Exposing Web Services to CORBA Clients

Adrian Trenaman,

Senior Consultant, Global Services.

adrian.trenaman@iona.com

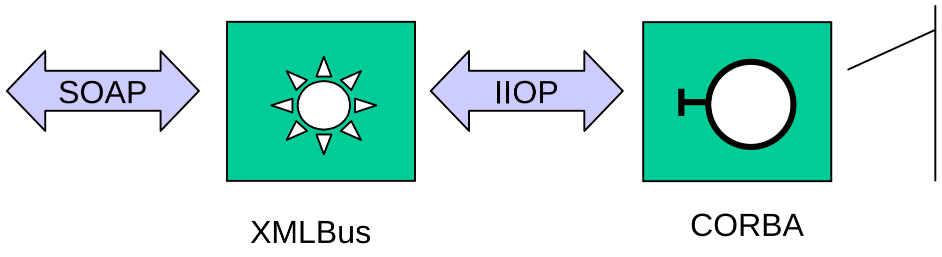
Overview

This presentation will:

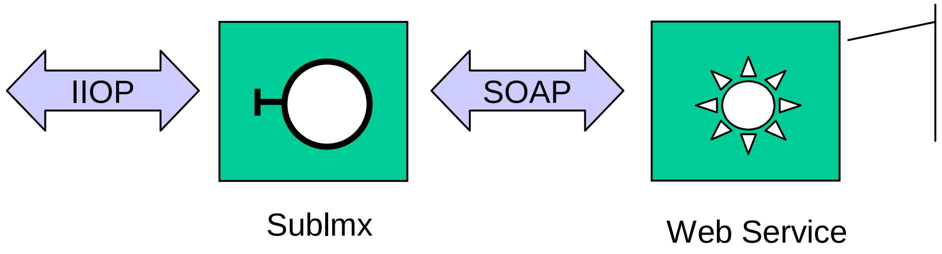
- Introduce an IIOP-SOAP adapter (SubImx), it's motivation, and the problem it solves.
- Discuss the architecture of SubImx
- Show SubImx features (security, configuration, object publishing)
- Discuss how to expose asynchronous web services to CORBA clients.
- Conclusion

The problem

- Expose web services to CORBA clients
- This differs from the well-addressed problem of "exposing CORBA services to Web Service clients": there are many SOAP-to-IIOP adapters, but not so many IIOP-to-SOAP adapters



This problem is well addressed with tools such as XMLBus



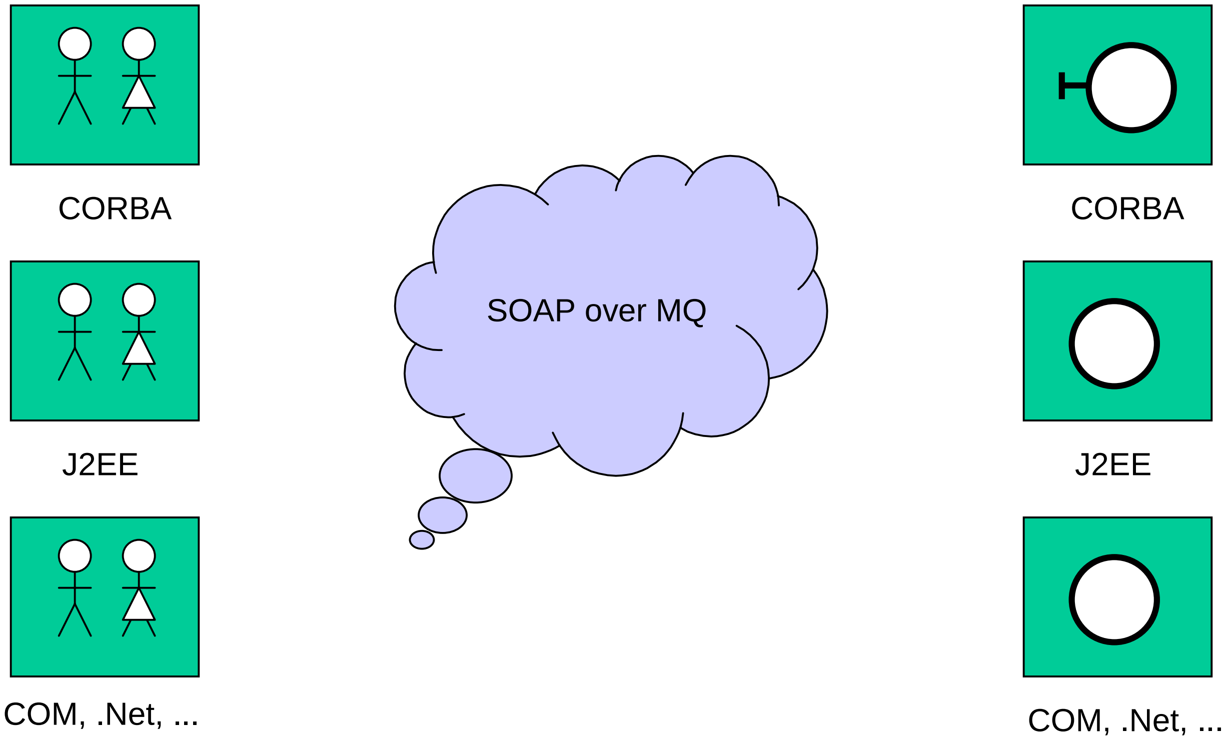
SubImx addresses the problem of exposing Web Services to CORBA clients

The Motivation

- Why expose Web Services to CORBA Clients?
  - Surely it would be easier for WS Client developers to use SOAP toolkits?
- We were approached by a customer with a vision for a Web Services based middleware infrastructure.
  - Major CORBA backend systems, with highly service-oriented architecture
  - Diverse middleware already in place
  - They wished to expose all services as Web Services, and then transport SOAP messages over MQ-Series.
- The customer wished to maintain “middleware transparency” and let clients access new services without knowledge of service implementation details.

