Project Tango: Adding Quality of Service and .NET Interoperability to the Metro Web Services Stack

Version 1.0.2
Last Updated: Oct 12, 2007

Arun Gupta
http://blogs.sun.com/arungupta

Table of Contents

1.0 Introduction

This document provides an overview of Project Tango. Project Tango is an open source implementation from Sun Microsystems of the key enterprise Web services specifications, commonly known as WS-*, that provides interoperability with .NET 3.0.

This document is not intended to be a Web services tutorial and requires you to have a basic understanding of Web services. The references section provides a resource if you want to understand how to build a Web service.

2.0 What Is Project Metro?

Project Tango is a key component of Project Metro (also referred to as Metro), which is the Web services stack in GlassFish V2. To understand Project Tango better, it's important to be familiar with Metro.

Project Metro is the Web services stack from Sun Microsystems. This stack is integrated in GlassFish V2, a high-performance, production-quality, Java Platform, Enterprise Edition (Java EE) 5 technology-compatible Application Server. Figure 1 below shows all the components of Metro.

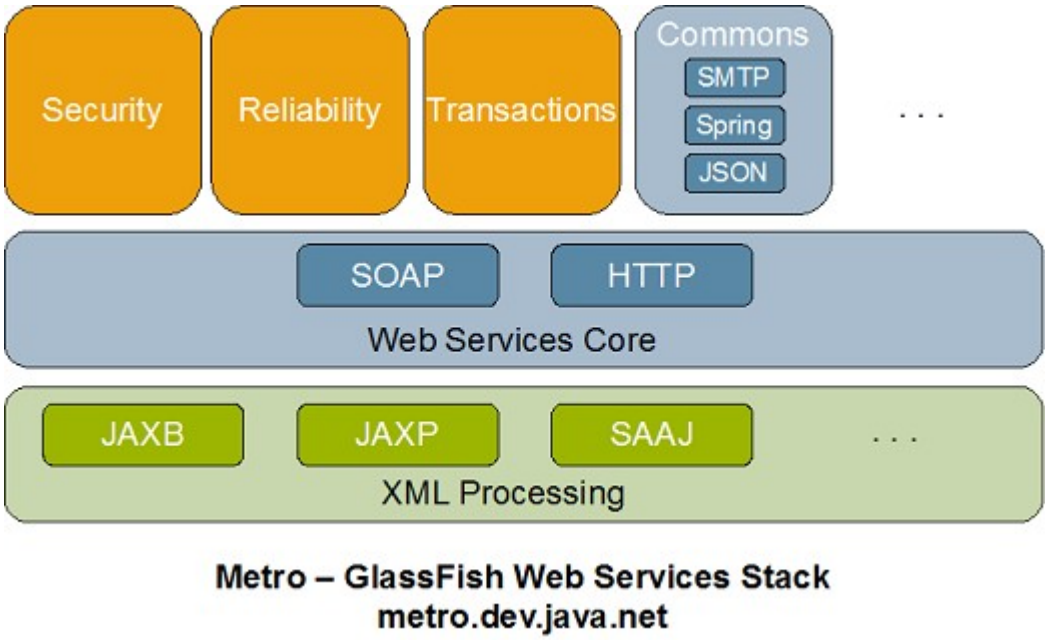

Figure 1: Metro – GlassFish Web Services Stack

The main components of Metro can be divided in two categories:

- JAX-WS RI – The core Web services platform
- Project Tango (also referred to as Tango) – An implementation of Reliability, Security, and Transactions WS-* specifications and interoperability with .NET 3.0

Java API for XML Web Services (JAX-WS) RI provides the Core Web services platform. This includes all the SOAP message functionality, including WS-Addressing and MTOM. The JAX-WS RI is an implementation of the JAX-WS specification that is developed as JSR 224 in the Java Community Process (JCP).

Project Tango implements support for Security, Reliability, and Transactions, using the protocols and mechanisms defined by several WS-* specifications, on this Core layer. This allows a Java client to communicate with a Java endpoint using these protocols. In addition, these protocols also enable interoperability with the Windows Communication Foundation component of .NET 3.0 framework.

JAX-WS has an extensible architecture and a Commons area to collect useful plug-ins and extensions around the JAX-WS RI. For example, SMTP Transport support is used to send and receive messages using email and JSON, instead of SOAP or Plain Old XML (POX), for encoding and decoding the messages. Several XML processing APIs, such as JAXB for XML<->Java Data Binding and JAXP for formulating and accessing the XML messages in clients and servers are also used.

All the applications on Metro can be easily developed using NetBeans 5.5.1 architecture and deployed on GlassFish V2.

This document explains Project Tango.

3.0 What Is Project Tango?

Project Tango (also known as WSIT for Web Services Interoperability Technology), an integrated part of GlassFish V2, is an open source implementation of the key enterprise Web services technologies, commonly known as WS-*. The technologies are divided into three main areas: Security, Reliability, and Transactions.

One of the main goals of Project Tango is to provide interoperability with Windows Communication Foundation, the Web services stack bundled with the .NET 3.0 platform.

Figure 2 shows a more detailed view of Figure 1. It shows the different WS-* specifications implemented by Project Tango in the areas of Security, Reliability, and Transactions. This figure also shows Metadata, a new component of Project Tango, which is explained below.

A common English-language idiom is “it takes two to tango.” Project Tango provides an implementation of key WS-* technologies with a focus on interoperability with the .NET 3.0 framework. And so the name signifies a tango, or dance, between the two Web services stacks. The logo shows Duke striking a pose on a Tango dance floor with a rose in his mouth.

In Security, WS-Security 1.0 and 1.1 provide a basic framework for SOAP message level security in Web services. WS-Trust defines a framework for issuing, renewing, and validating security tokens, and to broker trust relationships within different trust domains. WS-Secure Conversation increases the overall performance and security by defining semantics for secure message exchange for multiple message exchanges. WS-Security Policy enables Web service endpoints to specify their security requirements to potential clients in an interoperable manner.

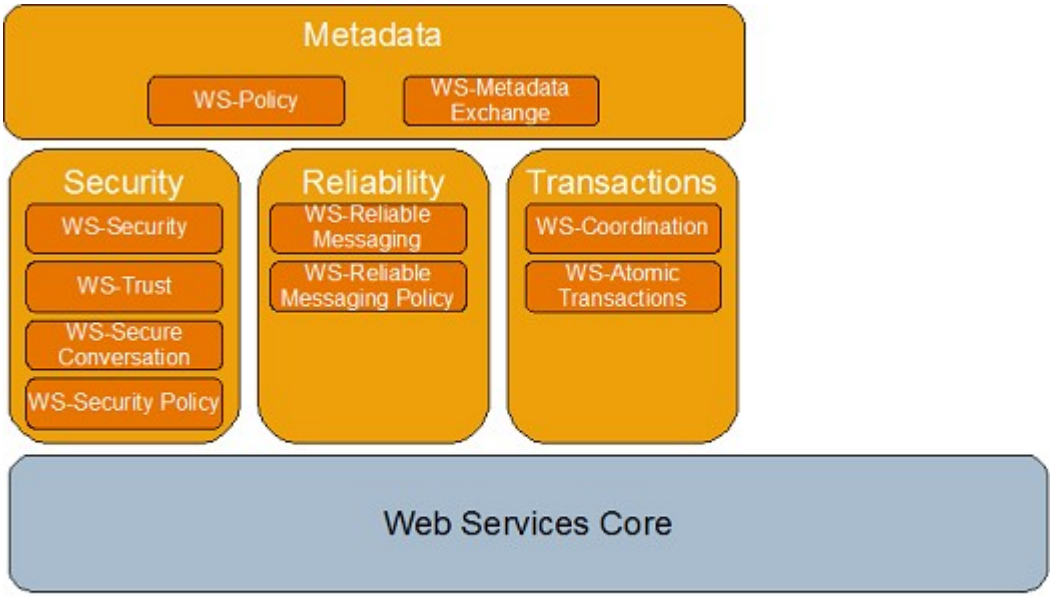

Figure 2: Project Tango – Key Components

In Reliability, WS-Reliable Messaging defines a messaging protocol to identify, track, and manage the reliable message delivery between two parties, a source and a destination. WS-Reliable Messaging Policy enables a Web service endpoint to indicate that a reliable message delivery is required.

