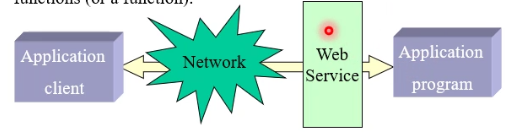
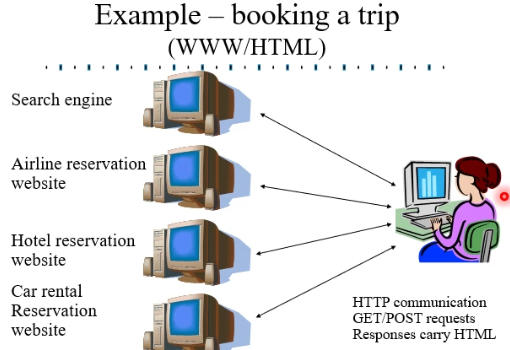
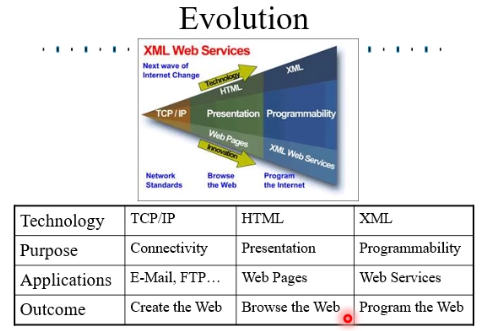
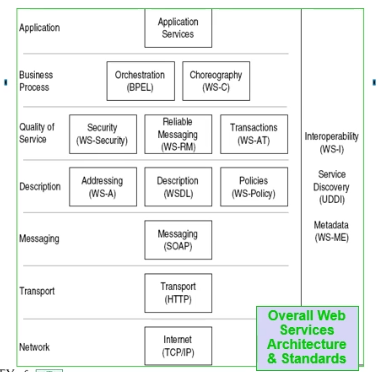
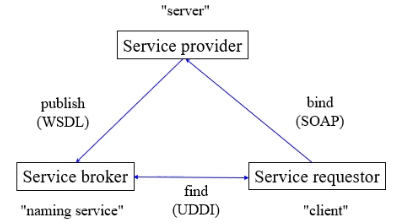
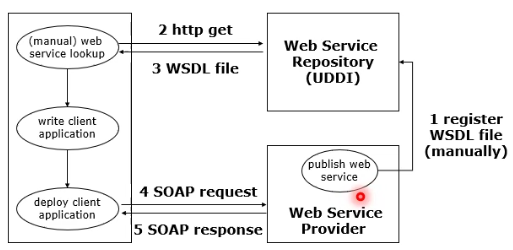
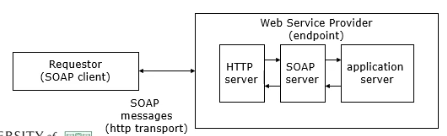
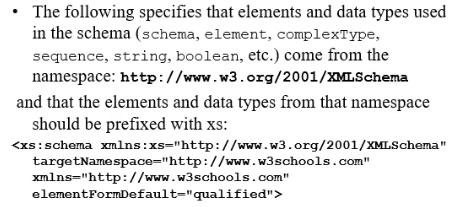
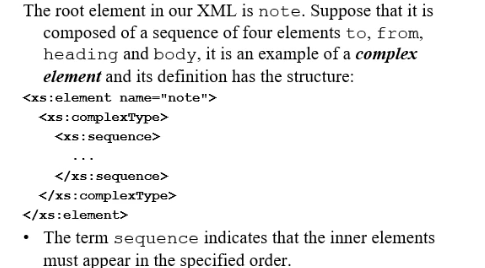
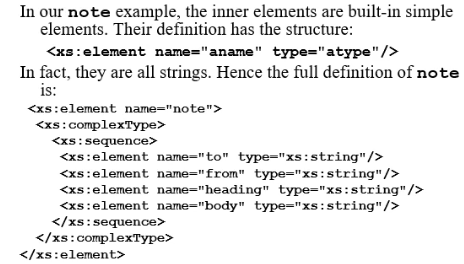
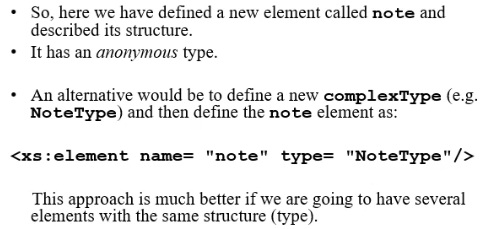
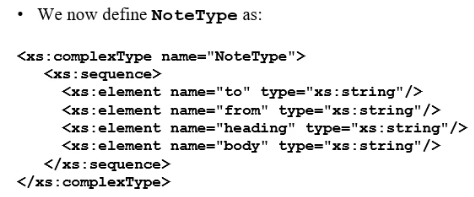
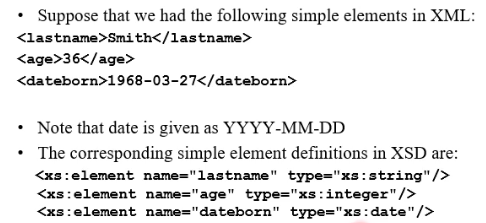
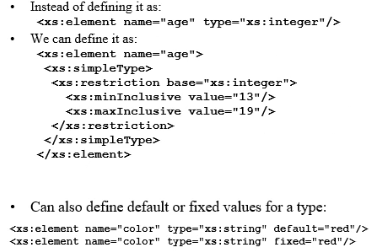
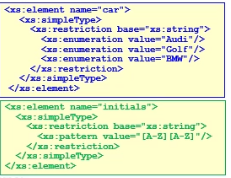
# Web Service Introduction