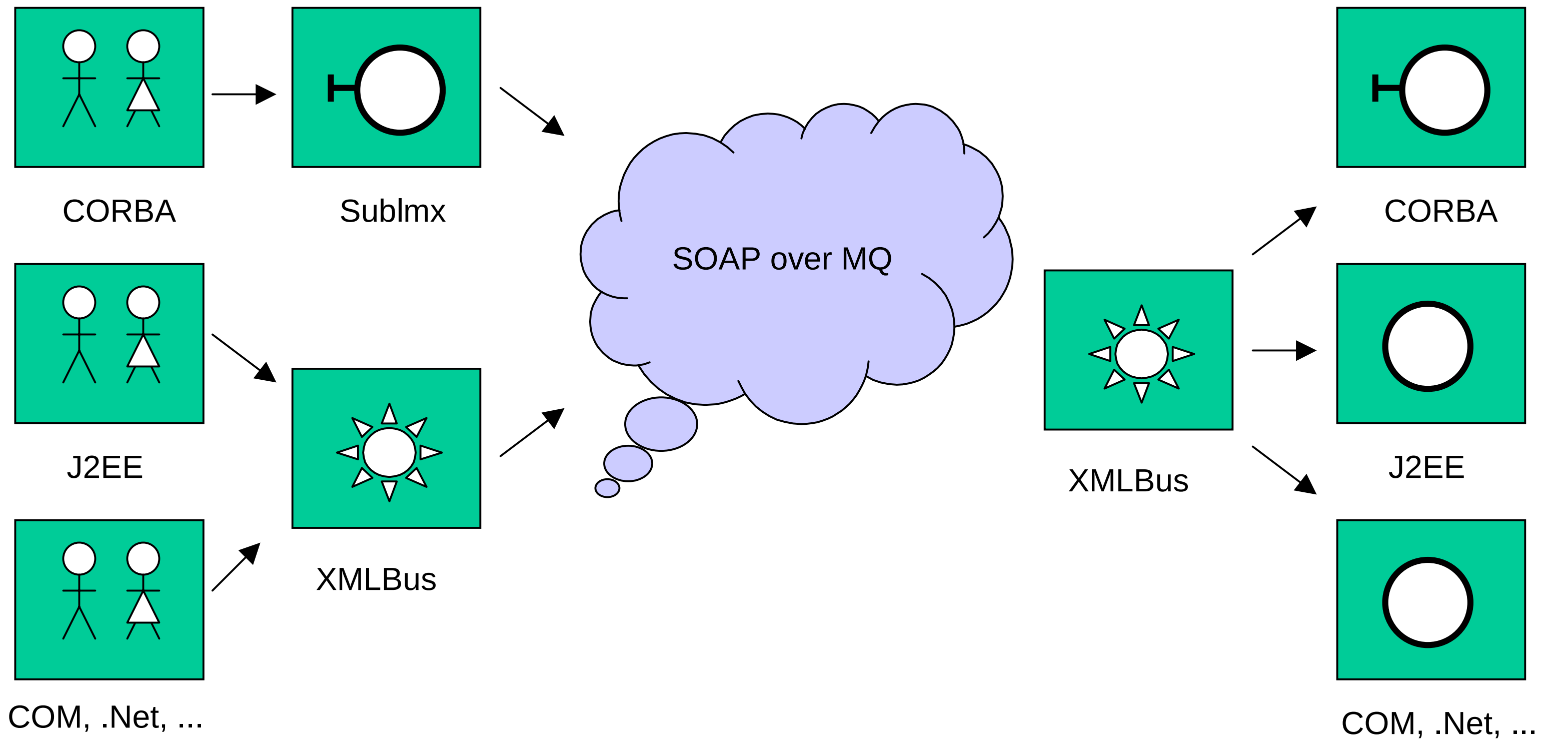
The Vision

The customers vision was a “soap-over-MQ” infrastructure.



The Solution

- IONA's XMLBus provided much of the functionality: but a new component, Sublmx, was created to allow IIOP-to-SOAP functionality.



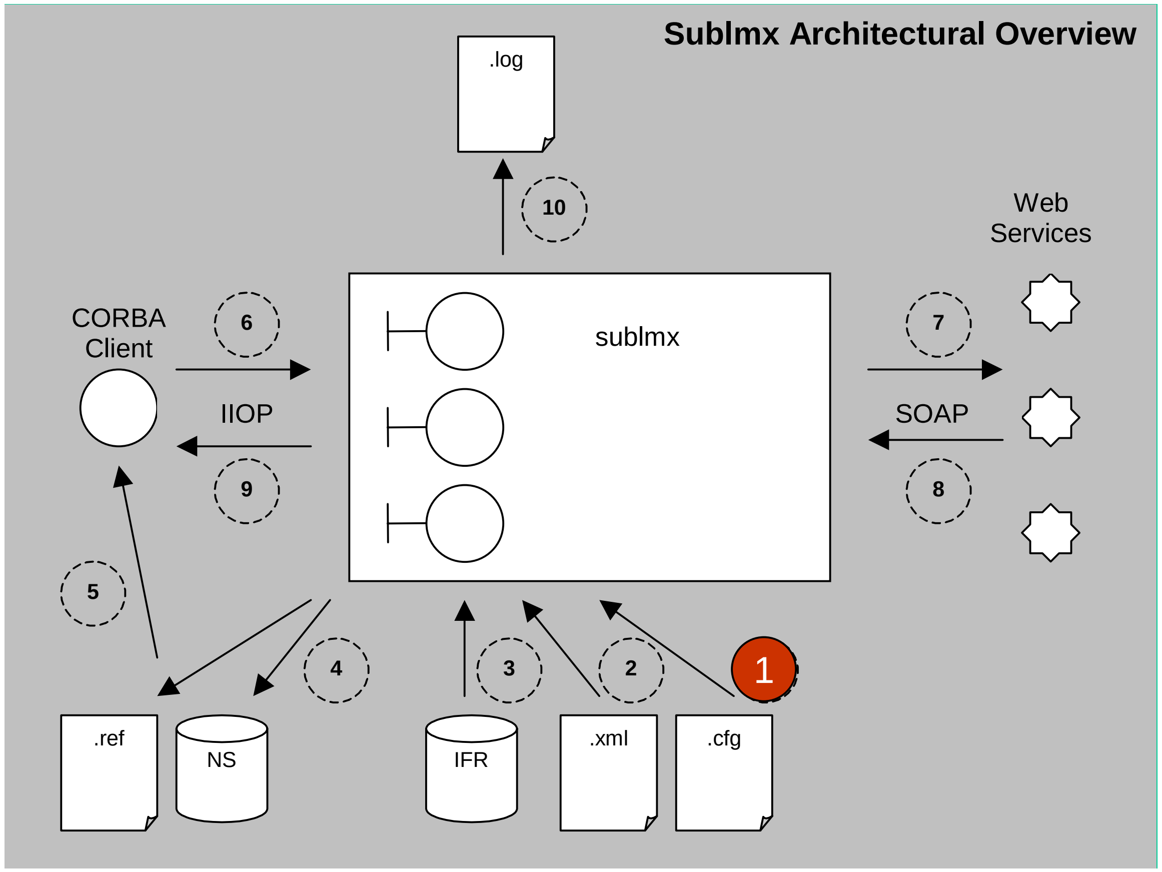
The Madness?

