Introduction to SOAP
(Simple Object Access Protocol)
What is SOAP?

- SOAP is a simple, lightweight XML protocol for exchanging structured and typed information on the Web
- Overall design goal: KISS (keep it simple and stupid)
  - Can be implemented in a weekend
  - Stick to absolutely minimum of functionality
- Make it Modular and Extensible
  - No application semantics and no transport semantics
  - Think “XML datagram”
SOAP Contains Four Parts:

- An extensible envelope expressing *(mandatory)*
  - *what* features and services are represented in a message;
  - *who* should deal with them,
  - *whether* they are optional or mandatory.
- A set of encoding rules for data *(optional)*
  - Exchange instances of application-defined data types and directed graphs
  - Uniform model for serializing abstract data models that can not directly be expressed in XML schema
- A Convention for representation RPC *(optional)*
  - How to make calls and responses
- A protocol binding to [HTTP](http) and [HTTP-EF](http) *(optional)*
SOAP Example in HTTP

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

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP:Header>
        <t:Transaction xmlns:t="some-URI" SOAP:mustUnderstand="1">
            5
        </t:Transaction>
    </SOAP:Header>
    <SOAP:Body>
        <m:Deposit xmlns:m="Some-URI">
            <m:amount>200</m:amount>
        </m:Deposit>
    </SOAP:Body>
</SOAP:Envelope>
```
SOAP Example in SIP (session initial protocol)

```xml
SERVICE sip:broker.ubiquity.net SIP/2.0
To: sip:broker.ubiquity.net
From: sip:proxy.ubiquity.net
Call-ID:648324@193.195.52.30
CSeq: 1 SERVICE
Via: SIP/2.0/UDP proxy.ubiquity.net
Content-Type: text/xml
Content-Length: 381

<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope"
    SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP:Body>
        <m:SetCreditStatus xmlns:m="http://www.ubiquity.net/sipservices">
            <m:user>sip:jo@ubiquity.net</m:user>
            <m:status>super</m:status>
        </m:SetCreditStatus>
    </SOAP:Body>
</SOAP:Envelope>
```
... or SOAP by Itself...

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope"
    SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP:Header>
        <m:MessageInfo xmlns:m="http://www.info.org/soap/message">
            <m:to href="mailto:you@your.com"/>
            <m:from href="mailto:me@my.com"/>
            <m:contact href="mailto:someone@my.com"/>
        </m:MessageInfo>
    </SOAP:Header>
    <SOAP:Body>
        <msg:Message xmlns:m="http://www.info.org/soap/message">
            <msg:subject>Your house is on fire!</msg:subject>
            <msg:feed href="ram://livenews.com/yourhouse"/>
        </msg:Message>
    </SOAP:Body>
</SOAP:Envelope>
```
SOAP Stack Examples

SOAP
Protocol Binding
HTTP
TCP

SOAP
Protocol Binding
SIP
TCP

SOAP
Protocol Binding
SIP
UDP

SOAP
Protocol Binding
MIME Multipart
SMTP
TCP

SOAP
Protocol Binding
SMTP
TCP

SOAP
Protocol Binding
UDP
Note Again: SOAP is a Protocol!

- What does this mean?
  - It is *not* a distributed object system
  - It is *not* an RPC system
  - It is *not even* a Web application
- Your application decides what your application is!
  - You can build a tightly coupled system
    ...or...
  - You can build a loosely coupled system
- Tunneling is a property of the application, not the protocol
SOAP is Designed for Evolvability

- How are features and services deployed in the Web?
  - Often by extending existing applications
  - Spreading from in the small to the large over time
- This means that:
  - Applications have different capabilities at all times
  - We have to support this
- This requires that:
  - Applications supporting a particular feature or service should be able to employ this with no prior agreement;
  - Applications can require that the other party either understand and abide by the new feature or service or abort the operation
Why Not Roll My Own XML Protocol?

- SOAP allows you to define your particular feature or service in such a way that it can co-exist with other features and services within a SOAP message.
- What is a feature or a service?
  - Authentication service
  - Payment service
  - Security service
  - Transaction management service
  - Privacy service
- Not owning the message means easier deployment and better interoperability.
What is Interoperability?

