

# XQuery at your Web Service

Jérôme Siméon

IBM T. J. Watson Research Center

## Part I

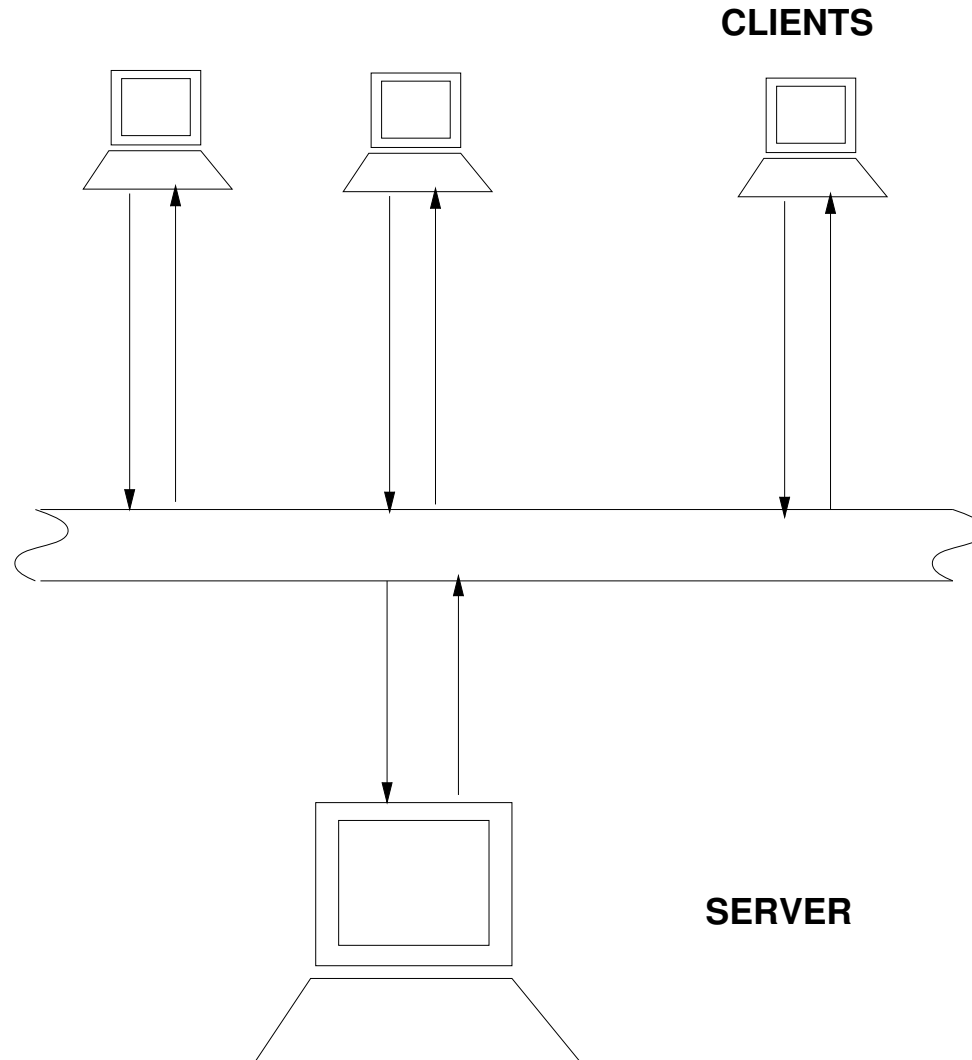
# XQuery at your Web Service



## Overview

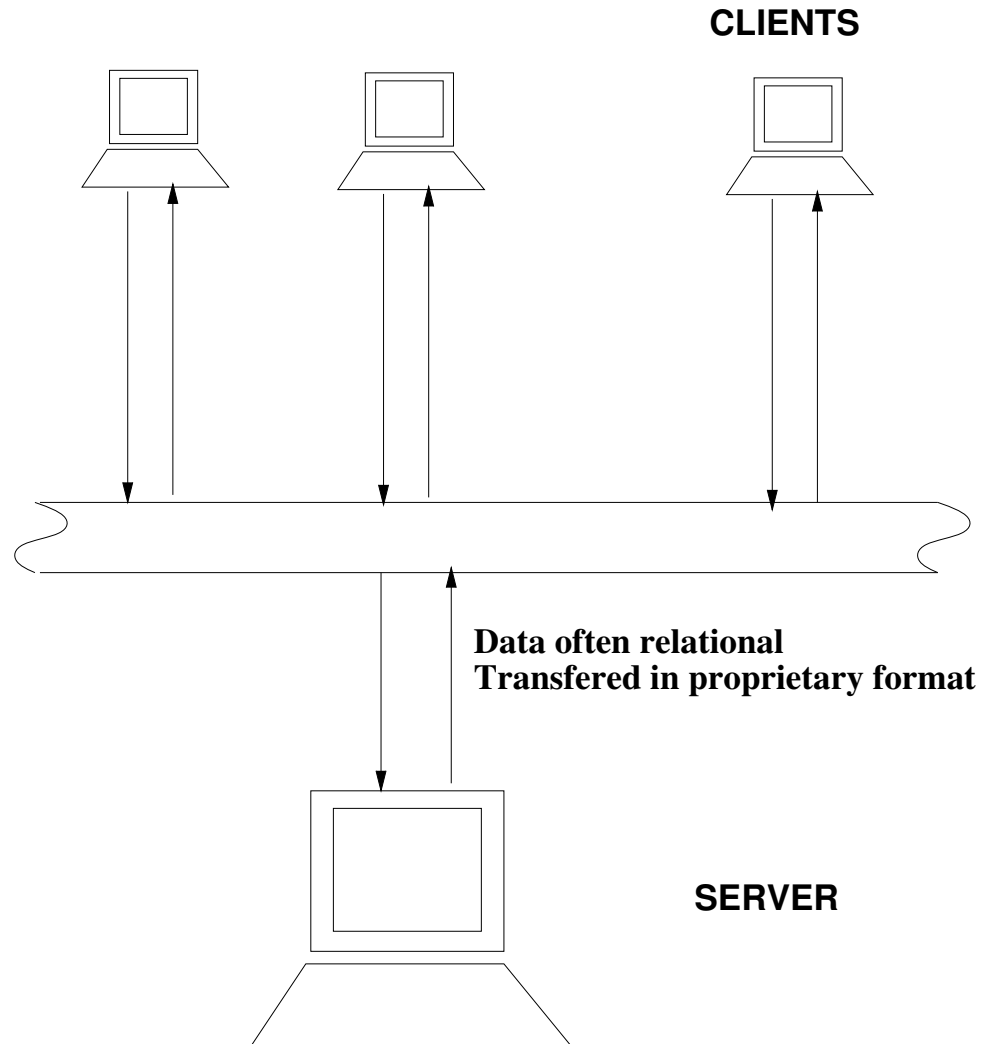
# The world used to be simple

- ▶ Centralized and stable
- ▶ Client-server architecture



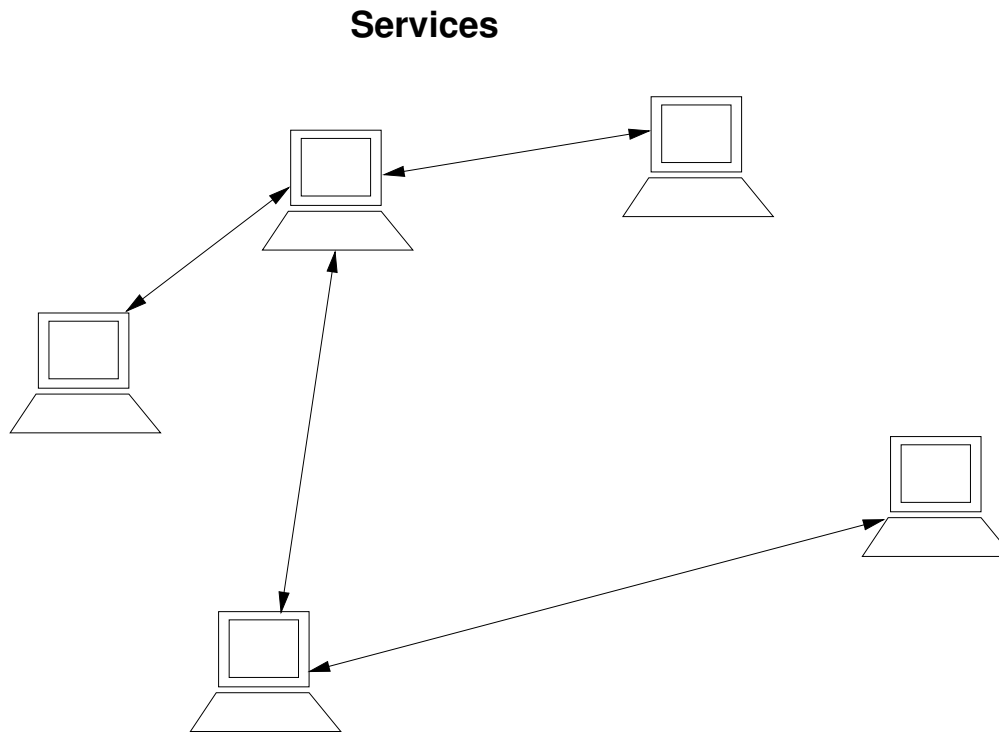
# The world used to be simple

- ▶ Centralized and stable
