



Customization Using Interceptors

Using an interceptor-based framework for providing customized client-side and server-side Web Service extension behaviour

Jorgen Thelin
Chief Scientist
Cape Clear Software Inc.

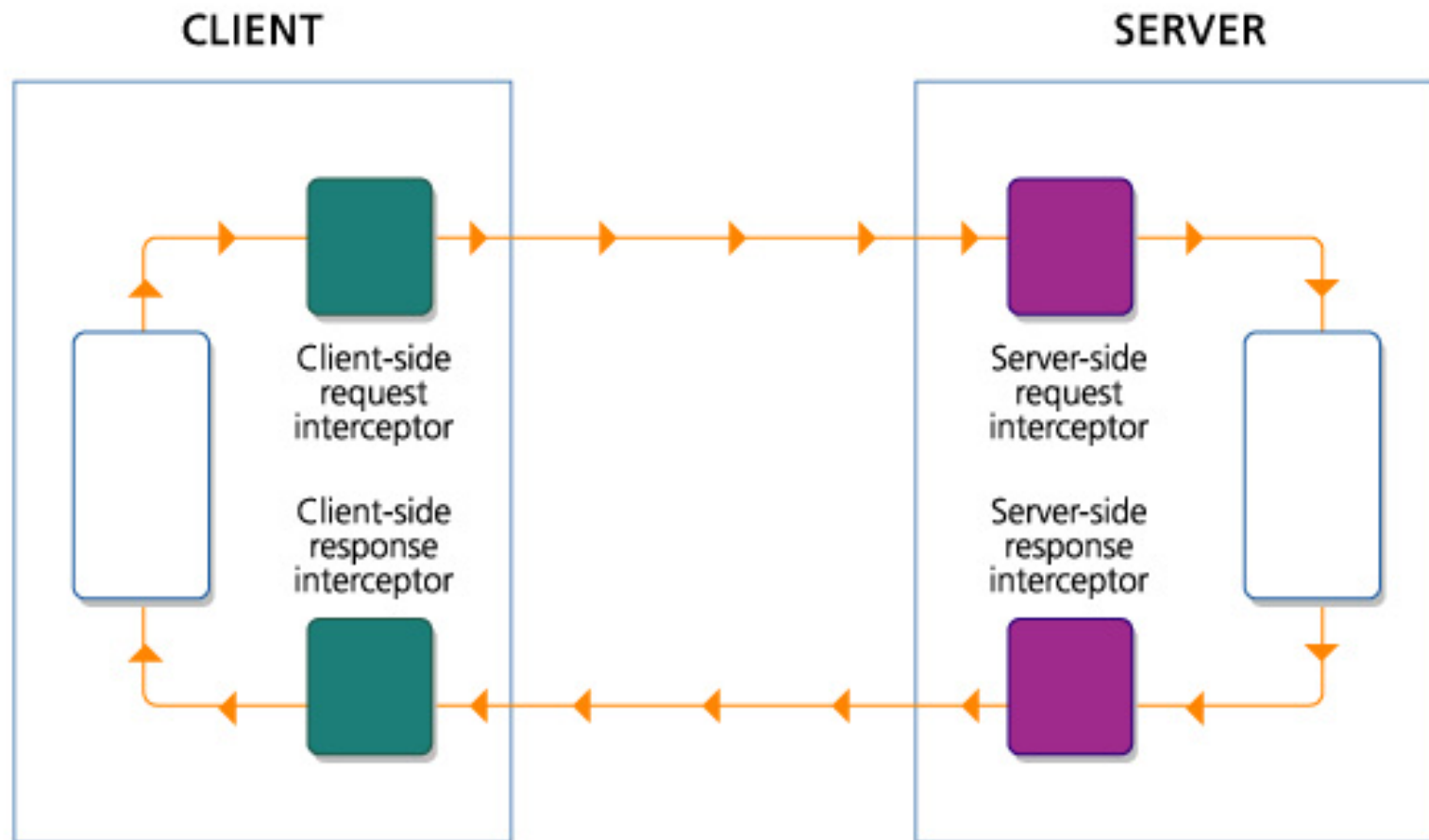


Interceptors

- ✦ Interceptors are a general purpose concept for customizing and controlling message processing
- ✦ Interceptors provide a framework for changing the steps involved in processing a message
- ✦ Interceptors can be used both server-side and client side