## Overview

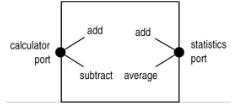
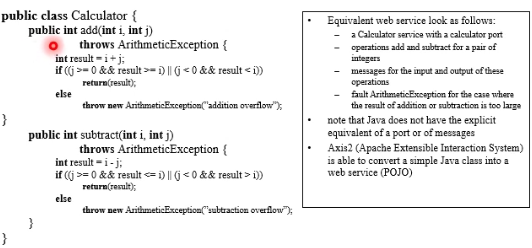
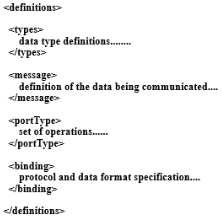
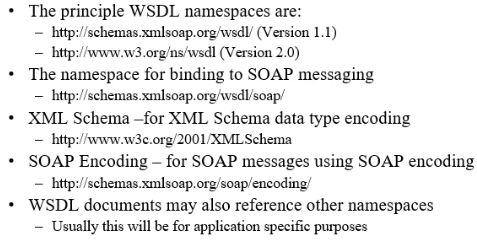
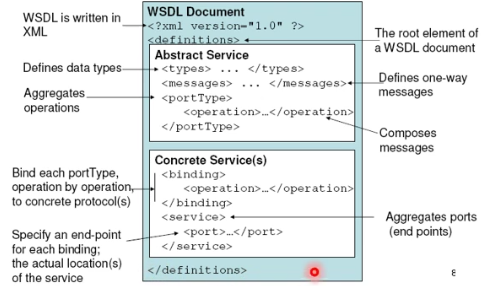
* Origins
  + Web services focus on B2B communication (Business to Business)
* Characteristics
  + Communication among apps rather than users
  + Follow open standards supported by industry
  + Architecture loosely coupled – WS can be designed in isolation and can interwork even if not explicitly designed to do so
  + Supported by 3 classes: service consumers (clients), service providers (servers) and service brokers (registries)
* What is a Web Service
  + Network accessible interface to application programs, built using standard Internet technologies
  + Clients of web services do not need to know how it is implemented
  + Web service is a URL-addressable software resource that performs functions (or a function)
  + 
  + 
  + 
* Usage
  + Web services are used by business for managing partnerships
    - Supply chains
    - Outsourcing
    - Contracting
    - Combined services
  + Web services support virtual organisations across the boundaries of conventional organisations
* Web services standards
  + Standards for Web services defined by
    - IETF
    - OASIS
    - W3C
  + Web Services share communication mechanisms with conventional use of the web
    - HTTP – to support message exchange between web services
    - TCP – for reliable transfer of data across collection of subnetworks
    - IP – for routing across a collection of subnetworks
  + Many web service aspects have been standardised in areas such as
    - Service description
    - Service Discovery
    - Service Addressing
    - Security and authentication
    - Reliable messaging and transaction
    - Service orchestration and choreography
    - Service policies
  + 
* Web Service Usage
  + 
  + 
* Web Services Implementation
  + Application server (Web service-enabled)
    - Provides implementation of services and exposes it through WSDL/SOAP
    - Implementation in Java, as EJB, as /NET (C#) etc
  + SOAP server, implements the SOAP protocol
  + HTTP server, standard web server
  + SOAP client, implements the SOAP protocol on the client side
  + 
* SOA/SOC
  + Web services support what is generally called SOA (Service Oriented Architecture) in support of SOC (Service Oriented Computing)
  + SOC views a distributed system from the perspective of the services it offers and how these relate to each other
  + Previous work on distributed computing was object oriented, describing the (close) coupling among distributed objects
  + Instead, web services maintain a loose coupling – only services offered by a distributed application are exposed
    - Legacy apps can easily be given a web service wrapping
    - The internal design of a web service can readily be changed
    - New services can be created by combining existing services

