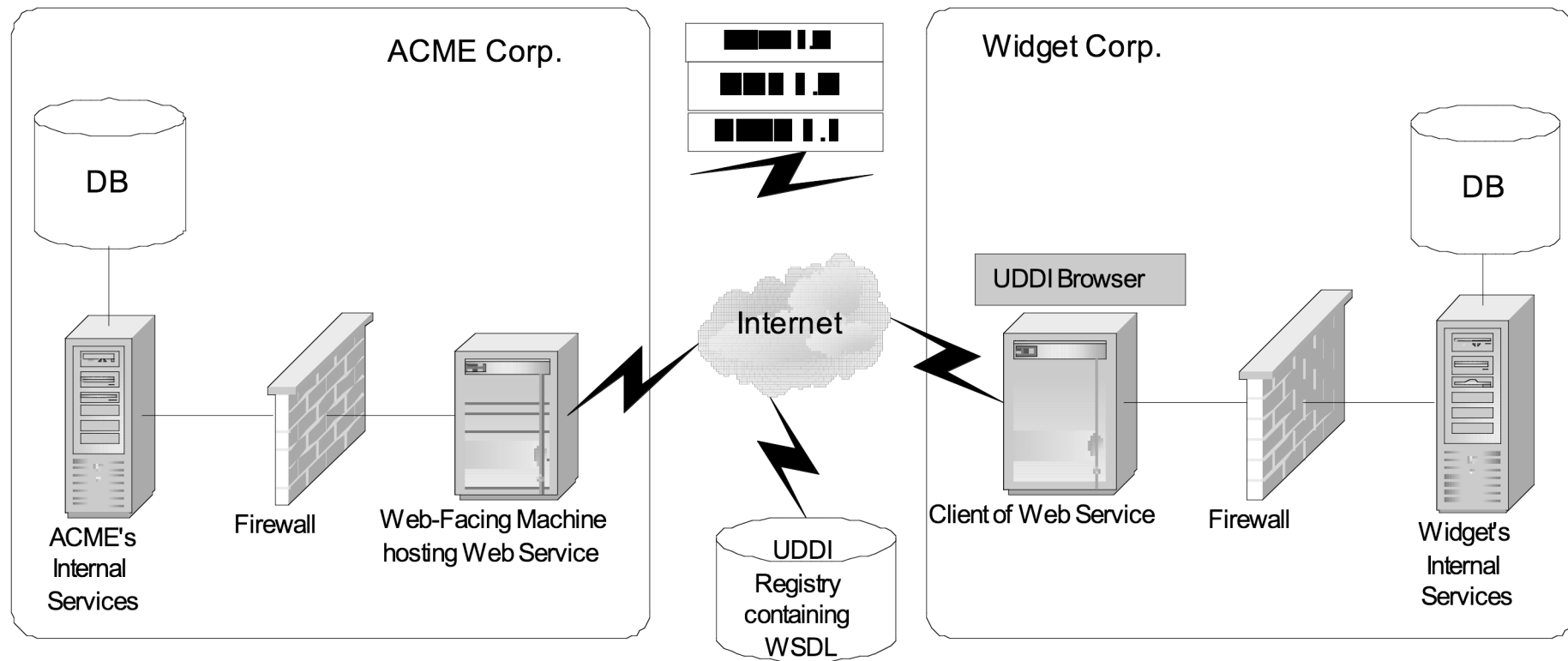
- Configured to allow popular ports to work, including port 80 used for HTTP, port 21 used for FTP, port 23 used for telnet and combinations of ports 25, 110 and 143 for e-mail protocols.

Java RMI and CORBA can be used to communicate between systems through different TCP/IP ports.

- For most organizations all but the most popular ports (see above) are closed off, which means the RMI and CORBA solutions will most likely no longer work without re-configuration of the firewall.

Simple Object Access Protocol (SOAP) uses HTTP as its transport and doesn't require any firewall re-configuration.

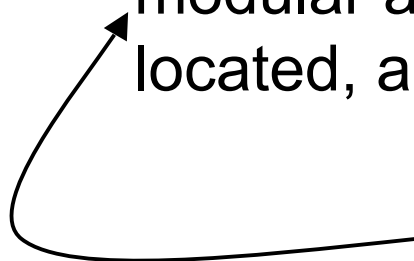
# Web Services Architecture



# What are Web Services?

---

Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Internet.



The general meaning of the term is services offered via the Web

## Loosely Coupled Services

- cf. success of loose coupling of e-commerce and other services over the web

# Web Services Properties

---

Properties of Web Services include:

- Based upon open standards- UDDI, SOAP, HTTP
- Accessible via widely deployed protocols such as HTTP and SMTP.
- Support loosely-coupled service-oriented architectures.
  - *Services advertise their existence in some form of registry or directory service.*
  - *Clients look up services in the directory.*
  - *Hide language/platform details from all participants.*

# Enterprise Level Possibilities

---

Enterprise applications have largely remained corporation-bound

- CORBA, J2EE and DCOM/.NET applications have largely remained behind corporate firewalls

Web Services describes a new way to perform inter-organisational distributed computing

- Uses traditional Internet protocols such as HTTP and SMTP to carry application data between organisations and beyond corporate firewalls

# The Web Services Musketeers

---

## A Contract Definition Language

- Web Service Description Language (WSDL).
- De Facto standard.

## Standardized Look-up

- Universal Description Discovery and Integration (UDDI).

## Interoperability standards

- XML to describe messages & Simple Object Access Protocol (SOAP) for RPC

In short, Web Services are described using WSDL, are published and located via the UDDI, the application data is encoded in XML and they are invoked using SOAP over HTTP.

---

---

# Web Services Musketeers

---

XML

SOAP

WSDL

UDDI

Web Services Example



# eXtensible Mark-up Language

---

The ISO-standard SGML without the silly bits

Defines a standard way of defining a document's structure

- A mixture of text and control elements
- Schemas & Document Type Definitions (DTDs)
  - *What elements are allowed, how they can be nested, what can and must appear*
  - *The XML Schema language is also referred to as XML Schema Definition (XSD)*

A Web Services standard

- For describing all aspects of Web Services
  - *SOAP, WSDL, UDDI Schemas*

# General form

---

## Looks extremely like HTML

- Ordinary text with mark-up in angle brackets

XML Element {  
    <?XML version="1.0"?>  
    <note>  
        <to>Paddy</to> <from>Evelyn</from>  
        <heading>Reminder</heading>  
        <body>Mow the lawn!</body>  
    </note>

This identifies the document as XML

Unlike HTML, XML is case-sensitive & preserves white spaces - which is going to cause some real problems...

