VisualAge Smalltalk Web Services Experience Report

Alison Dixon dixona@attglobal.net

Current Environment

- Large financial application
- Large user base spread over wide geographic area
- VAST 6.0 Fat Client
- OS/2 moving to Windows XP
- OS/2 server for near future

Current Environment Requirements

- High uptime
- High speed
- Peripheral support

Why Web Services?

- Re-use of current business logic
- Access to information from external and internal providers
- Possible migration path to newer client technologies

Web Service Prototypes

- Simple string concatenation
- Insurance example
- Client locate - complex input and output
- External service consumption with complex input and output

Various Client Types

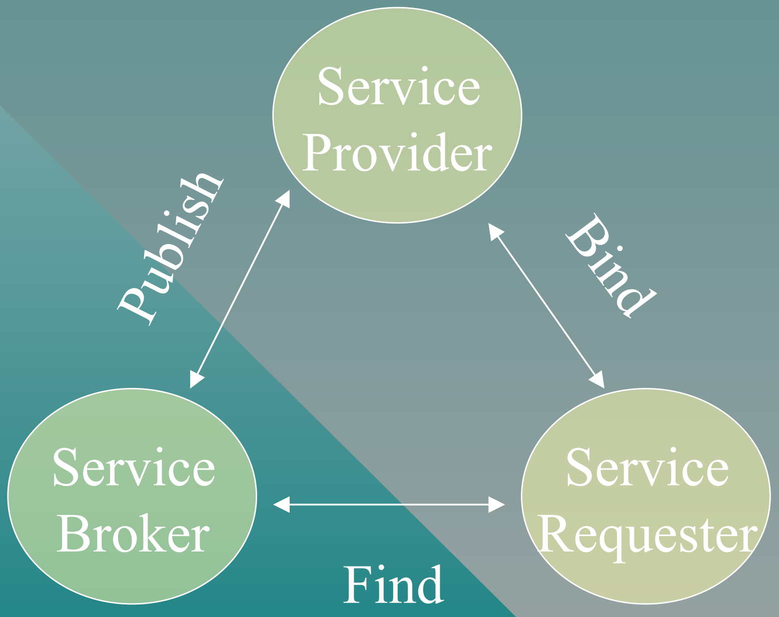
- VA client
- C# .net
- ASP.net
- Java thick and thin (created using WSAD)

What is a Web Service?

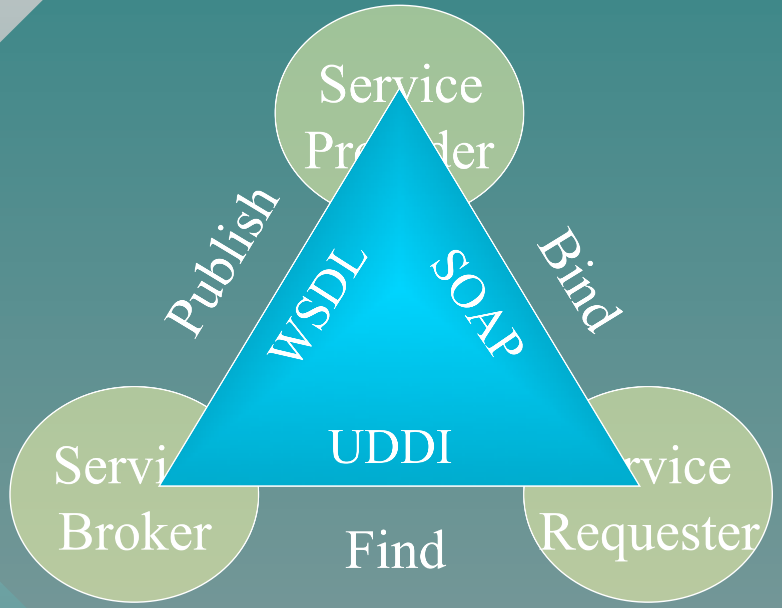
- Web Services are self-contained, modular applications that can be:
  - Described
  - Published
  - Found
  - Bound
  - Invoked
  - Composed

Web Services

- How does it work?