## XML Key Concepts

* XML
  + Meta-language
  + Uses markup to describe data
    - Used to develop our own markup languages
  + Text files
  + XML and HTML are for diff. purposes
    - HTML is concerned with display (e.g. <body>, <p>)
    - XML concerned with data representation
  + The markup facilities in HTML are predefined
  + In XML we define our own
  + 
* Background XML cont
  + XML attributes further define elements
  + #Required vs #Implied
  + Well formed vs valid XML
  + Two ways to specify the structure of XML documents
    - Document Type Definitions (DTDs)
    - XML schemas
  + Need to specify
    - What elements (tags) will be used
    - How the various elements may be nested
    - What attributes they may contain
    - What types of data an element can contain
* DTD vs Schemas
  + DTDs rather limited in how they describe the content of data
  + XSD (XML Schema Definition) is now the preferred way to define applications of XML, giving:
    - The structural elements in the data, defining the types of data they contain
    - Definitions of data types, including sophisticated constraints on their contents
    - The relationships among the structural elements
    - The attributes of elements
  + XSI (XML Schema Instance) defines XML documents as instances of their schemas
* Namespaces
  + Essentially just unique string
  + Usually take form of URI
  + Namespace URI is typically URL for where schema is defined
  + Namespace URI may simply be a URN that gives a relatively unique identifier
  + Namespace URI often lenghtly, commonly referred to by a short prefix, a string that is unique within a document
* Defining Namespace
  + Namespace prefixes defined and used asfollows (xmlns means XML NameSpace)
    - 
  + mstd:sequence is an example of using a namespace prefix
  + Documents may declare a default namespace (xmlns on its own) for elements and attributes that are used without an explicit namespace prefix
  + Documents may also declare a target namespace (targetNamespace) that applies to all elements and attributes it defines
  + Common to have prefix corresponding to target namespace; typically but need not be, named tns
  + Namespace prefix for XML Schema Definition is usually xsd (though sometimes xs, while that for XML Schema Instance usually xsi)
* XML Schema
  + When the XML file is specified by an XSD document held in note.xsd, the attributes within the note element in the XML file are:
  + 
  + Notation is therefore in XML
  + Also have defined namespaces
  + Default namespace in this example is <http://www.w3schools.com>
  + The <schema> element is the root element of every XML Schema and it normally has attributes
  + 
  + Elements used by an XML document which are declared in this schema must be namespace **qualified**
* Example
  + 
  + 
  + 
  + Built in data types prefixed with **xs:**
* A new type
  + 
* XML Schema Datatypes
  + XML Schema have a lot of built-in data types – examples are (all prefaced with xs: )
    - String
    - Decimal
    - Integer
    - Boolean
    - Date
    - Time
  + Gives us a lot more control than with DTDs to specify what can go into our XML document
  + If XML document contains a value of the wrong type then it will not validate
* Example datatypes
  + 
* Another new type
  + Can also define a new simpleType
  + Start from existing simpleType (the base type) and impose restriction by means of a facet
  + Example facets are:
    - maxInclusive and minExclusive
    - minInclusive and minExclusive
    - pattern
    - enumeration
* Restrictions on types
  + Suppose we want to restrict the range of allowable values in our age element
  + 
  + 

