An Analysis of Web Services Mediation Architecture and Pattern in Synapse

Chen Wu and Elizabeth Chang
Digital Ecosystems and Business Intelligence Institute
Curtin University of Technology, Perth, Australia
E-mail:{chen.wu, elizabeth.chang}@cbs.curtin.edu.au

Abstract

Web services mediation – the process of bridging disparate service requesters and providers – is of paramount importance. However, few existing open projects are directly working on service mediation. Although mediation has become an essential part of commercial Enterprise Service Bus (ESB) products, they use proprietary solution, which cannot lead to a high level understanding of the mediation architecture. And the open source ESBs do not really focus on modeling the mediation[1]. After exploring the nature of service mediation problem, this paper presents a systematic evaluation of the support for mediation architecture and patterns in Synapse, a leading Apache open source project that provides a mediation framework for Web services. Based on this evaluation analysis, the paper identifies several interesting mediation research directions and, more importantly, reveals the linkage between Synapse and our ongoing work in providing a distributed web services mediation network.

1. Introduction

The key premises of Web services are standardization and interoperability, which pave the underpinning for Internet-scale integration. However, Web services R&D is still in its emerging phase. As such, the technologies and specifications that various organizations have adopted and implemented can be very different. Such difference might be caused by, among other factors, a) the ambiguous understanding of these complicated WS-* standards, b) insufficient comprehension of the interactions between service providers (SP) and service requesters (SR). Moreover, from the business perspective, web services are typically provided by different organizations and hence are designed not to be dependent of any collective computing entity but to reflect a certain degree of autonomy. This local autonomy brings about service mismatch in terms of, for example, the formats of data and the semantics of interfaces.

To remedy the heterogeneity issue without degrading the degree of loose-coupling, the notion of service mediation is first proposed in the Enterprise Service Bus (ESB) industry community [1-3]. In general, service mediation enables a Service Requester to connect to a relevant Service Provider regardless of heterogeneity. Being a central part of ESB, service mediation sits between the service requester and provider and works in a transparent way – neither of them needs to be aware of its existence. In addition to facilitating interaction, service mediation can also be used for service management by monitoring messages between requester and provider.

In this paper, we approach web services mediation from the architectural perspective since "mediation is primarily an architectural concept" [4]. Commercial ESB products use proprietary solution, which cannot lead to a high level understanding of the mediation architecture due to its unclear internal component design, implementation, and prohibitive purchase cost. Some open source ESBs do not really focus on modeling the mediation[1]. This paper presents a systematic evaluation of the mediation architecture and patterns in Synapse[5], a leading Apache open source project. We start by formally defining service mediation in terms of data, message, and process. This clarification identifies the source/target of the mediation problem/solution in web services contexts. We then link the abstract mediation patterns with concrete Synapse mediator solutions. This provides an effective evaluation metrics for the development of Synapse. In addition, we provide a detailed analysis of Synapse architecture and the mediator extension framework, which, together with the pattern evaluation result, lead to several future research issues and directions in the area of service mediation.

2. Preliminary Concepts

The early research of mediation [4, 6] is driven by two central data issues – data abstraction and data mismatch. Going beyond the data, mediation, in a broader sense, comprises a cognitive process of reconciling two mutually interdependent sides. In

21st International Conference on
Advanced Information Networking and Applications Workshops (AINAW'07)
0-7695-2847-3/07 $20.00 © 2007 IEEE

SOA, we define the service mediation as *an intelligent process reconciling differences in data, message, and process between service requester and service provider*. This definition of mediation is depicted in Figure 1, where three levels of mediation can be observed – data, message, and process.

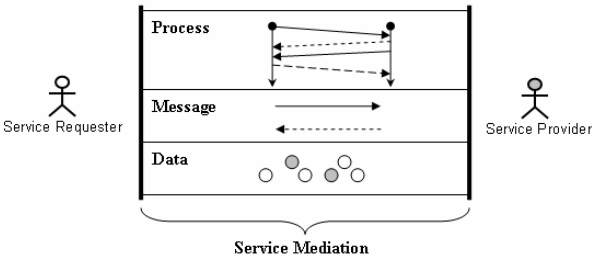


Figure 1. Three levels of Service Mediation

Data mediation has received substantial research since web services are typically provided by different organizations with different policies. This brings about enormous dissimilarities regarding the data type, data format, and data semantics. Data mediation thus includes tasks such as data transformation – translation, reconciliation, and creation – and data synthesis such as data integration from multiple data sources.