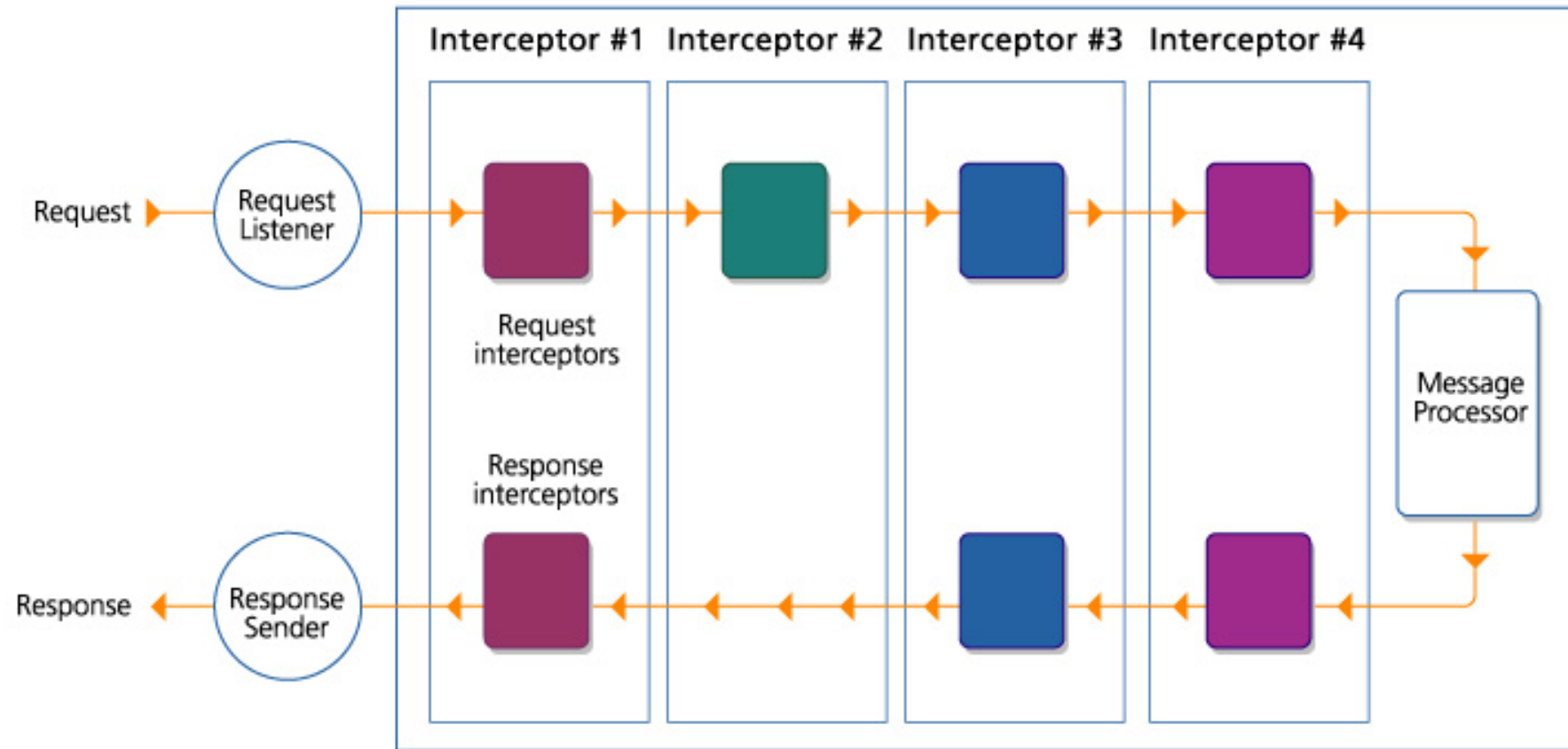


Interceptor Plugin Architecture



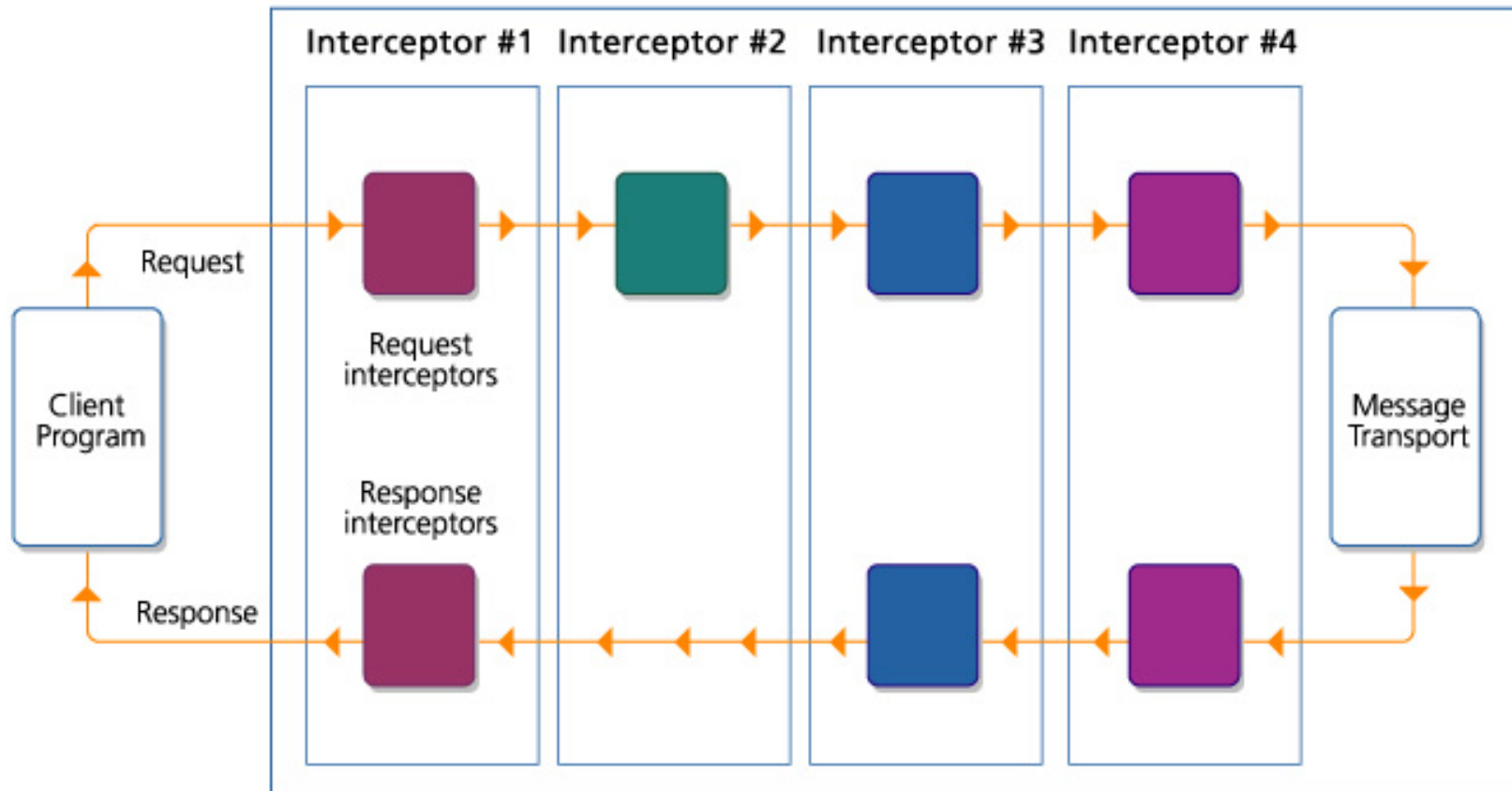


Interceptor Plugins – Server-side





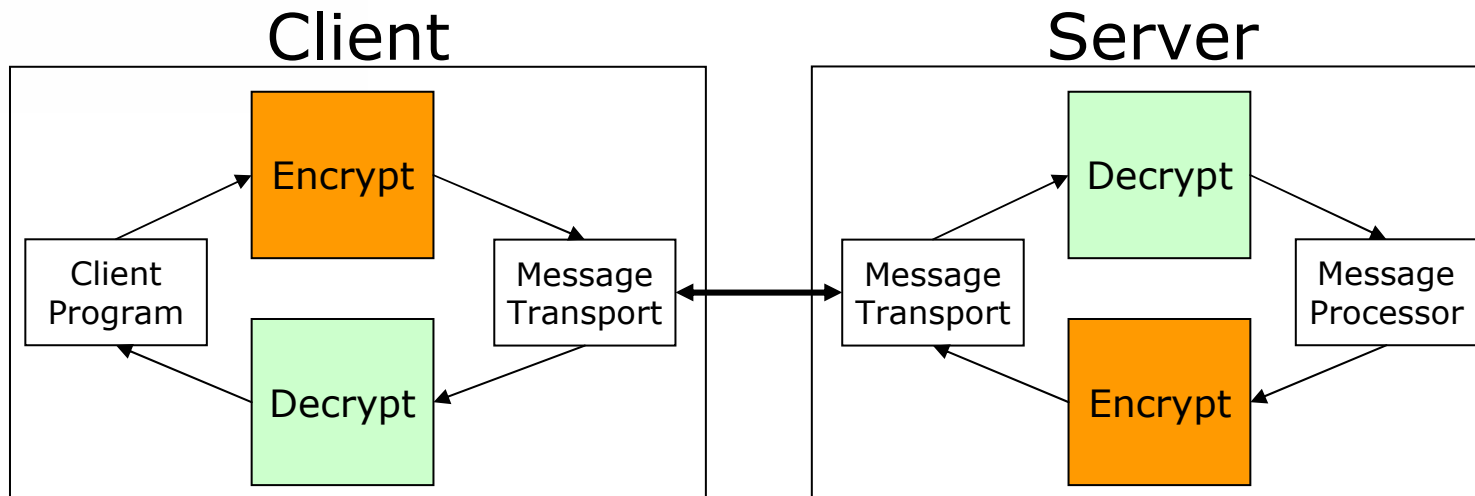
Interceptor Plugins – Client-side





Inbound and Outbound Interception Points

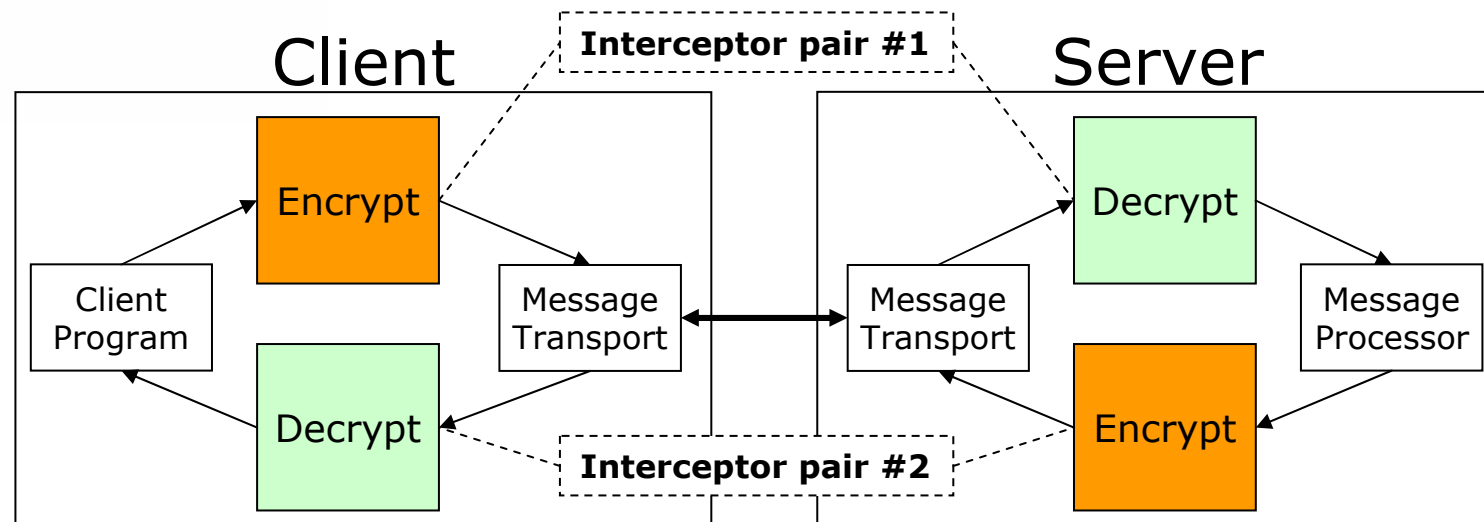
- ✧ Often have similar functionality at “opposite” interceptor points
 - Eg. Server-side **response** interceptor may **en**crypt data in the reply just like the client-side **request** interceptor **en**crypts data in the request message
- ✧ The **request** message is:
 - Outbound for the client
 - Inbound for the server
- ✧ The **response** message is:
 - Inbound for the client
 - Outbound for the server





The “Interceptor-Pair” Concept