- ▶ Client-server architecture



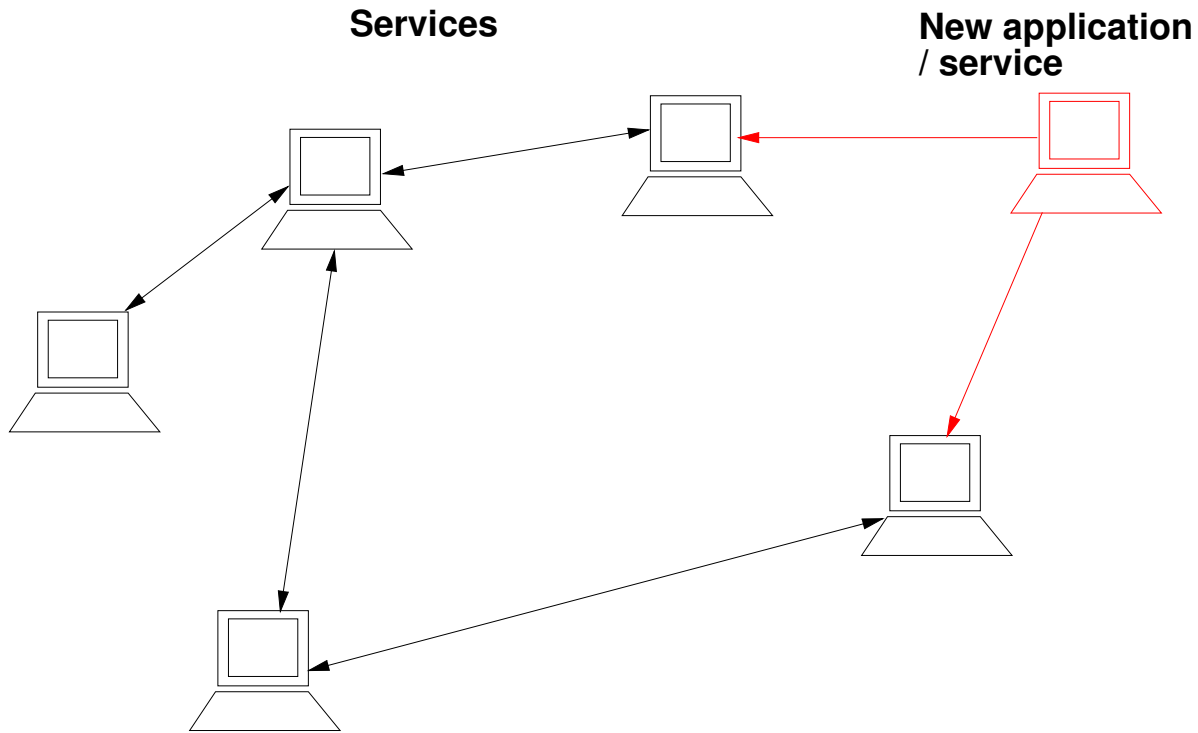
# The world now

- ▶ Distributed and evolving
- ▶ Service oriented architecture (SOA)



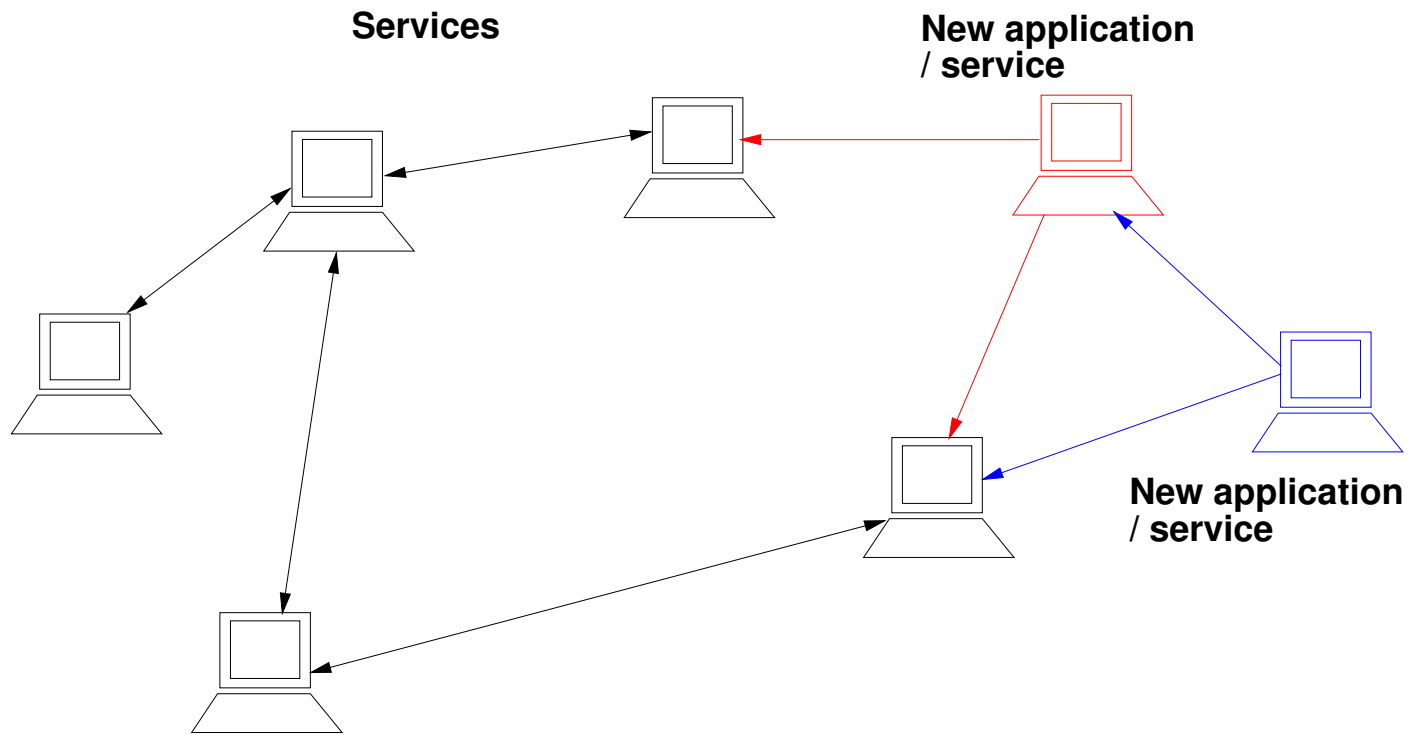
# The world now

- ▶ Distributed and evolving
- ▶ Service oriented architecture (SOA)



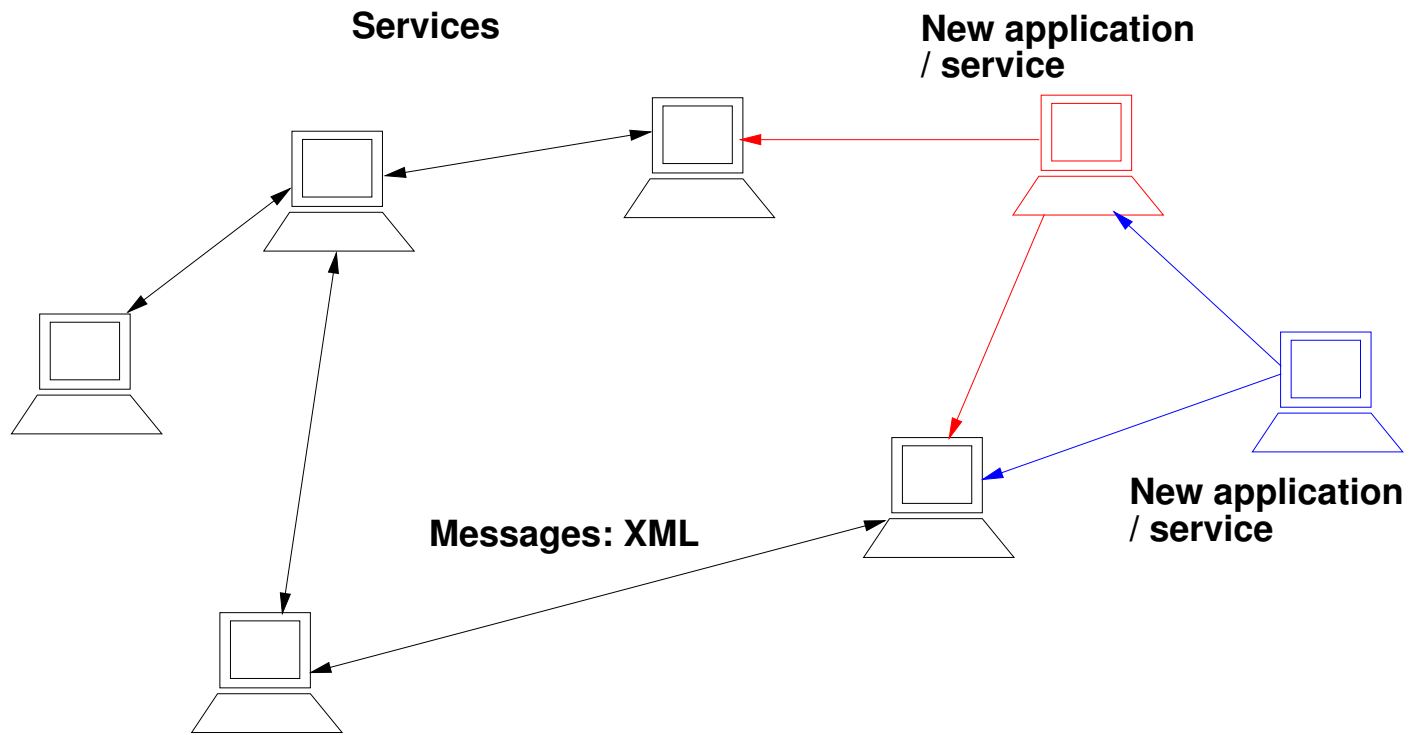
# The world now

- ▶ Distributed and evolving
- ▶ Service oriented architecture (SOA)