- On first viewing, this may look a little ambitious:
  - What about the overhead of transforming to SOAP, then back again, each way?!
  - What about the overhead of having a dynamic Java-based bridge (sublmx) using the DSI?!
  - What about marshalling of object references between different protocols?!
  - What about security & transactions?!
- Answer:
  - Performance needed to be acceptable in s-time, not ms-time.
  - The architecture was service-oriented, with no call-backs and no passing of object-references
  - Security addressed in a straightforward manner, transactions are a nice-to-have but not essential

What is Sublmx?

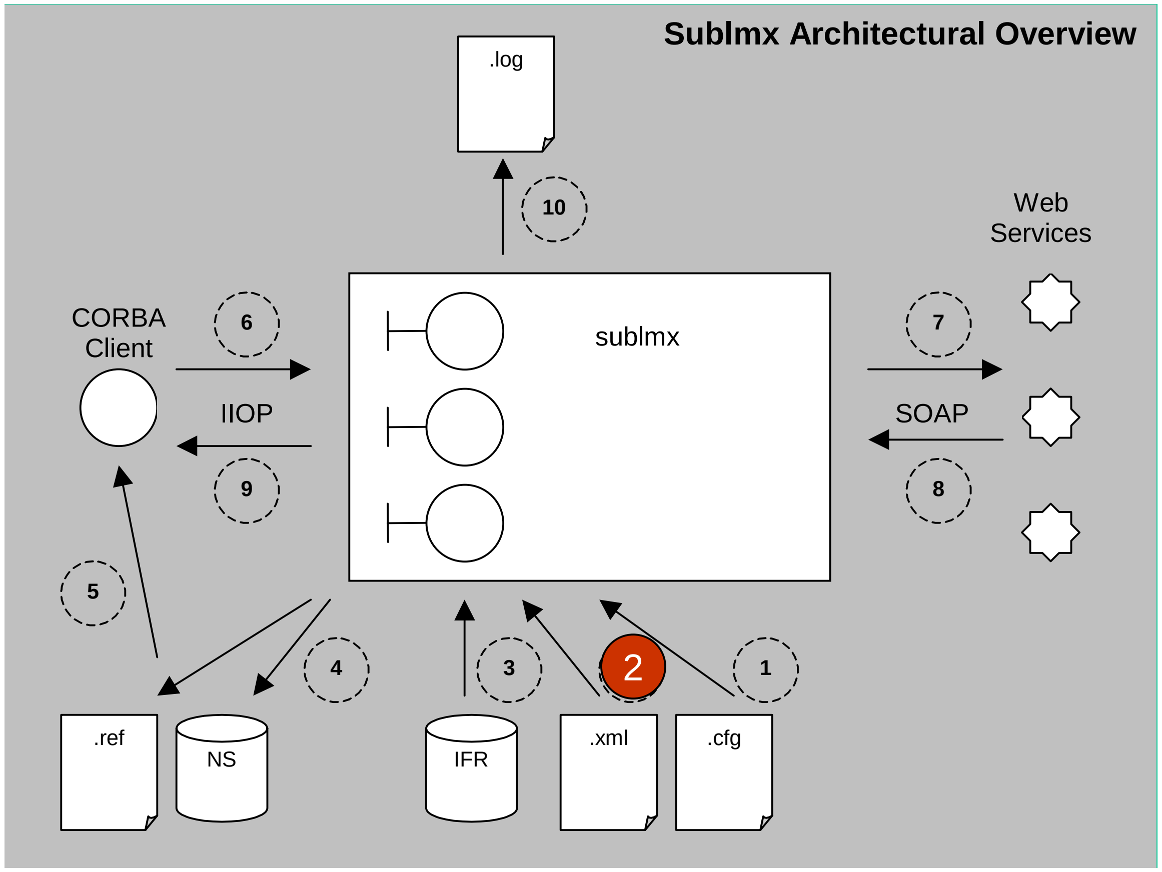
- Sublmx is an IIOP to SOAP bridge that allows CORBA clients to communicate with web services.
- It is implemented as a standalone CORBA server, and requires a CORBA interface repository (IFR).

SubImx Architecture



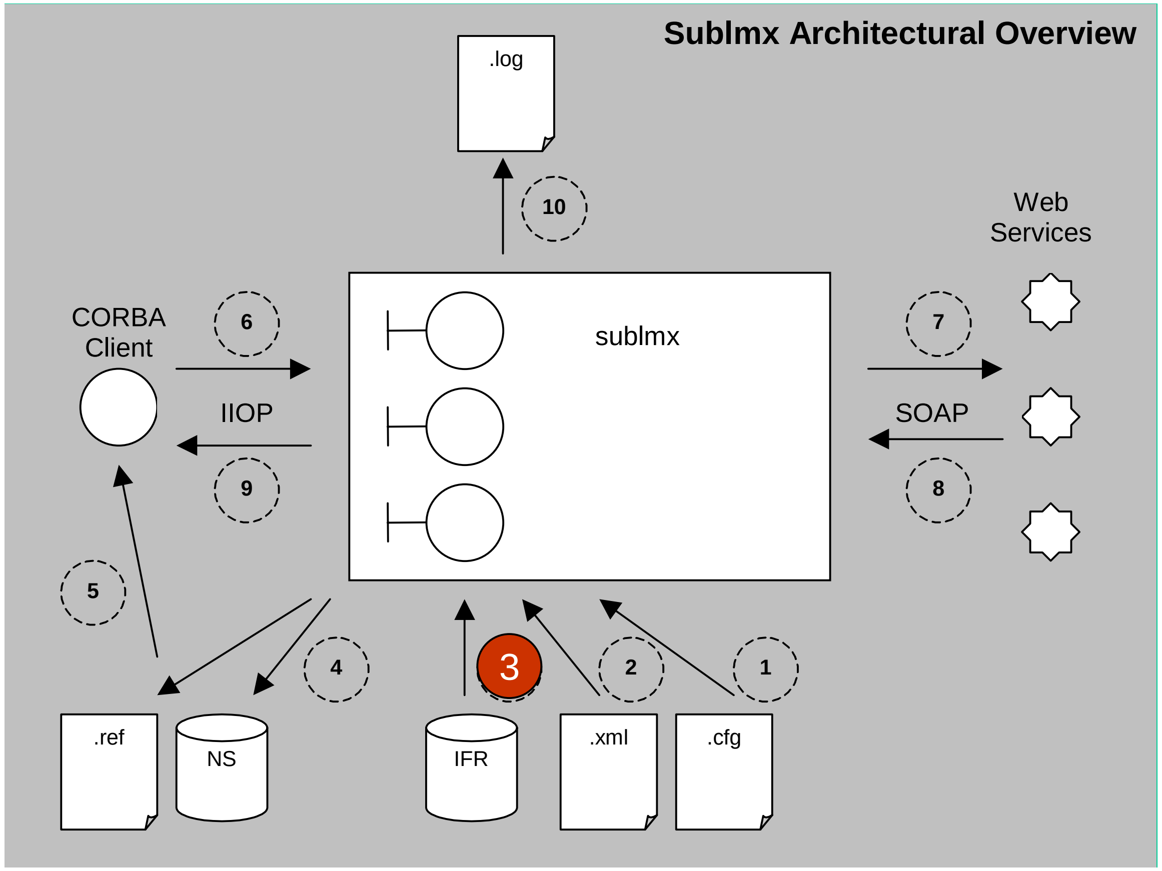
1. SubImx is configured on startup from settings in the Orbix ASP configuration file / CFR