## Differences

- No pre-defined elements – no built-in capabilities, everything has to be defined and implemented by applications
- Strict rules on validity and well-formedness – applications can afford to be less tolerant

# Openness and its Consequences

---

## XML has *no* in-built semantics or capabilities

- *Everything to be represented has to be defined externally*
- *...and the definition has to be agreed by all parties who want to share the data*

## No silver bullet

- *Data format like any other – it has no **intrinsic** advantages over any other text format, except for wide adoption, but...*

## Chicken-and-egg adoption cycle

- *Not useful until widely adopted*

## Adoption by Web Services Community

- *Already agreed XML Schemas for SOAP, WSDL, UDDI*
- *...and these are shared across the web services community, even if they're not "open" in terms of participation in their definition*

# XML Extensibility

---

## XML documents are independently extensible

- They can be extended without breaking applications that are dependent on the XML document
  - *Cf. Distributed Object Interfaces & Microsoft applications that link with DLLs*
- E.g. An application that extracts the <to>, <from>, and <body> elements from the old XML document to produce the output below continues to work with the new XML document.

**Message**  
**To: Paddy**  
**From: Evelyn**  
**Mow the lawn!**

### Old XML document

```
<?XML version="1.0"?>
<note>
<to>Paddy</to>
<from>Evelyn</from>
<body>Mow the lawn!</body>
</note>
```



### New XML document

```
<?XML version="1.0"?>
<note>
<to>Paddy</to>
<from>Evelyn</from>
<heading>Reminder</heading>
<body>Mow the lawn!</body>
</note>
```

# XML Elements

---

## XML Elements have Relationships

- Elements are related as parents and children.

## XML Elements can have different content types

- mixed content, simple content, empty content or **attributes**

## XML elements can have attributes.

- Data can be stored *either* in child elements *or* in attributes.

# XML Attributes

---

The "to" element below has the attribute "email"

```
<to email="ft@ft.org">Paddy</to>
```

- Attribute values must always be quoted

## Problems with attributes

- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- attributes cannot describe structures (child elements can)
- attributes are more difficult to manipulate by program code
- attribute values are not easy to test against a Schema

# XML Attributes versus Elements

---

Date Element Used Here



```
<?XML version="1.0"?>
<note>
<date>12/11/99</date>
<to>Paddy</to> <from>Evelyn</from>
<heading>Reminder</heading>
<body>Mow the lawn!</body>
</note>
```

Date Attribute Used Here



```
<?XML version="1.0"?>
<note date="26/09/2002">
<to>Paddy</to> <from>Evelyn</from>
<heading>Reminder</heading>
<body>Mow the lawn!</body>
</note>
```

Expanded Date Element Used Here



```
<?XML version="1.0"?>
<note>
<date>12/11/99
    <day>12</day>
    <month>11</month>
    <year>99</year>
</date>
<to>Paddy</to> <from>Evelyn</from>
<heading>Reminder</heading>
<body>Mow the lawn!</body>
</note>
```

# Well Formed XML

---

A well-formed XML document is a document that conforms to the XML syntax rules:

- must begin with the XML declaration
- must have one unique root element
- all start tags must match end-tags
- XML tags are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- XML entities must be used for special characters



# XML Validation

---

XML documents are validated against a Schema or DTD to ensure they contain valid xml.

Parsers are used to help you validate your XML files

- Document Object Model (DOM) Parser
  - *Parses an XML document into a tree*
- SAX (Simple API for XML) Parser
  - *Communicate parsing events to the user (such as start/end of document)*
- See here for more details on Parsers for XML:
  - [http://www.w3schools.com/xml/xml\\_parser.asp](http://www.w3schools.com/xml/xml_parser.asp)

# XML Schema Definition

---

XML Schema is an XML based alternative to DTD

- XML Schema language is referred to as XML Schema Definition (XSD)

Defines the legal elements and the legal nesting of XML documents.

- An O-O analogy is that Schemas are classes, documents are objects

Provides an agreed syntax around which to build a semantics

XML Schema is superseding the DTD format since its advantages include:

- written in XML
  - support data types
  - support namespaces
  - are extensible to future additions
-

# XML Schema Properties

---

## An XML Schema defines

- elements that can appear in a document
- attributes that can appear in a document
- which elements are child elements
- the order of child elements
- the number of child elements
- whether an element is empty or can include text
- data types for elements and attributes
- default and fixed values for elements and attributes

## XML Schemas are Extensible

- Since they are written in XML

# XSD - The <schema> Element

---

The <schema> element is the root element of every XSD and it may contain attributes

Schema  
Element

```
<?xml version="1.0"?>
<xsd:schema>
...
...
</xsd:schema>
```

Schema  
Attributes

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  targetNamespace="http://www.dsg.cs.tcd.ie"
  xmlns=http://www.dsg.cs.tcd.ie
  elementFormDefault="qualified">
...
...
</xsd:schema>
```

# XSD Elements

---

## XSD Simple Elements

- A simple element is an XML element that can contain only text.

```
<xsd:element name="xxx" type="yyy"/>
```

```
<xsd:element name="color" type="xsd:string" default="red"/>
```

## XSD Complex Elements

- A complex element contains other elements and/or attributes
- ◆ We can define a complex element in an XML Schema in *many different ways*
  - See Examples on next 2 slides for different Schemas for the same XML
- There are 4 types of complex elements
  - ☐ *empty elements*
  - ☑ *elements that contain only other elements*
  - ✓ *elements that contain only text*
  - ✓ *elements that contain both other elements and text*

# XSD Complex Empty Element

---

Complex Empty XML element, "product", which contains no content

```
<product prodid="1345" />
```

## XML Schema Definition No. 1

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:attribute name="prodid"
      type="xsd:positiveInteger"/>
  </xsd:complexType>
</xsd:element>
```

## XML Schema Definition No. 2

```
<xsd:element name="product" type="prodtype"/>
<xsd:complexType name="prodtype">
  <xsd:attribute name="prodid"
    type="xsd:positiveInteger"/>
</xsd:complexType>
```

---

# XSD Complex Nested Elements

---

Complex XML element, "employee", which contains only other elements

## XML Schema Definition No. 1

```
<xsd:element name="employee">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstname"
        type="xsd:string"/> <xsd:element
        name="lastname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

## XML Schema Definition No. 2

```
<xsd:element name="lecturer" type="personinfo"/>
<xsd:element name="student" type="personinfo"/>
<xsd:element name="employee">
  <xsd:complexType name = "personinfo">
    <xsd:sequence>
      <xsd:element name="firstname"
        type="xsd:string"/> <xsd:element name="lastname"
        type="xsd:string"/> </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