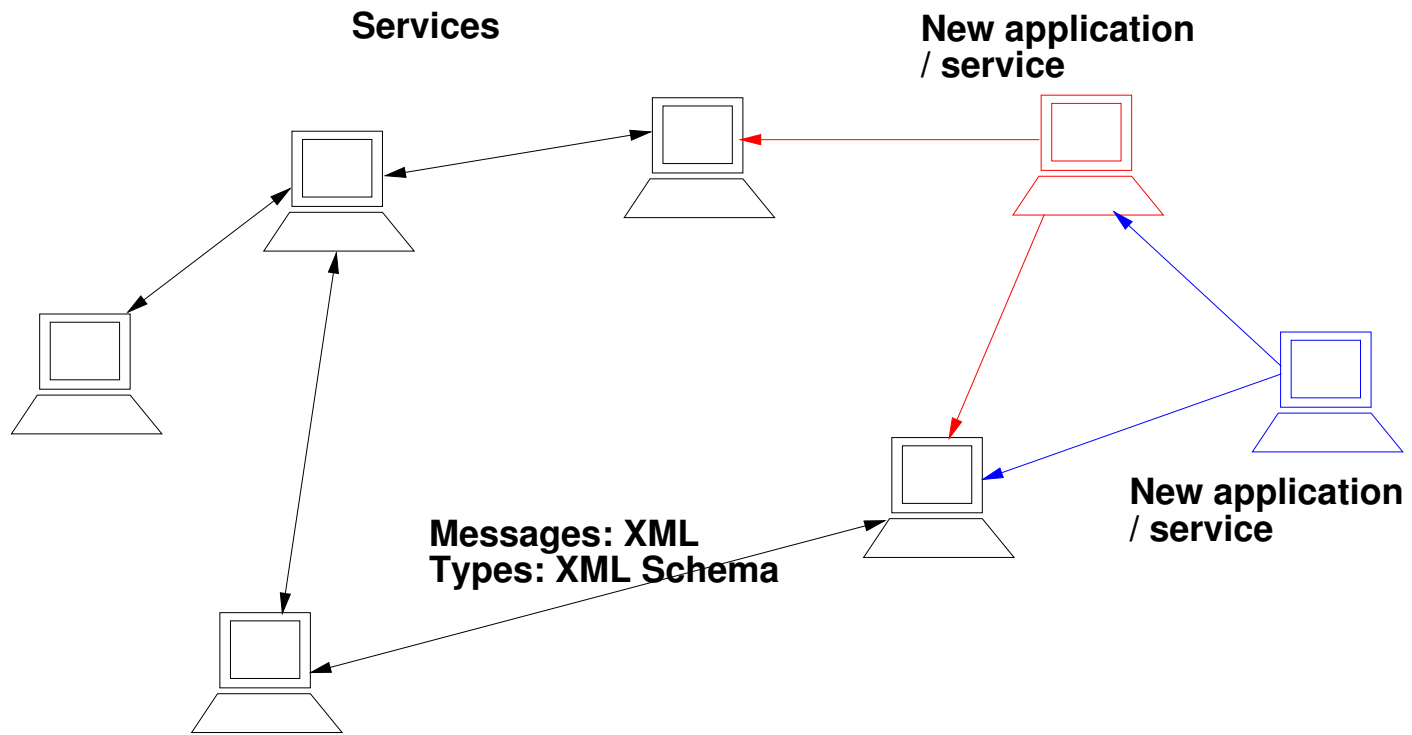
# The world now

- ▶ Distributed and evolving
- ▶ Service oriented architecture (SOA)



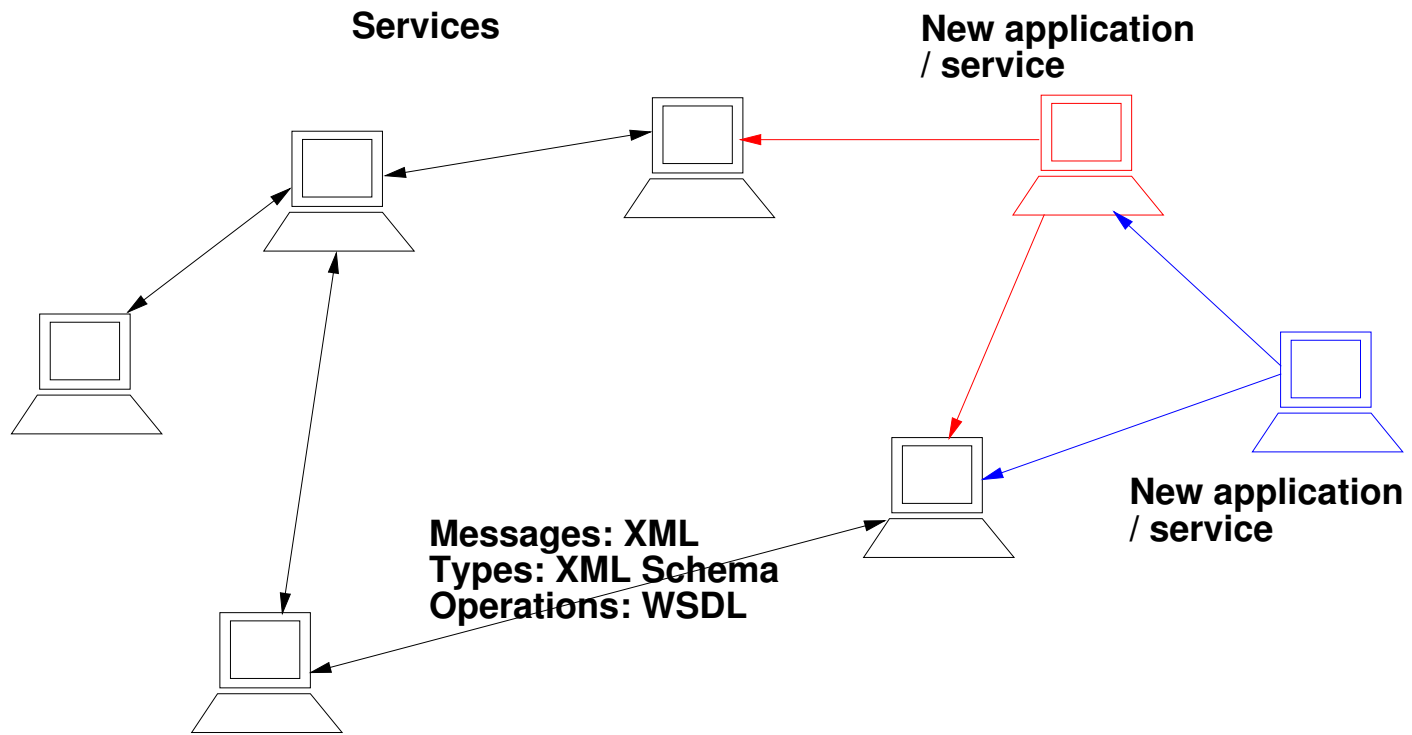
# The world now

- ▶ Distributed and evolving
- ▶ Service oriented architecture (SOA)



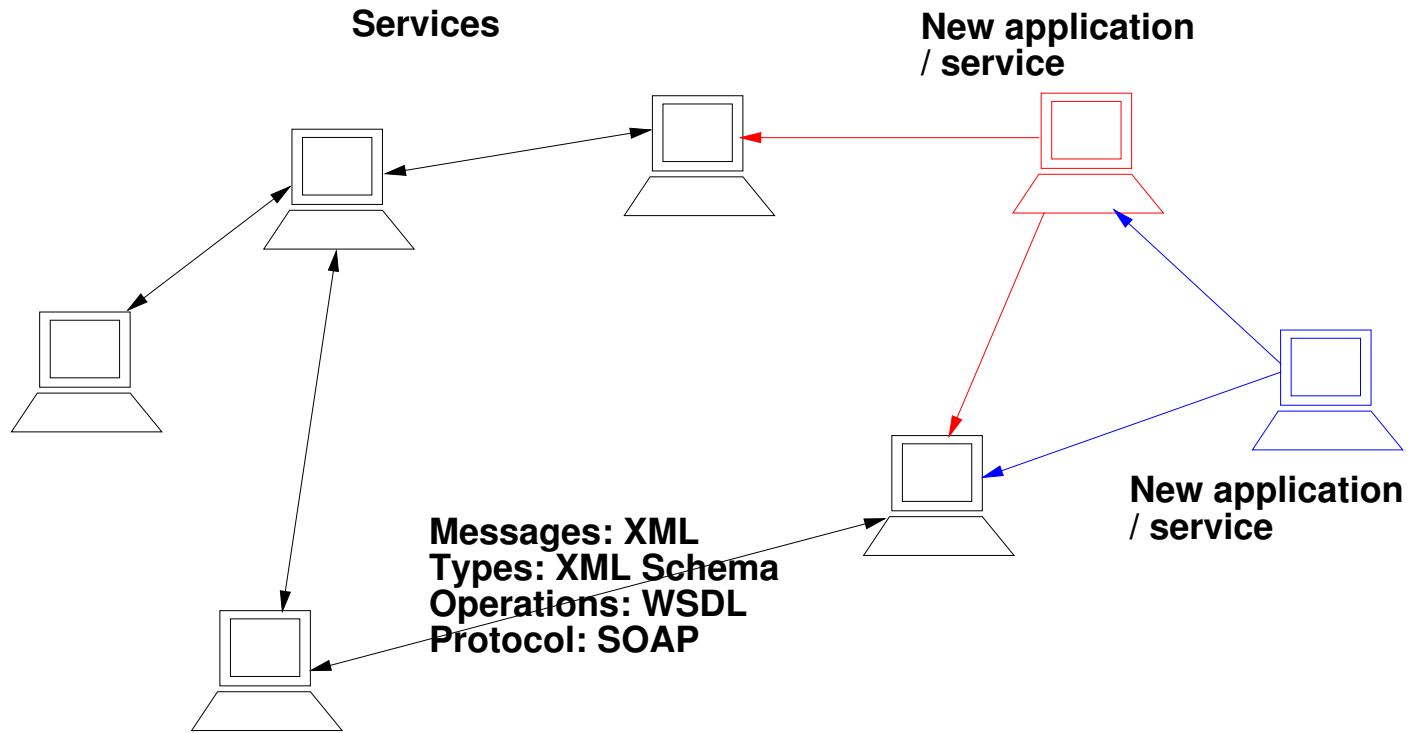
# The world now

- ▶ Distributed and evolving
- ▶ Service oriented architecture (SOA)