- ✧ Usually require a **pair** of interceptors to fulfil a task
 - One on the client side
 - eg. Request interceptor which encrypts the data in a field
 - One on the sever side
 - eg. Request interceptor which decrypt the field data
- ✧ The functions of the interceptor pair need to match up
 - Eg. Both request interceptors cannot both encrypt or both decrypt
- ✧ This is often referred to as an input-output interceptor pair





JAX-RPC Handlers (aka "SOAP Interceptors")

✦ JAX-RPC Handlers

- Provide a standardized interface for a "SOAP interceptor" in Java
- Operate on the SOAP object model defined by SAAJ (SOAP with Attachments API for Java)
- Are part of the JAX-RPC extension module which will be in J2SE 1.4 and J2EE 1.4
- Provides a MessageContext object for passing call-related context information between interceptors and callbacks



JAX-RPC handler interface: `javax.xml.rpc.handler.Handler`

✧ JAX-RPC interface `javax.xml.rpc.handler.Handler`

- Interception Operations:

- `boolean handleRequest(MessageContext context)`
 - Called with each request message
- `boolean handleResponse(MessageContext context)`
 - Called with each response message
- `boolean handleFault(MessageContext context)`
 - Called with each fault message

- Interrogation Operations:

- `QName[] getHeaders()`
 - The names of the header blocks processed by this Handler

- Lifecycle Operations:

- `void init(HandlerInfo config)`
 - Called at the start of the lifecycle of the Handler instance
- `void destroy()`
 - Called at the end of the lifecycle for the Handler instance.