# XSD Data Types

---

## Common XML Schema Data Types

- `xsd:string`, `xsd:decimal`, `xsd:integer`, `xsd:boolean`, `xsd:date`, `xsd:time`

## Examples

### Schema Declarations:

```
<xsd:element name="customer" type="xsd:string"/>
<xsd:element name="prize" type="xsd:integer"/>
```

### Sample Elements:

```
<customer>John Smith</customer>
<prize>999</prize> or <prize>-999</prize>
```

## The "Date" Type is specified in the following form "CCYY-MM-DD"

### Schema Declaration:

```
<xsd:element name="startdate" type="xsd:dateTime"/>
```

### Sample Element:

```
<startdate>2002-05-30T09:00:00</startdate>
```



# Modularity – XML namespaces

---

Introduce SOAP as a prefix  
for elements/attributes  
within this element

More information on this  
namespace is at this URL

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header> <!-- content --> </Header>
  <SOAP:Body> <!-- content --> </Body>
</SOAP:Envelope>
```

Elements are defined in the Schema's namespace

Web Services middleware understands the SOAP namespace  
and can accept and process any XML document that uses it

# Example XSD

---

`<?XML version="1.0"?>`

`<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

`targetNamespace="http://www.dsg.cs.tcd.ie"`

`xmlns="http://www.dsg.cs.tcd.ie"`

`elementFormDefault="qualified">`

`<xsd:element name="note">`

`<xsd:complexType>`

`<xsd:sequence>`

`<xsd:element name="to" type="xsd:string"/>`

`<xsd:element name="from" type="xsd:string"/>`

`<xsd:element name="heading" type="xsd:string"/>`

`<xsd:element name="body" type="xsd:string"/>`

`</xsd:sequence>`

`</xsd:complexType>`

`</xsd:element>`

`</xsd:schema>`

elements and data types used in this schema  
come from this namespace

elements defined by this schema  
come from this namespace

default namespace declaration

elements used by the XML instance document  
declared in this schema must be namespace qualified

---

# Example XML Document

---

```
<?XML version="1.0"?>  
  
<note xmlns="http://www.dsg.cs.tcd.ie"  
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xsi:schemaLocation=  
        "http://www.dsg.cs.tcd.ie/schema/note.xsd">  
  
  <to>Paddy</to>  
  <from>Evelyn</from>  
  <heading>Reminder</heading>  
  <body>Don't forget to mow the lawn!</body>  
</note>
```

default namespace declaration

schemaLocation attribute used is the one in the XMLSchema-instance namespace

"http://www.dsg.cs.tcd.ie" namespace is defined by note.xsd.

# Schema Design Issues

---

Anonymous type definitions

Attribute Groups

Substitution Groups

- (polymorphism in XML)

Namespaces, Schemas & Qualification

Unique Particle Attribution Constraint

Parsers

- DOM or SAX
  - *Xerces*

Conformance

# XML Analysed

---

"XML – it's a data format, like any other"

...and anyone who says otherwise  
is selling something...

Its significance comes from its adoption as the basis for  
common web-based formats, such as SOAP

All the common features of the web – mark-up, layout,  
hyperlinking, composition, extensibility – are being  
given XML definitions

- Re-engineering what we've got already into a format we can build  
on tomorrow

# XML References

---

W3's site is a good source for the XML standards

- <http://www.w3.org/TR/REC-XML>

## XML Tutorials

- <http://www.w3schools.com/XML/>
- <http://www.w3schools.com/schema/default.asp>

---

---

# Web Services Musketeers

---

XML

SOAP

WSDL

UDDI

Web Services Example



# Simple Object Access Protocol

---

SOAP is a protocol specification that defines a uniform way of passing XML-encoded data.

- All SOAP messages are encoded using XML
- Similar to XML-RPC, XML is SOAP's transfer syntax
  - *Marshalling complexity is high.*
  - *Parsing XML at the server-side is slow.*
  - *Marshalling overhead can degrade performance and scalability significantly.*

Used in combination with a variety of Internet protocols and formats

- such as HTTP, SMTP, and MIME.

Language Independent

Stateless protocol

Supports messaging systems and RPC

# SOAP Messages

---

SOAP messages are carried over standard Internet Protocols such as HTTP, SMTP, and MIME.

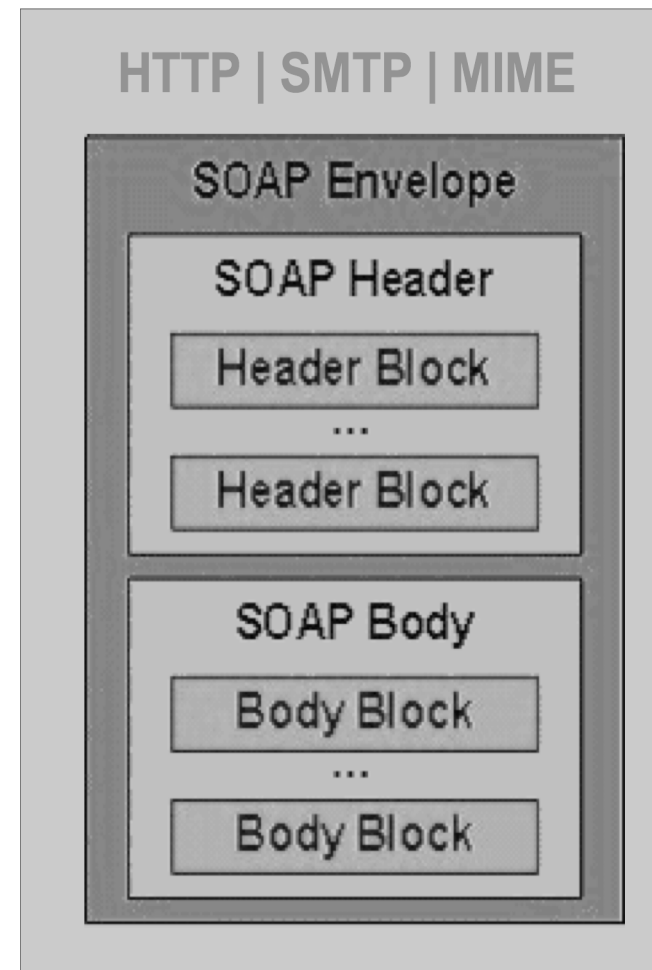
- E.g. inside a HTTP packet

A SOAP message is an XML document

- Often carried in HTTP packets

## SOAP Message Structure

- Consists of 3 Elements
  - **Envelope** [Mandatory]
  - **Header** [Optional]
  - **Body** [Mandatory]



# SOAP Architecture

---

There are three main parts in the SOAP architecture:

- An envelope that describes the contents of a message and how to process it.
  - *Based on an SOAP Envelope XML Schema*
    - <http://schemas.xmlsoap.org/soap/envelope/>
- A set of encoding rules for expressing instances of application-defined data types (marshalling).
  - *Based on SOAP Encoding XML Schema*
    - <http://schemas.xmlsoap.org/soap/encoding/>
- A convention for representing remote procedure calls, responses and faults.
  - *SOAP Body element part of SOAP Envelope Schema*

# Structure of a SOAP Message

---

**envelope** → `<SOAP:Envelope`  
**encoding rules** → `xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"`  
`SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">`  
`<SOAP:Header>`  
`<!-- content of header goes here -->`  
`</SOAP:Header>`  
**body** → `<SOAP:Body>`  
`<!-- content of body goes here -->`  
`</SOAP:Body>`  
`</SOAP:Envelope>`

# Critique of SOAP

---

## Flexibility.

- Server can be CGI, CORBA, EJB, etc

## Simpler Clients.

- Using HTML means less programming.

## Easy to add security, firewall support.

- Use HTTP as transport
- HTTP/SSL for security

## Widely Accepted Standard

## Performance and Scalability.

- Marshalling complexity is high.

# SOAP References

---

## SOAP Specification

- <http://www.w3.org/TR/SOAP/>

## Tutorials on SOAP

- <http://www.w3schools.com/soap/default.asp>

## For information on available SOAP Implementations

- <http://www.soaplite.com>
- <http://www.soapware.org>

---

---

# Web Services Musketeers

---

XML

SOAP

WSDL

UDDI

Web Services Example



# Web Services Description Language

---

WSDL is a template for how services should be described and bound by clients

- WSDL is written in XML and is an XML document
- WSDL is used to describe Web services
  - *Describes a Web service's interface*
- WSDL is also used to locate Web services
  - *provides users with a point of contact (communication end point)*
- Cf. CORBA's Interface Definition Language

WSDL was developed by Ariba, IBM and Microsoft

- WSDL is not a W3C standard yet

# Example WSDL Definition

[Note: Example is missing namespace info. See <http://www.w3.org/TR/wsd> for full version]

---