In Transactions, WS-Coordination provides an extensible framework for defining coordination context and types for protocols that coordinate distributed actions. WS-Atomic Transactions provides the definition of transaction context and atomic transaction coordination type that is to be used with the framework defined by WS-Coordination. This enables transactions flowing over Web services.

The message formats and mechanisms defined by the WS-* specifications above enable on-the-wire interoperability. However, WSDL interoperability is equally important for a client to understand the requirements published by an endpoint, and for the endpoint to understand the client requirements. The new box labeled Metadata in Figure 2 serves that

The version of specifications implemented by Project Tango are the following:

- OASIS WS-Security 1.0 and 1.1
- WS-Trust (Feb 2005)
- WS-Secure Conversation (Feb 2005)
- WS-Security Policy (Jul 2005)
- WS-Reliable Messaging (Feb 2005)
- WS-Reliable Messaging Policy (Feb 2005)
- WS-Coordination (Aug 2005)
- WS-Atomic Transactions (Oct 2004)
- WS-Metadata Exchange (Oct 2004)
- WS-Policy (W3C Member Submission)
- WS-Policy Attachment (W3C Member Submission)

These are the same versions implemented by .NET 3.0. The exact reference to the specifications are listed at `http://wsit.dev.java.net/specification-links.html`. The specification versions in the next version of Tango will be aligned with .NET 3.5.

purpose. This box identifies the mechanisms that allow the Security, Reliability, and Transactional capabilities of an endpoint to be published and consumed by a client in an interoperable manner. WS-Policy defines a general purpose framework to express the capabilities of an endpoint. This extensible framework is then used to define the domain-specific policy assertions. WS-Metadata Exchange and WS-Transfer are used by the client to retrieve the information about the endpoint.

Project Tango provides all the above functionality built as an extension to JAX-WS RI.

3.1 What Is Tango's Value Proposition?

The two main value propositions of Project Tango are the following:

- An implementation of the key WS-* specifications
- Interoperability with .NET 3.0 framework

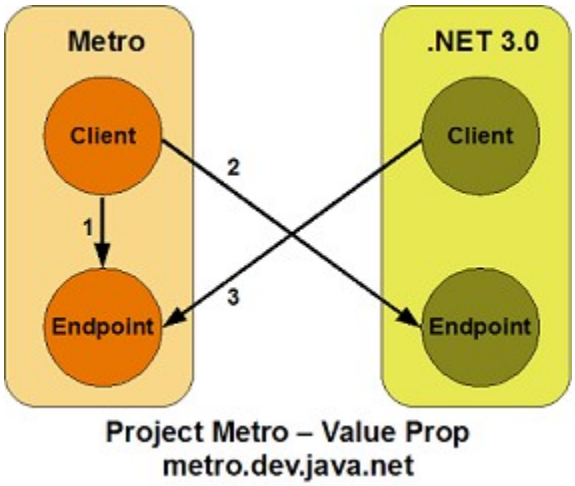

Figure 3: Project Metro – Value Proposition

Tango provides an implementation of the key enterprise Web services technologies, commonly known as WS-*, with a focus to provide interoperability with Windows Communication Foundation, which is a component of .the NET 3.0 framework that provides Microsoft's unified framework to build distributed applications.

Figure 3 shows how Metro, with Tango as a key component, provides a complete Web services solution for both homogeneous and heterogeneous architectures.

For any Web service communication, there is a Client that invokes an Endpoint. The Endpoint advertises its capabilities as *metadata* which the Client uses to bootstrap communication with the Endpoint. This metadata indicates which of the capabilities—Security, Reliability, and Transactions—are supported at the Endpoint. The figure shows a pair of Client and Endpoint, one using Metro and the other using .NET 3.0. The Metro Client and Endpoint can be deployed on any of the GlassFish V2 supported platforms. The .NET 3.0 Client and Endpoint can be deployed on any of the

.NET 3.0 supported platforms.

For a homogeneous architecture, where both Client and Endpoint are using Metro, the arrow labeled 1 shows a Metro Client invoking a Metro Endpoint. If this serves your purpose, then all you need is GlassFish V2 for Metro runtime, NetBeans IDE for tooling, and the WSIT tutorial that provides detailed steps to develop, deploy, and invoke such an endpoint.

For a heterogeneous architecture, where only one of the Client or Endpoint is using Metro and the other is based on .NET 3.0, the arrows labeled 2 and 3 show that a Metro client can invoke a .NET 3.0 endpoint and a .NET 3.0 client can invoke a Metro endpoint respectively. Again, the NetBeans IDE allows you to publish a Web service endpoint using any combination of Tango features and to invoke a Web service endpoint hosted on .NET 3.0 runtime.

Sun has participated in multiple interoperability plug-fests with Microsoft to ensure that the implementations of the two Web services stack are fully interoperable.

What is an interop plug-fest ?

Interoperability Plug-fest is a Web services interoperability event, focused on WS-* specifications, hosted by Microsoft and attended by other vendors. In each plug-fest, the participants carry their Web services stack and interoperate on scenarios and test cases which are defined using these specifications. Details about upcoming events and scenarios can be found at `http://www.mssoapinterop.org/ilab/`. Sun's participation in plugfests can be seen at `http://blogs.sun.com/arungupta/tags/plugfest`.

4.0 Tango Programming Model

The beauty of Project Tango is that it does not introduce a new programming model. It leverages the existing JAX-WS and EJB programming models and allows you to define Security, Reliability, and Transactional capability on the endpoints by bundling an additional configuration file with your application. Figure 4 shows the WSIT configuration file in the center.

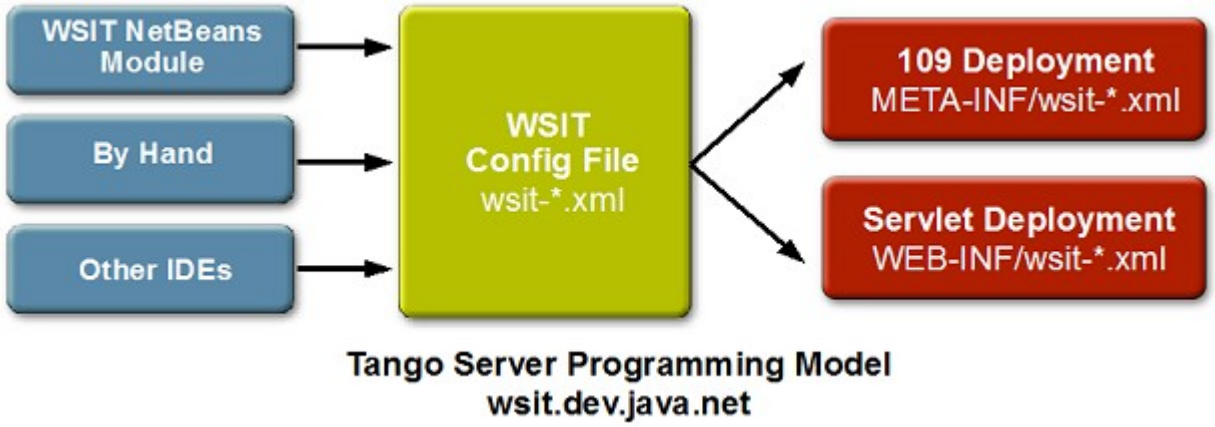

Figure 4: Tango Server Programming Model

The configuration file can be easily generated using the NetBeans 5.5.1 IDE updated with the WSIT plug-in. The WSIT plug-in can be installed from the NetBeans Update Center. Alternatively, the configuration file can be hand-crafted, but this requires knowledge of the different domain-specific policy languages. Therefore, using the NetBeans IDE to generate the configuration file is the preferred approach. The WSIT configuration file can be generated in other IDEs once the WSIT plug-in is available there. A sample of the configuration file, with Security, Reliability, and Transactions enabled, is provided in Appendix A.