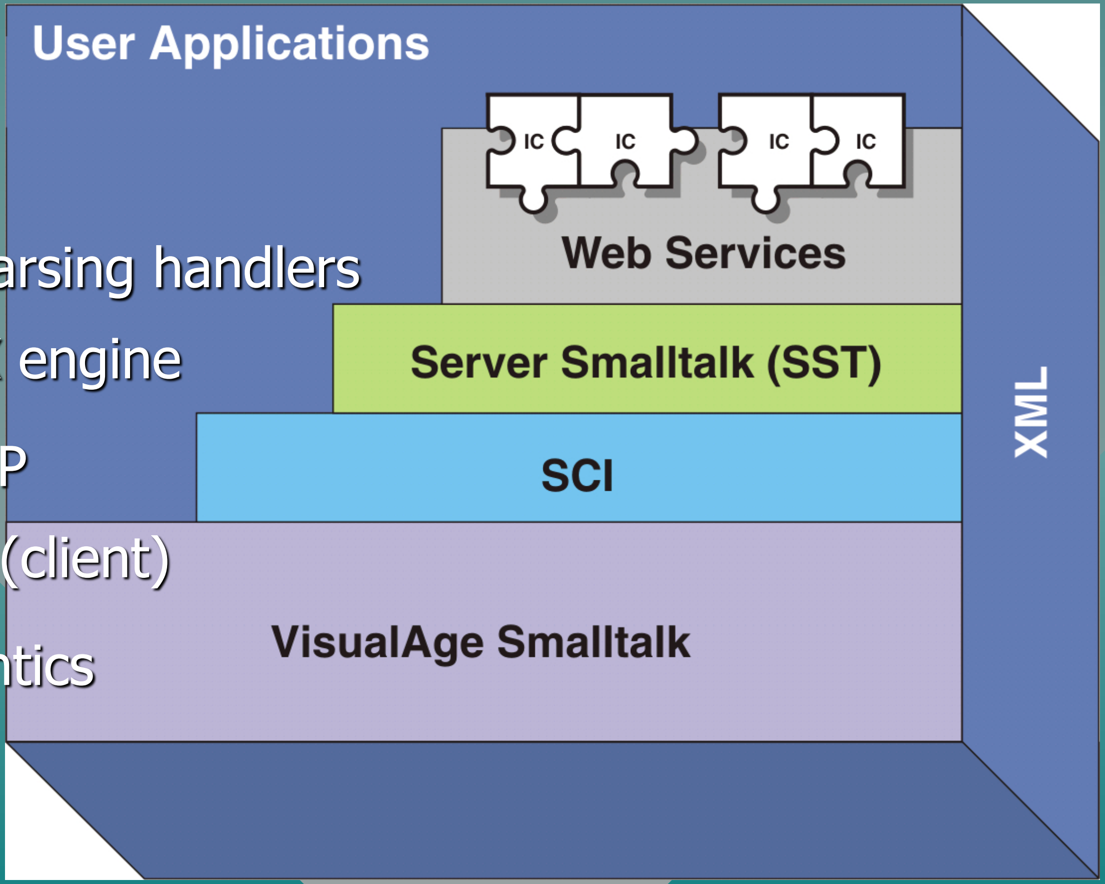
Service Oriented Architecture



- Web Service Definition Language: an XML based interface definition language for network based services
- Universal Description Discovery & Integration: a standards based architecture specification for service description and discovery. (www.uddi.org)
- Simple Object Access Protocol: a lightweight XML based protocol for the exchange of information in a decentralized, distributed environment.

VA Web Services Platform

- SST
  - Transports
  - Dispatching
- XML
  - Schema based SAX parsing handlers
  - Uses the existing SAX engine
- Supports SOAP over HTTP
  - 0.40 beta has HTTPs (client)
- Maintain Smalltalk Semantics
  - Messaging
  - Faults



The Files

- .wsdl (parent and imported)
- .xml (client and server)
- .map (client and server)

WSDL

- Allows separation of implementation and interface
- High-level WSDL has interface specs – location of service
- Imported file contains schema and service messaging specs

XML

- Allows specification of the Smalltalk deployment
- Define mapping specs and wsdl
- Custom handlers
- Able to specify client and server side
- Allows user to specify whether service will run remote or local

Mapping

- Allows conversion between web service elements or types and Smalltalk classes
- Useful for mapping between Array and OrderedCollection

Web Services Container

- The key element in the implementation, the Services Container is responsible for the storage and manipulation of all information relating to deployed Web services
- Client and server have their own separate containers

Serialization Manager

- Makes available cached information (retrieved when the service was deployed) that enables serialization and deserialization of SOAP messages
- Responsible for loading/storing XML Mapping specs named in VA ST Web services deployment descriptor
- Responsible for loading/storing XML schemas referenced in deployed WSDL
- Two level resolution of resources
  - First me
  - Then AbtXmlObjectCache

Handlers

- VA web services supports various handlers
- Add to deployment descriptor for customized processing

Current Status

- Functioning prototypes with various clients
- Packaged client runtime

Struggles Along the Way

- Proxy server
- Use of arrays in C# and Java
- Multi-references in C# - interoperability
- Organizing the files and discovering the purposes
- Beta code