```
<?xml version="1.0"?>
<definitions name="StockQuote"

<types>
  <element name="TradePriceRequest">
    <complexType> <all>
      <element name="tickerSymbol" type="string"/> </all>
    </complexType>
  </element>
  <element name="TradePrice">
    <complexType> <all>
      <element name="price" type="float"/> </all>
    </complexType>
  </element>
</types>
<message name="GetLastTradePriceInput">
  <part name="body" element="xsd1:TradePriceRequest"/>
</message>
<message name="GetLastTradePriceOutput"> <part name="body"
  element="xsd1:TradePrice"/>
</message>

<portType name="StockQuotePortType">
```

# Example WSDL Definition CTD

[Note: Example is missing namespace info. See <http://www.w3.org/TR/wsd> for full version]

---

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

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

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

---

# The WSDL Schema

---

## The WSDL Schema Defines

- **Types**— a container for data type definitions using some type system (such as XSD)
- **Message**— an abstract, typed definition of the data being communicated.
- **Operation**— an abstract description of an action supported by the service.
- **Port Type**—an abstract set of operations supported by one or more endpoints.
- **Binding**— a concrete protocol and data format specification for a particular port type.
- **Port**— a single endpoint defined as a combination of a binding and a network address.
- **Service**— a collection of related endpoints.

# Separating Applications from Protocols

---

WSDL separates application-level service functionality from access protocol details

- allows the reuse of the following abstract definitions: **messages**, **port types** (abstract collections of **operations**),
- Concrete protocol and data format specifications for a particular port type constitutes a reusable **binding**

# wsdl:definitions

---

A WSDL document is simply a **set of definitions**.

- There is a **definitions** element at the root, and definitions inside.

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri">
  <import namespace="uri" location="uri"/>*
  <wsdl:documentation .... /> ?
  ...
  ...
</wsdl:definitions>
```

# wsdl:types

# wsdl:message

---

A **types** element encloses data type definitions that are relevant for exchanged messages:

```
<definitions .... >
  <types> <xsd:schema .... />*
</types>
</definitions>
```

A **message** element consists of one or more logical **parts**

- Each part is associated with a type
- **element:** Refers to an XSD element using a QName.
- **type:** Refers to an XSD simpleType or complexType using a QName.

```
<definitions .... >
  <message name="nmtoken"> *
    <part name="nmtoken" element="qname"? type="qname"?/> *
  </message>
</definitions>
```

# wsdl:portType

---

A port type is a named set of abstract operations and the abstract messages involved:

```
<wsdl:definitions .... >  
  <wsdl:portType name="nmtoken">  
    <wsdl:operation name="nmtoken" .... /> * </wsdl:portType>  
  </wsdl:definitions>
```

WSDL has four operation types that an endpoint can support:

- **One-way.** The endpoint receives a message.
- **Request-response.** The endpoint receives a message, and sends a correlated message.
- **Solicit-response.** The endpoint sends a message, and receives a correlated message.
- **Notification.** The endpoint sends a message.



# wsdl:operation

---

## Grammar for a request-response operation:

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

For the 3 other operation types, see  
<http://www.w3.org/TR/wsdl>

# wsdl:service    wsdl:port

---

A service groups a set of related ports together:

```
<wsdl:definitions .... >
  <wsdl:service name="nmtoken"> *
    <wsdl:port .... />*
  </wsdl:service>
</wsdl:definitions>
```

A port defines an individual endpoint by specifying a single address for a binding:

```
<wsdl:definitions .... >
  <wsdl:service .... > *
    <wsdl:port name="nmtoken" binding="qname"> *
      <!-- extensibility element (1) -->
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

# wsdl:binding

---

The binding element tells how a given interaction occurs over the specified protocol

## Concrete Binding Information

- what communication protocol to use (such as SOAP over HTTP),
- how to accomplish individual service interactions over this protocol, and
- where to terminate communication (the network address).

# wsdl:binding

---

## Grammar for a binding:

```
<wsdl:definitions .... >
  <wsdl:binding name="nmtoken" type="qname"> *
    <!-- extensibility element (1) --> *
    <wsdl:operation name="nmtoken"> *
      <!-- extensibility element (2) --> *
      <wsdl:input name="nmtoken"? > ?
        <!-- extensibility element (3) -->
      </wsdl:input>
      <wsdl:output name="nmtoken"? > ?
        <!-- extensibility element (4) --> *
      </wsdl:output>
      <wsdl:fault name="nmtoken"> *
      <!-- extensibility element (5) --> *
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
</wsdl:definitions>
```

# wsdl:binding Extensions

---

WSDL provides binding extensions for the following protocols and message formats:

- SOAP 1.1
  - *SOAP Binding Schema*
- HTTP GET / POST
  - *HTTP Binding Schema*
- MIME
  - *MIME Binding Schema*

# WSDL References

---

## WSDL Tutorial

- <http://www.w3schools.com/wSDL/default.asp>

## WSDL 1.1 specification

- <http://www.w3.org/TR/wSDL.html>

---

---

# Web Services Musketeers

---

XML

SOAP

WSDL

UDDI

Web Services Example



# Universal Description Discovery & Integration

---

## Universal Description Discovery and Integration

- Specification for Web Service Information Registries
- Stores WSDL files
  - *UDDI is a directory of web service interfaces described by WSDL*
- Search engine for Web Services
- UDDI communicates via SOAP
- Cf. CORBA's Naming Service or Java's JNDI

# UDDI Possibilities

---

- Making it possible to discover the right business from the millions currently online
- Defining how to enable commerce once the preferred business is discovered
- Reaching new customers and increasing access to current customers
- Expanding offerings and extending market reach
- Solving customer-driven need to remove barriers to allow for rapid participation in the global Internet economy
- Describing services and business processes programmatically in a single, open, and secure environment

Will it solve any chronic technical distributed systems problems?

---

# Business Forces at Work (1)

---

## Manufacturing Demand

- Predictions for growth of web services

But....

- Customers are wary
  - *predictions for growth of e-commerce versus actual growth of e-commerce*

## Planned Technology Depreciation

- Depreciate your own technology before your competitors do

But....

- The lifetime of software products is extending

# Business Forces at Work (2)

---

## Global Marketplace

- Many companies have found out-sourcing primary manufacturing to be cheaper in developing countries

Web Services provides a mechanism for creating more efficient global markets for low tech industries

- Commodities
- Primary Manufacturing


Footwear manufacturers took 10 years to get the price of labour down from 2 dollars/hr to .50 cents/hr [No Logo, Naomi Klein]


- Will Web Services improve (sic) on this? ☹

# UDDI Registry

---

UDDI provides two basic specifications that define a service registry's structure and operation:

 a definition of the information to provide about each service, and how to encode it;

 a query and update API for the registry that describes how this information can be accessed and updated.

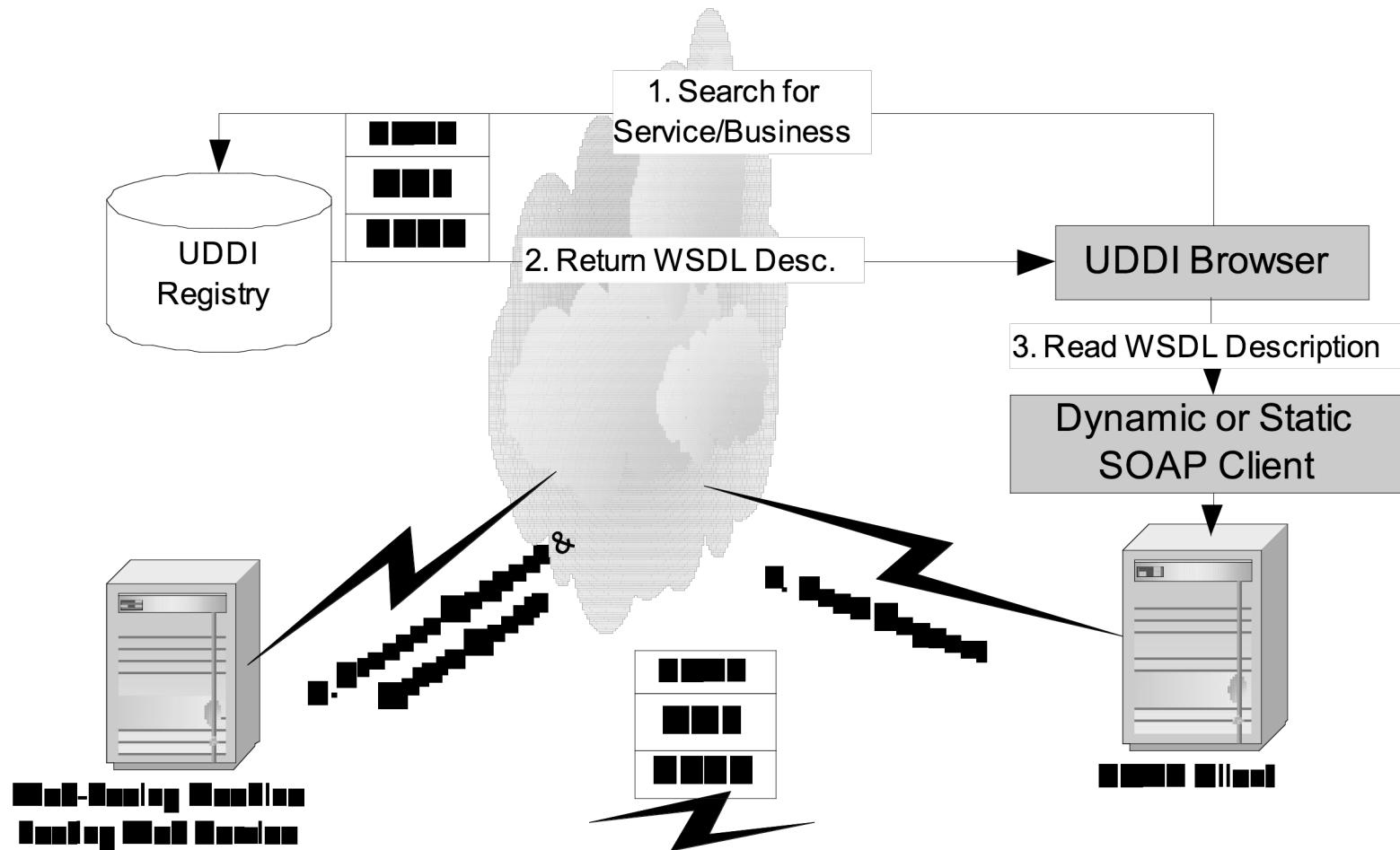
Registry access is accomplished using a standard SOAP API for both querying and updating.

Public registries: Microsoft, IBM and HP.

- UDDI is built into the Microsoft .NET platform

# Service Discovery and Usage with UDDI

---



# UDDI Organising Structure

---

UDDI encodes three types of information about Web services:

- "white pages" information includes name and contact details;
  - *businessEntity XML Element*
- "yellow pages" information provides a categorization based on business and service types;
  - *businessService XML Element*
- "green pages" information includes technical data about the services.
  - *tModel XML Element*

# White Pages Query

---

## UDDI API Query:

```
<find_business generic="1.0" xmlns="urn:uddi-org:api">  
  <name>Acme Travel</name>  
</find_business>
```

## Result:

Detailed listing of <businessInfo> elements currently registered for Acme Travel, which includes information about the UDDI service itself.



# UDDI Returns ... businessEntity Structure

---

```
<businessEntity businessKey="A687FG00-56NM-EFT1-3456-098765432124">
  <name>Acme Travel Incorporated</name>
  <description XML:lang="en">Acme is a world leader in
    online travel services</description>
  <contacts>
    <contact useType="US general">
      <personName>Acme Inc.</personName>
      <phone>1 800 CALL ACME</phone>
      <email useType="">acme@acme-travel.com</email>
      <address>
        <addressLine>Acme</addressLine>
        <addressLine>12 Maple Avenue</addressLine>
        <addressLine>Springfield, CT 06785</addressLine>
      </address>
    </contact>
  </contacts>
  <businessServices> ...
</businessServices>
  <identifierBag> ...
</identifierBag>
  <categoryBag> ...
    <keyedReference tModelKey=
      "UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
      keyName="Electronic check-in"
      keyValue="84121801"/>
  </categoryBag>
</businessEntity>
```

---

# Yellow Pages Query

---

## UDDI API Query:

```
<find_service generic="1.0" xmlns="urn:uddi-org:api">  
  <name>FlightPrices</name>  
</find_service>
```

## Result:

Detailed listing of FlightPrices <service> elements  
currently registered at the UDDI Registry.

# Simplified businessService Structure

---

