In diesem Kapitel

- Um was geht's?
- Axis Architektur – Eine Übersicht
- Subsysteme
  - *Message Flow*
    - Handlers und Chains (Handler Ketten)
    - Message Contexts
  - *Adminstratives Subsystem*
  - *SOAP Message Modell Subsystem*
  - *Message Elemente*
  - *Deserialisierung*
  - *Encoding*
  - *WSDL Tools*

Apache AXIS Architektur

1.1. Übersicht

1.1.1. Vorbemerkungen

Die Informationen über AXIS sind eher provisorisch und beziehen sich auf die Version 1.x. Es kann also noch viele Änderungen geben!

1.1.2. Handlers und Message Pfad in Axis

Axis wurde von Grund auf darauf ausgelegt, Messages zu verarbeiten, also nicht nur SOAP. Der ursprüngliche Ansatz war umfassender, weil man mit weiteren XML Protokollen gerechnet hat (nach XML-RPC, SOAP und den Bemühungen der XML Protokoll Work Group von W3C).

Kern von Axis ist eine Reihe von unterschiedlichen *Handlern*, die in einer bestimmten Reihenfolge aufgerufen werden. Die Reihenfolge selbst wird aufgrund des Deployment Descriptors und weiterer Faktoren bestimmt. An den Handler selber wird ein sogenannter *MessageContext* übergeben. Ein MessageContext ist eine Datenstruktur, welche primär aus folgenden Teilen besteht:

- eine Request Message
- eine Response Message
- Properties

Als nächstes muss zwischen Client und Server unterschieden werden:

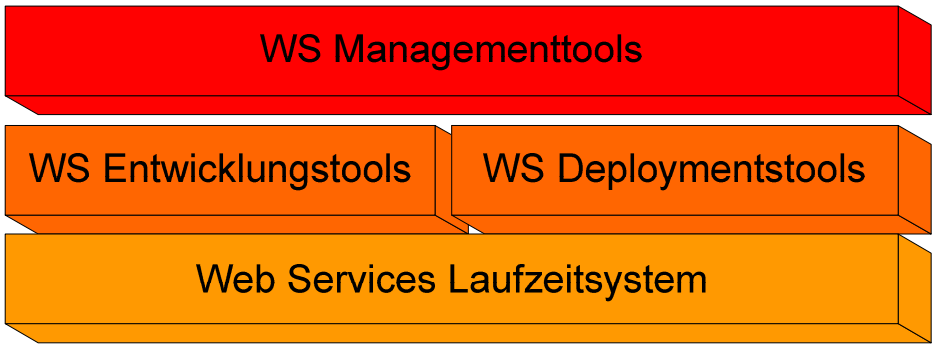
- **Server**: in diesem Fall wird ein *TransportListener* einen MessageContext kreieren und das Axis Framework aufrufen
- **Client**: in diesem Fall wird ein MessageContext kreiert kreiert und das Axis Framework aufgerufen

Das Axis Framework muss den MessageContext durch die konfigurierten Handler durchschleusen. Jeder Handler hat dabei die Möglichkeit mit dem kreierten MessageContext zu arbeiten, entsprechend der Auslegung des Handlers.

1.2. Grobarchitektur von Axis

1.2.1. Architektur einer Web Services Anwendung - Entwicklersicht

Aus Sicht des Entwicklers besteht ein Web Services Umfeld grob aus folgenden Layern, aus folgender Layer-Architektur (Patterns: siehe Buschmann et al: Software Architektur Patterns):

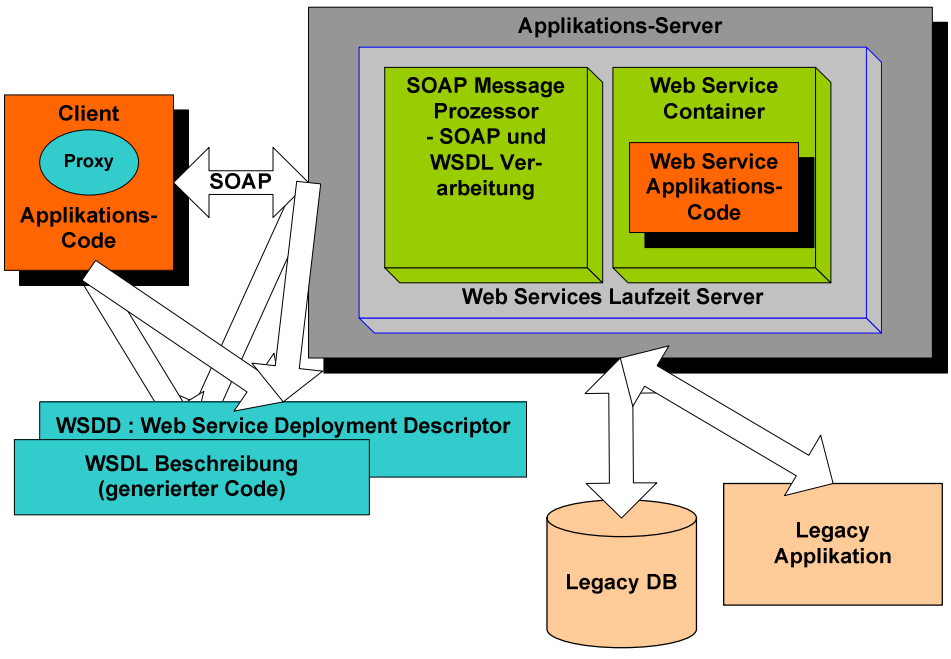


- Tools: WSDL und Proxy Generierung
  Administrationstools
  Monitore (SOAP, TCP, …)
- Management Monitore
  Logging
- Laufzeitsystem SOAP Message Processing
  Security
  Service Provisioning

Schauen wir uns diese einzelnen Layer etwas genauer an.

1.2.1.1. WS - Laufzeitsystem

Das Laufzeitsystem besteht aus Client und Server seitigen Komponenten: das Server-seitige **das Laufzeitsystem für Web Services** muss in der Lage sein, SOAP Messages und Web Services zu „hosten“.