Proxy Server Authentication

- Setting of credentials with Base64 encoding

```
(SstTransport configurationRegistry at: 'http')
proxyCredentials: 'Basic
   CCQwXXXNNN5kaWRR'.
```

- Setting of proxy url

```
(SstTransport configurationRegistry at: 'http')
proxyUrl: ('http://proxy:8080') sstAsUrl.
```

Content of an External WSDL

```
(SstXmlResourceReader new fetch:
('http://www.xmethods.net/sd/
   TemperatureService.wsdl' sstAsUrl)) inspect.
```

```
(SstXmlResourceReader new fetch:
('http://www.alethea.net/webservices/
   zipcode.asmx?wsdl' sstAsUrl)) inspect.
```

Invoking Remote Web Services

```
[ | aContainer aServiceCollection|
aContainer := SstWSContainer containerNamed:
    SciSocketManager default getHostName.
aServiceCollection := aContainer deploy:
    'http://www.xmethods.com/sd/StockQuoteService.wsdl'.
( aServiceCollection first getQuote: 'QCOM' ) inspect] fork
```

Troubleshooting

- Settings for disabling trapping of exceptions
  - Tools > SST > Trap exceptions
  - Tools > SST > Forward exceptions
- Forking of actions

Use of Handlers for Performance Timings

- How we took timings
  - Use of web service handlers
  - Addition of performance info to the wsdl
- What we learned about our application

Client-side Output Handler

- Add a handler to the client container

```
| chain |

chain := (SstWSContainer containerNamed:
'EnterpriseBankerServices') handlerFactory
handlerNamed:
'wsClientInputMessageConstructor'.

chain addHandler:
(WcWSClientContainerOutputHandler new
name: 'containerOutputHeaderHandler')
```

Client-side Output Handler

- The following adds information to the SOAP header

```
invoke: anSstWSMessageContext
	" Write performance timings to the output message."
	| containerHeaderElement |
	containerHeaderElement := self newElementFrom: anSstWSMessageContext.
	containerHeaderElement
		timeParsingAtClient:  Time now asMilliseconds asString;
		timeInvokedAtClient: WcPerformanceTimings timeInvokedAtClient.

anSstWSMessageContext currentMessage addHeaderElement:
	containerHeaderElement
```

Client-side Input Handler

- Add a handler to the client side xml file

```
<handlers namespace="urn:vastPerformanceGlobals">
                                <handler name="vastPerformance"
      class="WcWSClientContainerInputHandler"/>
</handlers>
```

- Add code to the invoke: method of your class 'WcWSClientContainerInputHandler'

```
invoke: anSstWSMessageContext
      " Perform any client-side handler processing "

| performanceInfo |
performanceInfo := anSstWSMessageContext propertyNamed: WSCustomHandlerElement.
WcPerformanceTimings timeInvokedAtClient: performanceInfo      timeInvokedAtClient;
timeParsingAtClient: performanceInfo timeParsingAtClient;
```

……….(cont)

Server-side Input Handler

- Add a handler to the server side xml file

```
<handlers namespace="urn:vastPerformanceGlobals">
                <handler name="vastPerformance"
        class="WcWSServerContainerInputHandler"/>
        </handlers>
```

