Introduction to SOAP

Henrik Frystyk Nielsen
<frystyk@microsoft.com>

SOAP/1.1 Spec Status

- SOAP/1.1 was submitted to W3C and became a W3C Note on May 8, 2000
  - Intent was to start standards process
- W3C was the natural place because SOAP is intended as general infrastructure
- Discussion happens on public lists
  - soap@discuss.develop.com
  - xml-dist-apps@w3.org

SOAP Development

- Developed in traditional Web style:
  - Very distributed community with broad support
- Several Implementations
  - MSDN Toolkit, MS .Net Framework, Apache SOAP, Developmentor SOAP toolkits in Perl and Java, SOAP::lite (Perl), libwww based SOAP, IONA, and many more
- New spec for SOAP binding to MIME Multipart
  - John Barton, HP Labs, Satish Thatte, MS, and myself
- SOAP/1.1 issues list

SOAP/1.1 Authors

- Don Box, DevelopMentor
- David Ehnebuske, IBM
- Gopal Kakivaya, Microsoft
- Andrew Layman, Microsoft
- Noah Mendelsohn, Lotus
- Henrik Frystyk Nielsen, Microsoft
- Satish Thatte, Microsoft
- Dave Winer, UserLand Software

W3C Submitters

- Ariba, Inc.
- Commerce One, Inc.
- Compaq Computer Corporation
- DevelopMentor, Inc.
- Hewlett Packard Company
- International Business Machines Corporation
- IONA Technologies
- Lotus Development Corporation
- Microsoft Corporation
- SAP AG
- UserLand Software Inc.

The W3C XML Protocol Activity

- In Sep 2000, W3C started XML Protocol Activity
- Contains the XP Working Group
  - Chair is David Fallside, IBM
  - Very large (70+ members)
  - Public charter and mailing list xml-dist-app@w3.org
- SOAP/1.1 is the starting point for this work
  - Will be evaluated against requirements
- Very focused charter with Four Deliverables:
  - Protocol Envelope
  - Mechanism for serializing abstract data models
  - Convention for use with RPC
  - Binding to HTTP

What is SOAP?

- SOAP is a simple, lightweight XML protocol for exchanging structured and typed information on the Web
    - One way message based
    - Decentralized evolvability
    - Loosely coupled, stateless interactions
- Overall design goal: KISS
    - 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 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
- Protocol bindings to HTTP and HTTP-EF *(opt)*

SOAP Example in HTTP

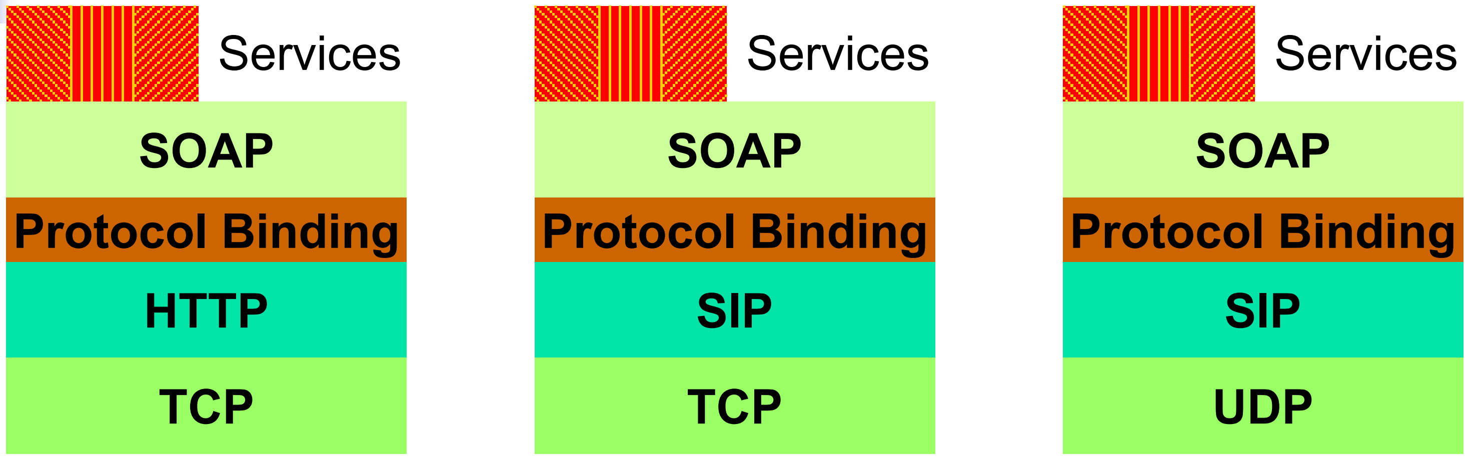
```
POST /Accounts/Henrik HTTP/1.1
Host: www.somebank.com
Content-Length: nnnn
Content-Type: text/xml; charset="utf-8"
SOAPAction: "Some-URI"
```