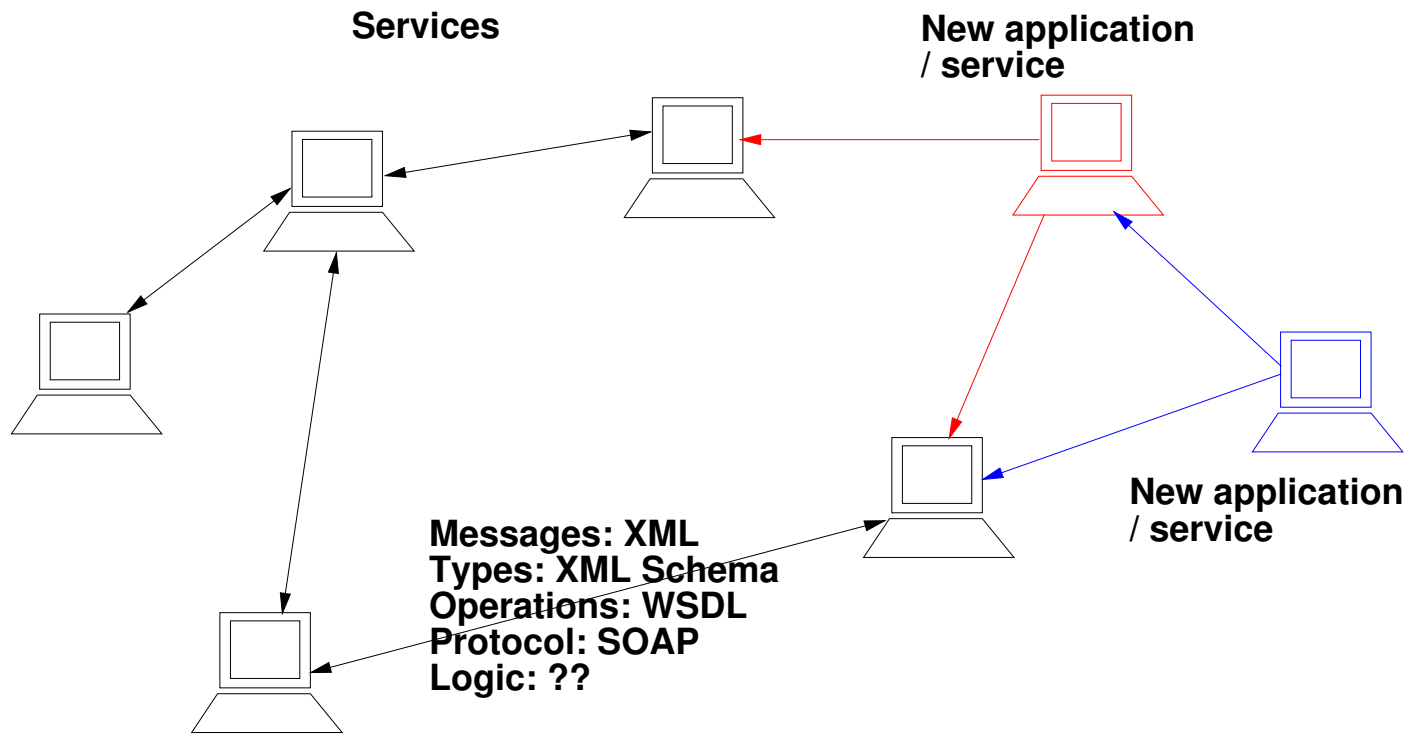
# The world now

- ▶ Distributed and evolving
- ▶ Service oriented architecture (SOA)



# The world now

- ▶ Distributed and evolving
- ▶ Service oriented architecture (SOA)



# What is the talk about?

- ▶ How to use XQuery in SOA

# What is the talk about?

- ▶ How to use XQuery in SOA
- ▶ How to program Web services with XQuery
  - ▶ How to hook up XQuery to Web services infrastructure

# Hooking up XQuery to Web services

- ▶ Binding between WSDL and XQuery

# Hooking up XQuery to Web services

- ▶ Binding between WSDL and XQuery
- ▶ XQuery extensions
  - ▶ Importing Web service as an XQuery module

```
import service
  namespace yh = "http://YooHoo.net/";
...

```

# Hooking up XQuery to Web services

- ▶ Binding between WSDL and XQuery
- ▶ XQuery extensions
  - ▶ Importing Web service as an XQuery module

```
import service
  namespace yh = "http://YooHoo.net/";
...

```

- ▶ Deploy an XQuery module as a Web service

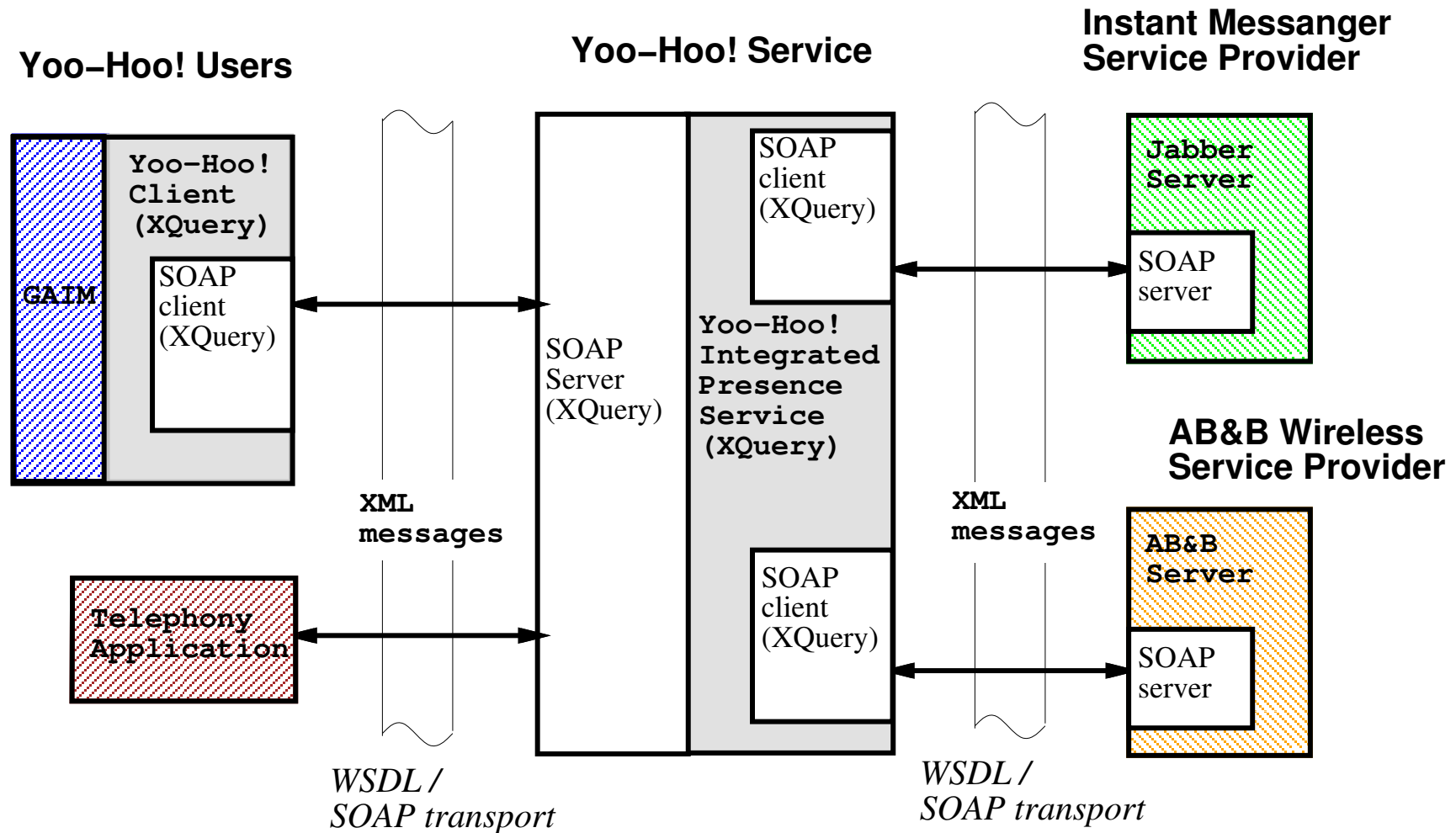
```
xquery2soap -installdir ... YooHoo.xq
```

# What is the talk about?

- ▶ How to use XQuery in SOA
- ▶ How to program Web services with XQuery
  - ▶ How to hook up XQuery to Web services infrastructure
- ▶ Example building presence service
  - ▶ Unified access to *presence* information
  - ▶ Integrates presence from multiple service providers
  - ▶ *“the requested subscriber is on-line at Jabber”*
  - ▶ *“the subscriber’s cell phone is busy”*

# Sample application

- ▶ Building the Yoo-Hoo! presence service



# Organization

- ▶ XML Schema / XQuery / WSDL Crash Course
- ▶ Building Yoo-Hoo! with XQuery and WSDL
- ▶ WSDL-XQuery Binding
- ▶ Challenges and Opportunities

## Part II

### XQuery at your Web Service



XML Schema / XQuery / WSDL

# Presence Information in XML Schema

```
<xs:schema targetNamespace="http://YooHoo.net">
  <xs:element name="presence" type="PresenceType"/>

  <xs:complexType name="PresenceType">
    <xs:sequence>
      <xs:element ref="service" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="service">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="location" type="xs:string"/>
        <xs:choice>
          <xs:element name="online" type="xs:string"/>
          <xs:element name="offline" type="xs:string"/>
          <xs:element name="chat" type="xs:string"/>
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

# XQuery in one slide

- ▶ XQuery is a language to process XML
  - ▶ On W3C recommendation track. Last call period.
  - ▶ Understands XML, namespaces, XML Schema *natively*
  - ▶ XPath 2.0 is a subset of XQuery 1.0
- ▶ Benefit from database and programming language experience
  - ▶ Small functional language (like ML)
  - ▶ Database primitives (like SQL)
- ▶ Has a both an XML and a non-XML syntax:

```
for $p in distinct(//publisher)
let $a := avg(//book[publisher = $p]/price)
return
  <publisher>
    <name> { $p/text() } </name>
    <avgprice> { $a } </avgprice>
  </publisher>
```

# XQuery is like SQL

- ▶ Concise notation for iteration, selection, aggregation
- ▶ E.g., find average price of books for each publisher
  - ▶ In SQL, Select-From-Where:

```
SELECT P.Name, AVG B.Price
FROM Publishers P, Books B
WHERE B.BookPub = P.PubId
GROUP BY P.Name
```

- ▶ In XQuery, FLWOR expressions:

```
for $p in distinct(//publisher)
let $a := avg(//book[publisher = $p]/price)
return
  <publisher>
    <name> { $p/text() } </name>
    <avgprice> { $a } </avgprice>
  </publisher>
```

# XQuery is like ML

▶ In ML:

```
# let f x = if (x > 0) then x+2 else -x+2;;  
val f : int -> int = <fun>  
# (* one call to f *)  
  let a = 1 in f(a);;  
- : int = 3
```

▶ In XQuery:

```
declare function f($x as xs:integer) as xs:integer {  
  if ($x > 0) then $x + 2 else -$x+2  
};
```

(: One call to f :)

```
let $a := 1 return f($a)
```

==>

```
- : xs:integer = 3
```

# Static typing in XQuery

▶ In ML:

```
# let a = "1" in f(a);;
```

Characters 17-18:

```
let a = "1" in f(a);;
```

^

This expression has type string but is here used with type int

▶ In XQuery:

```
let $a := "1" return f($a)
```

==>

Static Type Error

- ▶ Detects some errors at compile time
- ▶ All based on XML Schema

# XQuery Modules

- ▶ Contains:
  - ▶ Type declarations
  - ▶ Function definitions

- ▶ Fibonacci module:

```
module "m1.example.com";
import schema my = "my.integers.com";

declare function fibo($n as my:integer) as xs:integer
{
  if ($n = my:integer(0)) then 0
  else if ($n = my:integer(1)) then 1
  else (fibo($n -1)+ fibo($n -2))
};
```

- ▶ Calling module:

```
import module "m1.example.com";
import schema my = "my.integers.com";

let $e := my:integer(10)
return fibo( $e )
```

# XQuery Modules Limitations

## ▶ No notion of 'hiding'

```
module "m1.example.com";
import schema my = "my.integers.com";

declare function test0($n as my:integer) { ($n = 0) };
declare function test1($n as my:integer) { ($n = 1) };

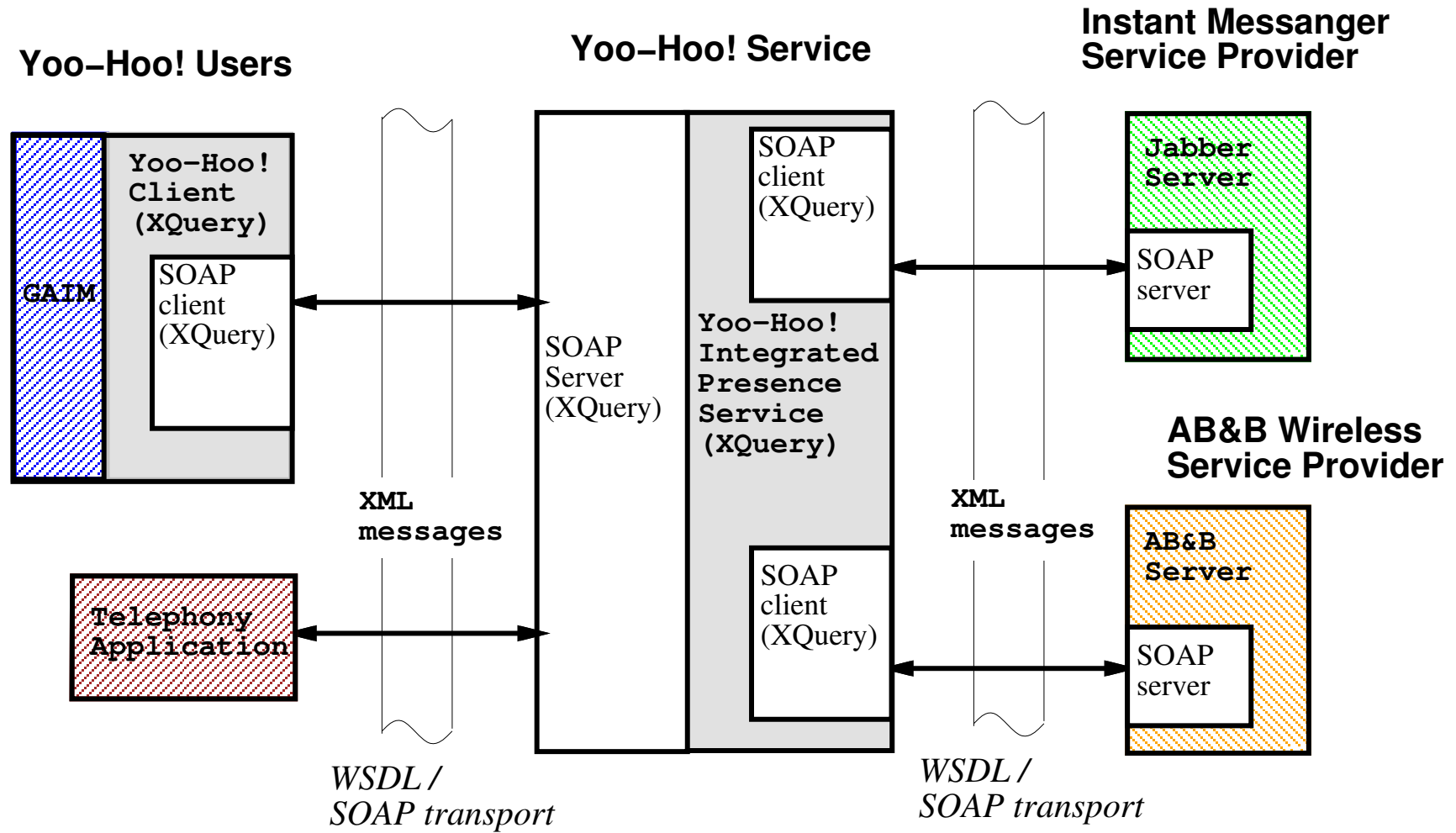
declare function fibo($n as my:integer) as xs:integer
{
  if test0($n) then 0
  else if test1($n) then 1
  else (fibo($n -1)+ fibo($n -2))
};
```

## ▶ Which types are exported?

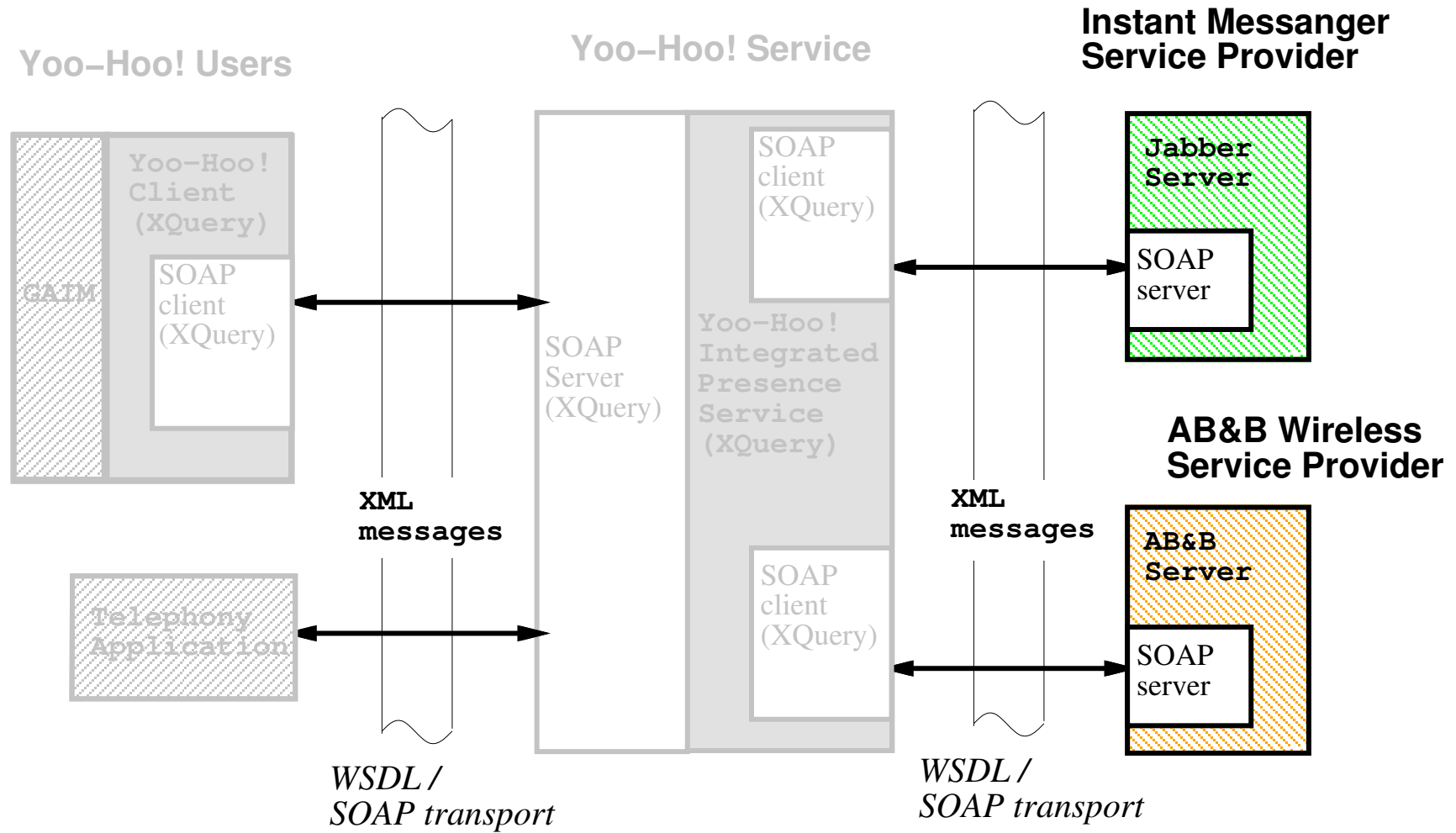
```
import module "m1.example.com";

let $e := my:integer(10)
return fibo( $e )
```

# Describing the Service



# Describing the Service



# Web Service Description Language