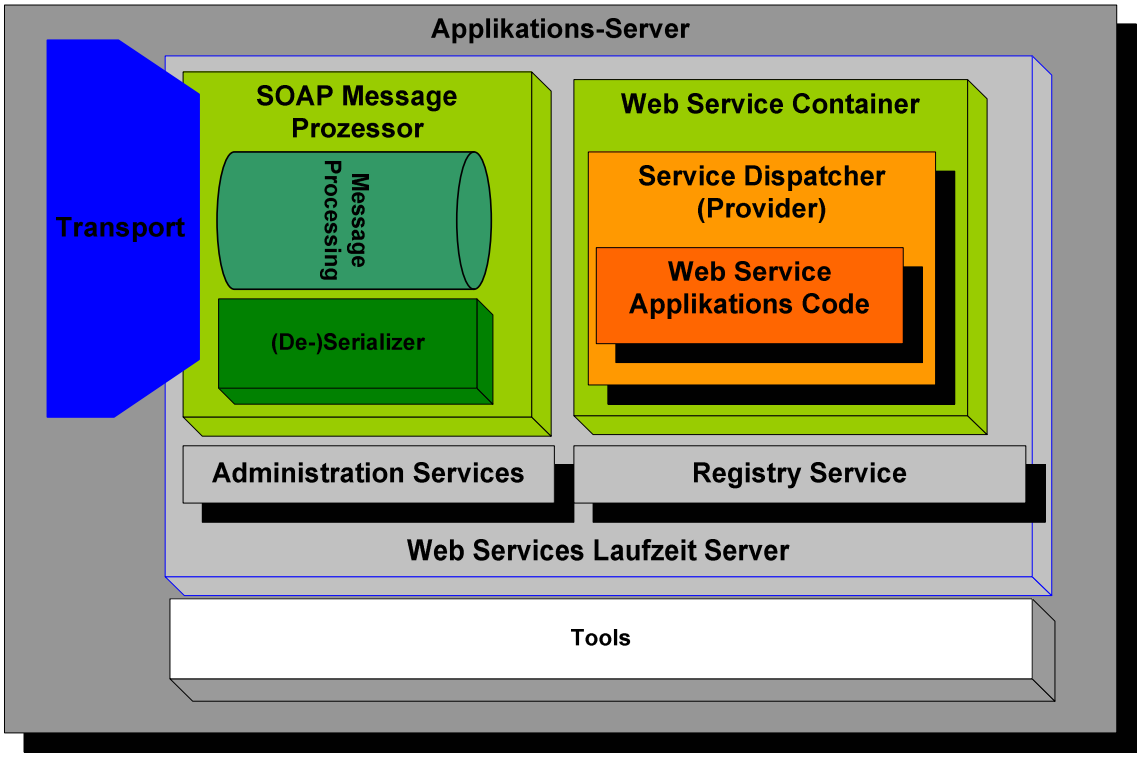


Kein Tool kann die **Client oder Web Service Applikation** generieren.

Dagegen ist dies bei **WSDD, WSDL und Proxy** der Fall.

1.2.2. Web Services Plattform

Nun wollen wir die allgemeine Architektur konkretisieren und Axis darauf abbilden (oder umgekehrt):

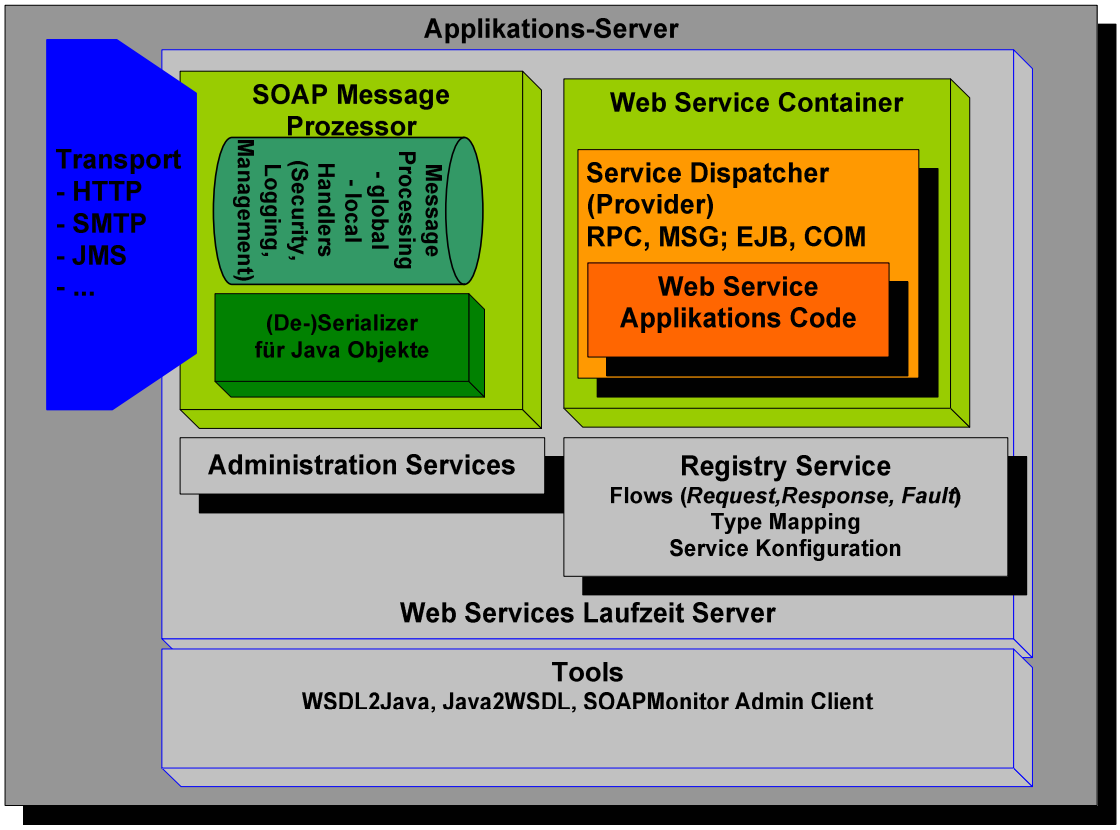


Ein Applikations-Server muss, wenn er SOAP fähig sein will, in der Lage sein, mit unterschiedlichen Transport-Protokollen zu kommunizieren (Transport Komponente).