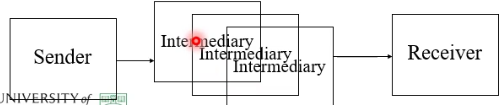
# Web Service Description Language (WSDL)

## WSDL

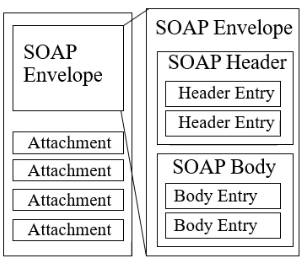
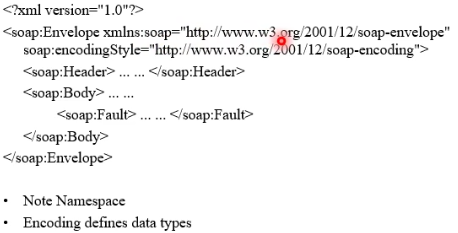
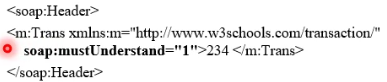
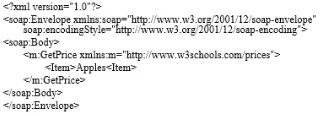
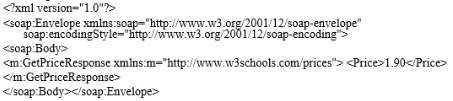
* Used to describe web services, that is the interface to web apps
* Web services
  + A **service** supports
    - **Ports** whose interfaces are defined by **port types**
    - **Operations** at ports that can take an input, return an output, and cause a fault
    - **Messages** that are sent to or by an operation
    - **Faults** that indicate failure of the service (not the underlying communications)
    - 
  + Note that
    - Diff ops with same name may be supported at diff ports
    - Op parameters are optional, e.g. op may not produce output, may send output without input, may not cause a fault
    - Although web service may offer multiple ports, each w/ multiple ops, in practice services have just one port w/ multiple ops
* Calculator class – example
  + 
* WSDL
  + Web Service Definition Language (XML)
  + Namespace typically wsdl
  + A WSDL document describes how to interact with a web service in terms of **data** **types**, **operations** provided and their **parameters**, **protocols** used, **location** of the service
  + WSDL deals w/ **syntax**(how to call ops) and not **semantics**(what ops do), so other info needed before service can be fully understood
  + **Contract** between service provider and requestor
  + Described services can be implemented in **any language** & on **any platform**
* Uses
  + Used by app devs as spec of the web service
    - Helps w/ dev of both web services and web service clients
    - Source code for (parts of) service & client can be generated from WSDL
    - WSDL can also be generated from web service implementation
  + Used by apps to invoke web service
    - Dynamically generating call to the web service based on its description
  + Published in service registries
    - Aids discovery & use of web services
  + WSDL-described web service can be communicated w/ using any agreed protocol
    - SOAP (most common)
    - SMTP/MIME
    - HTTP/REST (used for simple cases)
* WSDL messages
  + Web services handle messages in one of two basic styles
    - Document style means each message carries an XML doc
    - Rpc style (cf. remote procedrure) means request message carries the name of the op to be invoked plus its parameters, and resulting response message carries op result
  + Each of these has two encodings
    - Encoded – types of all values explicitly stated
    - Literal – values just given literally, types implicit
  + In practice, only literal styles used
  + Disadvantage of doc/literal over rpc/literal – ops cannot be overloaded (different ops cannot have the same parmeters)
* WSDL documents
  + WSDL separates the abstract description of a service interface from how it is actually supported
  + Abstract Service Interface Definition
    - Describes what web service does (what ops it offers)
  + Concrete Service Implementation
    - Binds abstract ops to concrete protocols – how to call those ops using those protocols
  + Services support one or more ports (typically just one)
  + Each port supports one or more ops
  + Each op may have an input, output, and zero or more faults
* Abstract Service Definition
  + Data types used in service
    - Typically XML Schema type definitions
    - Simple types, e.g. int, float, string, Boolean
    - Complex types, e.g. Customer, Address, Stock Item
    - Types used within messages
  + Messages sent/received by service
    - Message is the payload of a single, one-way communication
    - Message consists of one or more parts
    - Each part is of a certain data type (as defined in data types)
    - Messages used to make operations
  + How messages combine to form ops
    - At most one input message (input parameters)
    - At most one output message (output parameters)
    - Optional fault descriptions (Exceptions)
  + Ops combine to form portType
* Port types
  + Describes interface(s) of a web service
  + Represent logical aggregation of operations
  + 
* Concrete Service Implementation
  + Concrete bindings of the abstract service interface definition
    - Describe an implementation of a portType
    - Input, output, fault messages in the ops of each port type mapped to:
      * Transport protocol(s) used
      * Message style (document or rpc)
      * Data encoding style (encoded or literal)
    - Although binding info has to be repeated for each port and op parameter, these are usually all the same
  + Overall service defined
    - Name of service
    - Each port has name, binding and location
  + Entire web service is exposed via one or more ports (end point)
    - Each binding corresponds to a single port
    - A port is the actual address where the service can be found, e.g:
      * <http://some.web/service> if binding to HTTP
      * [Some.web@service.com](mailto:Some.web@service.com) If binding to SMTP
* Tools for WSDL
  + All packages for web services include support for WSDL
  + Apache Axis2 supports
    - Parsing WSDL, interpreting SOAP messages in context of this
    - WSDL2Java converts from WSDL to Java, creating stubs and skeletons
    - Conversely, Java2WSDL converts from Java to WSDL
    - Basic XML types have direct Java mapping (e.g. Boolean, float, int)
    - More complex XML types map onto Java classes
* WSDL limitations
  + Unable to describe complex business processes
    - E.g. sequences of related messages
  + Does not describe business level requirements of the service
    - E.g. quality of service, security
* Important Namespaces
  + 
* Summary
  + 