The configuration file is located in the appropriate directory based upon JSR-109 or the Servlet deployment model. This step is transparent if NetBeans IDE is used to package or deploy the application.

On the client side, an optional WSIT configuration file may be used to specify certain client-side parameters such as timeout for Reliable Messaging or the locations of trust and keystores. This file is located in the META-INF directory on the classpath. The complete details about the WSIT configuration file are available in the WSIT tutorial.

5.0 Containers and Tools

The Web service endpoints that are developed using any of the Project Tango features can be deployed on the following platforms:

- **GlassFish V2** – Project Tango is integrated in GlassFish V2 and is the recommended platform.
- **Apache Tomcat** – Project Tango's standalone build (available from `wsit.dev.java.net`) provides an install script to install the implementation on Apache Tomcat.
- **Jetty** – Experimental support is documented at `http://blogs.sun.com/arungupta/entry/tango_on_jetty`.
- **Endpoint APIs in Java SE** – Experimental support is documented at `http://blogs.sun.com/theaquarium/entry/glassfish_web_services_stack_tango`.

GlassFish V2, a high-performance, production-quality and Java EE5 compatible Application Server provides recommended deployment platform for WSIT-enabled endpoints. GlassFish can be download from: http://glassfish.java.net

Metro can be installed on any Servlet 2.4 compliant Web container.

The NetBeans IDE provides first-class support for developing, configuring, and deploying WSIT-enabled Web service endpoints starting from Java. The IDE also provides mechanisms to invoke a Web service endpoint described using a WSDL.

The IDE needs to be updated with the WSIT plug-in from

NetBeans IDE 5.5.1 and WSIT plug-in provides tool time support for developing, configuring and deploying WSIT-enabled endpoints. The IDE can be downloaded from: http://netbeans.org

NetBeans Update Center to configure WSIT capabilities on a Web service endpoint. After a Web service is created, a contextual menu on the Web service with the name Edit Web Service Attributes gets enabled. Figure 5 shows you a snapshot of the window that pops up when this menu item is selected.

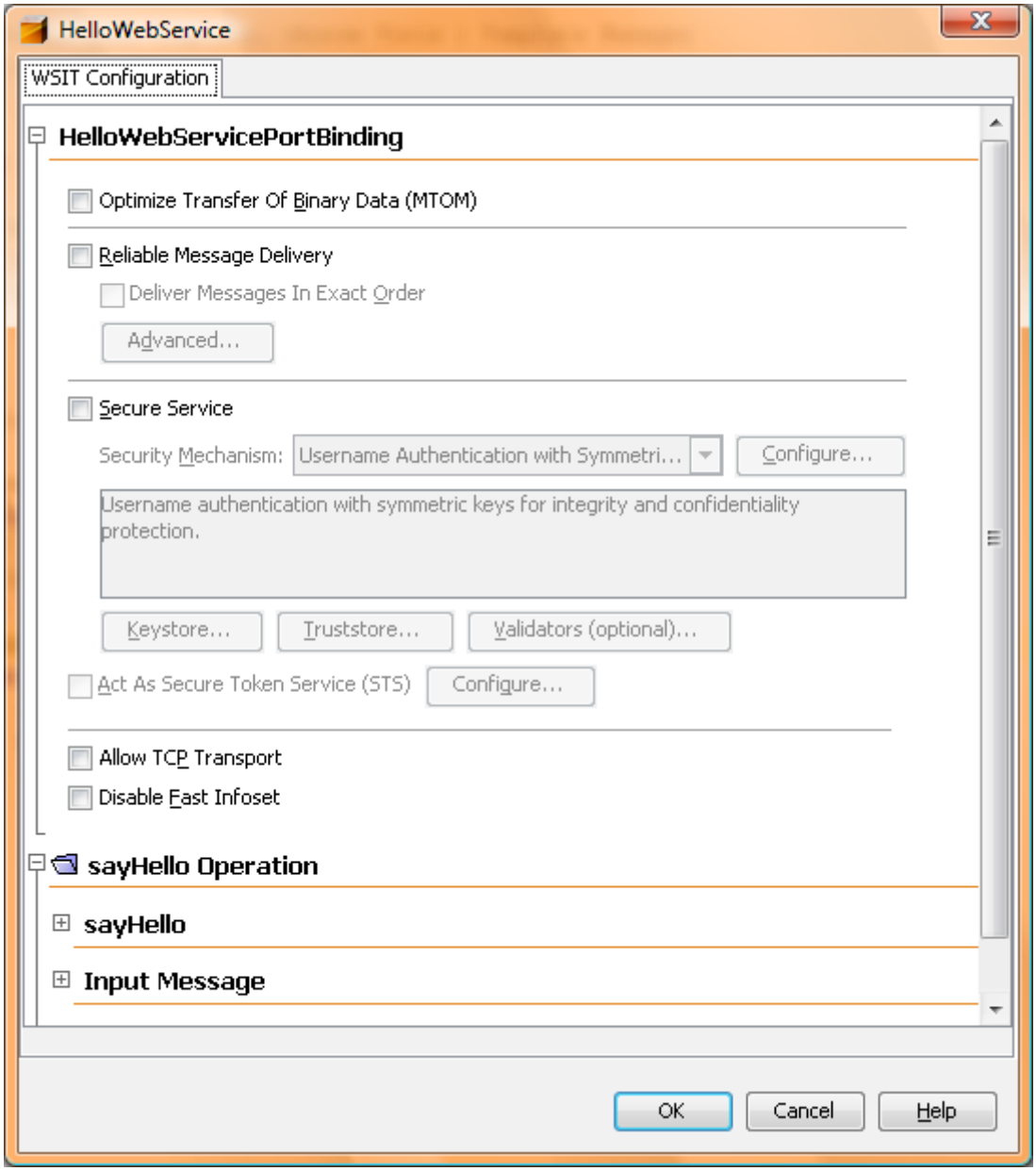

Figure 5: WSIT Configuration in NetBeans IDE

To enable Reliable Messaging, you only need to select the Reliable Message Delivery check box. The WSIT configuration file is generated after you click OK. The configuration file is then packaged appropriately as explained in Section 4.0. This configuration file is then used to generate the WSDL with correct policy assertions.

Similarly, Security can be enabled on an endpoint or on a per-operation basis. Figure 6 shows different security profiles, defined in Project Tango, that can be applied on the client and endpoint. Each method of the endpoint can also selectively enable Transactions support. In each case, appropriate domain specific policy assertions are generated and captured in the configuration file.