Die Messages werden im SOAP (oder XML Format) empfangen. Je nach eingesetzten Datentypen müssen eigene Serializer und Deserializer eingesetzt werden (insbesondere wenn Sie Datentypen verwenden, welche durch die Standards nicht unterstützt werden).

Diese Architektur wurde in etwa in Axis realisiert. Unabhängig von Axis macht der Aufbau Sinn. Sprachspezifische „Handler“ kennen selber erstellt und mit dem Framework kombiniert werden.

1.2.3. Die Axis Komponenten



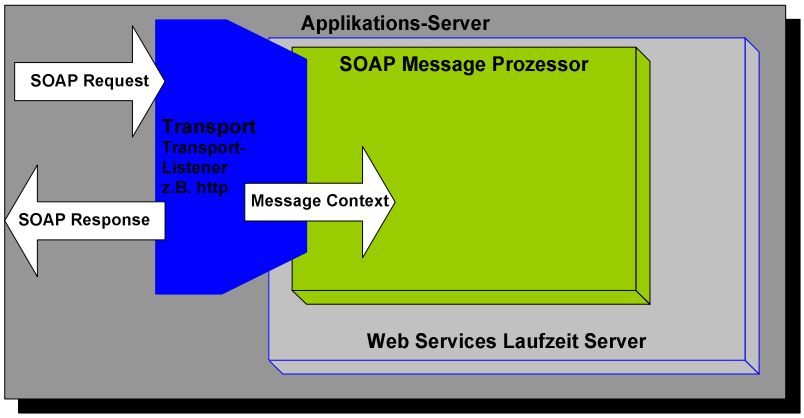
Die konkrete Axis Architektur kenn verschiedene Handler Ebenen:

- globale (aus server-config.wsdd)
- eigene ergänzende Handler (wie beispielsweise für eine Logging Applikation: die Anzahl der Zugriffe auf einen bestimmten Service werden gelogged).

1.2.3.1. Der Transport

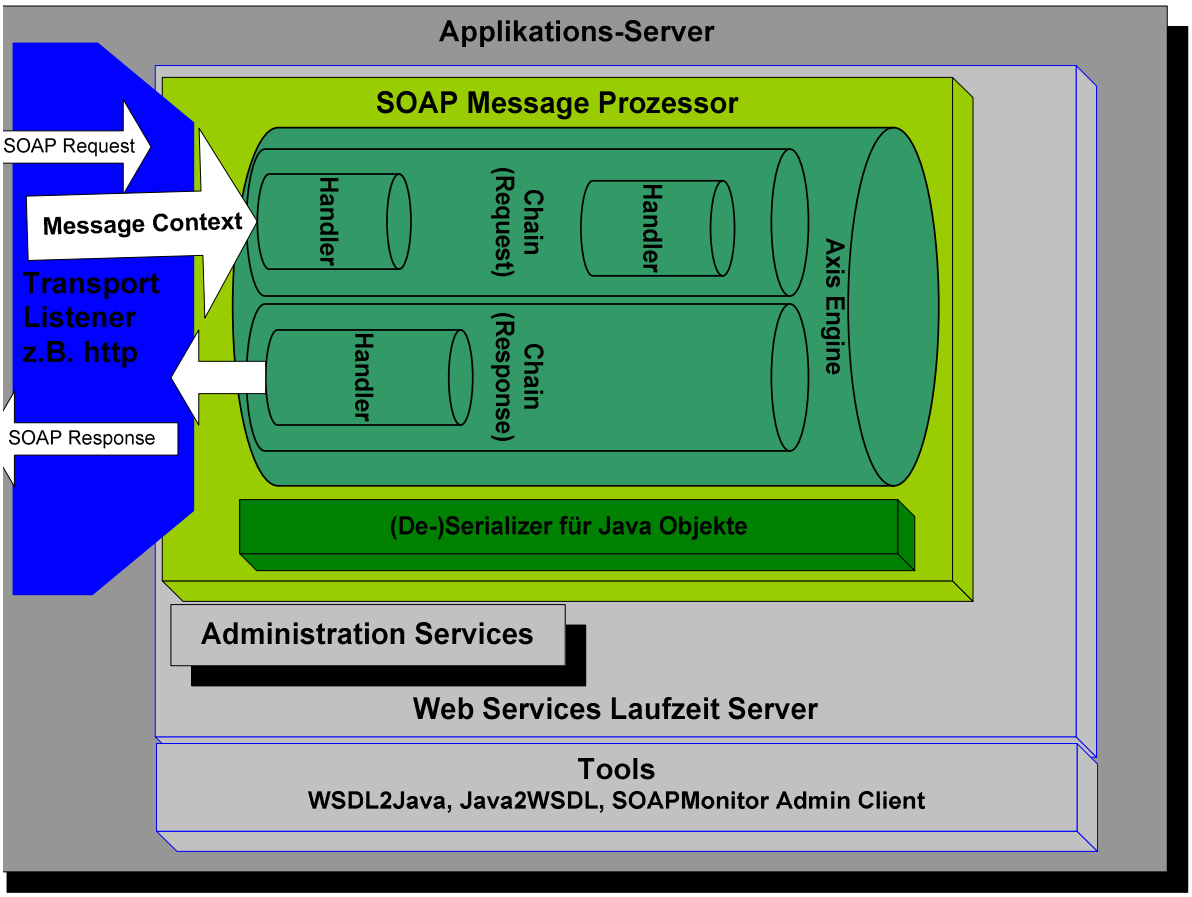
Die Transport-Komponente ist verantwortlich für:

- das Interface auf die physikalische Netzwerk-Komponenten
- für den Empfang der SOAP Anfragen
- für das Kreieren des Message Inhalts (SOAP konform)
- für das Versenden der SOAP Message



1.2.3.2. Message Processing

Die eigentliche Verarbeitung geschieht mithilfe von **Handlern**. Mehrere Handler können zu einer **Chain** zusammengefügt werden.