# Simple Object Access Protocol (SOAP)

## SOAP Overview

* Simple Object Access Protocol
  + Structured XML message format
  + Protocol for exchanging messages
  + Encoding scheme for representing data types in those messages
  + Uses underlying transport protocol (HHTP, SMTP, etc) through binding
* SOAP
  + Provides platform neutral
    - Message and information exchanging
    - Invocation of remote functionality
  + SOAP enables
    - Distributed applications
    - Business-to-business integration
    - Web services
  + SOAP version 1.2
    - W3C recommendation (standard)
    - From XML Protocol Working Group
    - <http://www.w3.org/TR/soap/>
* Why SOAP
  + Many apps comm using Remote Procedure Calls (RPC) between objects like DCOM and CORBA
  + RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this traffic
  + Better way to communicate between apps is over HTTP because HTTP is supported by all internet browsers and servers – SOAP was created to accomplish this
  + SOAP provides a way to comm between apps running on diff operating systems, with different technologies and programming languages
* SOAP messages
  + Stateless
  + One-way
  + Composable
    - One way
    - Request-response
    - Solicit-response
    - Notification
  + Transferred between SOAP nodes (apps)
* SOAP nodes
  + SOAP sender
    - Generates and sends the message
  + SOAP receiver
    - Ultimately receives and processes the message
    - May generate a SOAP response, message or fault as a result
  + SOAP intermediary
    - Zero or more
    - Receives, processes and resends the message
    - 
* SOAP intermediaries
  + Forwarding intermediaries
    - Uses and updates the SOAP header blocks to pass the message (body unchanged) on to the next node
  + Active intermediaries
    - Perform additional processing on the soap message before sending
* SOAP binding to Transport Protocol
  + SOAP messages can be sent in many diff ways
    - Over HTTP
    - Over HTTP/SSL
    - Over SMTP
  + A binding specifies how SOAP messages are passed using an underlying transport protocol