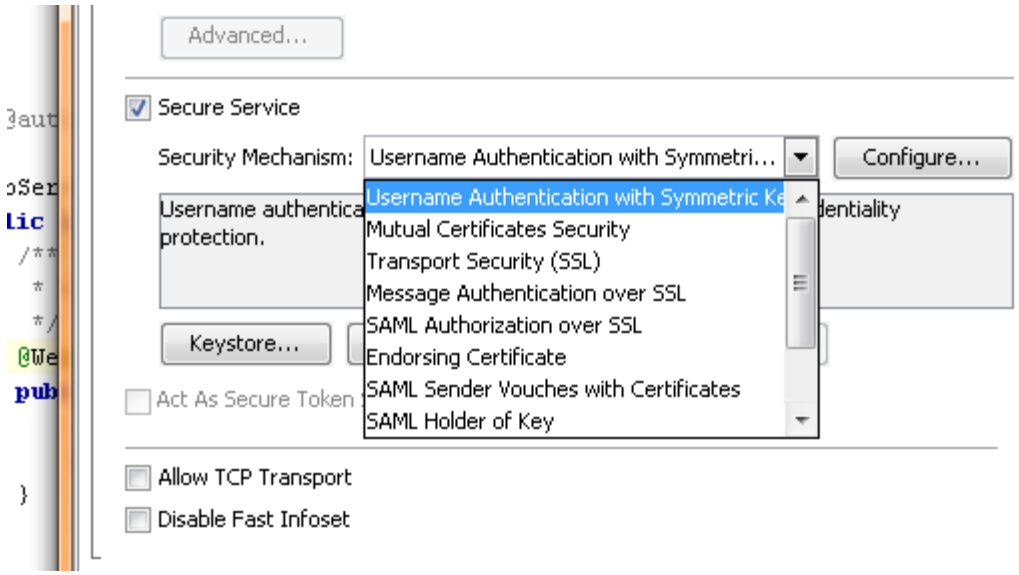

Figure 6: Security Profiles in WSIT Plug-in

More details about configuring each Tango feature (Reliability, Security, Transactions) are available in respective sections later in this document. The WSIT configuration file is generated, and packaged, based upon the WSIT features selected in the project. The WSDL with appropriate policy assertions is generated by the Metro runtime in the container. The NetBeans IDE also provides a contextual menu to deploy the project in a selected container.

The NetBeans IDE allows client-side artifacts to be generated from a WSDL that uses features supported by the WSIT plug-in. A new *Web Service References* node is added to the NetBeans project as shown in Figure 7.

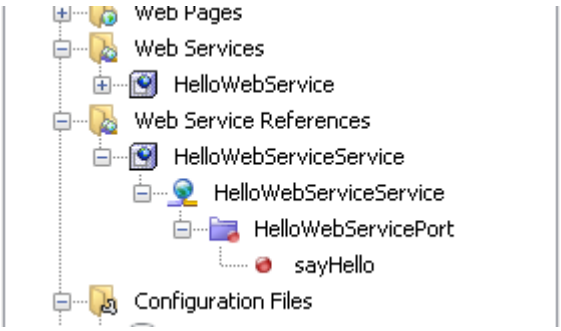

Figure 7: Client-side Artifacts in NetBeans IDE

This node contains child nodes for each of the operations exposed by the Web service. The child node can be dragged and dropped into any Java code (for example, JSP or Java class) to generate the boiler-

plate code to invoke the Web service.

In short, NetBeans IDE and GlassFish V2 provide you with first-class support for developing, deploying, and invoking a Web service endpoint that uses any of the WSIT capabilities.

6.0 Security

The Security support in Tango is an implementation of OASIS WS-Security 1.0 and 1.1, February 2005 WS-Trust and WS-Secure Conversation, and July 2005 WS-Security Policy specifications. This implementation provides a framework to secure SOAP message exchange between client and endpoint in an interoperable manner.

OASIS Web services Technical Committee (TC) produced WS-Security (WSS) 1.0 and 1.1 specifications. Built upon existing security technologies such as XML Digital Signature, XML Encryption, and X.509 Certificates, these specifications provide an industry-standard framework for SOAP message level security in Web services. WSS supports multiple security token formats, such as Username Token, that describes how a Web service endpoint can supply a UsernameToken as a means of identifying the requestor by “username”, and optionally using a “password” to authenticate that identity to the client. The NetBeans IDE defines several security profiles to configure security on Web service client and endpoint. These profiles allow a developer to select the integrity and confidentiality of a message, and also to choose a token format by selecting an option in the IDE.

WS-Trust defines a framework for issuing, renewing, and validating security tokens, and to broker trust relationships within different trust domains. This specification addresses the issue of *trust interoperability*. That means even if a given security token's format is acceptable to a Web service endpoint, interoperability at the syntax level does not guarantee that the consumer will be able to trust the token. For example, for a Web service endpoint to support an X.509 token from a client does not mean it will accept X.509 tokens from any Certificate Authority (CA). The endpoint will need to have necessary trust with the CA to verify the X.509 certificate. WS-Trust addresses this concern by defining a general framework used by a *Security Token Service* (STS) for token issuance.

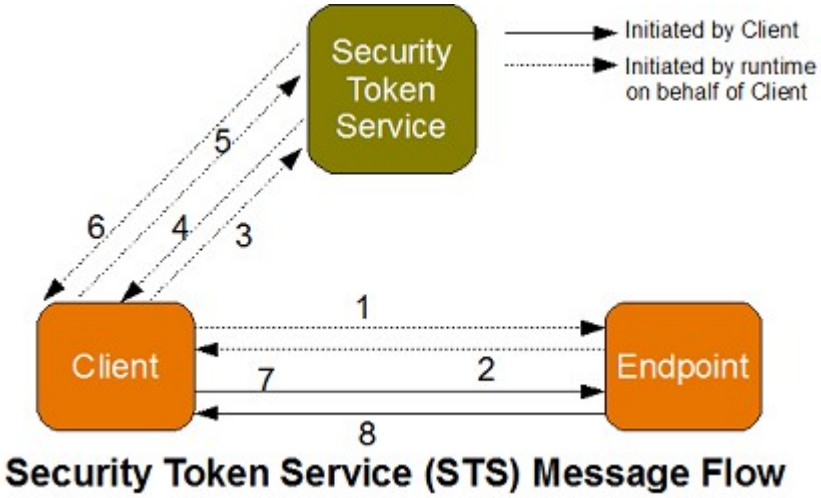

Figure 8: Security Token Service (STS) Message Flow

Figure 8 shows the message flow between Client, Endpoint, and STS when STS-issued tokens are used for securing the message exchange. For Client and Endpoint to communicate, the client requests the metadata (arrow labeled 1) from Endpoint. The dashed arrow indicates that the WSIT runtime on the client side (completely transparent to the application) initiates the request on its behalf. If the metadata received (arrow labeled 2) indicates that a security token is required from a specified STS, then the Client issues another request (arrow labeled 3) to obtain STS's metadata. The response message from the STS (arrow labeled 4) indicates the type of security token to be used for further communication with it. After fulfilling security requirements gathered from the STS metadata, the Client requests a security token from the STS (arrow labeled 5) and receives a response (arrow labeled 6) that contains the issued token. The Client then invokes the Endpoint with the issued token (arrow labeled 7). The Endpoint then verifies the request from the Client, processes the request, and returns the response to the Client (arrow labeled 8). The arrows labeled 1 through 6 are protocol messages, initiated by the WSIT runtime on behalf of the Client, to gather metadata which is then subsequently used in application-level messages (arrows 7 and 8).

An application may span multiple security domains where each domain has its own STS. These multiple STSs can be chained with each other to broker trust among multiple security domains. This pattern is called a *brokered trust* and is supported by Project Tango. Appendix B provides a description of a real-life scenario using brokered trust.