- Add code to the invoke: method of the above class

Server-side Output Handler

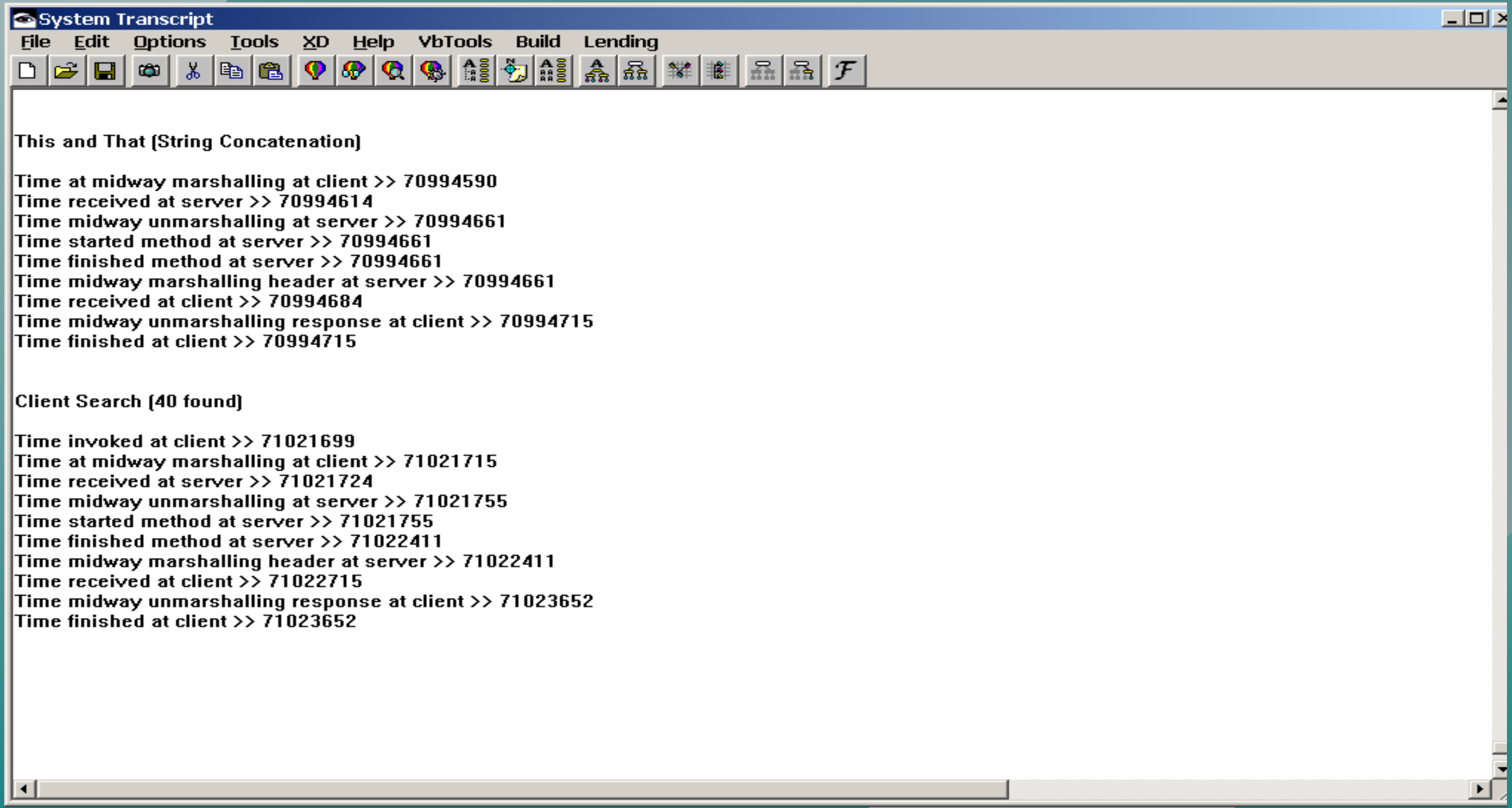
- Add a handler to the server container

```
<handler
   name="wsGlobalResponseServerHandler"


   class="WcEBankerContainerHeaderOutpu
   tHandler"/>
```