Microsoft Web Service Extensions (WSE)

- ✧ Web Services Enhancements 1.0 for Microsoft .NET
- ✧ WSE is a class library that implements additional Web Services functionality and advanced protocols
 - Diagnostics and tracing
 - WS-Security
 - WS-Routing
 - WS-Referral
- ✧ Provides a SoapContext object for passing call-related context information between interceptors and callbacks



WSE Filter Interfaces

✧ Interface **SoapOutputFilter**

- Interception Operations:
 - `void ProcessMessage(SoapEnvelope envelope)`
- Lifecycle Operations:
 - *Standard object constructor*
 - *Standard object destructor*

✧ Interface **SoapInputFilter**

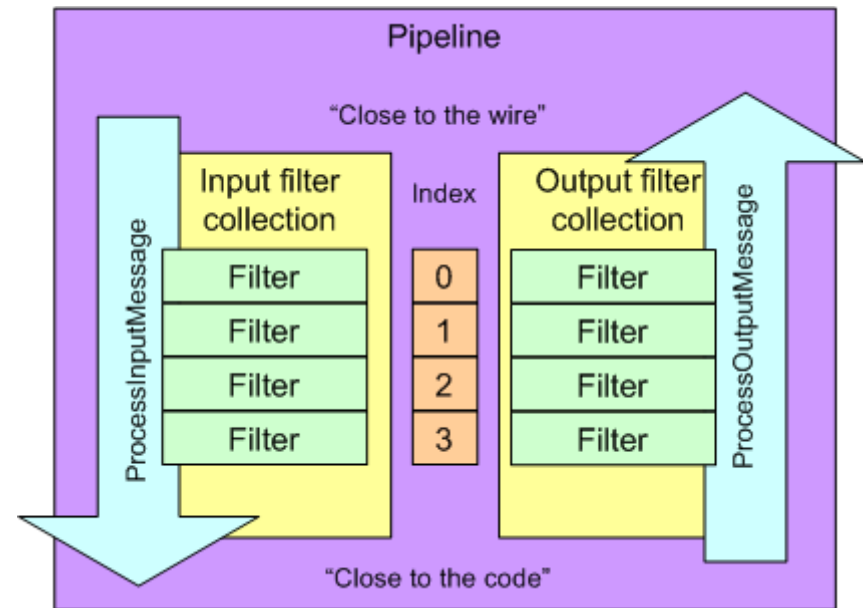
- Interception Operations:
 - `void ProcessMessage(SoapEnvelope envelope)`
- Interrogation Operations:
 - `bool CanProcessHeader(XmlElement header, SoapContext context)`
- Lifecycle Operations:
 - *Standard object constructor*
 - *Standard object destructor*



Microsoft Web Service Extensions (WSE) Framework

- ✧ WSE is based on the architectural model of a pipeline of **filters** that process inbound and outbound SOAP messages
- ✧ The pipeline controls execution order
- ✧ Output filters are called in reverse order
- ✧ Filters can be integrated with ASP.NET or used in standalone code

Filter ordering in the WSE pipeline



Source: MSDN

More info: <http://msdn.microsoft.com/library/en-us/dnwebsrv/html/insidewsepipe.asp>



WSE Built-in Filters

Namespace	Input Filter	Output Filter	Purpose
Microsoft .Web.Services .Diagnostics	TraceInputFilter	TraceOutputFilter	Write messages to log files to help with debugging
Microsoft .Web.Services .Security	SecurityInputFilter	SecurityOutputFilter	Authentication, signature and encryption support (WS-Security)
Microsoft .Web.Services .Timestamp	TimestampInputFilter	TimestampOutputFilter	Timestamp support (WS-Security)
Microsoft .Web.Services .Referral	ReferralInputFilter	ReferralOutputFilter	Dynamic updates to routing paths (WS-Referral)
Microsoft .Web.Services .Routing	RoutingInputFilter	RoutingOutputFilter	Message routing (WS-Routing)