Message mediation concerns the way of moving data. Web services technology is built on top of the messaging exchange mechanism. The need for message mediation can be easily seen from the service interface specification – WSDL2.0, where eight sets of Message Exchange Patterns (MEP) are defined [7]. It is not feasible for all SPs or SRs to implement all these MEPs due to various reasons such as technical complexity (e.g. the state maintenance induced by the 'Asynchronous Out-In'), business policies and rules, etc. Moreover, many WS applications only support WSDL1.1, where only four patterns are used. Different message patterns represent different means to interact with. Without appropriate message mediation only a small number of SPs and SRs can communicate.

Process mediation occurs when orchestrating sets of messages into business scenario engaged by both SR and SP. It involves determining "how two public processes can be matched in order to provide certain functionality" [8], such as online ticket booking from a virtual SP. Process mediation is particularly important in service transactions which require: a) mediation for sets of shared message exchanges agreed upon by both requester and provider, b) mediation for monitoring, enforcing terms, and QoS guarantee[9]. The sets of message exchange here may be composed of the technical-level MEPs discussed in the message mediation, but it reflects the semantic choreography which stipulates the business-level interaction agreed upon by both sides.

Service mediation is an essential part of Enterprise Service Bus (ESB), an SOA enabling middleware providing integration facilities built on top of web services industrial standards [3]. ESB considers the mediation as "manipulating messages in-flight on the 'bus'" [2] in which messages initially sent by a service requester are transformed into messages understood by a incompatible service provider. It covers all three levels of service mediation. Nevertheless, commercial ESB products use proprietary solution, which cannot help us to achieve a thorough comprehension of the mediation architecture. While the open source ESBs can be studied extensively, they do not really focus on modeling the mediation as stated in [1]. In this paper, we choose Apache Synapse [5] that is dedicated to the role of mediation in SOA solutions. It allows messages flowing through, into, or out of an organization to be mediated by a set of mediators that can be easily re-configured. The built-in mediators support features such as: simple interception based on regular expression and *XPath* rules, logging, routing, *XSLT* transformation of payload, and stages in/out handling of messages. Although this project is still in its infancy (currently in the Apache incubation phase), its internal architecture and mediator framework is coherent with the concept of mediation and mediator that we have presented above. For example, Synapse treats mediator as a first-class 'citizen' in its architecture.

3. Mediation Patterns Evaluation

In this section we take a closer look at the fundamental mediation patterns that can be constructed with Synapse's native support or workaround solution depicted in the snippet. For each pattern, we provide a figure to visualize the message flow where a Mediator (M) is depicted as a circle; the Service Provider (SP) is denoted as a pentagon. The rectangle indicates the Service Requester (SR) that sends the request message (the connecting arrows) to the Mediator. The snippet explanation (if any) comes in the form of inline XML comments. Due to the page limit, the customized java mediators which contain business mediation logics are not included here for further examination.

3.1. Transform

Snippet 1

```
<synapse xmlns=
"http://ws.apache.org/ns/synapse">
   <xslt name="T-med"
xsl="/tranformation/warning.xsl" type="body"/>
</synapse>
```

In *Transform* pattern, the mediator translates the message payload (content) from the requester's schema to the provider's schema. This may include (de-)enveloping, data formatting transformation, data type conversion, data filtering, data re-presentation, or encryption. Synapse implements this pattern by introducing the XSLT mediator, an internal processor that converts the message payload against the specified XSLT document. In Snippet 1, the message is examined by the XSLT mediator based on the rule that expired transaction message content cannot be captured by the service providers and hence are filtered out and transformed into texts as if they are encrypted.