Sublmx Architecture



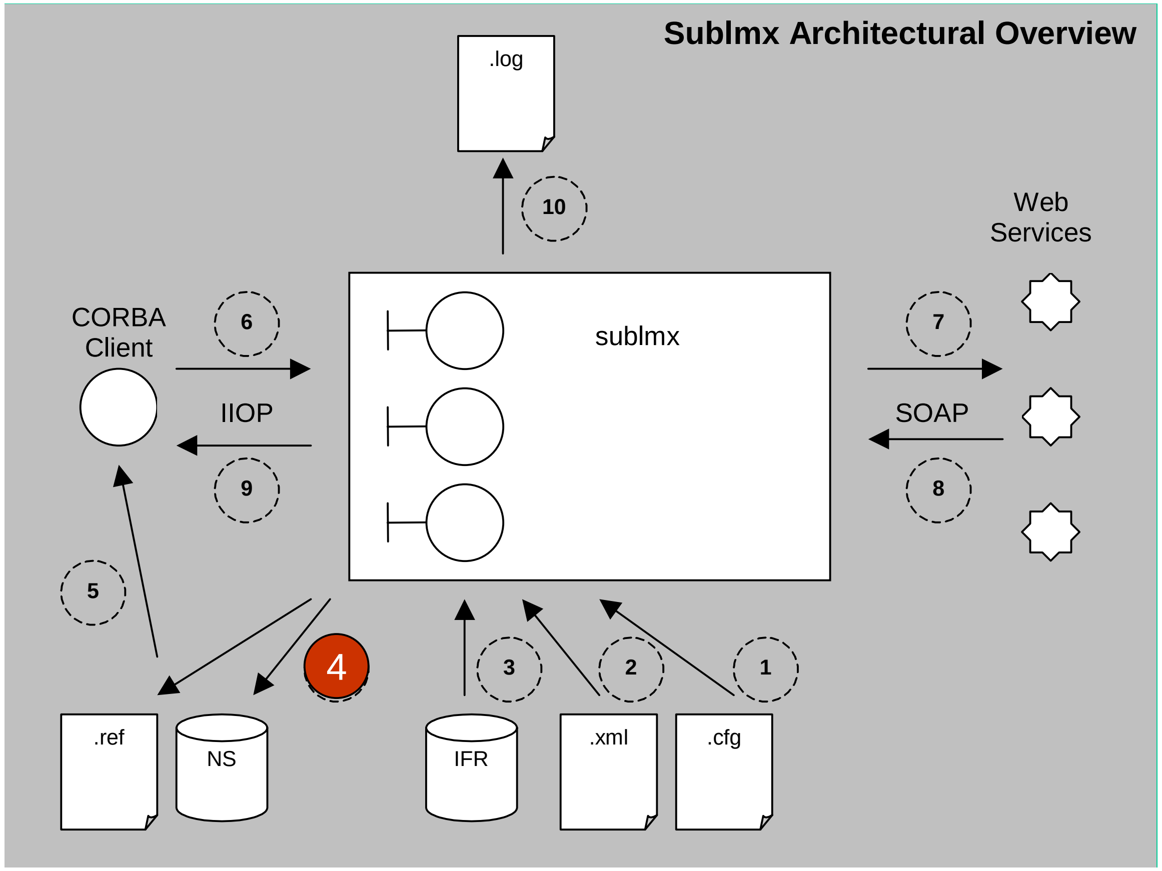
2. Information on the adapters to create are found in the sublmx configuration file, in xml format.