Server-side Interceptors

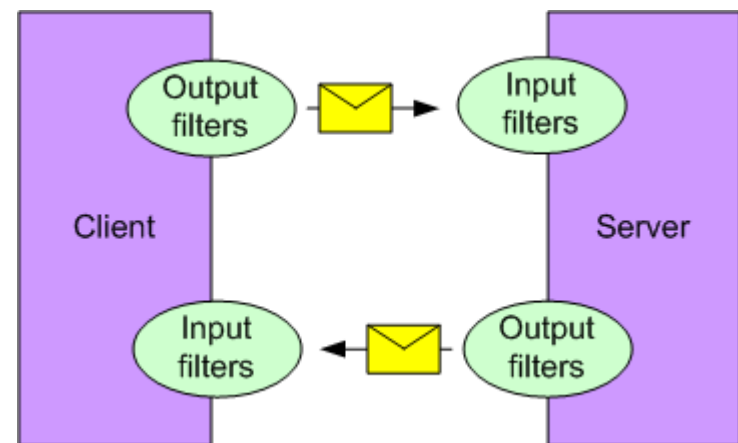
- ✧ Both JAX-RPC Handlers and .NET WSE Filters fit naturally as server-side interceptors
 - Inbound interception points handle requests
 - Outbound interception points handle response



Client-side Interceptors – WSE Filters

- ✧ .NET WSE Filters fit in naturally as client-side interceptors too
 - Output filters handle requests
 - Input filters handle responses
- ✧ Can directly reuse filters written for server-side use
 - Eg. Decryption interceptor is always an input filter

The WSE filter model

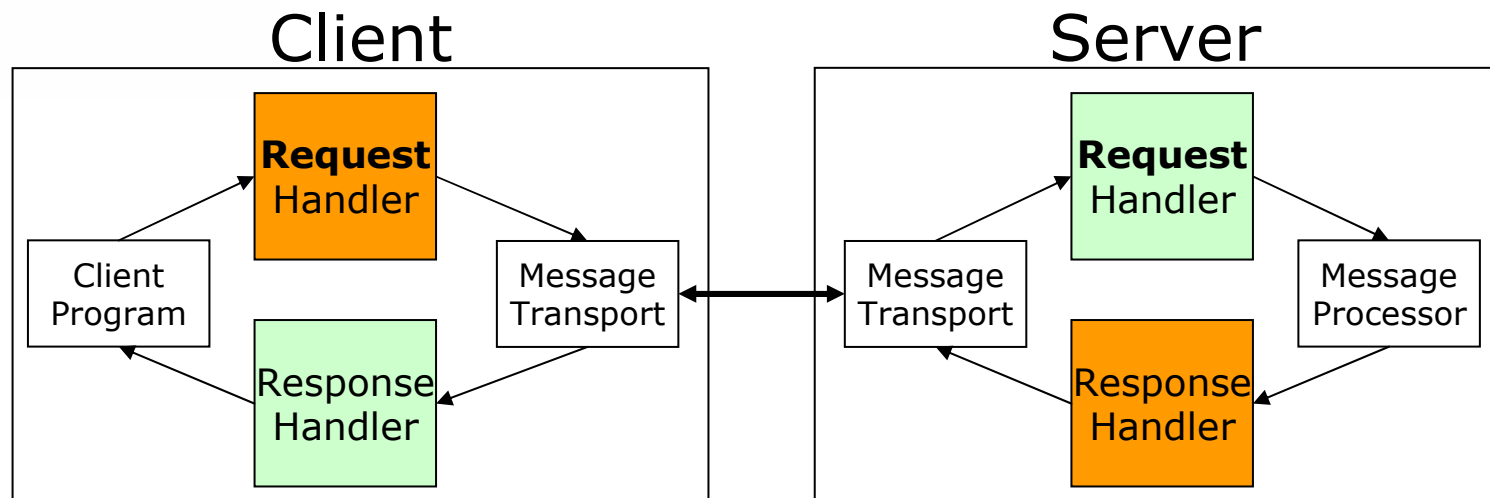


Source: MSDN



Client-side Interceptors – JAX-RPC Handlers

- ✧ JAX-RPC Handlers do not fit so cleanly as client-side interceptors:
 - **handleRequest** needs to do different things on the client-side (outbound) and server-side (inbound)
 - **handleResponse** needs to do different things on the client-side (inbound) and server-side (outbound)
- ✧ Need to write different JAX-RPC Handlers for client and server-side, or use a “configuration flag” to swap behaviour round