## SOAP Messages

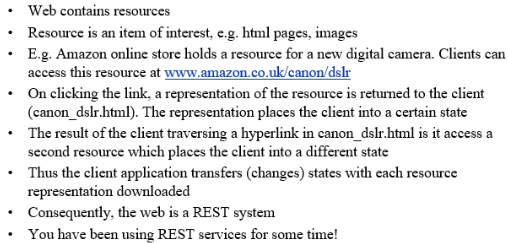
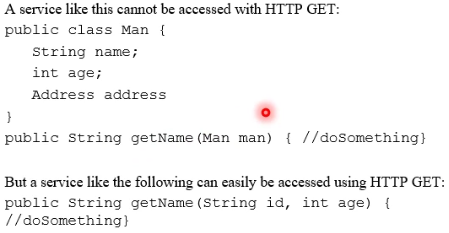
* SOAP message structure
  + 
  + Envelope
    - Identifies that this is a SOAP message
  + Header
    - Optional & app specific
    - Entries may be addressed to a particular SOAP node
  + Body
    - Mandatory
    - Contains message “payload”
  + Additional components:
    - Faults
      * Details of what and where something went wrong
    - Attachments
      * E.g. Binary Data (GIF, JPEG, MP3, etc)
      * Typically carried outside envelope
      * Uses multipurpose internet mail extensions (MIME)
* SOAP message example
  + 
* SOAP Header
  + Attributes
    - Soap:mustUnderstand
      * 
* SOAP Body
  + Request
    - 
  + Response
    - 
* SOAP Faults
  + Fault elements consists of:
    - <faultcode> - A code for identifying the fault
    - <faultstring> - A human readable explanation of the fault
    - <faultactor> - Information on who caused the fault to happen
    - <detail> Application specific error info related to the Body element
  + Fault codes
    - VersionMismatch - Found invalid namespace for the SOAP element
    - MustUnderstand – An immediate child element of the Header element, with the mustUnderstand attribute set to “1”, was not understood
    - Client: message incorrectly formed of contained incorrect info
    - Server: problem with the server: message could not proceed

## SOAP Communication Models

* Communication
  + SOAP provides two comm models
    - SOAP RPC
      * Synchronous request-response
      * Request encodes method & arguments
      * Response encodes result value or fault
    - SOAP Messaging (document)
      * Document-driven: XML
      * Normal XML description e.g. of products can be sent
      * No reference to operation names
      * Operations must have a single element
* SOAP RPC
  + Request body describes
    - Name of the method to invoke
    - Optional arguments to pass to that method
  + Includes the WSDL operation
  + Parameters are based on WSDL types
  + WSDL operation can include one or more parts
  + May be identified by order and/or by name
  + Response body describes
    - Return value(s) from the method or SOAP fault
* SOAP Message (Document)
  + Each message body is an XML document or “literal XML”
    - Can be validated against pre-defined XML schema document
    - A body element type typically identifies the message type
    - Therefore how/by what it should be handled
  + No operation name in SOAP message
  + Parts of a message are based on schema element definitions rather than WSDL types
  + Operations have a single part
* RPC vs Message
  + RPC is function-centric
    - RPC has tight coupling between the message and the implementation
  + Messaging is data-centric
    - Messaging has loose coupling between the message and the implementation
* Data encoding
  + Literals: XML fragments, defined in XML schema
    - Commonly used in XML messaging scenarios
  + Encoded values: defined in SOAP encoding
    - Set of rules for representing data types (not supported in Axis2)
  + Defines standard XML encoding for commonly observed programming language types
    - Simple types, enumerations
    - Compound types, e.g. structs, objects
    - Arrays, References