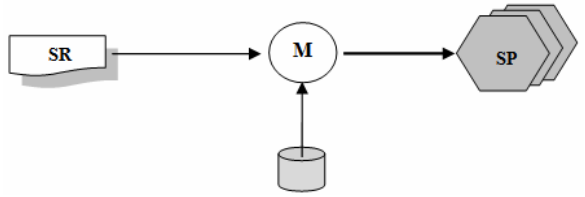
3.2. Protocol Switch

Snippet 2

```
<transportReceiver name="http"
<!—switch to HTTP protocol -->
class="org.apache.axis2.transport.http.
SimpleHTTPServer">
        <parameter name="port"
locked="false">6060</parameter>
</transportReceiver>
<transportReceiver name="jms"
<!—switch to JMS protocol -->
class="org.apache.axis2.transport.jms.
SimpleJMSListener">
</transportReceiver>
```

Protocol Switch pattern allows the broker to transform requests into the targeted service provider's preferred protocols. For example, it enables a service requester to dispatch its initial HTTP POST messages to a provider that expects different communication protocols, such as SOAP/HTTP, JMS, SMTP, or TCP etc. Synapse does not support this pattern in its mediation configuration language. But it implements this pattern with the underlying service communication platform – Axis. Snippet 2 is extracted from the Axis configuration file which includes the transport protocols supported by Axis1.1.

3.3. Enrich



Snippet 3

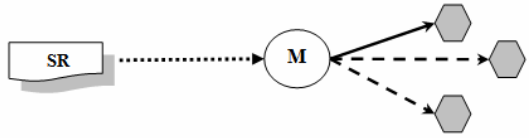
```
<synapse
xmlns="http://ws.apache.org/ns/synapse">
   <!-- request from the service requester -->
<in>
  <classmediator
<!— customized java mediator using mediator
extension framework -->
   class="com.acme.mediator.CheckCreditInfo">
    <send/>
</in>
</synapse>
```

In *Enrich* pattern, the mediator expands the message payload by attaching extra data, which comes from external data sources such as: a) customization parameters defined by the mediation; b) database queries to enrich the meaning and hence to improve the usefulness to the target SRs or SPs. *Enrich* pattern is not directly supported by Synapse mediators. However, Synapse provides the class mediator extension mechanism as shown in Snippet 3, where the incoming message is parsed to resolve the *customer ID*. The *CheckCreditInfo* mediator then retrieves the credit information, and appends the additional credit points to the original message, which is then sent (<*send*/>) to the service provider to process the loan application.

3.4. Route

In *Route* pattern, the mediator forwards the message to an appropriate provider based on the requester's intent. Selection criteria include message content and context, or the service provider's capabilities. It is a sort of 'on the fly' provider selection compared to the static service selections. Such a dynamic characteristic of *Route* pattern enables payload-level message routing for each message. Hence, *Route* pattern supports more flexible, loose-coupled, and fine-grained message exchange between SRs and SPs. Synapse implements *Route* pattern through two mediators: *regex* and *xpath*. The *regex* checks the regular expression pattern against the property or the header of the *SynapseMessage* (e.g. "to" header type) – the intent of the message, and passes on the message to its sub-mediators if the pattern matching is evaluated as 'true'. In this case the first sub-mediator – header sets the new header value (i.e. new stockquote service endpoint address) to the

headerType (e.g. 'to') of a *SynapseMessage*. In Snippet 4, this is specified in the header element attribute "type" and "value". The *xpath* mediator executes an Xpath (i.e., the "*expr*" attribute) test against the message envelope and then passes on the message to its sub-mediators rules if the test result is true. In this case, further routing is needed for a particular stock price.

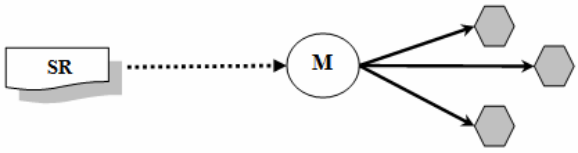


Snippet 4

```
<synapse
xmlns="http://ws.apache.org/ns/synapse">
<!-- Check if the URL matches the stockquote
pattern -->
<regex message-address="to"
pattern="/StockQuote.*">
   <header type="to"
value="http://www.webservicex.net/stockquote.as
mx" />
   <!-- check if the symbol is Acme -->
   <xpath expr="//*[wsx:symbol='ACME']"
          xmlns:wsx="http://www.webserviceX.NET/
          ">
     <!—further routing for this particular
stock -->
   <header type="to" value="http://www.au-
trader.net/stockquote.asmx" />
   </xpath>
</regex>
   </synapse>
```