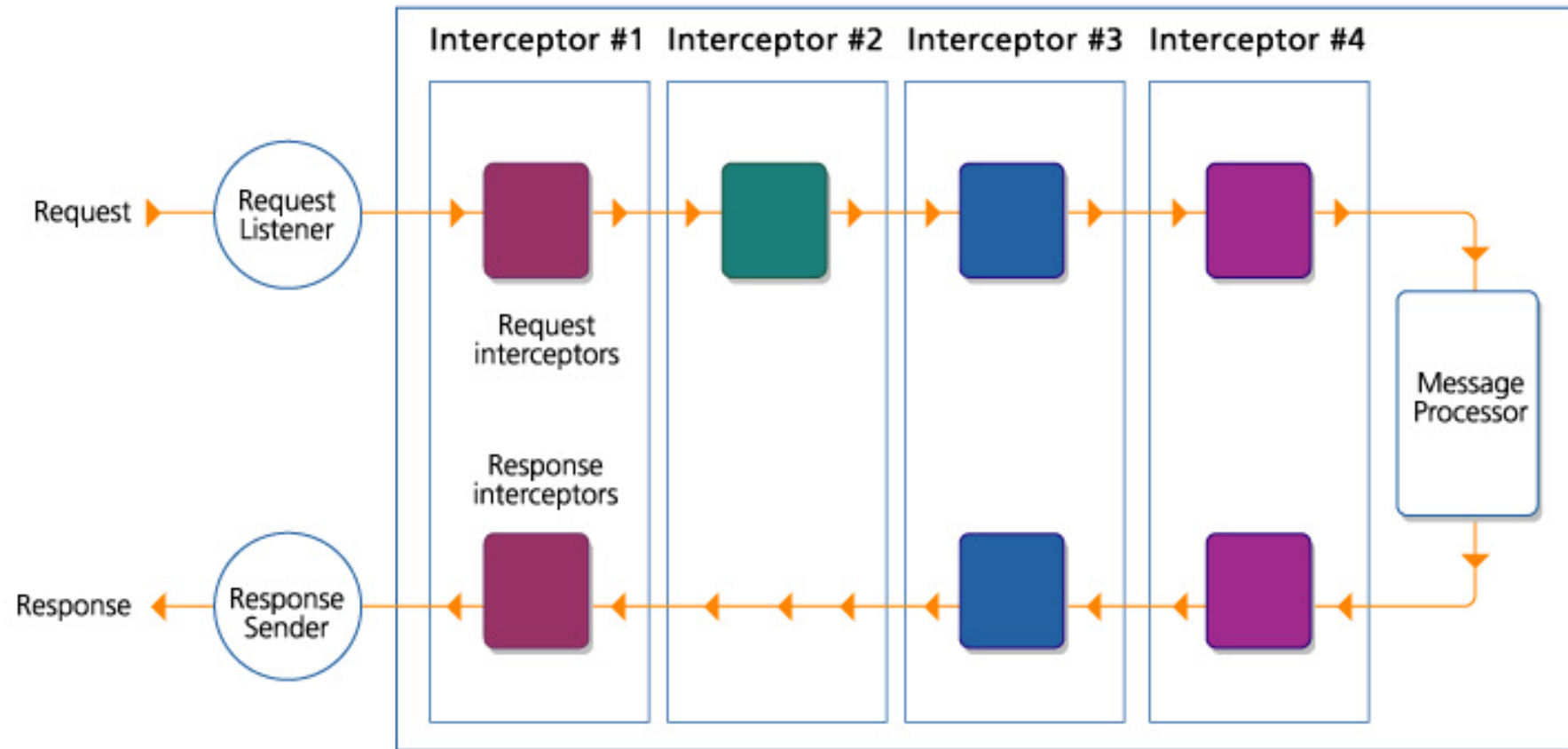


Combining Interceptors in "Chains"

- ✧ Interceptors are usually independent so can be combined to achieve a desired function
- ✧ For example, implementing a security policy for a Web Service:
 - An interceptor to decrypt a SOAP message which was transmitted using XML Encryption
 - An interceptor to recognise and decode SAML authentication assertion carried in the WS-Security header of the SOAP message
 - An interceptor to perform a role-based access control check that the caller is a member of the group of people permitted to call this operation



Interceptor Chain – Server-side





Interceptor Configuration in Cape Clear Server



Cape Clear Manager - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media Print

Address <https://localhost:8444/console/session.con?page=login> Go Links Post to MT Bl

Google Search Web Search Groups Search Directory News PageRank Page Info Up

CAPE CLEAR MANAGER  

HOME LOGOUT HELP

localhost

- Server
- Adapters
- Integration
- Router
- Security
- Transports
- Web Services
 - Deploy Web Service
 - Add JMS Web Service
- Web Applications

Configured Interceptors

Below is a list of configured Interceptors currently being used by this Web Service application. Configured Interceptors can be removed, or the information associated with the Interceptors can be edited. The position an Interceptor is invoked by the Web Service application can be changed by clicking on the arrows aligned to the current position.