```
<businessService serviceKey= "894B5100-3AAF-11D5-80DC-002035229C64"
    businessKey= "D2033110-3AAF-11D5-80DC-002035229C64">
    <name>ElectronicTravelService</name>
    <description XML:lang="en">Electronic Travel Service</description>
    <bindingTemplates>
        <bindingTemplate
            bindingKey="6D665B10-3AAF-11D5-80DC-002035229C64"
            serviceKey="89470B40-3AAF-11D5-80DC-002035229C64">
            <description>SOAP-based e-restrictions and flight info
        </description>
            <accessPoint URLType="http">http://www.acme-
                travel.com/travelservice</accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo tModelKey=
                    "D2033110-3BGF-1KJH-234C-09873909802">
                    ...
                </tModelInstanceInfo>
            </tModelInstanceDetails>
        </bindingTemplate>
    </bindingTemplates>
    <categoryBag> ... </categoryBag>
</businessService>
```

---

# UDDI Registry APIs

---

## Finding things Category

find\_business Inquiry

find\_service Inquiry

find\_binding Inquiry

find\_tModel Inquiry

## Getting details about things Category

get\_businessDetail Inquiry

get\_serviceDetail Inquiry

get\_bindingDetail Inquiry

get\_tModelDetail Inquiry

## Saving things Category

save\_business publishing

save\_service publishing

save\_binding publishing

save\_tModel publishing

## Deleting things Category

delete\_business publishing

delete\_service publishing

delete\_binding publishing

delete\_tModel publishing

## Security Category

get\_authToken publishing

discard\_authToken publishing

# UDDI References

---

## Tutorial

- <http://www.learnxmlws.com/tutors/uddi/uddi.aspx>

## Specification

- <http://www.uddi.org/specification.html>

## Provider

- <http://www-3.ibm.com/services/uddi/>

---

---

# Web Services Musketeers

---

XML

SOAP

WSDL

UDDI

Web Services Example

# Change to Google Example

---

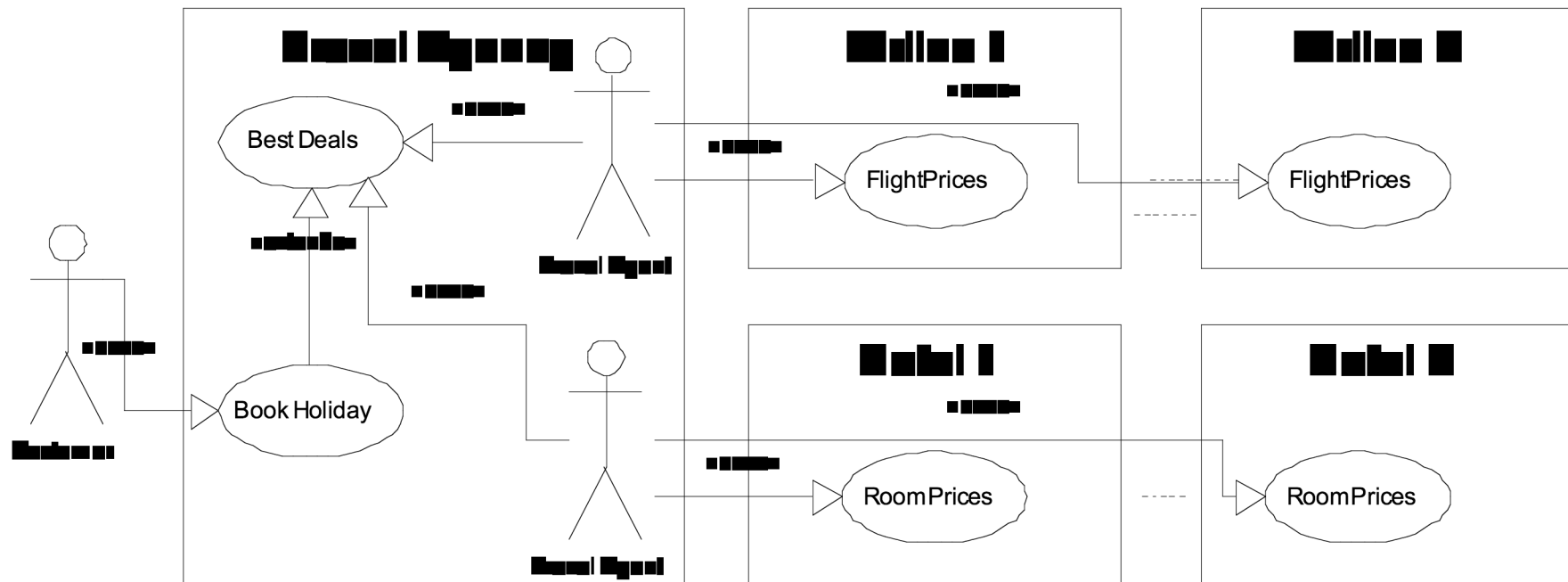
<http://www.cee.hw.ac.uk/courses/5nm1/14/index.htm>



# The Travel Agency Web Service

---

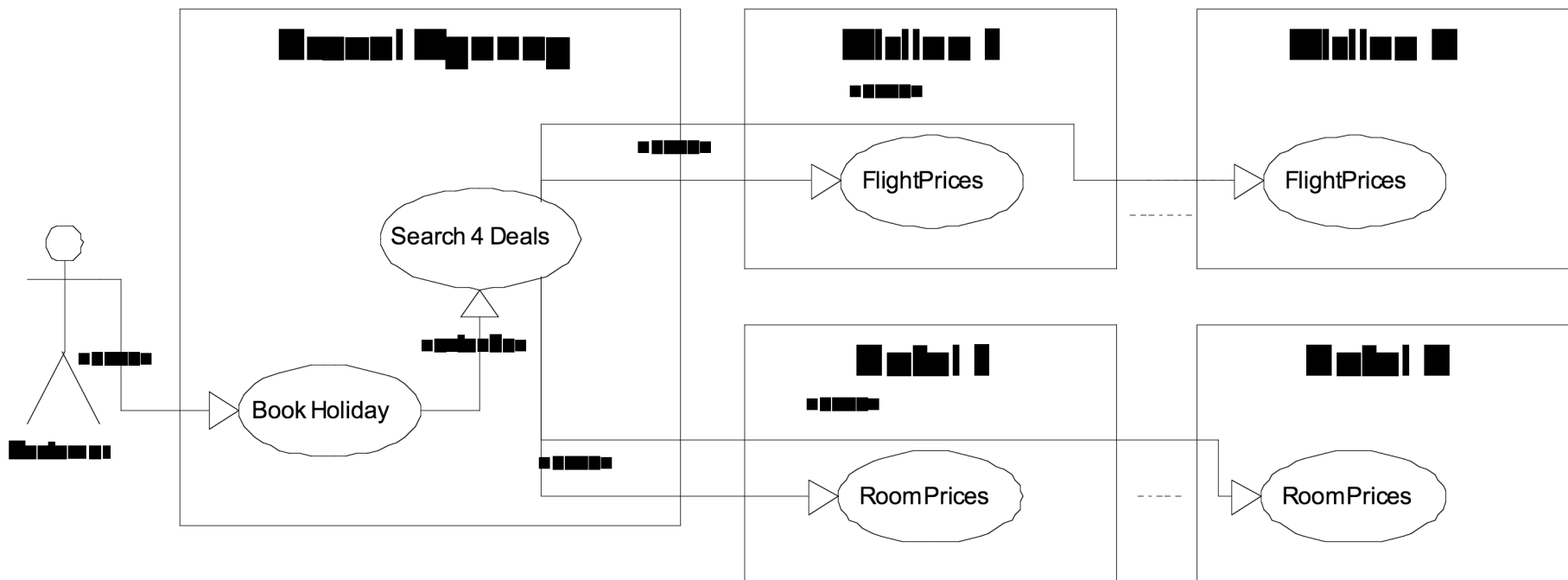
Use Case Diagram for purchasing a holiday package using an online travel agent (year 2002 || year 0 aws.)



# Booking a Holiday with Web Services

---

Use Case Diagram for purchasing a holiday package using an online travel agent that uses Web Services (year 10 aws.)



# Problems with Existing Technologies

---

The agency will have to poll multiple companies

No common service descriptions

- Each company will probably use incompatible applications for pricing and reservations

Incompatible communication protocols

- Each company could expose their services over different Internet protocols (e.g. IIOP, RMI, HTTP)

No support for Service Discovery

- Requirement for a standardized mechanism to locate services

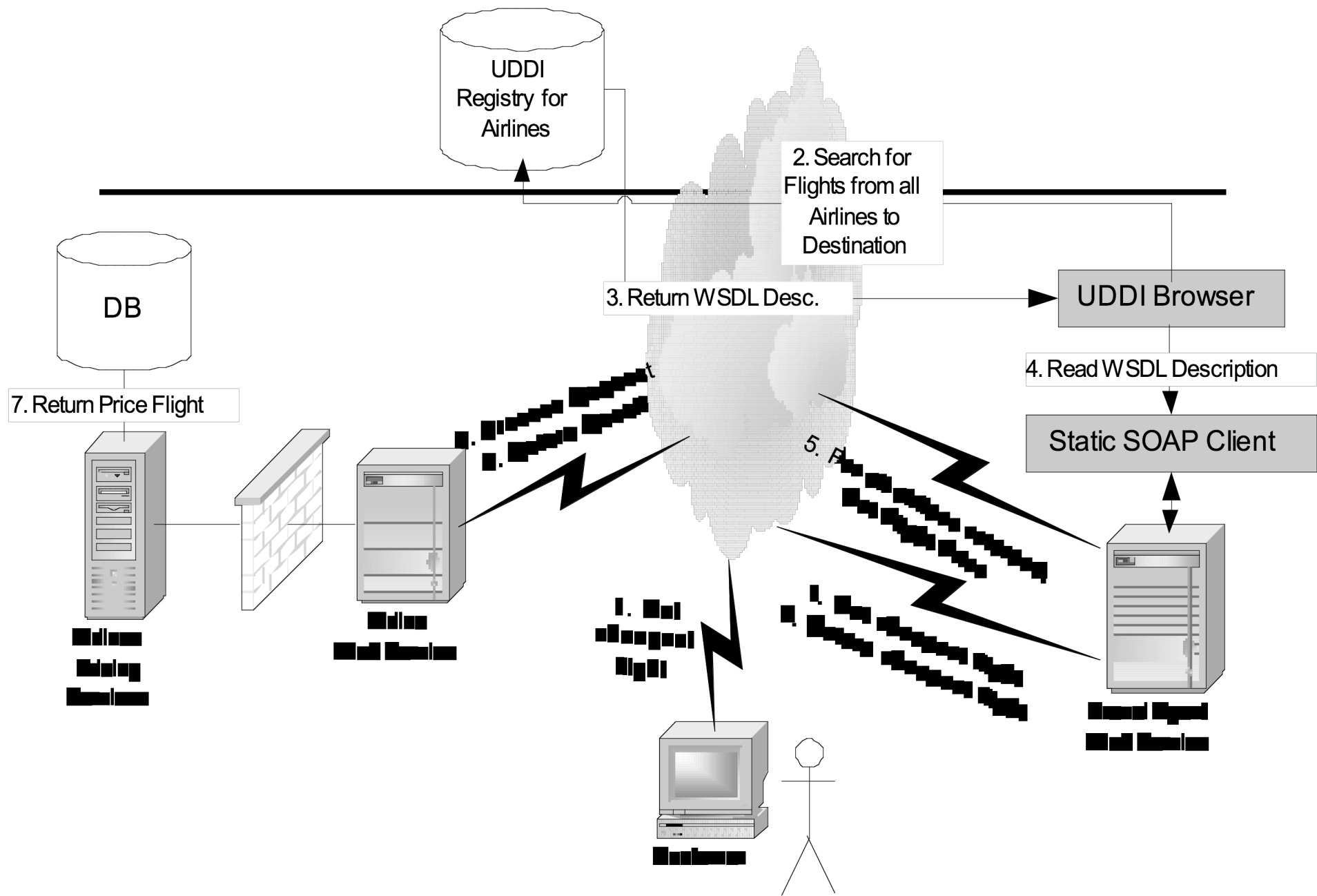
# Web Services Solution

---

Each airline and hotel application becomes an accessible Web service component.

An online travel service could thus use the same Web services framework to locate and reserve your package elements

- E.g. Flights & Hotel Rooms



# Steps 1, 2


---

Step 1. User issues request for cheap flight from HTML page

- CGI/Servlet call inside an HTTP Request Packet

Step 2. Travel Agent receives HTTP packet and invokes query operations on the Airline UDDI Registry:

Query:



```
<find_service generic="1.0" XMLNs="urn:uddi-org:api">  
    <BusinessKey>Airline</BusinessKey>  
    <TModelBag>FlightPrice</TModelBag>  
    <FindQualifiers>FlightInfo</FindQualifiers>  
</find_service>
```

Step 3. Result of Query returned:

Registry returns a detailed listing of <servicelist> elements currently registered for Airline, each containing WSDL description in tModelInstanceDetails. Repeat this procedure for all Airlines.

For Java API Details See [http://www.systinet.com/doc/wasp\\_uddi/api/org/idoxx/uddi/client/api/v1/UDDIProxy.html](http://www.systinet.com/doc/wasp_uddi/api/org/idoxx/uddi/client/api/v1/UDDIProxy.html)

---

## Steps 4

---

Step 4. Travel agent gets the location of the WSDL  
Description for each airline's `flightprice` webservice  
from `tModelInstanceDetails`

## Step 4. Travel Agent knows about tModel keys and gets WSDL Desc from tModel Element returned

---

Airlines have already registered a WSDL document as a tModel document  
with the Airline UDDI Registry.

```
<tModel tModelKey="FlightPrice">
  <name>http://www.travel.org/e-flightprice-
  interface</name>
  <description XML:lang="en"> Standard service interface
    definition for pricing airline flights
  </description>
  <overviewDoc>
    <description XML:lang="en">
      WSDL Service Interface Document
    </description>
  <overviewURL>
    http://www.travel.org/services/e-flightprice.wsdl
  </overviewURL>
  </overviewDoc>
  <categoryBag> ...
  </categoryBag>
</tModel>
```

---



## Step 4. Travel Agent program has generic code to operate on the Abstract WSDL Description

---

```
<message name="GetFlightPriceInput">
    <part name="date" type="xsd:datetime"/>
    <part name="flightNumber" type="xsd:int"/>
</message>
<message name="GetFlightPriceOutput">
    <part name="flightInfo" type="fixsd:FlightInfoType"/>
</message>
<message name="RestrictionsInput">
    <part name="body" element="xsd:Ticket"/>
</message>
<portType name="AirlineServicePortType">
    <operation name="GetFlightPrice">
        <input message="tns:GetFlightPriceInput"/>
        <output message="tns:GetFlightPriceOutput"/>
    </operation>
    <operation name="Restrictions">
        <input message="tns:RestrictionsInput"/>
    </operation>
</portType>
```

---

## ....and uses concrete binding information when binding to concrete Airline WebServices

---

```
<binding name="AirlineServiceSoapBinding" type="tns:AirlineServicePortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetFlightPrice">
    <soap:operation style="rpc"
      soapAction="http://acme-travel/flightinfo"/>
    <input>
      <soap:body use="encoded" namespace="http://acme-travel.com/flightinfo"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
      <soap:body use="encoded"
        namespace="http://acme-travel.com/flightinfo"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>
  <operation name="Restrictions">
    <soap:operation style="document"
      soapAction="http://acme-travel.com/restrictions"/>
    <input>
      <soap:body use="literal"/>
    </input>
  </operation>
</binding> <service name="travelservice">
  <port name="travelservicePort" binding="tns:AirlineServiceSoapBinding">
    <soap:address location="http://acmetravel.com/travelservice"/>
  </port>
</service>
```

---

## Step 5

---

Step 5. Travel agent uses previous WSDL description to issue a SOAP request for the flight's price to the Airline's Web Service

## Step 5. SOAP Message Embedded in HTTP Post

---

```
POST /travelservice HTTP/1.1
Host: www.jimstravelservice.com
Content-Type: text/XML; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://www.acme-travel.com/flightinfo"

<SOAP:Envelope XMLns:SOAP=
  http://schemas.xmlsoap.org/soap/envelope/
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Body>
    <m:GetFlightPrice
      XMLns:m="http://www.acme-travel.com/flightinfo"
      SOAP:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"
      XMLns:xsd="http://www.w3.org/2001/XMLSchema"
      XMLns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <airlineName xsi:type="xsd:string">UL</airlineName>
        <flightNumber xsi:type="xsd:int">506</flightNumber>
      </m:GetFlightPrice>
    </SOAP:Body>
  </SOAP:Envelope>
```

---

## Steps 6-8

---

Step 6. Airline's Web Service accepts request and forwards it through the firewall to backend systems for processing

Step 7. Airline's backend systems get the flight's price from the DB and return result to the Airline's Web Service through the firewall

Step 8. Airline's web service returns the flight's price to the travel agent

## Step 8. SOAP Message Embedded in HTTP Response

---

```
HTTP/1.1 200 OK
Content-Type: text/XML; charset="utf-8"
Content-Length: nnnn
<SOAP:Envelope xmlns:SOAP=
http://schemas.xmlsoap.org/soap/envelope/
SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Body>
    <m:GetFlightPriceResponse
      xmlns:m="http://www.acme-travel.com/flightinfo"
      SOAP:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi=
        "http://www.w3.org/2001/XMLSchema-instance">
      <flightInfo>
        <price xsi:type="xsd:int">1000</price>
        <status xsi:type="xsd:string">Economy
        </status>
      </flightInfo>
    </m:GetFlightPriceResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

---

## Step 9

---

Step 9. The travel agent enumerates all flight prices and returns them to the user who can then select the desired flight in HTML form

Phew ! 😊

---

---