Aufgaben des Message Processings in Axis

- Identifikation der Initialisierungs-Blöcke und des SOAP Knotens (wer?, was?)
- Abbildung der SOAP Message auf Java Objekte
- Ausführen der Handler und Chains
- Verarbeitung der SOAP Header Informationen
- Weiterleitung an den Service Dspatcher
-

Im Folgenden müssen wir den Aufbau der Handler genauer berücksichtigen. Wie oben erwähnt, kann zwischen globalen und eigenen oder lokalen Handlern unterschieden werden.

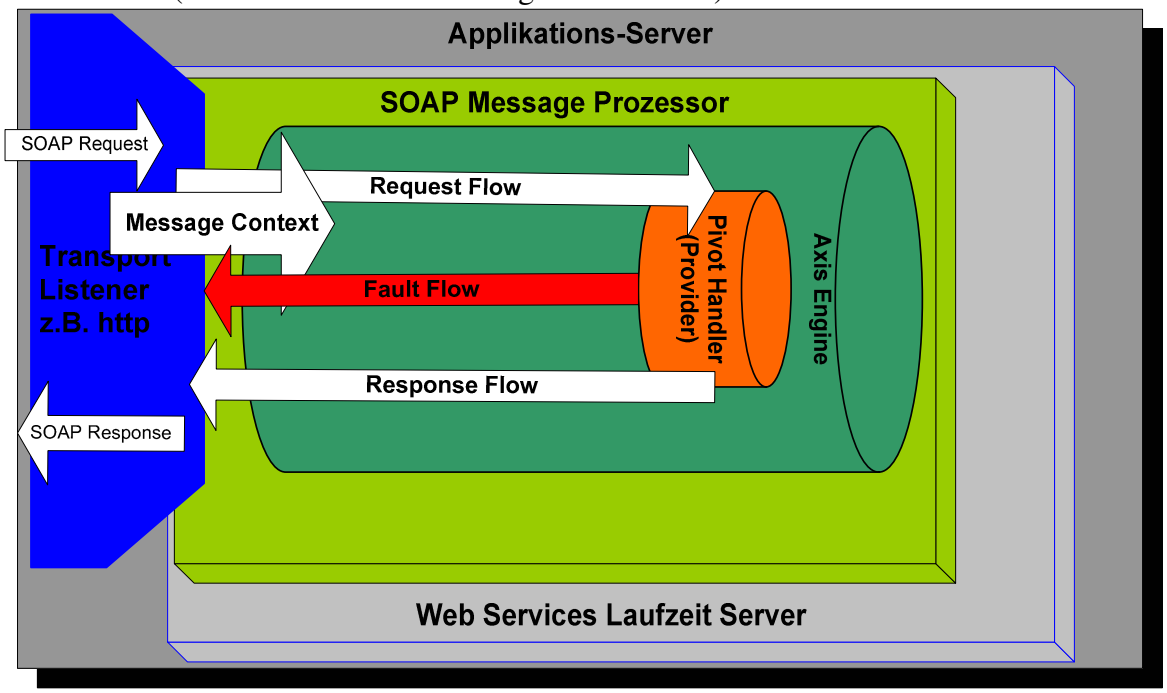
In diesem Zusammenhang istder Message Flow durch die Axis Engine wichtig.

1.2.3.3. Message Flows

Die Aufgabe eines Message Flow Systems, wie es mithilfe der Handler im Axis Engine implementiert wurde, sind:

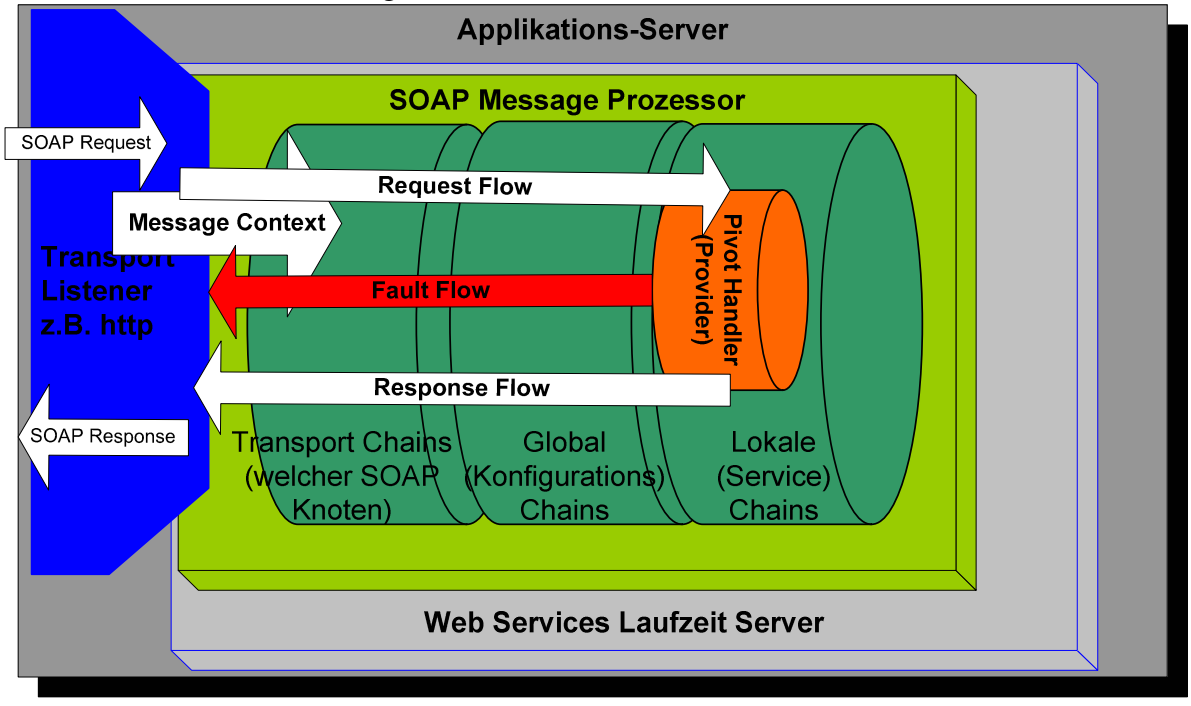
- die eindeutige Zuordnung der verarbeitenden Blöcke für den konkreten SOAP request
- Verarbeitung / Generierung der SOAP Response
- Generierung und Weiterleitung allfälliger Fehlermeldungen (Fault Flow).