Sublmx Architecture



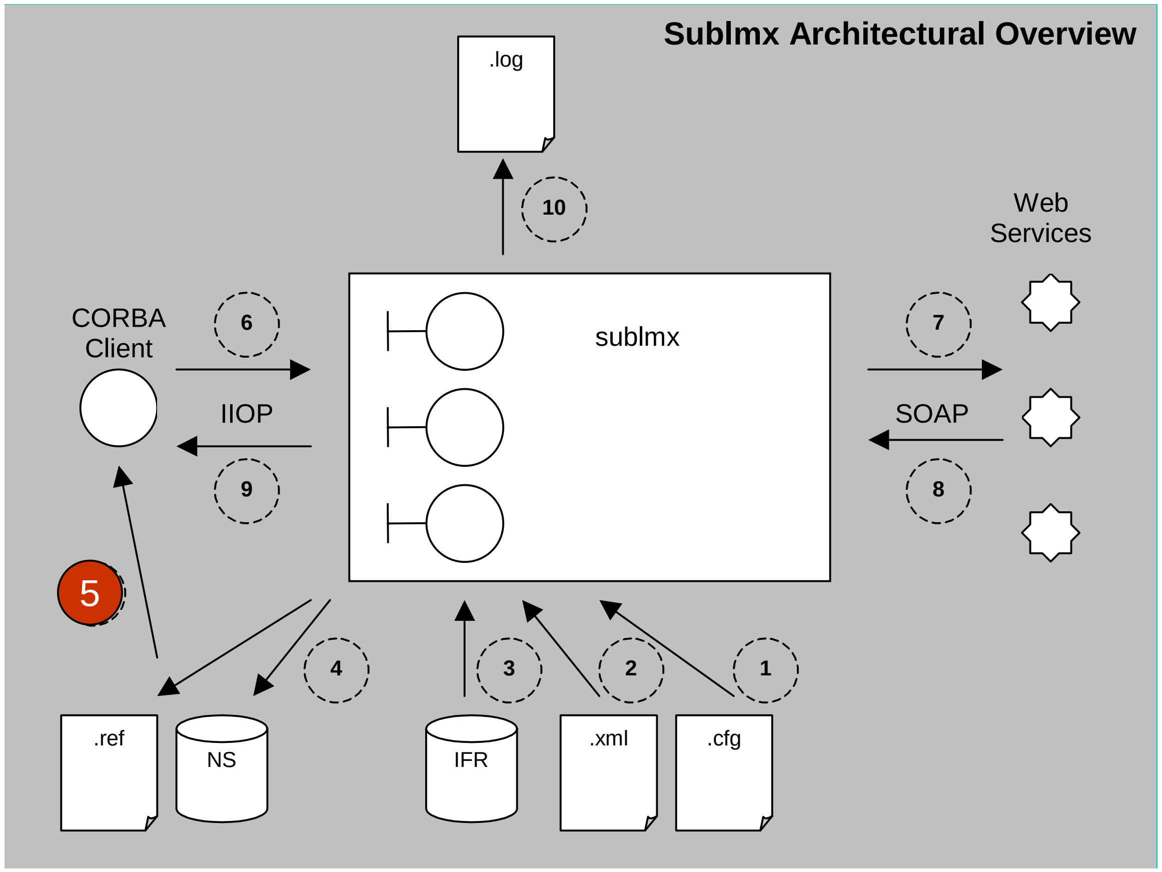
3. Interfaces are read from the IFR for each of the interfaces mapped.

Sublmx Architecture



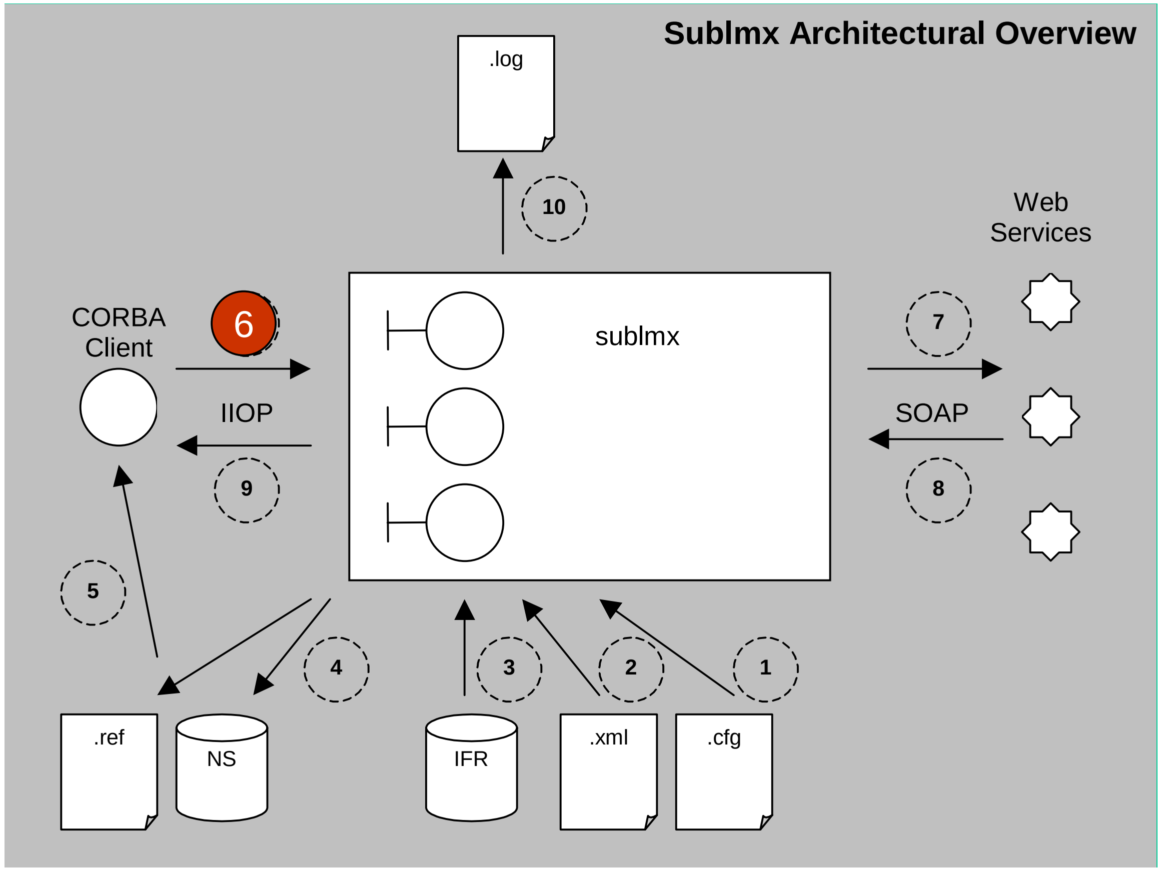
4. IOR's are published to file, Name Service or other publication mechanism, based on values in the sublmx configuration file.