WS-Secure Conversation increases the overall performance and security of messages by defining semantics for secure message exchange for multiple message exchanges. The specification defines mechanisms for establishing and sharing security contexts, and deriving keys from established security contexts. The security context is shared among the communicating parties for the lifetime of a communications session. Once the context is established, derived keys are used for each key usage in the secure context. The specification defines three ways by which security context can be created:

- By a Secure Token Service (STS)
- By one of the communicating parties and propagated with a message
- Through negotiations or exchanges

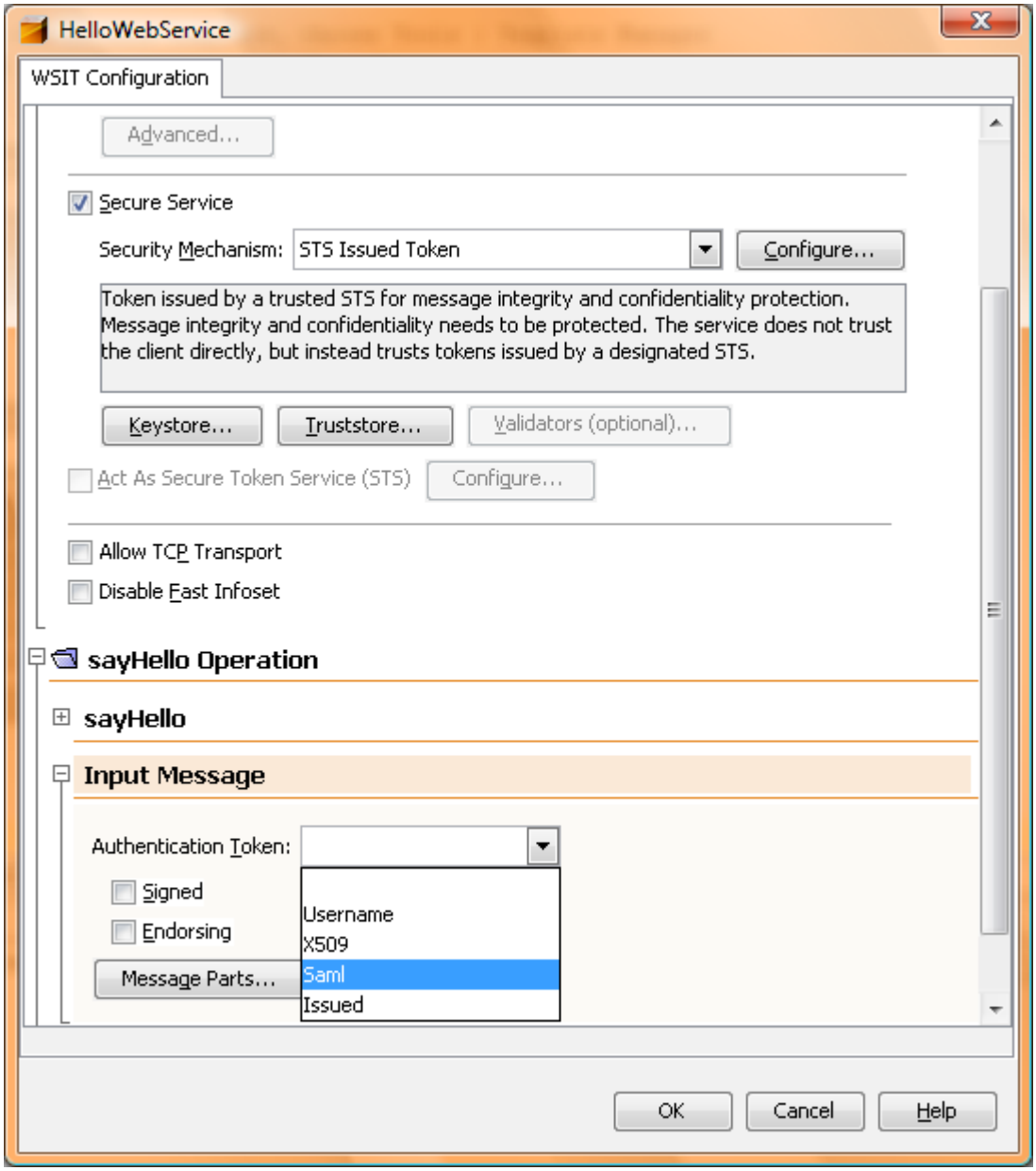

Figure 9: Security Configuration for "STS Issued Token" Profile

Using NetBeans IDE, security can be enabled on an endpoint or on a per-operation basis. Project Tango defines several security profiles to configure security on Web service client and endpoint. Figure 6 shows a sample of these security profiles that can be configured from within the IDE. For example, if

the “STS Issued Token” profile is chosen, then the security token on each operation can be configured as shown in Figure 9.

Tango allows fine-grained security configuration as shown in Figure 10. Using the NetBeans IDE, standard headers in the SOAP message or XPath-identified elements may be signed and encrypted.

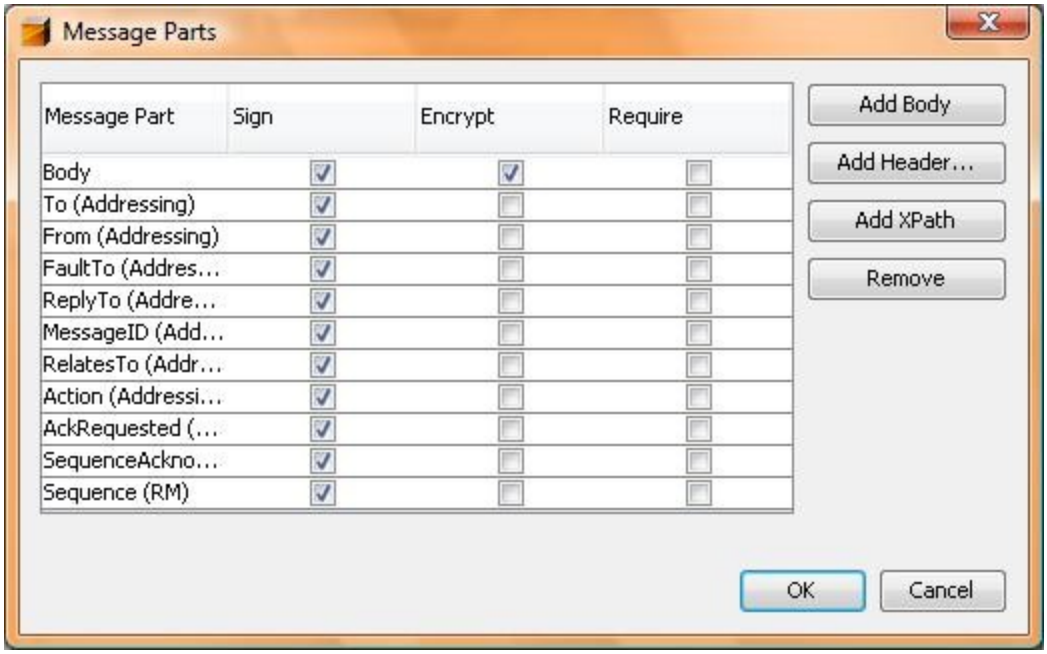

Figure 10: Fine-grained Security Configuration

After the security mechanism is chosen, appropriate WS-Security Policy assertions are generated in the WSIT configuration file. This configuration file is then used to generate the WSDL with correct policy assertions.

7.0 Reliability

The Reliability support in Tango is an implementation of February 2005 WS-Reliable Messaging and WS-Reliable Messaging Policy Assertion specifications.

WS-Reliable Messaging defines a wire-based messaging protocol, with a SOAP binding, to identify, track, and manage the reliable delivery of messages between two parties, a Source (for example, Client) and a Destination (for example, Endpoint). In a nutshell, the Source periodically asks the Destination to acknowledge which messages have arrived, and then based on that information, resends messages if necessary. This allows the Destination to reconstruct an exact stream of messages from the Source in the order in which they were sent. WS-Reliable Messaging introduces Reliable Messaging Source (RMS) and Reliable Messaging Destination (RMD) that encapsulate the functionality of send, resend, and reconstruct the stream of messages on Source and Destination respectively.

The extent to which RMD, and thus Destination, is able to reconstruct the message stream is called *Delivery Assurance* in Reliable Messaging terms. There are three levels of Delivery Assurance provided by RMD:

- **AtMostOnce** – Messages will be delivered at most once without duplication or an error will be raised. It is possible that some messages in the sequence may not be delivered.
- **AtLeastOnce** – Every message sent will be delivered or an error will be raised. Some messages may be delivered more than once.
- **InOrder** – All messages are delivered in the order that they were sent. This level requires that the sequence observed by the ultimate receiver be non-decreasing. It says nothing about duplications or omissions.