- Add code to the invoke: method of the above class

Performance Timings



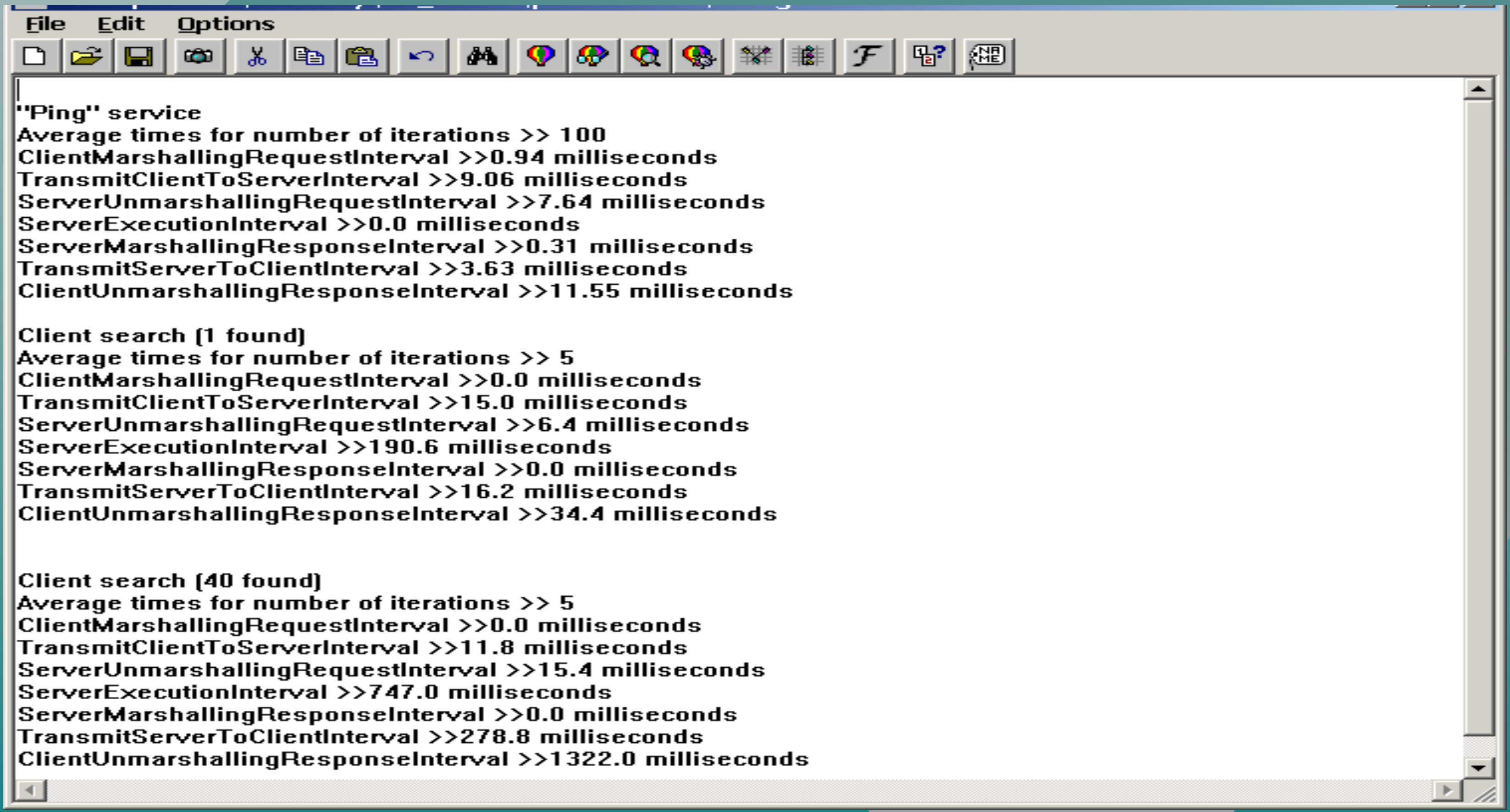
```
This and That (String Concatenation)

Time at midway marshalling at client >> 70994590
Time received at server >> 70994614
Time midway unmarshalling at server >> 70994661
Time started method at server >> 70994661
Time finished method at server >> 70994661
Time midway marshalling header at server >> 70994661
Time received at client >> 70994684
Time midway unmarshalling response at client >> 70994715
Time finished at client >> 70994715


Client Search (40 found)

Time invoked at client >> 71021699
Time at midway marshalling at client >> 71021715
Time received at server >> 71021724
Time midway unmarshalling at server >> 71021755
Time started method at server >> 71021755
Time finished method at server >> 71022411
Time midway marshalling header at server >> 71022411
Time received at client >> 71022715
Time midway unmarshalling response at client >> 71023652
Time finished at client >> 71023652
```

Performance Timings cont.



```
"Ping" service
Average times for number of iterations >> 100
ClientMarshallingRequestInterval >>0.94 milliseconds
TransmitClientToServerInterval >>9.06 milliseconds
ServerUnmarshallingRequestInterval >>7.64 milliseconds
ServerExecutionInterval >>0.0 milliseconds
ServerMarshallingResponseInterval >>0.31 milliseconds
TransmitServerToClientInterval >>3.63 milliseconds
ClientUnmarshallingResponseInterval >>11.55 milliseconds

Client search (1 found)
Average times for number of iterations >> 5
ClientMarshallingRequestInterval >>0.0 milliseconds
TransmitClientToServerInterval >>15.0 milliseconds
ServerUnmarshallingRequestInterval >>6.4 milliseconds
ServerExecutionInterval >>190.6 milliseconds
ServerMarshallingResponseInterval >>0.0 milliseconds
TransmitServerToClientInterval >>16.2 milliseconds
ClientUnmarshallingResponseInterval >>34.4 milliseconds


Client search (40 found)
Average times for number of iterations >> 5
ClientMarshallingRequestInterval >>0.0 milliseconds
TransmitClientToServerInterval >>11.8 milliseconds
ServerUnmarshallingRequestInterval >>15.4 milliseconds
ServerExecutionInterval >>747.0 milliseconds
ServerMarshallingResponseInterval >>0.0 milliseconds
TransmitServerToClientInterval >>278.8 milliseconds
ClientUnmarshallingResponseInterval >>1322.0 milliseconds
```

Next Steps

- Consuming a web service from another department
- Implementing our framework for web services
- Serverizing our code