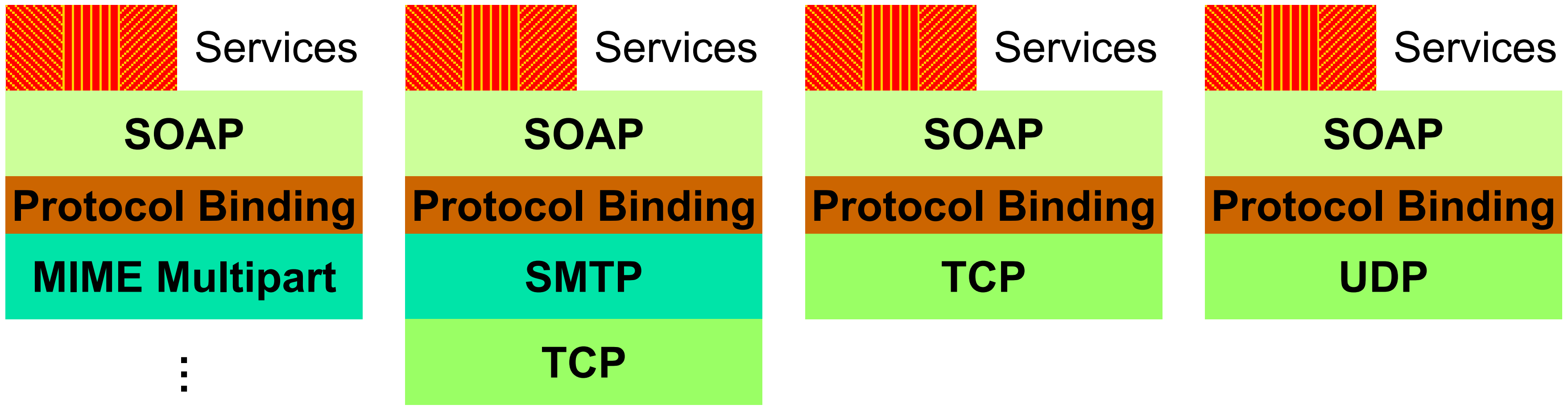
SOAP-HTTP Binding

HTTP Request
SOAP Body
SOAP Header
SOAP Envelope

```
<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 Stack Examples





Note: 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
  - You can tunnel through anything

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
- 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 use my own 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

Vertical Composability

- Allows for independent features to co-exist