Figure 11 shows a messaging system without any Reliable Messaging support. The Source and Destination are talking through Web service runtime with no Reliable Messaging support. The "x" in the arrows indicates that application messages may get lost or mishandled in either direction. In this case, without any Reliable Messaging support from machine and Web service runtime, Source and Destination are responsible for the recovery of messages.

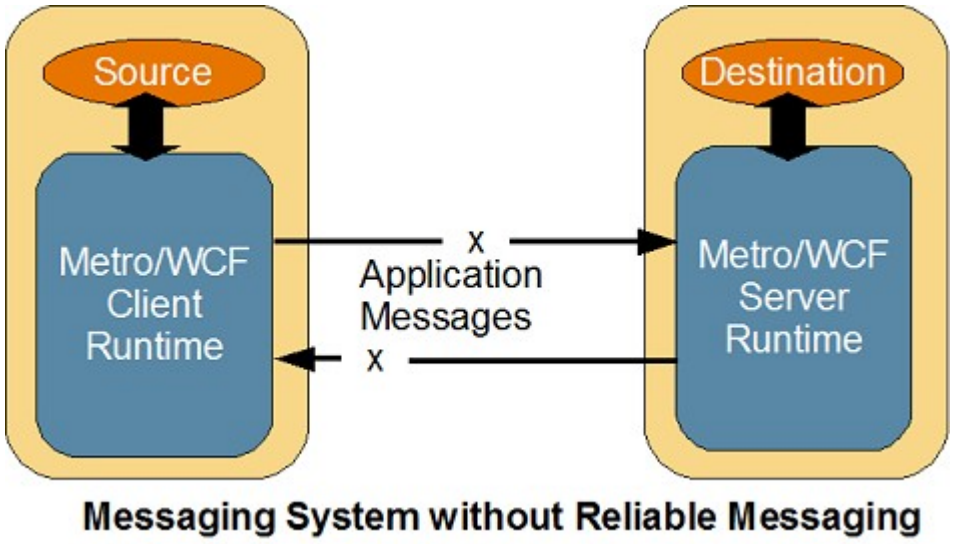

Figure 11: Messaging System Without Reliable Messaging

Figure 12 shows the messaging system with Reliable Messaging support. In this case, the messages between Source and Destination are exchanged through RMS and RMD that are embedded within the Web service runtime. Now using the Reliable Messaging protocol RMS and RMD can communicate in a reliable way, on behalf of Source and Destination respectively, and fulfilling the delivery assurance. Notice that there is no direct communication between the application code on either end and Reliable Messaging components. RMS and RMD are automatically enabled based upon the presence of Reliable

Messaging Policy assertions in the WSDL.

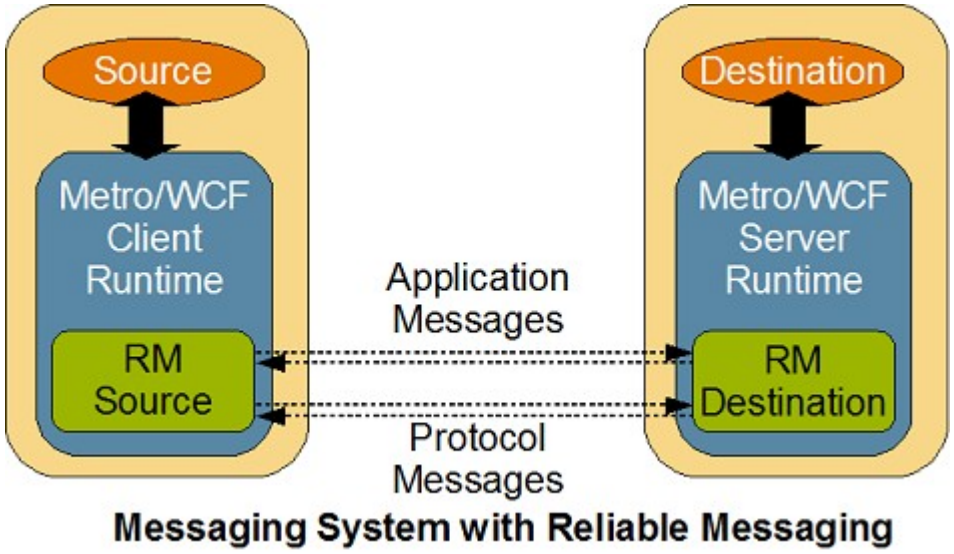

Figure 12: Messaging System With Reliable Messaging

Reliable Messaging support on a Web service endpoint can be easily enabled using NetBeans IDE by selecting Reliable Message Delivery as shown in Figure 13. The WSIT tutorial provides more details.

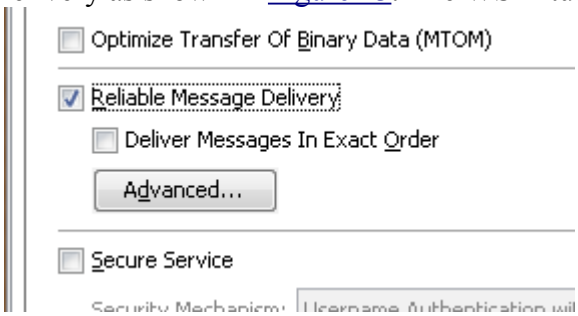

Figure 13: Reliable Messaging Support in WSIT plug-in

8.0 Transactions

Transaction support in Tango is an implementation of October 2004 WS-Coordination and WS-Atomic Transactions specifications.

The Transactions support in Tango enables Java EE transactions (defined by JTA) to work across heterogeneous systems, using Web services that support WS-Atomic Transactions (WS-AT) and WS-Coordination. This support provides consistent failure and recovery semantics, so the applications no longer need to deal with the mechanics of determining a mutually agreed upon outcome decision

between multiple parties, or to figure out how to recover from a large number of possible inconsistent states.
This section explains how WS-Coordination and WS-Atomic Transactions specifications provide the framework to enable JTA transactions over Web services. However, as a developer, you only need to enable Transactions support as mentioned at the end of this section.

The WS-Coordination specification defines an extensible framework that coordinates activities through a *Coordinator* and a set of coordination protocols. Each Coordinator may support multiple coordination protocols. The specification also defines the structure of *coordination context* and the requirements for propagating context between cooperating services. The coordination context flows with the application messages and contains the necessary information required by the other application.

The WS-Atomic Transaction specification provides the definition of atomic transaction coordination type that is to be used with the extensible framework defined in the WS-Coordination specification. This type is used to coordinate activities that have an "all or nothing" property, commonly known as Atomic, Consistent, Isolated, Durable (ACID). It also defines an *Atomic Transaction Context* to carry the transactional semantics on all application messages.

Project Tango supports the *Durable two-phase Commit* (Durable 2PC) protocol defined by the WS-Atomic Transaction specification. This protocol is used by participants managing durable resources such as a database. All XAResources in a Web service are mapped to a durable 2PC resource.

Project Tango creates an Atomic Transaction Context the first time a transacted Web service operation is invoked within a JTA transaction scope. For example, in the code fragment shown in Code Sample 1, WS-AT transaction context is created in line 05:

```
01 @Resource
02 javax.transaction.UserTransaction ut;
03
04 ut.begin();
05 bankWebService.makeWithdrawl();
06 ...
07 ut.commit();
```

Code Sample 1: WS-AT Transaction Context Creation

This does not impose any overhead on JavaEE transactions when the capability is not being used. This context flows along with the application messages. A request from a .NET 3.0 client that contains transaction context is also understood by Project Tango and participates in the transactional context.