Schematisch:(auch fehler müssen weitergeleitet werden).



1.2.3.4. Processing Scope – Handler und Chains in Aktion

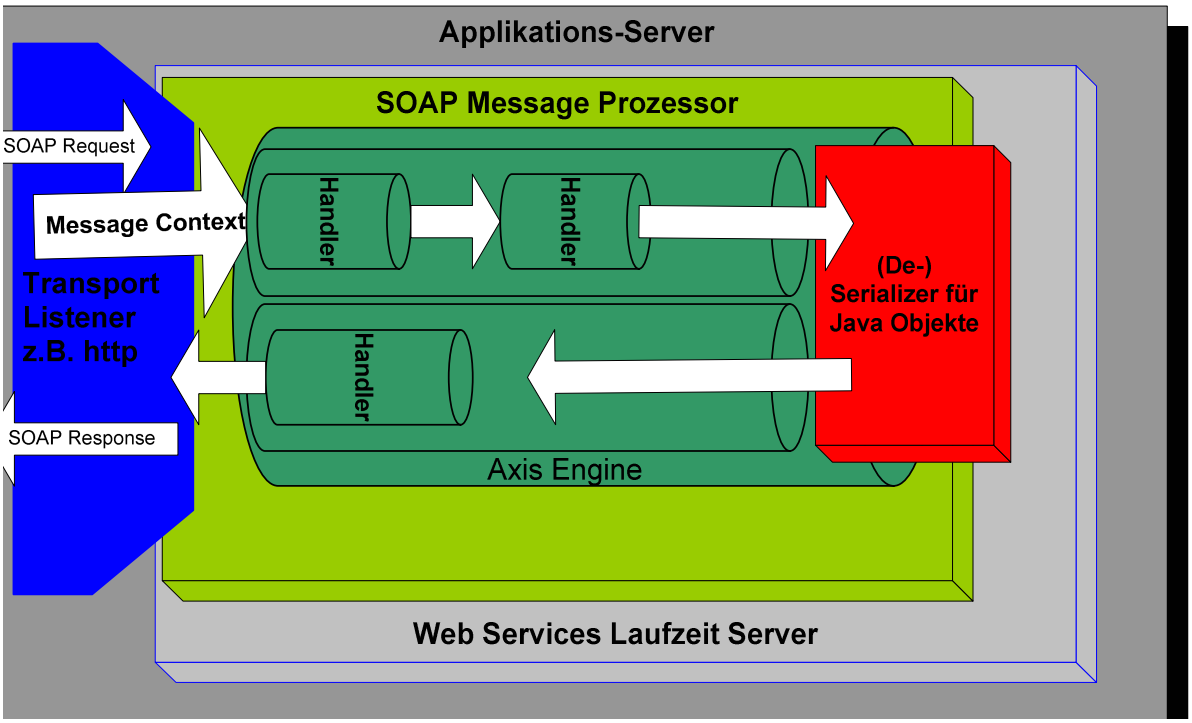
Handlers und Chains müssen gefunden und aktiviert werden. Allfällig auftretende Fehler müssen behandelt und weiter gemeldet werden.



1.2.3.5. Serialisierungs-Framework

Das Serialisierungs-Framework ist beispielsweise für das Datentyp-Mapping von XML auf Java und umgekehrt zuständig. Dieses muss allfällige RPC und Document Style Konventionen berücksichtigen (plus Schemata). Axis implementiert ein JAX-RPC konformes Mapping.

Das Mapping umfasst Basisdatentypen (int, char, float, double, boolean,...) aber auch generische Arrays und Java Beans, sowie eigene Serialisierungen.



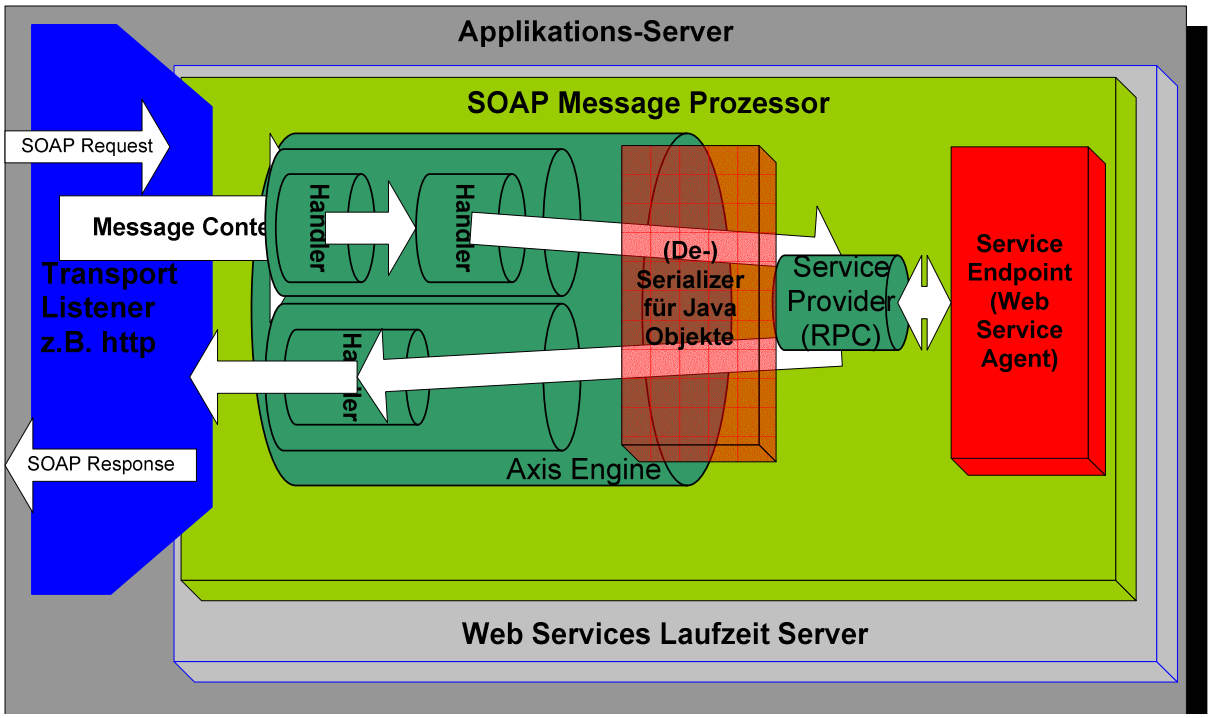
Best Practice:

- immer lesbar sind Strings / Dolumente
- Java Collections verbieten sich in heterogenem Umfeld oder falls der Dienst universell einsetzbar sein soll.
- Die Serialisierung und Deserialisierung ist beim Document Style einfacher als bei RPC (logisch: man übermittelt ja ein XML Dokument).

