Refactoring Context

Marc Hadley
Sun Microsystems, Inc

October 6, 2004

1 Introduction

In the current draft, the client-side context is subdivided into three separate compartments:

Request A property bag, of type `JAXRPCContext`, accessible from the message context as a property named `javax.xml.rpc.handler.context.request`. When a method is invoked on a `BindingProvider` (`Stub`, `Dispatch` or `Call` instance), this property bag is initialized by copying the contents of the `JAXRPCContext` accessible from `BindingProvider.getRequestContext()`.

Response A property bag, of type `JAXRPCContext`, accessible from the message context as a property named `javax.xml.rpc.handler.context.response`. For synchronous methods, the contents of this property bag are copied to the `JAXRPCContext` accessible from `BindingProvider.getResponseContext()` when a method invocation completes. For asynchronous methods, this property bag is made available via the `Response.getContext()` method.

Message A property bag, of type `MessageContext`, accessible only to handlers. The value of the `javax.xml.rpc.handler.context.response` property is exposed to client code when a method invocation completes, handlers can access all properties.

Table 1 shows context values before, during and after, an example method invocation.

This current compartmentalization scheme has a number of advantages and disadvantages:

+ The properties in the request context can be set up before a sequence of method invocations since the results of each invocation do not affect the request context of subsequent invocations.
+ Only a subset of the properties set by handlers are visible to client code.
+ Properties that will be made available to client code are clearly demarked from those that are private to handlers.

Table 1: Context value before, during and after, an example method invocation (based on current draft)

State	Context		
	Request	**Message**	**Response**
Before	`foo=bar`[a]	`null`	*Empty*

During	`foo=bar`	`foo2=bar2`[b]
		`javax.xml.rpc.handler.context`
		`.request={foo=bar`[c]`, foo3=bar3`[d]`}`
		`.response={foo4=bar4`[e]`}`
			Empty

After	`foo=bar`	`null`	`foo4=bar4`[f]

[a] An example property set by a client on a binding provider.
[b] An example message context property set by a handler.
[c] Copied from request context.
[d] An example request context property set by a handler.
[e] An example response context property set by a handler.
[f] Copied from response property in message context.

- The subdivisions are a client side only construct, writing a handler that works on client and server can be more complex because of this.
- Its not entirely intuitive that changes made to the request context property are not visible to the client code.
- There's no inheritance relationship between `MessageContext` and `JAXRPCContext` though they share many identical methods and a similar purpose.

2 Proposed Changes

In order to clean up the relationship between `MessageContext` and `JAXRPCContext`, to improve the symmetry between client and server sides, and to maintain the existing advantages listed above the following changes are proposed:

1. Make `MessageContext` extend `JAXRPCContext`.
2. Add the following to `MessageContext`:

```
1  public enum Scope {APPLICATION, HANDLER};
2  public void setPropertyScope(String name, Scope scope);
3  public Scope getPropertyScope(String name);
```

3. Eliminate the current client-side compartmentalization such that:
    - Properties set in the request context prior to a method invocation are used to seed the message context for outbound messages.

- Only properties whose scope is `APPLICATION` are made available in the response context after a method invocation.

4. Default scope is `HANDLER`. Properties have to be explicitly marked as visible to the application. Request context properties are not normally copied to the response context unless their scope is changed to `APPLICATION` by a handler.

Table 2 shows context values before, during and after, an example method invocation with the changes proposed above.

Table 2: Context value before, during and after, an example method invocation (based on proposed changes)

	Context		
State	**Request**	**Message**	**Response**
Before	`foo=bar`[a]	`null`	*Empty*
During	`foo=bar`	`foo(HANDLER)=bar`[b]	*Empty*
		`foo2(HANDLER)=bar2`[c]	*Empty*
		`foo3(APPLICATION)=bar3`[d]	*Empty*
After	`foo=bar`	`null`	`foo3=bar3`[e]

[a] An example property set by a client on a binding provider.
[b] Copied from request context with `HANDLER` scope.
[c] An example message context property set by a handler with `HANDLER` scope.
[d] An example message context property set by a handler with `APPLICATION` scope.
[e] Copied from message context.

If issue 12 is resolved by defining different types of endpoints:

Legacy An endpoint that can obtain a `MessageContext` or `SOAPMessageContext` from `ServletEndpointContext.getMessageContext`. Supported for backwards compatibility with JAX-RPC 1.1 endpoints.

Protocol neutral An endpoint that can obtain a `LogicalMessageContext` from `ServletEndpointContext.getMessageContext`.

Then it would be possible to extend the property scoping rules to the server side. If scoping is extended in this way then it might be appropriate to define a `ServiceContext` class to encapsulate the property scoping behavior.