To add an Interceptor to this Web Service application click: **ADD**

Configured Interceptors for Web Service application (cc-config)			
Position	Name	Type	Actions
(1) ▼	Authentication Credentials Interceptor	SoapInterceptor	EDIT REMOVE
(2) ▲	AdminRole Interceptor	CallInterceptor	EDIT REMOVE

BACK

Cape Clear Manager Local intranet



Aspect-oriented Programming (AOP)

- ✧ Aspect-oriented Programming (AOP) is a way of implementing **separation of concerns** (SOC) in software.
- ✧ AOP make it possible to **modularize** crosscutting “aspects” or “concerns” of a system.
 - For example:
 - Logging policies
 - Diagnostics
 - Transactional contexts
 - Security policy checks
- ✧ Separation of Concerns (SOC) makes software much easier to develop, construct and understand
- ✧ More info on AOP / AOSD at: <http://aosd.net/>



Interceptors as AOP

- ✦ Interceptors are a form of Aspect-oriented Programming (AOP)
 - Plug in an interceptor to deal with a specific function such as message validation or logging
 - Can change the external visible behaviour by adding an interceptor to modify data before or after processing
 - Can change the external visible behaviour by adding an interceptor to “short-circuit” processing
- ✦ Interceptors provide an ideal way to implement reusable “policy” aspects of a system
 - For example: Access-control checks
- ✦ Interceptors provide an extensibility framework for Web Service protocols and applications
 - For example: Adding WS-Security credentials



Example – Custom Authentication Headers

✦ Scenario:

- Client program needs to communicate with a Web Service which requires an session-based authentication dialog
- When calling a “start session” operation, the response message contains a custom header which must be resubmitted with all future requests



Example – Custom Authentication Headers

✦ Implementation:

- A custom client-side interceptor can be written to deal with this specific situation
- Client-side response interceptor:
 - Preserve the authentication token found in the SOAP header of the response message
- Client-side request interceptor:
 - Add any preserved authentication token into a SOAP header of the next request message



Example Code – Custom Auth Headers

```
public class SoapCorrelationHeaderInterceptor
    implements Handler
{
    SOAPHeaderElement correlationHeader; // OK for single-threaded client

    String correlationHeaderElementNamespace;
    String correlationHeaderElementName;
    Name correlationHeaderName;

    public void init( HandlerInfo info )
    {
        Map cfg = info.getHandlerConfig();

        correlationHeaderElementNamespace = (String) cfg.get( "header.ns" );
        correlationHeaderElementName = (String) cfg.get( "header.name" );
    }

    public void destroy()
    {
    }

    public QName[] getHeaders()
    {
        return new QName[] {
            new QName( correlationHeaderElementNamespace, correlationHeaderElementName )
        };
    }

    // More below
}
```



Example Code – Custom Auth Headers

```
public boolean handleResponse( MessageContext context )
{
    try {
        // Dig into the response message and extract the contents of the token header

        SOAPMessage soapMessage = ((SOAPMessageContext)context).getMessage();
        SOAPEnvelope soapEnvelope = soapMessage.getSOAPPart().getEnvelope();
        SOAPHeader soapHeaders = soapEnvelope.getHeader();

        if (this.correlationHeaderName == null) {
            this.correlationHeaderName = soapEnvelope.createName(
                correlationHeaderElementName, null,
                correlationHeaderElementNamespace );
        }

        if (soapHeaders != null) {
            this.correlationHeader =
                extractNamedHeader( correlationHeaderName, soapHeaders );
        }
    }
    catch (SOAPException se) {
        throw new JAXRPCExceptionImpl( se );
    }

    return true; // Continue processing
}
```



Example Code – Custom Auth Headers

```
protected SOAPHeaderElement extractNamedHeader(
    Name headerName, SOAPHeader soapHeaders )
    throws SOAPException
{
    Iterator iter =
        soapHeaders.getChildElements( headerName );

    if (iter.hasNext()) {
        SOAPHeaderElement soapHeaderField =
            (SOAPHeaderElement) iter.next();

        // Remove from SOAP Message element tree
        return soapHeaderField.detachNode();
    }
    else {
        return null;
    }
}
```



Example Code – Custom Auth Headers

```
public boolean handleRequest( MessageContext context )
{
    if (this.correlationHeader != null) {
        try {
            // Dig into the request message and add the contents of the token header

            SOAPMessage soapMessage = ((SOAPMessageContext)context).getMessage();
            SOAPEnvelope soapEnvelope = soapMessage.getSOAPPart().getEnvelope();
            SOAPHeader soapHeaders = soapEnvelope.getHeader();
            if (soapHeaders == null) { soapHeaders = soapEnvelope.addHeader(); }

            soapHeaders.addChildElement( this.correlationHeader );
        }
        catch (SOAPException se) {
            throw new JAXRPCExceptionImpl( se );
        }
    }

    return true; // Continue processing
}

public boolean handleFault( MessageContext context )
{
    return true; // Continue processing
}
```



Conclusion

- ✦ Interceptors provide a highly extensible processing architecture
- ✦ Interceptors allow incremental enhancements to functionality through writing small amounts of code
- ✦ Interceptors are the way enhanced protocol support is being added to SOAP platforms
- ✦ Interceptors provide an ideal way to implement custom security logic for Web Services