SubImx Architecture



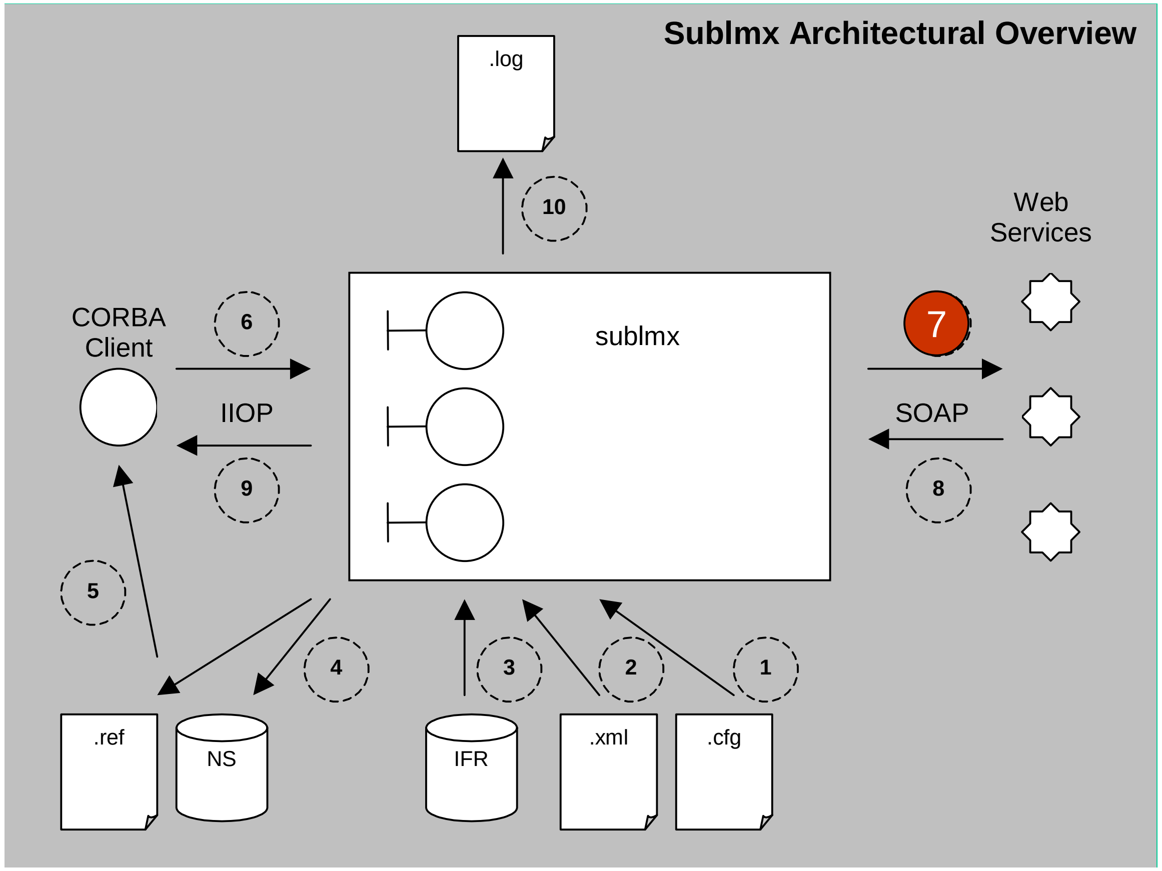
5. Client reads IOR.

