Architecture Guide

<!-- -->

1. Architecture Guide

1.1. Contents

1.2. Introduction

This guide describes the architecture of Axis C++ implementation.

1.3. Architecture Overview

Axis C++ is all about deploying C++ web services and processing SOAP messages. Axis C++ architecture closely follows Axis Java in Handler and message paths.

1.3.1. Handlers and the Message Path in Axis

Axis C++ implementation follows how handlers and message paths work in Axis Java implementation. When the central Axis processing logic runs, a series of Handlers are each invoked in order. The order of invocation is determined by two factors - deployment configuration and whether the engine is a client or a server. The object which is passed to each Handler invocation is a **MessageData**

- Deserializer
- Serializer
- A bag of properties
- We will be discussing more on each of this later on this Architecture Guide.

1.3.2. Message Path on the Server

figure 1.0

Above diagram shows you how the Axis C++ Engine architecture works to invoke functions of AxisC++ Server Engine.

Note:

1.4. Message Flow

1.4.1. Handlers and Chains

figure 3.0

A web service does not necessarily send a response message to each request message, although many do. However, response Handlers are still useful in the message path even when there isn't a response message, e.g. to stop timers, clean up resources, etc. A Chain is a composite Handler, i.e. it aggregates a collection of Handlers as well as implementing the Handler interface

A Chain also has similarities to the Chain of Responsibility design pattern in which a request flows along a sequence of Handlers until it is processed. Although an Axis Chain may process a request in stages over a succession of Handlers, it has the same advantages as Chain of Responsibility: flexibility and the ease with which new function can be added. Back to message processing -- a message is processed by passing through the appropriate Chains. A message Data is used to pass the message and associated environment through the sequence of Handlers. The model is that Axis Chains are constructed offline by having Handlers added to them one at a time. Then they are turned online and message data start to flow through the Chains. Handlers and Chains can be defined to have 'request', 'session', or 'application' scope.

1.5. 1.Axis Engine

1.6. 2.HandlerPool

1. Loads and keeps Transport and Global handlers.
2. Loads service specific handlers when needed and unloads when needed.
3. Loads target web service handler when needed and unloads when needed.

In order for the HandlerLoader to dynamically load a class, every DLL (or Shared object) must have following export functions.

int GetClassInstance(DCLInterface **inst);

int DestroyInstance(DCLInterface *inst);

AxisEngine has no idea of any web service methods in the deployed web service class that is dynamically loaded from a DLL. Therefore in order to communicate with loaded class we have to have a known interface. This interface is known as **BasicHandler**and is known to AxisEngine. This interface is implemented by every webservice and a handler.

1.7. 3.Message Model

1.8. 4.Soap Deserializer

Currently the Soap Deserializer is implemented using SAX2 parser. Soap Deserializer exposes and API such that the API is independent of the implementation.

1.9. 5.Soap Serializer

Soap Serializer's task is to generate the SOAP stream to be sent. There are a set of functions (API that is the opposite functionality with Soap Deserializer). Once the Serializer is given all the information that is required to generate a SOAP using the API, the getStream(..) function can be used to generate the SOAP message.

1.10. 6.WSDD Module

1.11. WSDL2Ws Tool

WSDL2Ws.html

1.12. Diagram Book

This Diagram Book includes Following Diagrames which are drawn to describe Axis Cpp Engine and it's process.

1) Use case diagram(s)

2) Sequence diagrams

3) Class diagrams(s)

4) Deployment diagram(s)

1.13. Open Issues