The Transactions feature in Tango supports the following:

- WS-AT Transactions flowing from the Web tier and EJB tier
- Automated mapping of Container Managed Transactions to semantically equivalent WS-Atomic Transaction policy assertions
- Auto-enlistment of XAResources – JMS implementation in GlassFish V2, Entity manager of TopLink Essentials using JavaDB, Entity Manager of Hibernate

The WS-AT specification also defines policy assertions to describe transactional capabilities of an endpoint. Transactional capabilities on a Web service endpoint can be easily enabled on a per-operation basis using NetBeans IDE. Figure 14 shows the different options available to configure WS-AT support on an operation. Selecting a Transaction type on an operation generates the appropriate WS-AT policy assertion in the WSIT configuration file. This configuration file is then used to generate the WSDL. The WSIT tutorial has more details.

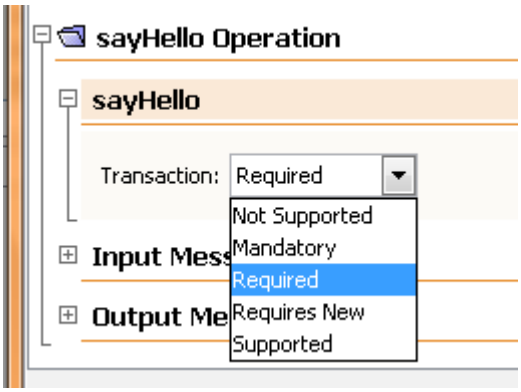

Figure 14: Atomic Transaction Configuration in WSIT Plug-in

The transactions feature of Project Tango works only with the GlassFish V2 container.

9.0 References

1. `wsit.dev.java.net` – Project Tango
2. `metro.dev.java.net - Project Metro`
3. `glassfish.dev.java.net` – GlassFish V2 Application Server
4. `jax-ws.dev.java.net` – JAX-WS
5. `netbeans.org` – NetBeans IDE
6. `http://sessions.sun.com/learning/javaoneonline/sessions/2007/TS-4865/index.html` – JavaOne 207 Technical Session with audio and synchronized transcript.
7. `java.sun.com/webservices/interop` – Landing page of Web Services Interoperability efforts at Sun
8. `planet.sun.com/webservices/group/blogs/ -` Aggregated blog from the Web services engineering team at Sun
9. `wsit-docs.dev.java.net/releases/m5/ -` WSIT Tutorial
10. `http://java.sun.com/javaee/5/docs/tutorial/doc/JAXWS.html#wp72279` – How to build Web services with JAX-WS
11. `wsit.dev.java.net/screencasts.html` – WSIT Screencasts

Appendix A – WSIT Configuration File Sample

This appendix contains Java code for a Web service endpoint and the WSIT configuration file for that endpoint, generated by NetBeans IDE, after enabling Security (STS Issued Token Profile), Reliability, and Transactions.

The Web service endpoint code is given below:

```
package server;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
 *
 * @author Arun Gupta
 */
@WebService()
public class HelloWebService {
    /**
     * Web service operation
     */
    @WebMethod
    public String sayHello(@WebParam(name = "name") String name) {
        return "Hello " + name;
    }

}
```

The WSIT configuration file, generated by NetBeans IDE, after enabling Security (STS Issued Token profile), Reliability, and Transactions is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
 <definitions
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="HelloWebServiceService"
targetNamespace="http://server/" xmlns:tns="http://server/"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" xmlns:wsaws="http://www.w3.org/2005/08/addressing"
xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust"
xmlns:sc="http://schemas.sun.com/2006/03/wss/server"
xmlns:wspp="http://java.sun.com/xml/ns/wsit/policy"
xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
```



```
xmlns:wsp2002="http://schemas.xmlsoap.org/ws/2002/12/policy"
 >
    <message name="sayHello"/>
    <message name="sayHelloResponse"/>
    <portType name="HelloWebService">
        <wsdl:operation name="sayHello">
            <wsdl:input message="tns:sayHello"/>
            <wsdl:output message="tns:sayHelloResponse"/>
        </wsdl:operation>
    </portType>
    <binding name="HelloWebServicePortBinding" type="tns:HelloWebService">
        <wsp:PolicyReference URI="#HelloWebServicePortBindingPolicy"/>
        <wsdl:operation name="sayHello">
            <wsp:PolicyReference
URI="#HelloWebServicePortBinding_sayHello_Policy"/>
            <wsdl:input>
                <wsp:PolicyReference
URI="#HelloWebServicePortBinding_sayHello_Input_Policy"/>
            </wsdl:input>
            <wsdl:output>
                <wsp:PolicyReference
URI="#HelloWebServicePortBinding_sayHello_Output_Policy"/>
            </wsdl:output>
        </wsdl:operation>
    </binding>
    <service name="HelloWebServiceService">
        <wsdl:port name="HelloWebServicePort"
binding="tns:HelloWebServicePortBinding"/>
    </service>
    <wsp:Policy wsu:Id="HelloWebServicePortBindingPolicy">
        <wsp:ExactlyOne>
            <wsp:All>
                <wsaws:UsingAddressing
xmlns:wsaws="http://www.w3.org/2006/05/addressing/wsdl"/>
                <wsrm:RMAssertion/>
                <sp:SymmetricBinding>
                    <wsp:Policy>
                        <sp:ProtectionToken>
                            <wsp:Policy>
                                <sp:SecureConversationToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/
AlwaysToRecipient">
                                    <wsp:Policy>
                                        <sp:RequireDerivedKeys/>
                                        <sp:BootstrapPolicy>
                    <wsp:Policy>
                        <sp:SymmetricBinding>
                    <wsp:Policy>
                        <sp:ProtectionToken>
                            <wsp:Policy>
                                <sp:IssuedToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/
AlwaysToRecipient">
```



```
                                    <sp:RequestSecurityTokenTemplate>
                                        <t:TokenType>http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1</t:TokenType>
                                        <t:KeyType>http://schemas.xmlsoap.org/ws/20
05/02/trust/SymmetricKey</t:KeyType>
                                        <t:KeySize>256</t:KeySize>
                                    </sp:RequestSecurityTokenTemplate>
                                    <wsp:Policy>
                                        <sp:RequireInternalReference/>
                                    </wsp:Policy>
                                </sp:IssuedToken>
                            </wsp:Policy>
                        </sp:ProtectionToken>
                        <sp:Layout>
                            <wsp:Policy>
                                <sp:Lax/>
                            </wsp:Policy>
                        </sp:Layout>
                        <sp:IncludeTimestamp/>
                        <sp:OnlySignEntireHeadersAndBody/>
                        <sp:AlgorithmSuite>
                            <wsp:Policy>
                                <sp:Basic128/>
                            </wsp:Policy>
                        </sp:AlgorithmSuite>
                    </wsp:Policy>
                </sp:SymmetricBinding>
                        <sp:Wss11>
                    <wsp:Policy>
                        <sp:MustSupportRefKeyIdentifier/>
                        <sp:MustSupportRefIssuerSerial/>
                        <sp:MustSupportRefThumbprint/>
                        <sp:MustSupportRefEncryptedKey/>
                    </wsp:Policy>
                </sp:Wss11>
                        <sp:Trust10>
                    <wsp:Policy>
                        <sp:MustSupportIssuedTokens/>
                        <sp:RequireClientEntropy/>
                        <sp:RequireServerEntropy/>
                    </wsp:Policy>
                </sp:Trust10>
                        <sp:EncryptedParts>
                            <sp:Body/>
                        </sp:EncryptedParts>
                        <sp:SignedParts>
                            <sp:Body/>
                            <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"/>
                            <sp:Header Name="From"
Namespace="http://www.w3.org/2005/08/addressing"/>
                            <sp:Header Name="FaultTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
```