- Interoperability is the intersection of features and service supported by two or more communicating peers.
Extensibility vs. Evolvability

- Extensibility: Cost per new feature/service increases over time
  - Feature set 1
  - Feature set 2
  - Feature set 3

- Evolvability: Cost per new feature/service is flat
  - Feature set 1
  - Feature set 2
  - Feature set 3
SOAP and Composability

- We are looking at two types of composability of features and services within a message:
  - **Vertical:** multiple features and services can exist simultaneously within the same message
  - **Horizontal:** features and services within a message can be intended for different recipients.
    - This is not boxcarring but rather the HTTP proxy model and as we shall see, the SOAP messaging model as well
Vertical Composability

- Allows for independent features to co-exist

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope"
    SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Header>
    <a:authentication ...>...</a:authentication>
    <s:security ...> ... </s:security>
    <t:transactions ...> ... </t:transactions>
    <p:payment ...> ... </p:payment>
  </SOAP:Header>
  <SOAP:Body>
    <m:mybody> ... </m:mybody>
  </SOAP:Body>
</SOAP:Envelope>
```
Horizontal Composability

- Allows for intermediaries

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope"
                xmlns:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Header>
    <a:authentication actor="intermediary a"/>
    <s:security actor="intermediary b"/>
    <t:transactions actor="intermediary c"/>
    <p:payment actor="destination"/>
  </SOAP:Header>
  <SOAP:Body>
    <m:mybody/>
  </SOAP:Body>
</SOAP:Envelope>
```
Modularity through XML Namespaces

- The SOAP envelope namespace
  - http://schemas.xmlsoap.org/soap/envelope/
  - Resolves to the SOAP Envelope Schema
- The SOAP encoding namespace
  - http://schemas.xmlsoap.org/soap/encoding/
  - Resolves to the SOAP Encoding Schema
- Separate namespaces help enforce modularity
- SOAP Envelope Schema provides formal definition of Envelope
The SOAP Envelope

- A SOAP envelope defines a SOAP message
  - Basic unit of exchange between SOAP processors
- SOAP messages are one-way transmissions
  - From sender through intermediaries to receiver
  - Often combined to implement patterns such as request/response
- Messages are routed along a "message path"
  - Allows for processing at one or more intermediate nodes in addition to the ultimate destination node.
  - A node is a SOAP processor and is identified by a URI
- Envelopes can be nested
  - Only outer envelope is "active" to the receiving SOAP processor
SOAP Headers

- Allows for modular addition of features and services
  - Open-ended set of headers
    - Authentication, privacy, security etc. etc.
  - Can address any SOAP processor using the "actor" attribute
  - Can be optional/mandatory using the "mustUnderstand" attribute
Semantics of SOAP Headers

- Contract between sender and recipient
  - Recipient is described by "actor" attribute
- Allows for different types of negotiation:
  - Take it or leave it directly supported
  - Let's talk about it can be built on top
- And for different settings:
  - Server dictated
  - Peer-to-peer
  - Dictated by third party
The SOAP actor Attribute

- The "Actor" attribute is a generalization of the HTTP Connection header field
  - Instead of hop-by-hop vs. end-to-end, the actor attribute can address any SOAP processor because it is a URI
- Special cases:
  - "next hop" has a special URI assigned to it
  - "end" is the default destination for a header
  - "end" is the destination for the body
The SOAP mustUnderstand Attribute

- The "mustUnderstand" is the same as "mandatory" in the [HTTP Extension Framework](http://www.w3.org/2001/10/servicedef/semantics.html)
  - Requires that the receiving SOAP processor must accept, understand and obey semantics of header or fail
    - It is up to the application to define what "understand" means
  - This allows old applications to gracefully fail on services that they do not understand
SOAP Body

- Special case of header
  - Default contract between sender and ultimate recipient
  - Different from HTTP's header/body separation
  - Defined as a header with attributes set to:
    - Implicit mustUnderstand attribute is always "yes"
    - Implicit actor attribute is always "the end"
SOAP Fault