Was noch fehlt, ist das Dispatchen des Requests an den zuständigen Web Service.

1.2.3.6. Service Dispatching – Der Web Service

Nachdem der Request aufbereitet ist, muss er an den vorgesehenen Service weitergeleitet werden.



Zu den Aufgaben des Service Dispatchers gehören:

- das Laden des Web Service Agenten
- das Formattieren des Endpoint Calls
- die Ausführung der Business Logik
- das Serialisieren der Rückgabe als SOAP Message.

1.2.3.7. Axis Registry

Die Registry ist, wie weiter oben bereits ersichtlich, für die Server Konfiguration und Flow Kontrolle zuständig. Informationen darüber finden wir in der server-config.wsdd Datei im Verzeichnis %AXIS_HOME%/WEB-INF/. Der Inhalt dieser Datei unterscheidet sich je nachdem wie viele Services Sie deployed haben und wie Sie Ihre Axis Engine konfiguriert haben:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <globalConfiguration>
        <parameter name="adminPassword" value="admin"/>
        <parameter name="attachments.Directory" value="C:\jakarta-tomcat-5.0.16\webapps\axis\WEB-
INF\attachments"/>
        <parameter name="sendMultiRefs" value="true"/>
        <parameter name="sendXsiTypes" value="true"/>
        <parameter name="attachments.implementation" value="org.apache.axis.attachments.AttachmentsImpl"/>
        <parameter name="sendXMLDeclaration" value="true"/>
        <parameter name="axis.sendMinimizedElements" value="true"/>
        <requestFlow>
            <handler type="java:org.apache.axis.handlers.JWSHandler">
                <parameter name="scope" value="session"/>
            </handler>
            <handler type="java:org.apache.axis.handlers.JWSHandler">
                <parameter name="scope" value="request"/>
                <parameter name="extension" value=".jwr"/>
            </handler>
```

```
        </requestFlow>
    </globalConfiguration>
    <handler name="soapmonitor" type="java:org.apache.axis.handlers.SOAPMonitorHandler">
        <parameter name="wsdlURL" value="/axis/SOAPMonitorService-impl.wsdl"/>
        <parameter name="serviceName" value="SOAPMonitorService"/>
        <parameter name="namespace" value="http://tempuri.org/wsdl/2001/12/SOAPMonitorService-impl.wsdl"/>
        <parameter name="portName" value="Demo"/>
    </handler>
    <handler name="LocalResponder" type="java:org.apache.axis.transport.local.LocalResponder"/>
    <handler name="SOAPMmonitorService" type="java:org.apache.axis.handlers.SOAPMonitorHandler">
        <parameter name="wsdlURL" value="/axis/SOAPMonitorService-impl.wsdl"/>
        <parameter name="serviceName" value="SOAPMonitorService"/>
        <parameter name="namespace" value="http://tempuri.org/wsdl/2001/12/SOAPMonitorService-impl.wsdl"/>
        <parameter name="portName" value="Demo"/>
    </handler>
    <handler name="URLMapper" type="java:org.apache.axis.handlers.http.URLMapper"/>
    <handler name="Authenticate" type="java:org.apache.axis.handlers.SimpleAuthenticationHandler"/>
    <handler name="tracker" type="java:LogHandler">
        <parameter name="filename" value="Tracking.log"/>
    </handler>
    <service name="AdminService" provider="java:MSG">
        <parameter name="allowedMethods" value="AdminService"/>
        <parameter name="enableRemoteAdmin" value="false"/>
        <parameter name="className" value="org.apache.axis.utils.Admin"/>
        <namespace>http://xml.apache.org/axis/wsdd/</namespace>
    </service>
    <service name="Version" provider="java:RPC">
        <parameter name="allowedMethods" value="getVersion"/>
        <parameter name="className" value="org.apache.axis.Version"/>
    </service>
    <service name="SOAPMonitorService" provider="java:RPC">
        <parameter name="allowedMethods" value="publishMessage"/>
        <parameter name="scope" value="Application"/>
        <parameter name="className" value="org.apache.axis.monitor.SOAPMonitorService"/>
    </service>
    <service name="urn:xmltoday-delayed-quotes" provider="java:RPC">
        <requestFlow>
            <handler type="soapmonitor"/>
        </requestFlow>
        <responseFlow>
            <handler type="soapmonitor"/>
        </responseFlow>
        <parameter name="allowedMethods" value="getQuote test"/>
        <parameter name="className" value="StockQuoteService"/>
    </service>
    <transport name="http">
        <requestFlow>
            <handler type="URLMapper"/>
            <handler type="java:org.apache.axis.handlers.http.HTTPAuthHandler"/>
        </requestFlow>
    </transport>
    <transport name="local">
        <responseFlow>
            <handler type="LocalResponder"/>
        </responseFlow>
    </transport>
</deployment>
```

Globale (ganz am Anfang) und lokale Handler sind klar ausgezeichnet.

Die einzelnen Services, welche deployed wurden, sind klar erkennbar:

- Name und Typ
- Parameter : Klasse, erlaubte Methoden
- Allfälliger alternativer Request / Response Flow (um beispielsweise eine Anzeige im SOAPMonitor zu ermöglichen).

1.2.4. Kurzzusammenfassung – Message Flow

Verarbeitungsketten (*Chains*) bestehen aus einer geordneten Menge von *Handlern*.

Die Message wird protokollspezifisch empfangen. Dafür ist ein **Transport Listener** zuständig. Dieser Listener kann typischerweise als http Servlet (vom Framework) zur Verfügung gestellt werden.

Der (Transport-) Listener wandelt die protokollspezifischen Daten in ein **`Message`** Objekt (`org.apache.axis.Message`) um und stellt dieses Objekt in den MessageContext.

Der Listener fixiert auch den **`transportName`** String im Message Context, beispielsweise „http“.

Der Listener übergibt den Message Context an die Axis Engine. Diese überprüft zuerst das **`Transport`** Objekt, welches Informationen über die *Request Chain* und eventuell die *Response Chain* enthält. Falls eine Request Chain vorliegt, wird den Message Context an die `invoke()` Methode übergeben und somit alle Handler aus der Request Chain aufgerufen.

Danach folgt die Bearbeitung einer allfällig vorhandenen *Global Request Chain* und der in der Chain enthaltenen Handler.

Falls einer der Handler das serviceHandler Attribut des Message Contexts setzt (die sgeschieht typischerweise im **`URLMapper`**, sofern das Protokoll http ist), dann kann der passenden Service aufgerufen werden. Der URL Mapper bildet beispielsweise die URL auf den in der Axis Engine vorhandenen Dienst ab (Beispiel: **`http://localhost:8080/axis/services/urn:xmltoday-delayed-quotes`** auf **`StockQuoteService`**).

Schliesslich muss ein *Provider* angegeben werden, also ein Handler, welcher die Business Logik des Services implementiert. Falls die Anfrage vom RPC Typus ist, wird der Handler duch die Klasse **`org.apache.axis.providers.java.RPCProvider.class`** repräsentiert. Der Aufbau der Klasse ist denkbar einfach:
- die Klasse fragt den Message Context ab und bestimmt Methode und Parameter