```
                            <sp:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
                            <sp:Header Name="MessageID"
Namespace="http://www.w3.org/2005/08/addressing"/>
                            <sp:Header Name="RelatesTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
                            <sp:Header Name="Action"
Namespace="http://www.w3.org/2005/08/addressing"/>
                            <sp:Header Name="AckRequested"
Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
                            <sp:Header Name="SequenceAcknowledgement"
Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
                            <sp:Header Name="Sequence"
Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
                        </sp:SignedParts>
                    </wsp:Policy>
                </sp:BootstrapPolicy>
                                    </wsp:Policy>
                                </sp:SecureConversationToken>
                            </wsp:Policy>
                        </sp:ProtectionToken>
                        <sp:Layout>
                            <wsp:Policy>
                                <sp:Strict/>
                            </wsp:Policy>
                        </sp:Layout>
                        <sp:AlgorithmSuite>
                            <wsp:Policy>
                                <sp:Basic128/>
                            </wsp:Policy>
                        </sp:AlgorithmSuite>
                        <sp:IncludeTimestamp/>
                        <sp:OnlySignEntireHeadersAndBody/>
                    </wsp:Policy>
                </sp:SymmetricBinding>
                <sp:Wss11>
                    <wsp:Policy>
                        <sp:MustSupportRefKeyIdentifier/>
                        <sp:MustSupportRefIssuerSerial/>
                        <sp:MustSupportRefThumbprint/>
                        <sp:MustSupportRefEncryptedKey/>
                    </wsp:Policy>
                </sp:Wss11>
                <sp:Trust10>
                    <wsp:Policy>
                        <sp:RequireClientEntropy/>
                        <sp:RequireServerEntropy/>
                        <sp:MustSupportIssuedTokens/>
                    </wsp:Policy>
                </sp:Trust10>
                <sc:KeyStore wspp:visibility="private" storepass="changeit"
type="JKS"
location="C:\testbed\b50\glassfish\domains\domain1\config\keystore.jks"/>
```



```
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    <wsp:Policy wsu:Id="HelloWebServicePortBinding_sayHello_Input_Policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <sp:EncryptedParts>
                    <sp:Body/>
                </sp:EncryptedParts>
                <sp:SignedParts>
                    <sp:Body/>
                    <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="From"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="FaultTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="MessageID"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="RelatesTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="Action"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="AckRequested"
Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
                    <sp:Header Name="SequenceAcknowledgement"
Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
                    <sp:Header Name="Sequence"
Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
                </sp:SignedParts>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    <wsp:Policy wsu:Id="HelloWebServicePortBinding_sayHello_Output_Policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <sp:EncryptedParts>
                    <sp:Body/>
                </sp:EncryptedParts>
                <sp:SignedParts>
                    <sp:Body/>
                    <sp:Header Name="To"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="From"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="FaultTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="MessageID"
Namespace="http://www.w3.org/2005/08/addressing"/>
```



```
                    <sp:Header Name="RelatesTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="Action"
Namespace="http://www.w3.org/2005/08/addressing"/>
                    <sp:Header Name="AckRequested"
Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
                    <sp:Header Name="SequenceAcknowledgement"
Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
                    <sp:Header Name="Sequence"
Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
                </sp:SignedParts>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    <wsp:Policy wsu:Id="HelloWebServicePortBinding_sayHello_Policy">
        <wsp:ExactlyOne>
            <wsp:All>
                <wsat:ATAssertion wsp:Optional="true" wsp2002:Optional="true"/>
                <wsat:ATAlwaysCapability/>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
</definitions>
```

Appendix B – Brokered Trust Sample

This appendix explains the Brokered Trust pattern using a real-life scenario.

Figure 15 shows the business scenario and message flow of a composite Web service. It consists of a Retail Quote Service (RQS), running in a Metro-hosted environment, that uses a Wholesale Quote Service (WQS), also running in the same security domain, to serve car quotes to clients running in the Metro and .NET environment.

This RQS also gets competitive bids from a WQS running in a different security domain hosted using .NET 3.0 framework. Each domain has its own Security Token Service that issue tokens for clients running in a different security domain.

Each client may be offered a special discount based upon their identity. To scale identity management, RQS also has an STS running on the Internet. The .NET client talks to this STS, running on the Internet, obtains the tokens, and uses the STS to authenticate with RQS. The client talks to RQS in a secure manner. The RQS talks to two WQSs in a secure and reliable manner.

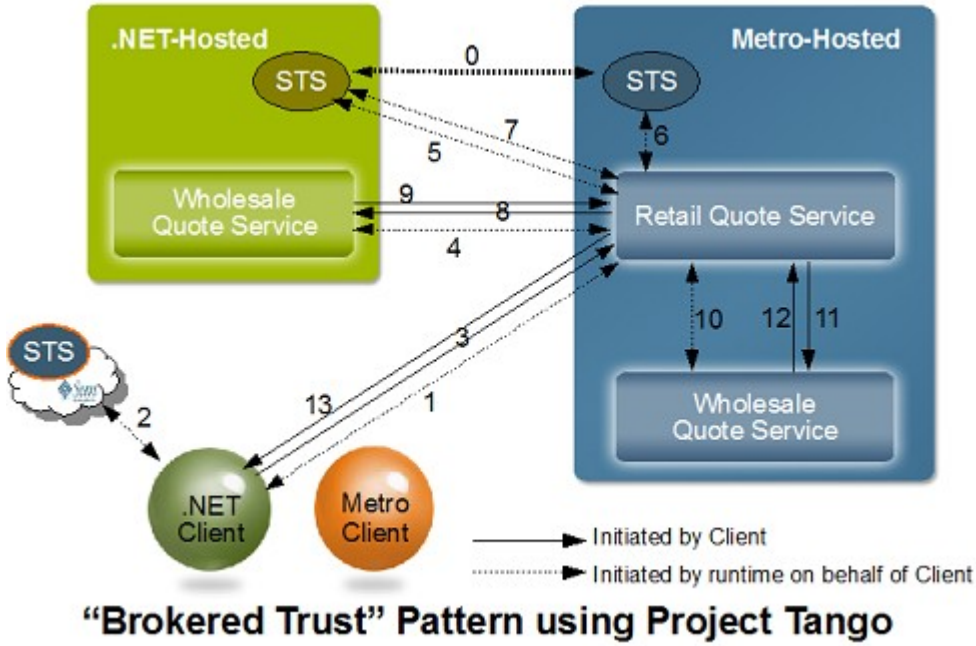

Figure 15: "Brokered Trust" Pattern Using Project Tango

The message flow is explained below. The numbers refer to the arrows in Figure 15.

0 Before the Web services in two security domains (Metro and .NET-hosted) can talk to each other, a trust relationship between these two security domains is established by an *out of band* trust between the two STSs.

1 NET client requests metadata from RQS and receives a response. The metadata provides the location of the external STS to obtain the security tokens.

2 NET client runtime then gets the security tokens from the specified STS.

3 Using the security tokens, the .NET client initiates a business request to RQS.

4 RQS obtains metadata from WQS in the .NET-hosted environment. The received metadata provides a location to the STS in the .NET-hosted environment.

5 RQS runtime requests the metadata from the specified STS. This metadata provides a location to the STS in the Metro-hosted environment.

6 Being in the same security domain, this STS provides tokens to RQS.

7 RQS runtime uses this token to obtain new tokens from STS running in the .NET-hosted environment.

8 Using these tokens, RQS sends a request to WQS with appropriate credentials.

9 WQS verifies the credentials, processes the request, and returns the response.

10 RQS obtains the metadata from WQS in the Metro-hosted environment.

11 RQS sends a request to WQS.

12 WQS verifies the credentials, processes the request, and returns the response.

13 Response is returned back to the client after aggregating responses from both WQS.

This pattern may be extended to chain multiple STSs together.

The only operations described above that were explicitly initiated by the developer were (arrow 1) to generate the client proxy from the WSDL provided by the service, and (arrow 3) to call a business method. All the other interactions are handled by the Tango infrastructure.