```
<definitions targetNamespace="http://YooHoo.net"
  xmlns:yh="http://YooHoo.net" ...>
  <types>
    <xs:schema targetNamespace="http://YooHoo.net"> ...
  </types>

  <message name="User">
    <part name="user" type="xs:string"/>
  </message>
  <message name="Presence">
    <part name="result" element="yh:presence"/>
  </message>

  <portType name="PresencePort">
    <operation name="presence">
      <input message="yh:User"/>
      <output message="yh:Presence"/>
    </operation>
  </portType>

  <binding name="PresenceSOAP" type="yh:PresencePort"> ...
  <service name="YooHoo">...</service>
</definitions>
```

# WSDL vs. XQuery Module

- ▶ XQuery modules
  - ▶ XML Schema type declarations
  - ▶ Functions body
  - ▶ Functions signature
- ▶ WSDL
  - ▶ XML Schema type declarations
  - ▶ Procedure call signatures...

**WSDL = XQuery Module Interface!**

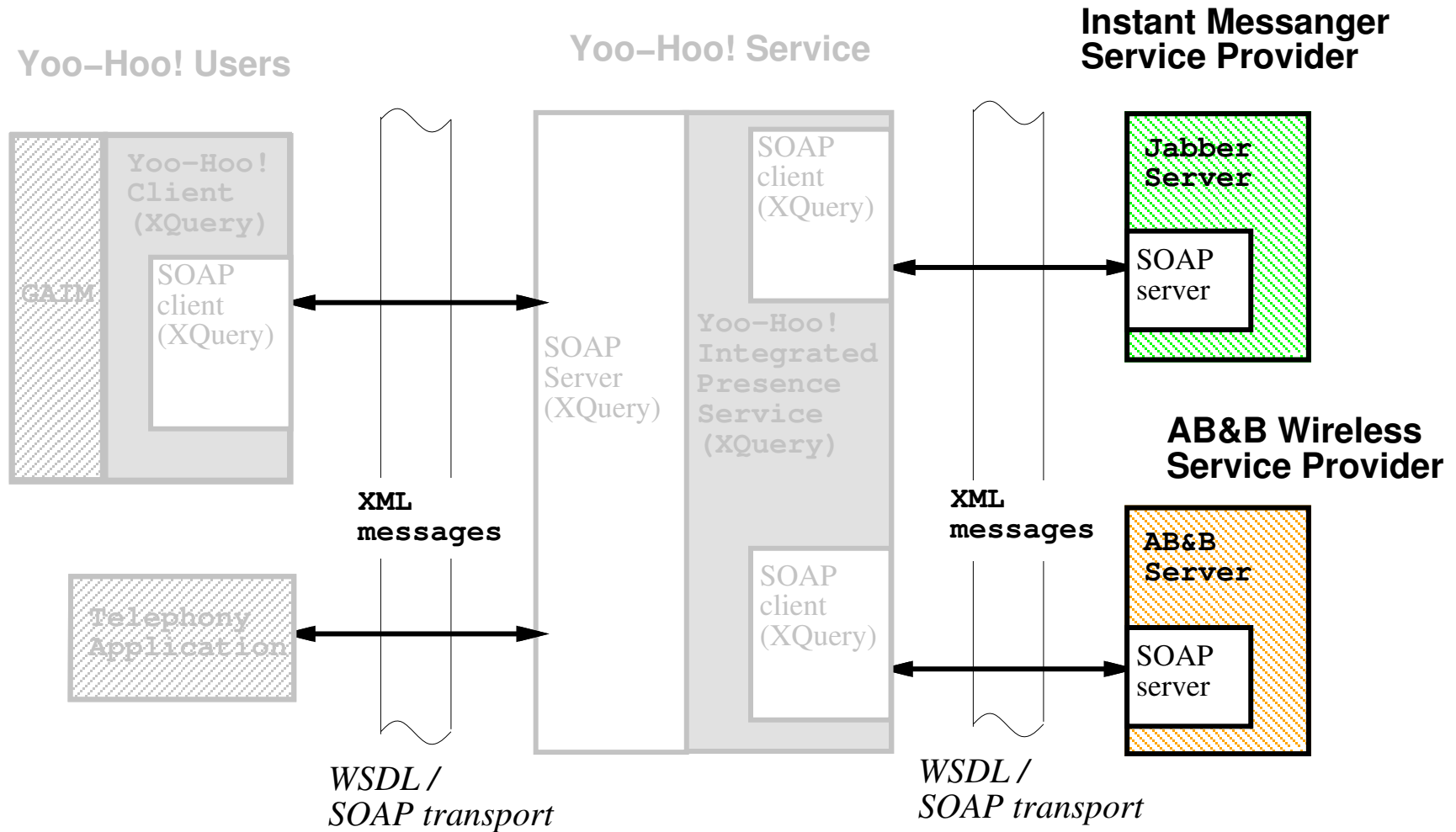
## Part III

# XQuery at your Web Service

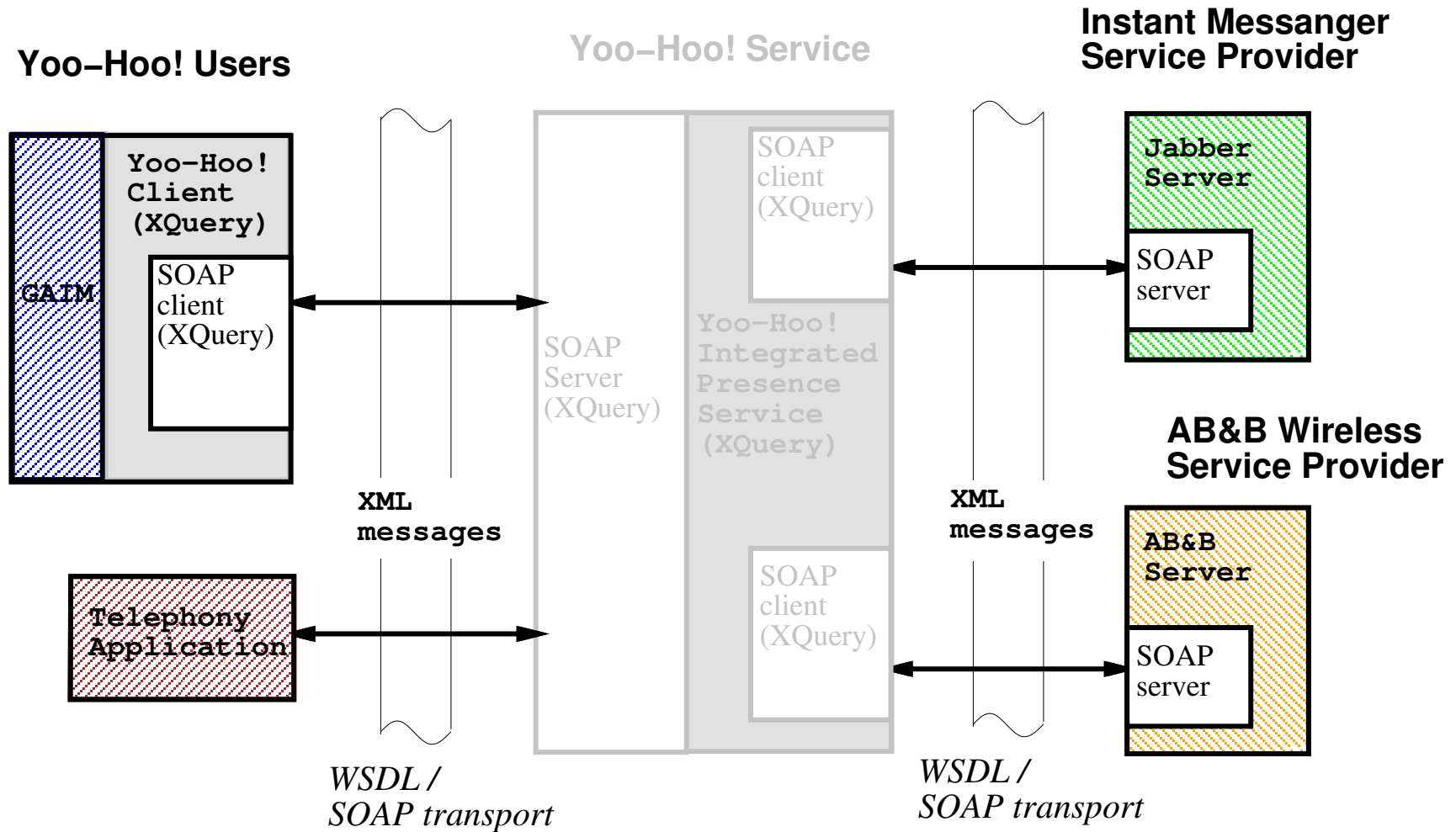


## Building a service with XQuery

# Building the Web Service Client



# Building the Web Service Client



# Importing Web services in XQuery

- ▶ Add “import service” declaration to the XQuery prolog:

```
import service
  (namespace NCName =)?      (: namespace prefix :)
  StringLiteral              (: target namespace :)
  (at StringLiteral)?       (: location hint :)
  name StringLiteral         (: service name :)
  (port StringLiteral)?     (: port name :)
```

- ▶ Target namespace used to identify the WSDL resource
- ▶ Services name to identify the service
- ▶ (Optional) Namespace prefix to bind the URL for the service to a local prefix in the XQuery module

# Importing Web services in XQuery