```
    public void processMessage(MessageContext msgContext,
                               SOAPEnvelope reqEnv,
                               SOAPEnvelope resEnv,
                               Object obj)
            throws Exception {

        SOAPService service = msgContext.getService();
        ServiceDesc serviceDesc = service.getServiceDescription();
        OperationDesc operation = msgContext.getOperation();
...
```

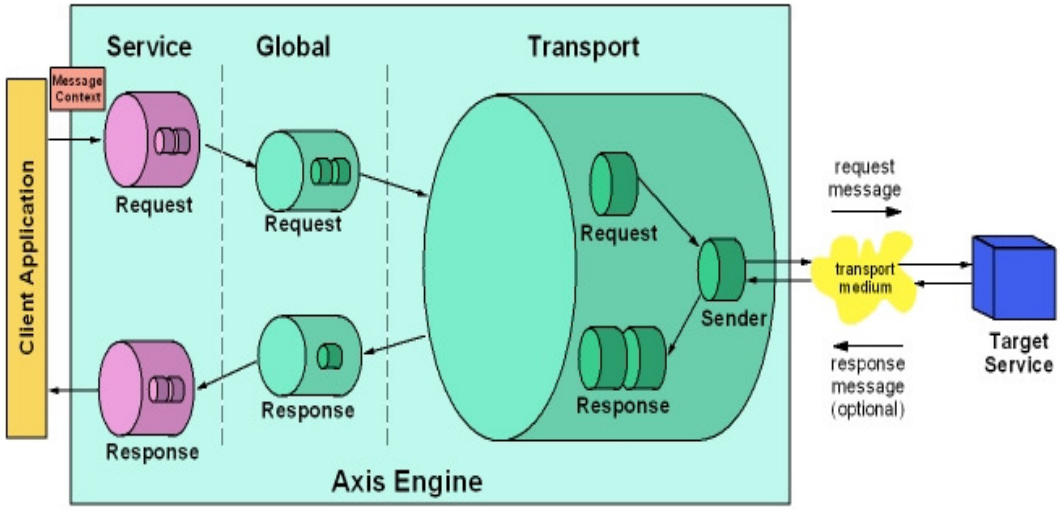
Der Handler kann auch auf die Informationen aus dem Deployment zugreifen. Diese Informationen werden in der Registry bzw. im obigen Server WSDD File abgespeichert.

1.2.5. Message Pfad beim Client

Die obigen Bilder sind symmetrisch d.h. beim Client verlaufen die Verarbeitungsketten im Wesentlichen umgekehrt:

- der Client Proxy empfängt die Message in einem Transport Handler, Teil des Client Axis Engine
- der *Transport* Handler reicht die Informationen an
- die *globalen Handler* weiter und diese schliesslich
- zum lokalen *Service* (Request / Response)
- der Provider Layer fehlt natürlich auf der Client Seite
- dieser Layer kommuniziert schliesslich mit der *Client Anwendung*.

Das folgende Bild aus der Axis Dokumentation veranschaulicht diesen Ablauf

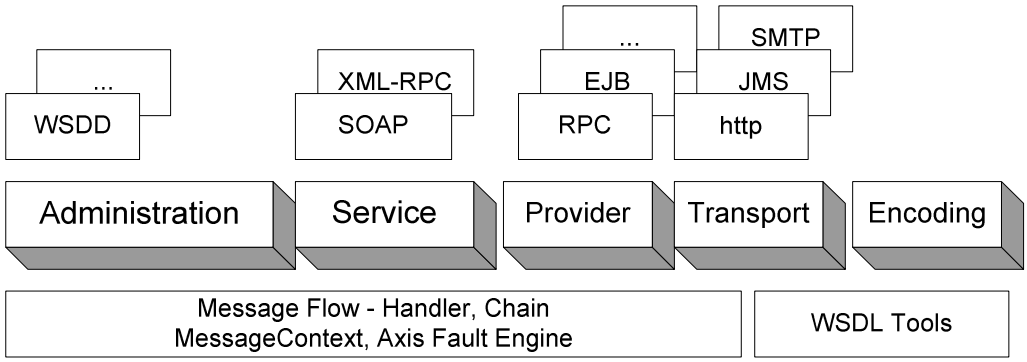


Die eigentliche Arbeit muss vom Transport *Sender* erledigt werden:

- dieser muss dafür sorgen, dass die Protokollanbindung korrekt durchgeführt wird
- eine allfällige Antwort eines Services wird ins responseMessage Feld des MessageContext gestellt und
- der Message Context wird dann an die zuständigen Verarbeitungsketten weiter gereicht
  - zuerst an die Transport Handler
  - dann an die globalen Handler
  - und schliesslich an die Service Handler

1.3. Axis Subsysteme

Axis bestehet aus mehreren Subsystemen, wobei die Aufteilung in einzelne Pakete im Sourcecode nicht genau der funktionalen Aufteilung entspricht.



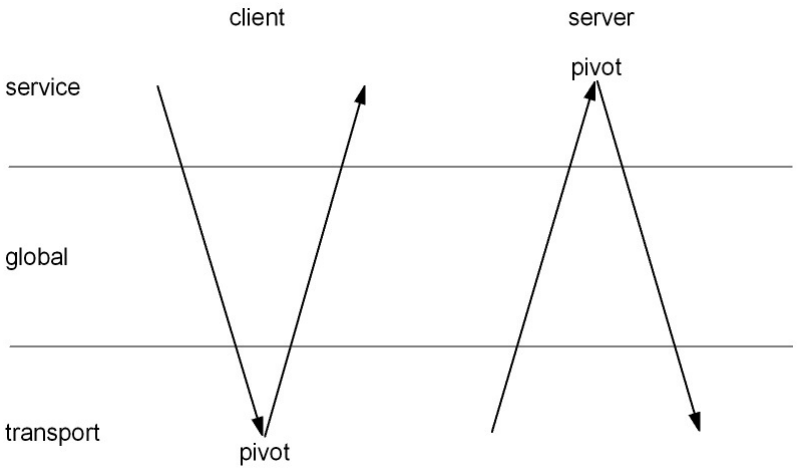
1.4. Message Flow Subsystem

Weiter oben haben wir einen Handler als Pivot bezeichnet. Um was handelt es sich dabei?

1.4.1. Handler und Verarbeitungsketten / Chains

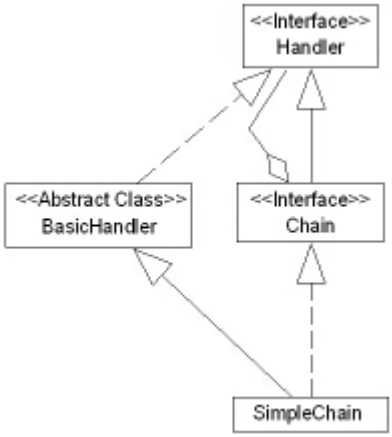