Sublmx Architecture



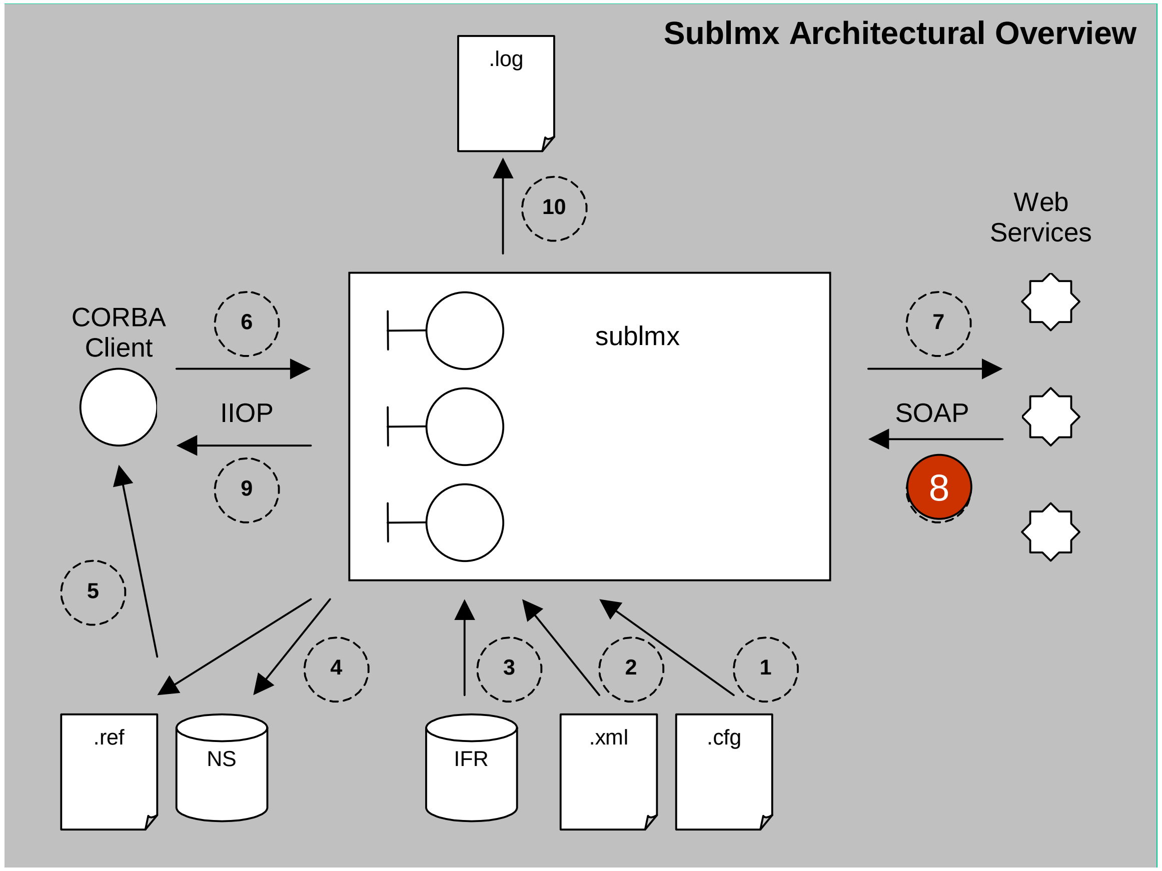
6. Client invokes on adapter

Sublmx Architecture



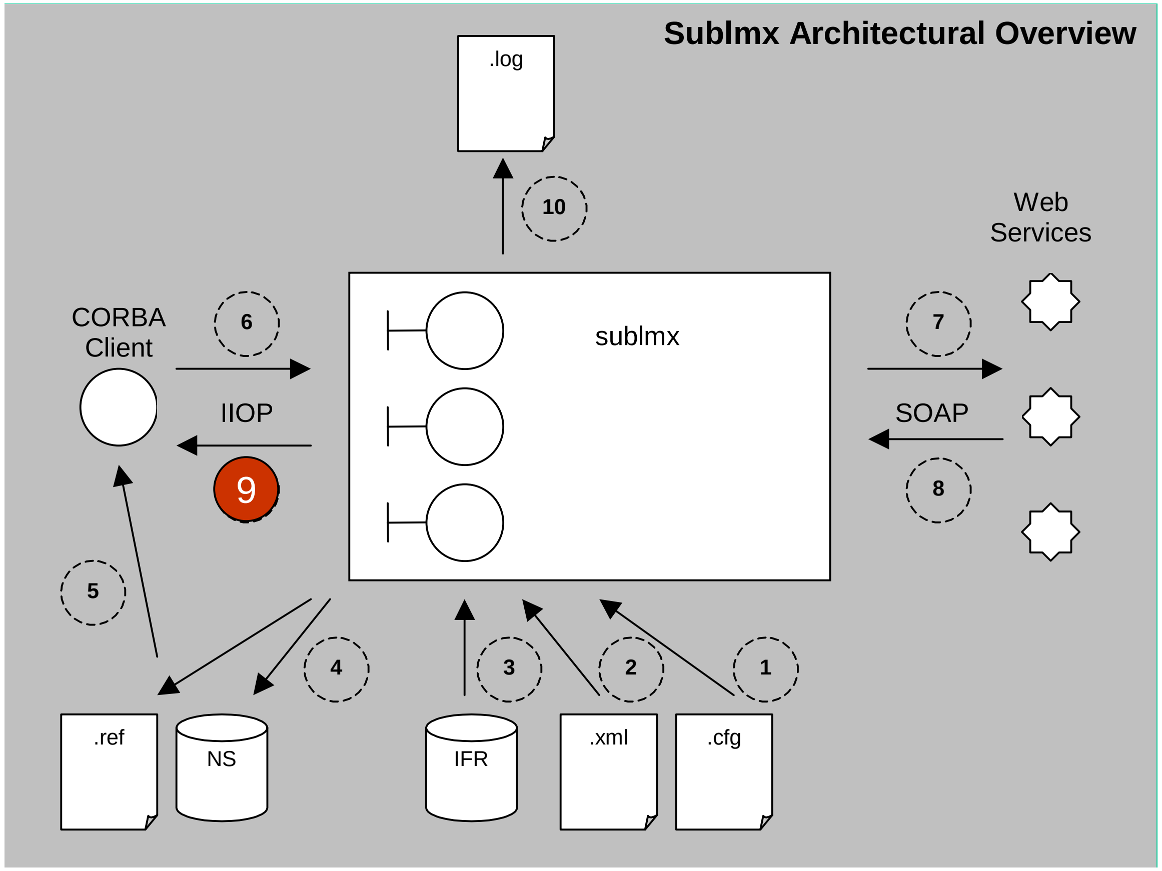
7. Message is transformed from IIOP to SOAP and sent to web service.

Sublmx Architecture



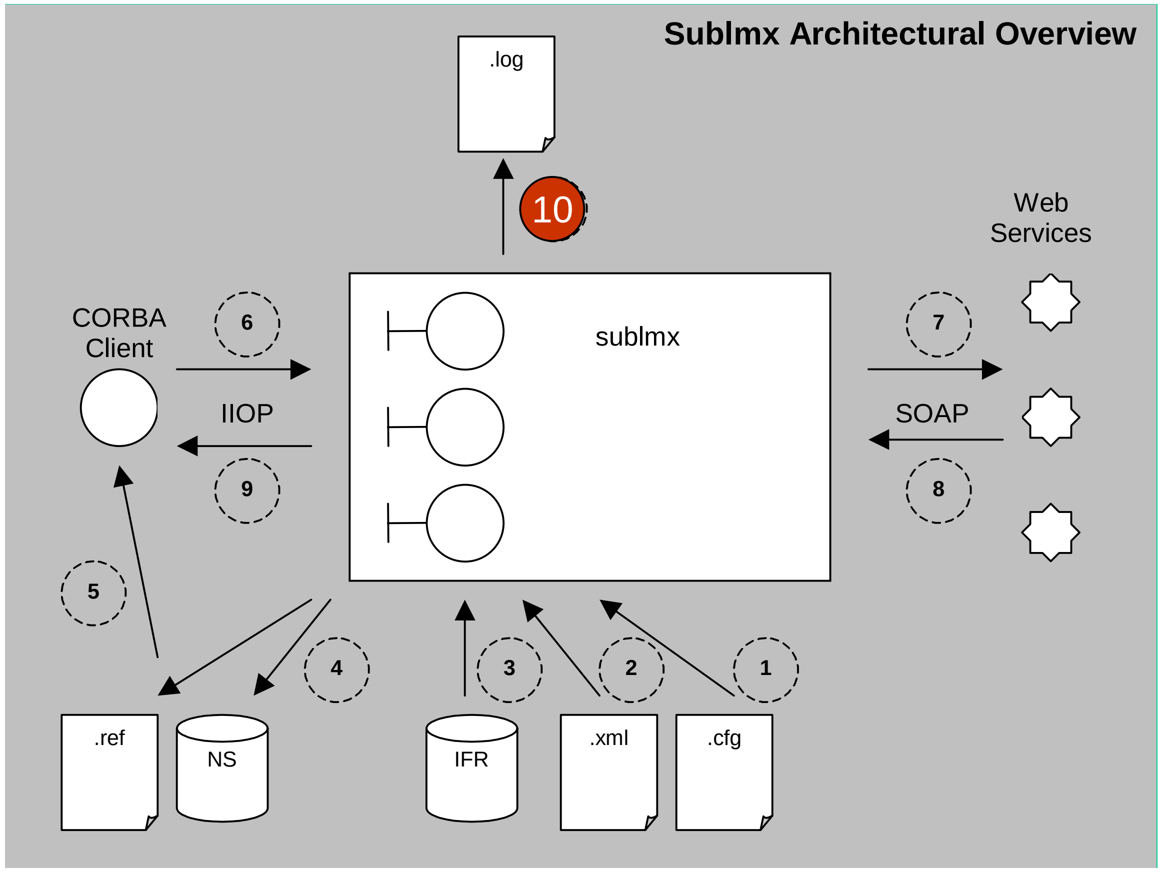
8. Web service responds

Sublmx Architecture



9. Response is transformed from SOAP to IIOP and sent to client.

Sublmx Architecture



10. Important events are logged using a configuration logging mechanism built on log4j.

Why use an adapter approach?

- It’s easy to expose Web Services to Java clients, but not so easy for C++ clients.
  - At the time of writing, there is no WSDL-to-C++ standard mapping.
- The customer wished to maintain “middleware transparency” and let CORBA clients access new services in a CORBA fashion.