# RESTful Web Services

## REST

* RESTful services
  + All the web services so far
    - Use SOAP for comm w/ service
    - Have service interfaces defined in WSDL
  + Some developers believe that the overheads and complexity of this kind of web service are undesirable
  + REST (Representational State Transfer) has been introduced more closely aligned with the original design of the web and is now supported by the major vendors and frameworks
  + REST based on the idea of resources and changing state depending on representations of resources being accessed
  + 
  + Web servers support four basic methods collectively known as CRUD:
    - Create – supported by POST in HTTP
    - Read – supported by GET in HTTP
    - Update – supported by PUT in HTTP
    - Delete – supported by DELETE in HTTP
  + Of these, GET (used in ordinary web browsing) and POST (often used w/ forms) are the best-known
  + The four HTTP operations are sufficient to implement a wide variety of web services
* More on RESTful Services
  + Key idea in REST is that a URI should identify a resource on which operaions can be performed, e.g:
    - An online shop has stocks considered as a resource; the whole stocks might be identified by the URI [www.shop.com/stocks](http://www.shop.com/stocks)
    - Each product is itself a resource within the overall stocks; one product might be identified by the URI [www.shop.com/stocks/product4059](http://www.shop.com/stocks/product4059)
  + HTTP methods can then be used to create, read, update or delete the whole stocks or an individual product in stock
  + Like other web services, RESTful services normally send and receive XML documents (though this is not essential)
  + RESTful services are best suited for database type systems
  + Simple databases can be transmitted as part of the request URI (GET requests)
  + Complex datatypes can be transmitted as message payload (XML), but cannot be accessed with the GET message
* RESTful services – parameters
  + 
* RESTful services
  + There is a general move away from SOAP to REST, however, SOAP has some advantages related to security and messaging
  + Several toolsets support REST, e.g:
    - Axis2 is widely used tool for supporting both REST and SOAP messaging
    - JDK has packages that support XML in general and REST in particular
    - JAX-WS defines an API for Java programs to use REST, also offers a REST-compatible application server called GlassFish
    - Restlet provides Java support for developing RESTful services
    - Spring io provides for an API for REST services

# Web Service Security

## Overview

* Requirements for web service security
  + Security reqs are diverse, depending on the purpose of the web service application
    - Maybe not required at all
    - Full security for commercial operations
  + Needs to be open and extensible
  + Needs to be interoperable and work between different organisations
  + Should leverage existing standards
* Security for SOAP WS
  + Family of related standards has been developed to address three major issues
  + WS-Security (Web Services Security)
    - Provides secure transfer over SOAP
  + WS-Trust (Web Services Trust)
    - Allows organisations to validate each other, and to develop chains of trust
  + WS-SecureConversation (Web Services Secure Conversation)
    - Uses security tokens to allow efficient encryption of messages
* Security for WS
  + A widely used approach is based on PKI (Public Key Infrastructure)
  + Encryption and decryption use inverse algorithms that make encryption w/ public key very easy, but decryption w/out the private key very difficult
    - Receiver publishes public key for messages it should receive
    - Sender uses this key to encrypt messages
    - Receiver also has a private key that it uses to efficiently decrypt messages
    - Although the public and private keys are related, should be impracticable to determine the private key from the public key
  + Digital certificates are widely used to establish the authenticity of a party:
    - CA (Certificate Authority) issues parties with digital certificates that guarantee they are who they claim to be
    - For large-scale use, certs are signed by chains of Cas starting from a root CA that everyone trusts (e.g. Thawte, Verisign)
    - Chain of trust might be Verisign, JISC (Joint Information Services Committee), Uni of Stirling,, M. Kolberg
    - HTTPS, HTTP over SSL (Secure Sockets Layer), employs digital certs for secure use of the web

## Case Study