```
<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

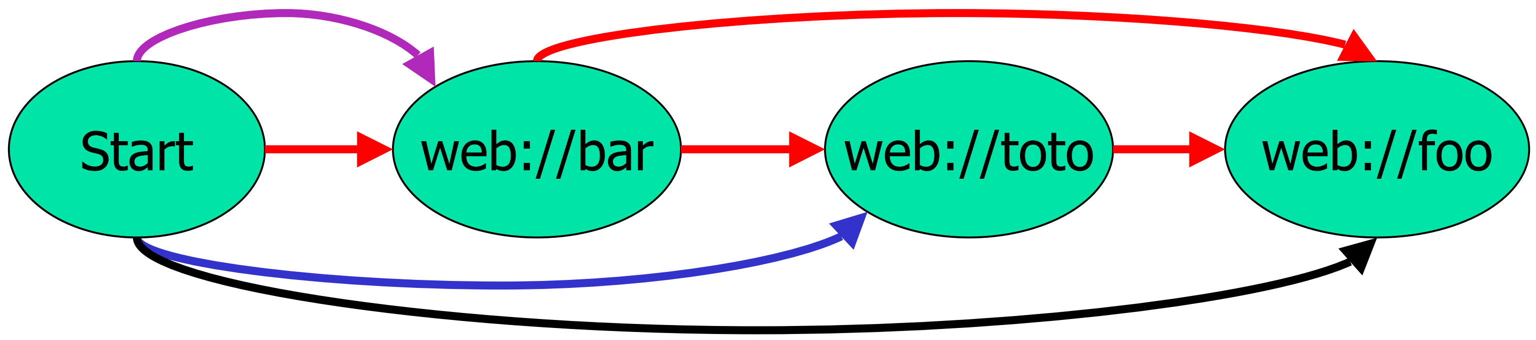
```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope”
  SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP:Header>
      <a:authentication actor="intermediary a"…>…</a:authentication>
      <s:security actor="intermediary b"…> … </s:security>
      <t:transactions actor="intermediary c"…> … </t:transactions>
      <p:payment actor="destination"…> … </p:payment>
   </SOAP:Header>
   <SOAP:Body>
      <m:mybody> … </m:mybody>
   </SOAP:Body>
</SOAP: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 (a SOAP processor) is identified by a URI
- Envelopes can be nested
  - Only outer envelope is "active" to the receiving SOAP processor

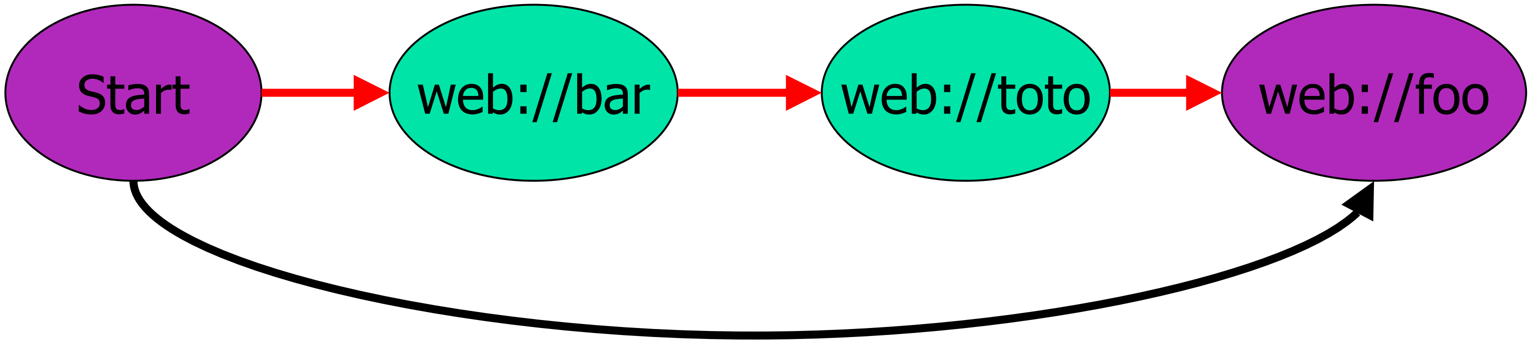
SOAP Headers

- Allows for open-ended addition of modular features and services
  - Address any SOAP processor using "actor" attribute
  - Optional/mandatory using "mustUnderstand" attribute



SOAP Body

- Special case of header
  - Default contract between sender and ultimate recipient
  - 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

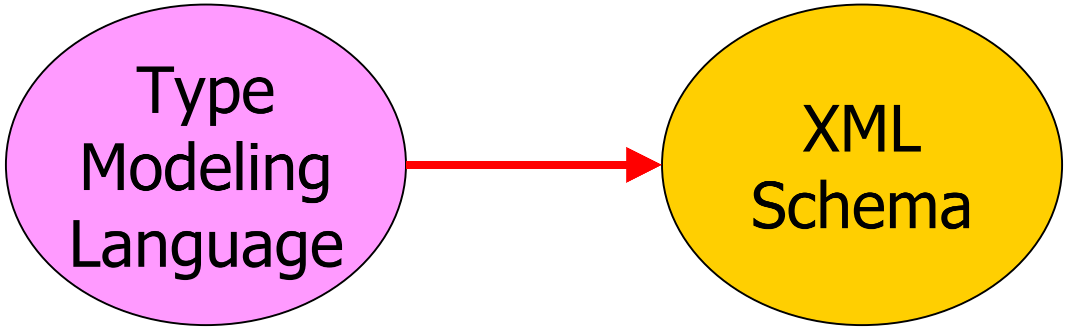
```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope”
  SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP:Header>
      <m:Authentication xmlns:m="http://www.auth.org/simple">
              <m:realm>Magic Kindom</m:realm>
      </m:Authentication>
   </SOAP:Header>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