Handler werden in einer bestimmten reihenfolge aufgerufen und verarbeiten dann die message. Falls ein Handler einen Request empfängt und eine Antwort versendet oder eine Anfrage bearbeitet anschliessend eine Antwort kreiert, dann wird dieser Handler als *Pivot* Punkt bezeichnet.

Wie bereits weiter oben gesehen, können Chains aus Transport, globalen und Service Handlern bestehen, oder ganzen Teilchains. Ein typischer Ablauf könnte etwa folgendermassen aussehen:



Die Pivots sind leicht erkennbar: bei diesen werden die Meldungen nicht einfach weitergeleitet sondern bearbeitet.

Der Aufbau einer Chain aus Handlern wird im folgenden UML Diagramm wiedergegeben:



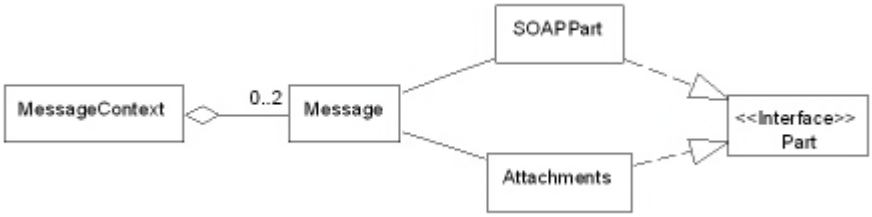
(das Diagramm lässt erkennen, dass eine Chain in etwa dem Design Pattern „Chain of Responsability“ entspricht).

Chains können, falls sie online sind, geklont werden, falls gleichzeitig mehrere Anfragen bearbeitet werden müssen.

Die Registry besitzt Factories, um unterschiedliche Handler und Chains zu bauen.

1.4.2. Message Contexts

Eine Skizze des UML Diagramms des Message Contextes zeigt, dass dieser aus Messages bestehend aus zwei Teilen, SOAP Part und Attachments, besteht, wobei beide Klassen aus dem Interface Part abgeleitet werden:

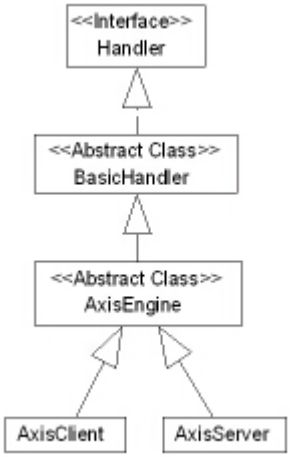


1.4.3. Axis Engine

Die Axis Engine besteht aus einer abstrakten Klasse, welche zwei konkrete Unterklassen besitzt:

- AxisClient
- AxisServer

Sie haben oben bereits gesehen, dass Client und Server symmetrisch sind. Jetzt sehen Sie den Grund dafür:



1.5. Schlussbemerkungen

Soweit zur Architektur von Axis. Es bleibt viel zu deuten und zu verstehen. Die beste Quelle für diese Zusammenfassung war für mich der Quellcode und einige Hilfstools, mit denen ich diesen genauer angeschaut habe.

Die Axis Architektur Dokumentation ist unvollständig, auch die Infos bei Axis sind unvollständig und vom Typus „tbd“ /to be done.

Axis ist aber ein System, ein Framework, welches sich auf dem richtigen Weg befindet:

- auffalend einfach
- erweiterbar
- ... (ich wiederhole die Marketingsprüche der Axis Bauer).

APACHE AXIS