```
import service namespace yh = "http://YooHoo.net/"
  name "YooHoo"

let $b := //buddies[name = "Elena Buchsbaum"]
for $s in yh:presence(b/@yhid)/service return
return
  let $presence :=
    if ($s[online|chat]) then "Available :)"
    else "Unavailable :("
  return
    fn:concat($presence, " on: ", $s/location)
```

- ▶ Makes *operations* available as *functions*
- ▶ Including XML Schema types!
- ▶ Deals only with synchronous calls

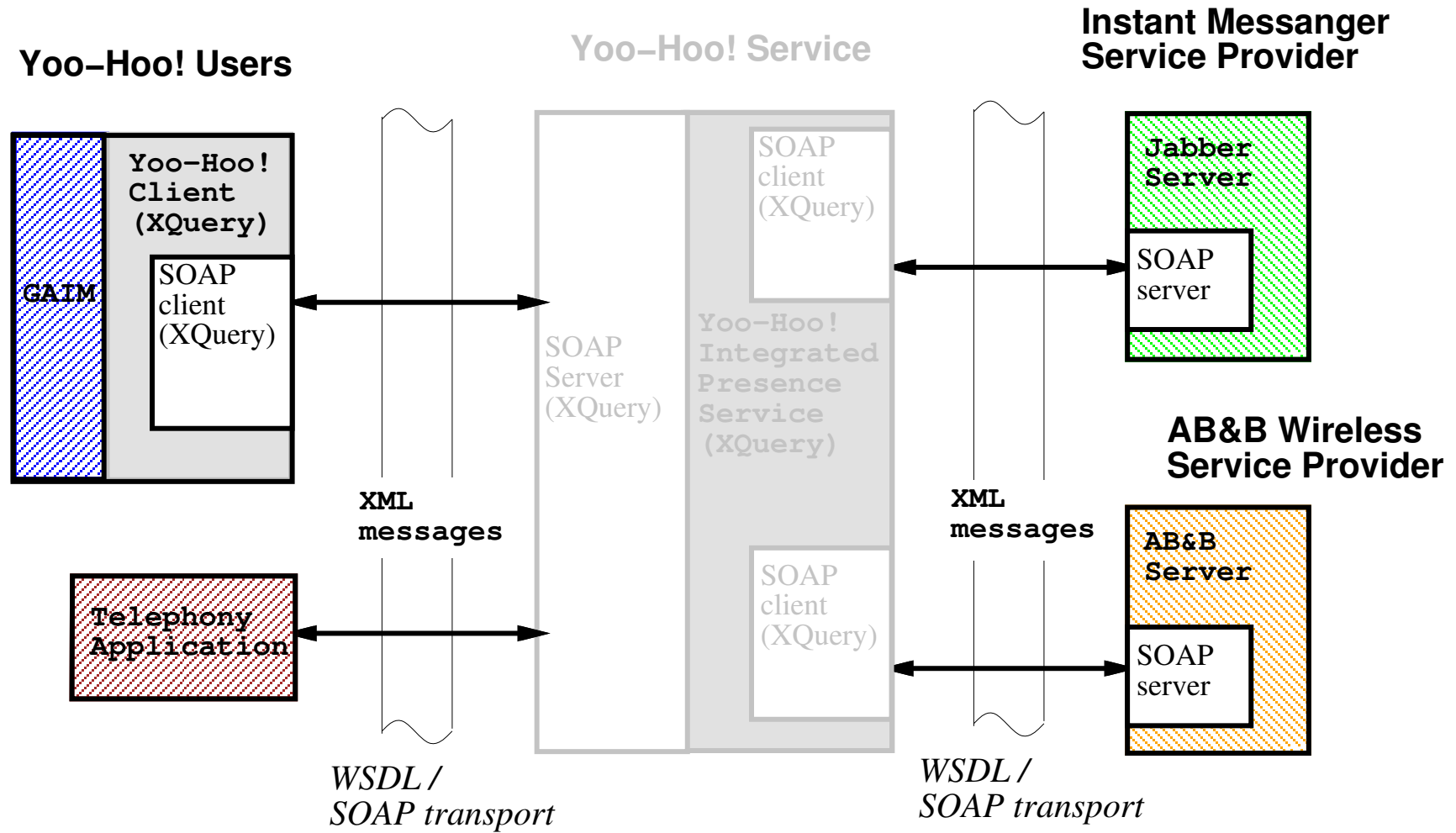
## Additional benefits

```
import service namespace yh = "http://YooHoo.net/"
  name "YooHoo"

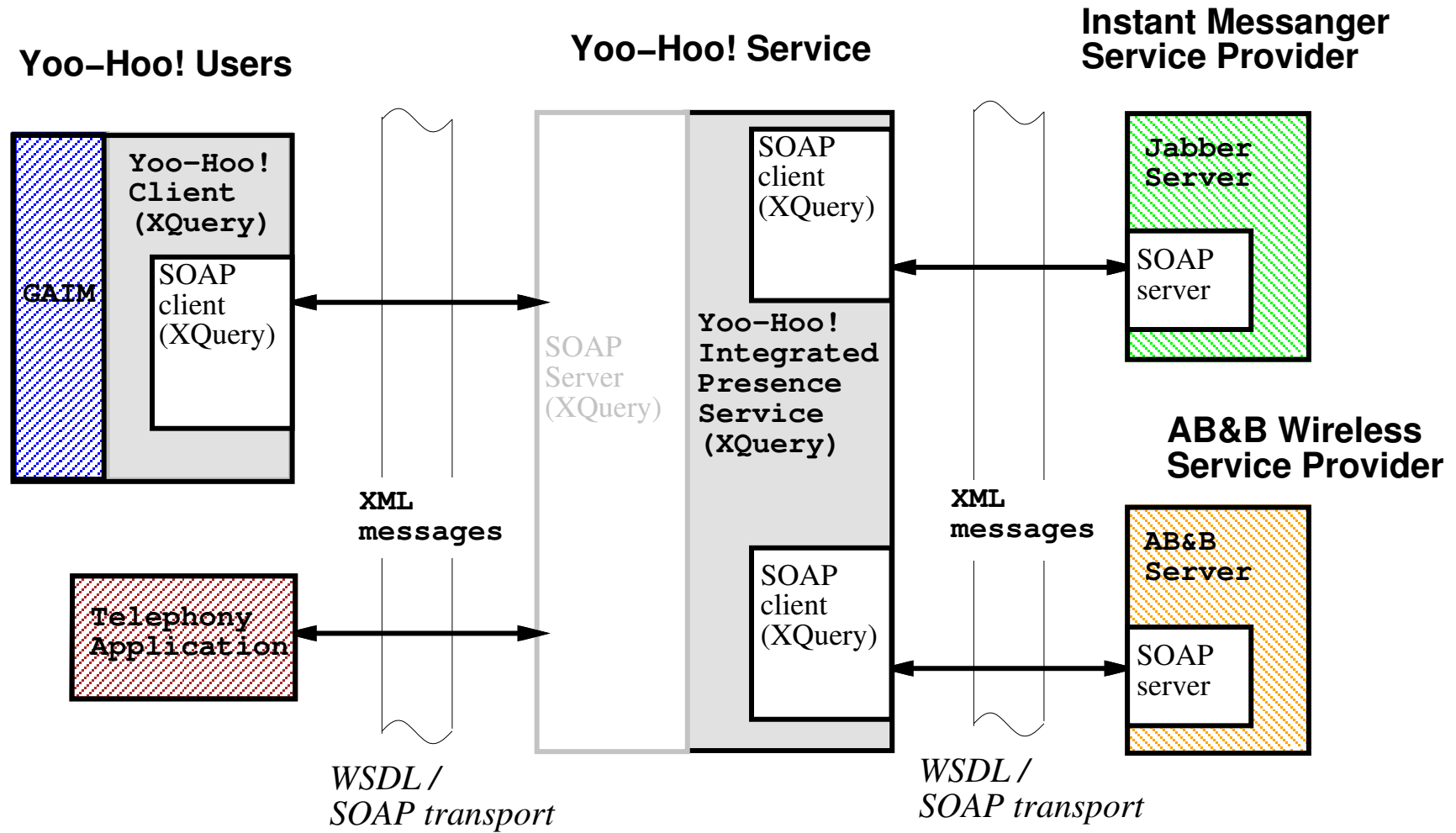
let $b := //buddies[name = "Elena Buchsbaum"]
for $s in yh:presence(b/@yhd)/service return
return
  let $presence :=
    if ($s[online|chat]) then "Available :)"
    else "Unavailable :("
  return
    fn:concat($presence, " on: ", $s/location)
```

- ▶ There is a bug in that program
- ▶ Can you find it?
- ▶ Static Typing in XQuery can

# Building the Service itself



# Building the Service itself



# Building the Service itself

```
module namespace yh = "http://YooHoo.net/YooHoo";

import service namespace im = ".." port "IM";
import service namespace jabber=".." port "Jabber";