- The SOAP Fault mechanism is designed to support the composability model
  - Is not a scarce resource as in HTTP where there can be only one (the Highlander principle)
- A SOAP message can carry one SOAP Fault element
  - Must be placed in the Body of the message
- The Fault Detail element carries information for faults resulting from the body
- Detail information for faults resulting from headers are carried in the header
- The SOAP fault code extension mechanism is for SOAP only
  - Application faults should use existing SOAP fault codes
  - Client code is for request faults
  - Server code is for processing faults
SOAP Fault Example

- A SOAP message containing an authentication service:

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope"
SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Header>
    <m:Authentication xmlns:m="http://www.auth.org/simple">
      <m:credentials>Henrik</m:credentials>
    </m:Authentication>
  </SOAP:Header>
  <SOAP:Body>
    … body goes here …
  </SOAP:Body>
</SOAP:Envelope>
```
SOAP Fault Example... 2

...results in a fault because the credentials were bad:

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope"
  SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP:Header>
    <m:Authentication xmlns:m="http://www.auth.org/simple">
      <m:realm>Magic Kindom</m:realm>
    </m:Authentication>
  </SOAP:Header>
  <SOAP:Body>
    <SOAP:Fault>
      <SOAP:faultcode>SOAP:Client</faultcode>
      <SOAP:faultstring>Client Error</faultstring>
    </SOAP:Fault>
  </SOAP:Body>
</SOAP:Envelope>
```
SOAP and "Binary" Data

- "Binary" can in fact mean any data that is to be tunneled through SOAP
  - Encrypted data, images, XML documents, SOAP envelopes as data
- Can be carried in two ways
  - Within the envelope as binary blob
  - Referenced from within the SOAP envelope
- References can point to anything including
  - MIME multipart, HTTP accessible resources etc.
  - Integrity can be obtained through manifest
Binding to HTTP

- The purpose of the HTTP protocol binding is two-fold
  - To ensure that SOAP is carried in a way that is consistent with HTTP’s message model
    - Intent is not to break HTTP
  - To indicate to HTTP servers that this is a SOAP message
    - Allows HTTP servers to act on a SOAP message without knowing SOAP
- Binding only works for HTTP POST requests
- SOAP intermediary is not the same as HTTP intermediary
  - Only HTTP origin server can be SOAP intermediary
HTTP Request

- Use HTTP POST request method name
- Use the SOAPAction HTTP header field
  - It cannot be computed – the sender must know
  - It should indicate the intent – not the destination
- SOAP request doesn't require SOAP response

 POST /Accounts/Henrik HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://electrocommerce.org/MyMessage"

SOAPAction: "http://electrocommerce.org/MyMessage"

<SOAP:Envelope...
HTTP Response

- Successful responses can 2xx status codes
- All 3xx, 4xx, and 5xx status codes work as normal
- SOAP faults must use 500 status code
- SOAP response doesn't require SOAP request
  - Response can in fact be empty

HTTP/1.1 200 Ok
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP:Envelope...
Purpose of SOAP Encoding

- Given a schema in any notation consistent with the data model defined by SOAP, a schema for an XML grammar may be constructed.
Given a type-system schema and a particular graph of values conforming to that schema, an XML instance may be constructed.
Given an XML instance produced in accordance with these rules, and given also the original schema, a copy of the original value graph may be constructed.
Simple Example

<Address id="Address-3">
    <street>28 Sea Dr #103</street>
    <city>Unicity</city>
    <state>CA</state>
</Address>

<Student id="Student-2567">
    <name>Michael</name>
    <dormaddr href="#Address-3"/>
    <attends href="#Course-19"/>
    <attends href="#Course-253"/>
</Student>
Basic Rules (in part)

- All values are represented as element content
- An element may be "independent" (top level of serialization) or "embedded" (everything else)
- Values can be single-reference or multi-reference
- A multi-reference value is represented as the content of an independent element. It has an unqualified attribute named "id" and of type "ID".
- An accessor can point to a multi-reference value with a local, unqualified attribute named "href" and of type "uri-reference"
- The root attribute can be used to indicate roots that are not true roots in a graph
Indicating the Type

For each element containing a value, the type of the value is represented by at least one of the following conditions:

- The containing element instance contains an xsi:type attribute,
- The containing element instance is itself contained within an element containing a (possibly defaulted) SOAP-ENC:arrayType attribute or
- The name of the element bears a definite relation to the type, that type then determinable from a schema.
Simple Types

- A "simple type" is a class of simple values
- SOAP uses all the types found in the section "Built-in data types" of "XML Schema Part 2: Datatypes"
- A simple value is represented as character data, that is, without any sub-elements
Simple Type Examples

```xml
<element name="age" type="int"/>
<element name="height" type="float"/>
<element name="displacement"
    type="negativeInteger"/>
<element name="color">
    <simpleType base="xsd:string">
        <enumeration value="Green"/>
        <enumeration value="Blue"/>
    </simpleType>
</element>
<age>45</age>
<height>5.9</height>
<displacement>-450</displacement>
<color>Blue</color>
```
Compound Types

- A “compound” type is a class of compound values.
- Each related value is potentially distinguished by a role name, ordinal or both (accessor).
- Supports traditional types like structs and arrays.
- Supports nodes with with many distinct accessors, some of which occur more than once.
- Preserves order but doesn't require ordering distinction in the underlying data model.
Struct Compound Type

• A compound value in which accessor name is the only distinction among member values, and no accessor has the same name as any other

```xml
<e:Book>
  <author>Henry Ford</author>
  <preface>When I…</preface>
  <intro>This is a book.</intro>
</e:Book>
```
Array Compound Type

- A compound value in which ordinal position serves as the only distinction among member values

```xml
<SOAP-ENC:Array
  SOAP-ENC:ArrayType="xyz:Order[2]">
  <Order>
    <Product>Apple</Product>
    <Price>1.56</Price>
  </Order>
  <Order>
    <Product>Peach</Product>
    <Price>1.48</Price>
  </Order>
</SOAP-ENC:Array>
```
General Compound Type

- A compound value with a mixture of accessors distinguished by name and accessors distinguished by both name and ordinal position

```xml
<PurchaseLineItems>
  <Order>
    <Product>Apple</Product>
    <Price>1.56</Price>
  </Order>
  <Order>
    <Product>Peach</Product>
    <Price>1.48</Price>
  </Order>
</PurchaseLineItems>
```
Serializing Relationships

- The root element of the serialization serves only as lexical container.
- Elements can reflect arcs or nodes
- Independent elements always reflect nodes
- Embedded elements always reflect arcs
- Element names correspond to node or arc labels
- Arcs are always encoded as embedded elements
1:1 Relationships

- A 1:1 relationship is expressed by simple containment. For example, if a student attends a course, the canonical XML looks like

```xml
<Student>
  <name>Alice</name>
  <attends>
    <name>Greek</name>
  </attends>
</Student>
```
1:n and n:1 Relationships

- A 1:many relationship is expressed by multiple elements for the 1:many direction or single element for the many :1 direction.

```xml
<Teacher id="Teacher-1">
  <name>Alice</name>
  <teaches>
    <name>Greek</name>
  </teaches>
  <teaches >
    <name>English History</name>
  </teaches>
</Teacher>
```
m:n Relationships

- A many:many relationship is expressed by using references in both directions.

```xml
<Student id="Student-1">
  <name>Alice</name>
  <attends href="#Course-1"/>
  <attends href="#Course-2"/>
</Student>
<Course id="Course-1">
  <name>Greek</name>
  <attendee href="Student-1"/>
</Course>
```
SOAP and RPC

- A method invocation is modeled as a struct
- A method response is modeled as a struct
- Struct contains an accessor for each [in] or [in/out] or [out] parameter.
- The request struct is both named and typed identically to the method name.
- The response struct name is not important
- The first accessor is the return value followed by the parameters in the same order as in the method signature
Summary

- SOAP envelope provides
  - Composability in the vertical (Shopping basket)
  - Composability in the horizontal (Amtrak)
- SOAP can be used with many protocols
  - Easy to deploy with existing infrastructure
- SOAP is fundamentally a one-way message
  - Supports request/response, RPC etc.
  - Your application decides what it is!