SubImx Features

1. Support for all primitive IDL types, enums, sequences, structs and arrays
2. Support for methods inherited from base-interfaces.
3. Secure configuration (using SSL/TLS).
4. Extendable mechanism for extraction and retransmission of caller's credentials to web services using SOAP Security Headers.
5. Clustering of the SubImx server through Orbix ASP high-availability mechanisms.
6. Configurable logging of key events (using log4j)
7. Extendable mechanism of object reference publication

Sublmx configuration

- Configuration is performed through Orbix ASP configuration variables, for example:

```
sublmx {
    sublmx:logging:logfile = "c:\sublmx\logs\bubba.log";
    sublmx:adapter_config_file = “c:\sublmx\cfg\sublmx.xml”;
    sublmx:logging:log_to_console = "true";
    sublmx:logging:level = "DEBUG";
    sublmx:soap_security_header_helper_class =
      "com.iona.sublmx.UseCNFromClientCert";
};
```

Configuring Adapters

Adapters are instantiated by specifying them in the SubImx configuration file.

```
<config>
  <adapter>
    <interfaceName>
      **::echo::test_primitives**
    </interfaceName>
    <endpointAddress>
      http://localhost:8080/…/echoPrimitiveService/echoPrimitivePort/
    </endpointAddress>
    <publishingInstructions>
      file#h:\temp\echo_test_primitives.ref
    </publishingInstructions>
  </adapter>
</config>
```

Publishing IOR's

IOR's can be published in an extensible way using the `publishingInstructions` tag.

- "**file#\c:\iors\myior.ref**"- to the file system, in location c:\iors\myior.ref.
- "**name_service#/myservices/myIor**" - to the name service under the name "myservices/myIor"
- "**exec#itadmin ns bind -object -path myservices/myIor %IOR%"** using the system command specified. The IOR is inserted into the command at the placeholder "%IOR%": if the place holder is not present then it is added to the end of the command.
- "**java_class#com.mycompany.object_publisher**" - uses a custom algorithm implemented by the class com.mycompany.object_pulisher, an implementation of com.iona.corba.utils.custom_import_export.

Security

- As SubImx is an Orbix ASP server, security can be provided through Orbix ASP's SSL/TLS configuration.
- The release version of SubImx includes configuration scopes to allow subImx to be run in secure, semisecure or insecure modes.

SOAP Security Header Helpers (1)

- When a client invokes on sublmx, sublmx tries to place the calling client's identity into a SOAP Security Header.
- There are a number of different algorithms appropriate to determining the identity of the caller- for example the principal can be extracted from the caller's X509 certificate or from IIOP service context information.
- The algorithm that creates the security header for the SOAP message has been abstracted into a class as follows...

class SoapSecurityHeaderHelprer

```
public abstract class SoapSecurityHeaderHelper
{
    protected org.omg.CORBA.ORB orb_;

    public SoapSecurityHeaderHelper(org.omg.CORBA.ORB orb) {
        orb_ = orb;
    }
    public abstract void insertSecurityHeader
     (ServerRequest sr,
      SOAPMessage message);

}
```

SOAP Security Header Helpers (2)

- Concrete versions of this class implement different algorithms for determination of the principal and creation of the Security Context.
- The algorithm that gets used is determined in the configuration of Sublmx by the configuration variable `sublmx:soap_security_header_helper_class`, for example:

```
sublmx:soap_security_header_helper_class =
  "com.iona.sublmx.UseCNFromClientCert";
```

Exposing Asynchronous Web Services to CORBA Clients

- A web service is neither “synchronous” nor “asynchronous”.
  - However, its implementation may provide results in non-immediate time, and so may be better suited for asynchronous access.
  - In this case, it is desireable to provide an asynchronous interface for CORBA clients.
  - There is currently no provision made to export WSDL to IDL to support asynchronous invocations.
  - As a solution, consider an approach where a WSDL-to-IDL compiler would generate additional IDL methods for asynchronous calls (a technique similar to the CORBA Asynchronous Messaging Interface “implied IDL”)

Synchronous IDL

A WSDL-to-IDL compiler might generate the following synchronous IDL:

```
interface blah {
  long doSomething(in long x,
           inout long y,
           out long z);
};
```

Asynchronous IDL

The WSDL-to-IDL compiler would then generate the following method which will be used for asynchronous invocation of `doSomething()`.

The string returned is a message handle used to get the result later.

```
interface blah {
  long doSomething(in long x,
          inout long y,
          out long z);

  string asynch_doSomething(in long x,
                    in long y);
 };
```

Asynchronous result polling

Results would then be polled using a `asynch_poll_doSomething()` method:

```
interface blah {
  long doSomething(in long x,
           inout long y,
           out long z);

  string asynch_doSomething(in long x,
                   in long y); boolean

  asynch_poll_doSomething(string id, out long ret,
                   out long y, out long z);
 };
```

Asynchronous Result Callback (1)

For callback style asynchronous process, a callback could be registered using the `asynch_register_callback_doSomething()` method:

```
interface blah {
  long doSomething(in long x,
           inout long y,
           out long z);
  string asynch_doSomething(in long x,
                    in long y); boolean

  asynch_poll_doSomething(string id, out long ret,
                    out long y, out long z);

  void asynch_register_callback_doSomething(string id,
       blahCallback cb);
};
```

Asynchronous Result Callback (2)

Callbacks would implement the following generated interface, used by the adapter to send a return or exception response.

```
interface blahCallback
{
  void doSomething_return (in long ret,
                           in long y,
                           in long z);

  void doSomething_exception(ExceptionHolder
   exception_holder);
};
```

Conclusion

- This presentation has given an overview of the Sublmx IIOP-to-SOAP adapter
- The server can expose web services to CORBA clients and is mature enough to be deployed onsite in proof of concept scenarios.
- The server allows CORBA clients to treat Web Services as CORBA services in a transparent fashion, and enables secure transmission and principal delegation.

Additional Support Slides

Custom IOR Publishers

Custom IOR publishing algorithms can be created by implementing the interface `com.iona.corba.utils.custom_import_export`, and then specifying this class name using the “java_class#” publishing instructions prefix.

```
public interface custom_import_export {

    public void export_obj_ref(org.omg.CORBA.ORB orb,
                               String instructions,
                               org.omg.CORBA.Object obj)
        throws import_export_exception;

    public org.omg.CORBA.Object import_obj_ref
        (org.omg.CORBA.ORB orb,
        String instructions)
        throws import_export_exception;
}
```

Custom IOR Publishers (2)

- As the entire instructions string is sent to the custom algorithm, the custom implementation can parse the string to obtain parameters.
- For example:

  “java_class#com.customer.factory_pattern -n blah -f -p”