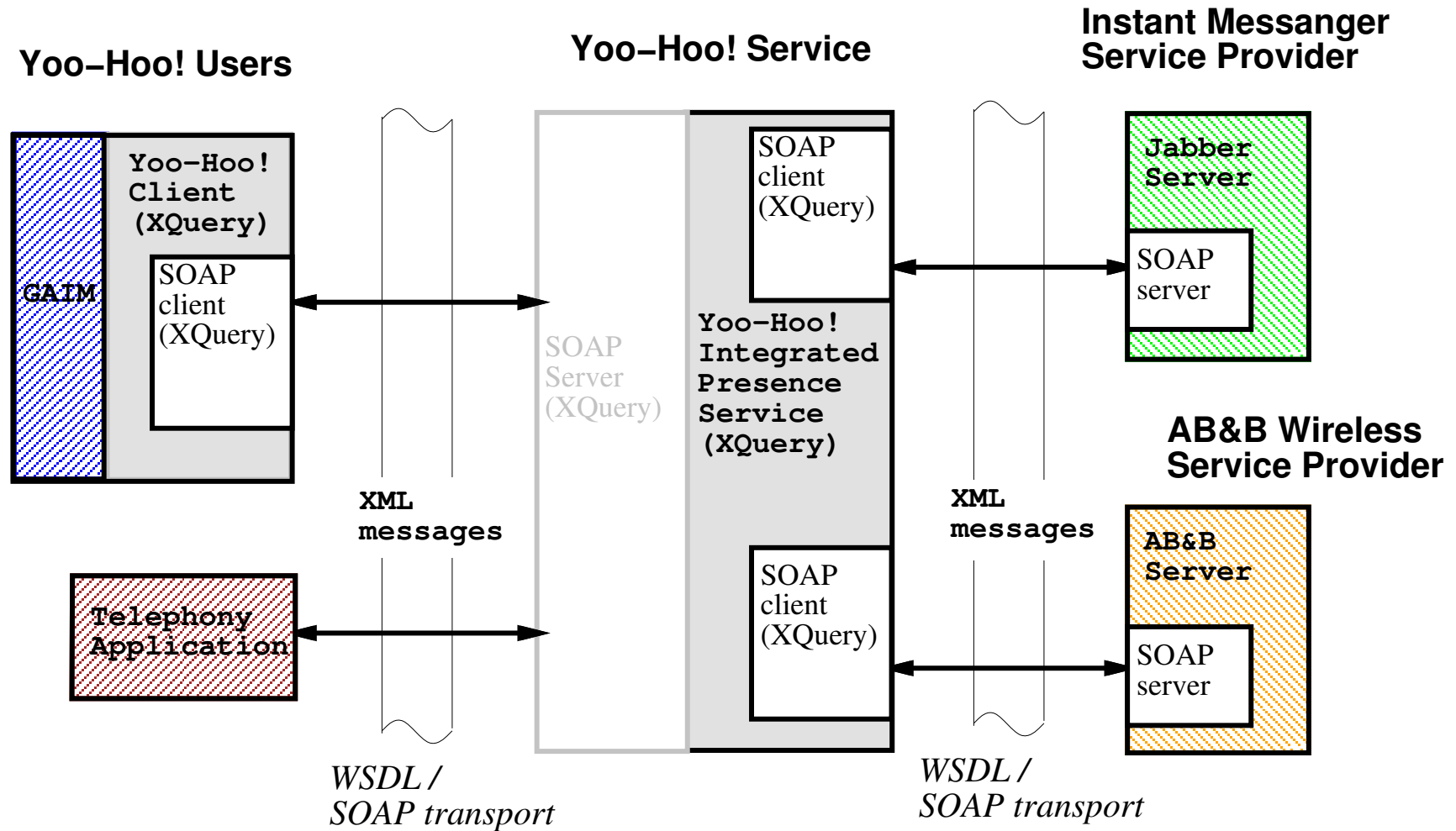
declare function yh:presence($user as xs:string)
  as element(yh:presence) {
  <presence>
    <service>
      <location>IM</location>
      { im:status($user) }
    </service>
    <service>
      <location>Jabber</location>
      { jabber:onlineinfo($user) }
    </service>
  </presence>
};
```

- ▶ Easy to import many services

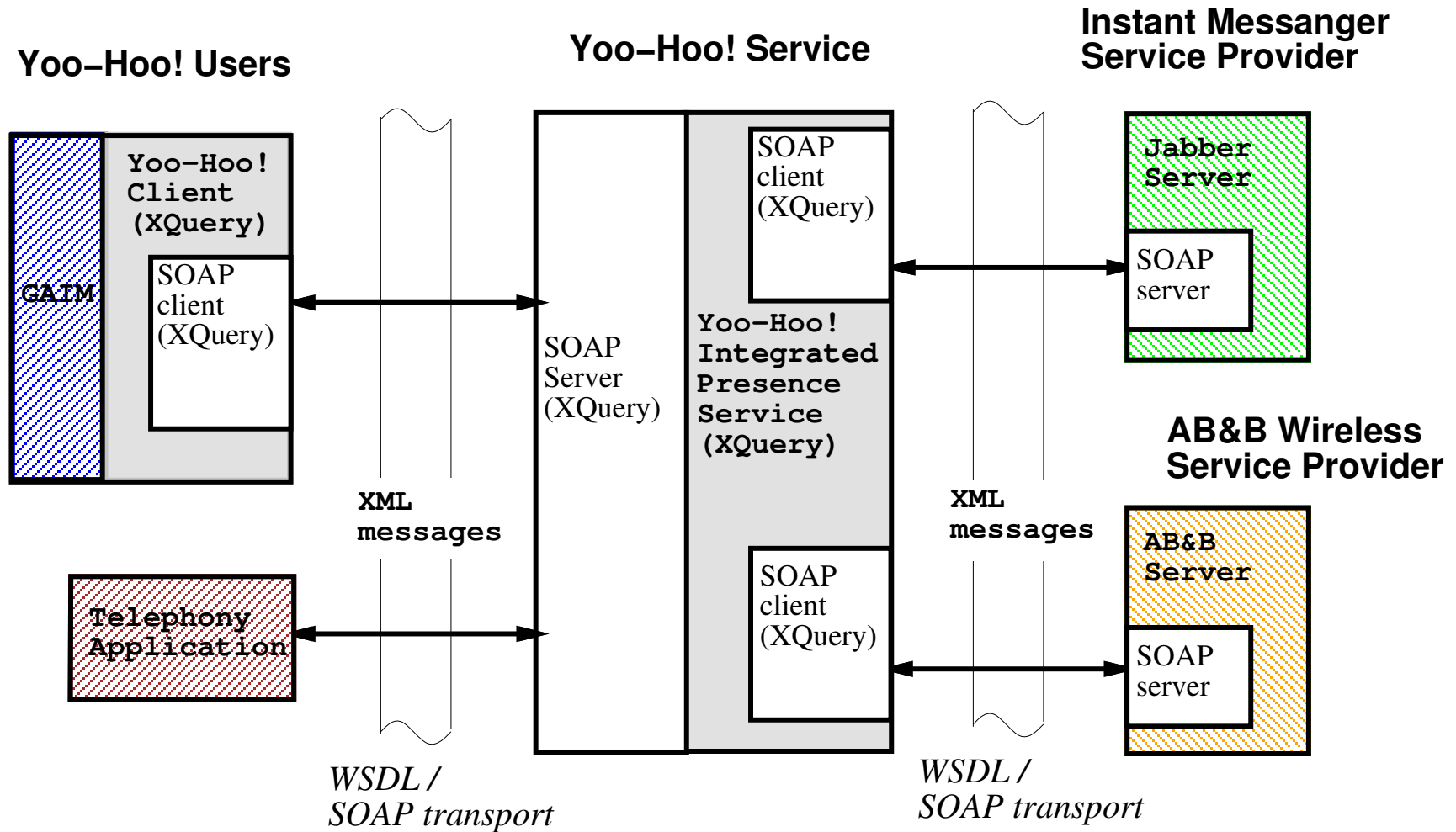
# One word about implementation

- ▶ Import service implemented as
  - ▶ Build a *stub* in XQuery itself
  - ▶ Stub deals with SOAP message encoding/decoding
  - ▶ Import that stub as any XQuery module!

# Deploying the service



# Deploying the service



# Deploying the service

```
xquery2soap
```

```
-installdir "/var/www/services"
```

```
-interfacedir "/var/www/services/wsdl"
```

```
-address "http://YooHoo.net/YooHoo.xqs"
```

```
YooHoo.xq
```

- ▶ Generates a WSDL if necessary
- ▶ Generates a SOAP/http stub
- ▶ Registers the service with an apache server

## Part IV

XQuery at your Web Service

—

WSDL / XQuery Binding

# WSDL / XQuery Binding

- A WSDL portType is mapped to an XQuery module:

```
[ <portType>Operations</portType> ]Bind  
== [ Operations ]Bind
```

- A WSDL operation is mapped to an XQuery function:

```
[ <operation name="QName">  
  Input Output  
</operation> ]Bind  
==  
declare function QName([ Input ]Input)  
[ Output ]Output external
```

# WSDL / XQuery Binding

- A WSDL input message is mapped by mapping each of its parts to an XQuery variable declaration:

$$[\langle \text{message} \rangle \textit{Parts} \langle / \text{message} \rangle ]_{\text{Input}} \\ == [ \textit{Parts} ]_{\text{Input}}$$
$$[\langle \text{part name} = \text{"QName"} \ \textit{Type} / \rangle ]_{\text{Input}} \\ == \$\text{QName} \text{ as } [ \textit{Type} ]_{\text{Type}}$$

- A WSDL output message is mapped only if it has a single part, which is mapped to an XQuery sequence type:

$$[\langle \text{message} \rangle \textit{Part} \langle / \text{message} \rangle ]_{\text{Output}} \\ == [ \textit{Parts} ]_{\text{Output}}$$
$$[\langle \text{part name} = \text{"QName"} \ \textit{Type} / \rangle ]_{\text{Output}} \\ == [ \textit{Type} ]_{\text{Type}}$$

# WSDL / XQuery Binding

- A WSDL part type is mapped depending on the method used to identify the type, into an XQuery sequence type:

$$[\text{element}=\text{"QName"}]_{\text{Type}} \\ == \text{element}(\text{QName})$$
$$[\text{type}=\text{"AtomicType"}]_{\text{Type}} \\ == \text{AtomicType}$$
$$[\text{type}=\text{"ListType"}]_{\text{Type}} \\ == \text{AtomicType}^*$$

Where *AtomicType* is the atomic type from which the list type is derived.

$$[\text{type}=\text{"ComplexType"}]_{\text{Type}} \\ == \text{element}(*, \text{ComplexType})$$

# What is missing?

- ▶ Mismatch between WSDL types and XQuery sequence types
  - ▶ “WSDL” types do not have any equivalent in XQuery
  - ▶ \*, +, ? in sequence types cannot be expressed in WSDL
- ▶ WSDL not yet quite a module interface
  - ▶ Global variables in XQuery modules do not have equivalent in WSDL
- ▶ XQuery is functional, but Web services may not be
  - ▶ Need for XML updates
  - ▶ Need for exception handling
  - ▶ Need to deal with asynchronous calls
  - ▶ etc.

Part V

Conclusion

# Conclusion

XQuery + XML Schema + WSDL + SOAP

=

an XML-native, distributed, computing platform

# Web Services in Galax

- ▶ Approach implemented in Galax
- ▶ Galax is complete, open-source, XQuery 1.0 implementation
  - ▶ Infos and downloads are available at:

<http://www.galaxquery.org/>

Complete description of the system in: “XQuery at your Web Service”, Nicola Onose & Jérôme Siméon, WWW'2004, May 2004, New York.