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

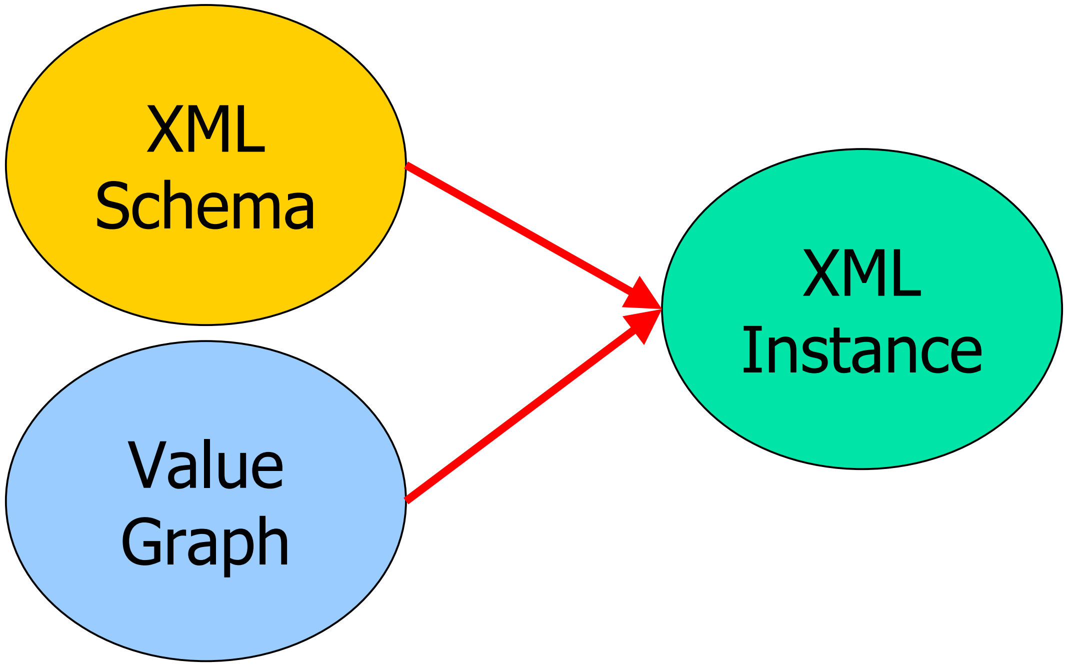
Purpose of SOAP Encoding

- Given a schema in any notation consistent with the data model defined by SOAP, a schema for an XML grammar may be constructed



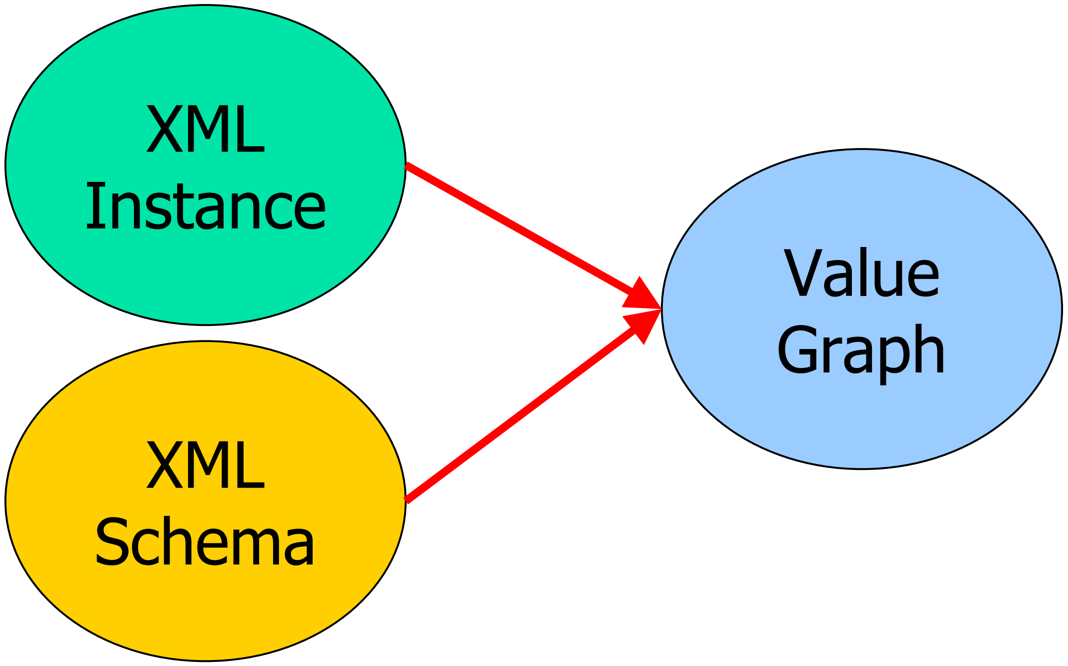
Purpose of SOAP Encoding… 2

- Given a type-system schema and a particular graph of values conforming to that schema, an XML instance may be constructed.



Purpose of SOAP Encoding… 3

- 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.



SOAP Encoding Types

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

Type Examples

- Simple Type Example

```
<age>45</age>
<height>5.9</height>
<displacement>-450</displacement>
<color>Blue</color>
```

- Compound Struct Type

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

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!