3.5. Distribute (a.k.a. Clone)

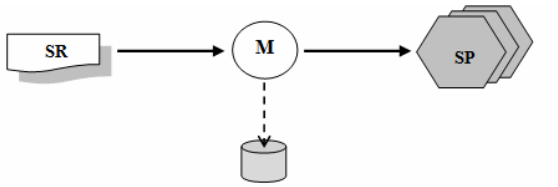
In *Distribute* pattern, the mediator distributes the message to a set of interested SPs based on their capabilities. A major difference from *Distribute* is that the message initiated by the SR is forwarded to, at most, one of multiple targets – a SP or the next mediator. Whereas the *Distribute* pattern supports a one-to-many communication, which allows a single message from the source service requester to be distributed to multiple targets concurrently. Hence this pattern is also known as *Clone* pattern, where the mediator in effect needs to make a copy of a message and modifies its route. The distribution rule can be set against SP's interests or advertisement provided to the mediator. In current Synapse, the *Distribute* pattern is not supported natively. Firstly, the concurrent message distribution cannot be expressed. The repeated configuration of *send* mediators will not help because in current single message sequential setting, whenever a <*send*> mediator is invoked the whole mediation process ceases. Moreover, the message clone is not supported by any existing mediators. However, we create a workaround solution – the distribute mediator – to implement this pattern as shown in Snippet 5.



Snippet 5

```
<synapse
xmlns="http://ws.apache.org/ns/synapse">
<!-- Check if the property matches the book
search pattern -->
<regex property="subject" pattern="search*">
   <distribute>
      <branch>
        <header type="to"
   value="http://api.google.com/search/beta2"
   />
        <send>
      </branch>
      <branch>
        <header type="to" value="
   http://soap.amazon.com/onca/soap?Service=AW
   SECommerceService " />
        <send>
      </branch>
   </distribute>
</regex>
</synapse>
```

3.6. Monitor



Snippet 6

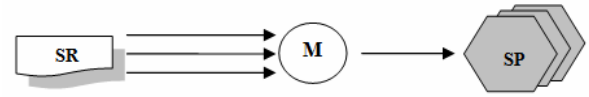
```
<synapse
xmlns="http://ws.apache.org/ns/synapse">
   <distribute>
      <branch>
        <send/>
      <branch>
      <branch>
        <log/> <!-- log the message using
log4j at the same time -->
      <branch>
   </distribute>
</synapse>
```

In *Monitor* pattern, the mediator logs the messages as they pass through the mediation without making any changes. This pattern can be used to check the quality of service, determine the message problems, meter usage for subsequent billing to users, or trace business-level events, such as transaction over a certain amount of money. It can also be used for data inspection, or for service management. In Synapse, the *log* mediator supports the *Monitor* pattern as shown in Snippet 6.

The log mediator in current Synapse simply records the basic message address such as *To*, *From*, *ReplyTo*, and implements the dummy serialization of the *SoapEnvelope* using the *Log4j*[1] – a popular java logging library. However, advanced log functionality can be easily added by enhancing this mediator. In this example, the log mediator is also used jointly with the *distribute* mediator so that the *send* mediator is executed as normal. The *log* mediator uses the same copy of the original message. This prevents the extra data monitoring from becoming the bottleneck that would affect the performance of the message flow.

3.7. Cache

In *Cache* pattern, the mediator shortens the service response time by caching service response messages. When a duplicated request message reaches the mediator, the mediator attempts to find from its local cache the previous response message and returns it directly to the SR without forwarding the request message to the remote SP. This pattern improves the performance by reducing the service response time. In Snippet 7 we illustrate how Synapse can implement this pattern using a customized external processor dedicated to providing cache services. We use *in* mediator to enclose sub-mediators that are only applied to the message initiated from the SR. Likewise, *out* mediators only deal with messages returned from the SP. Synapse does not have built-in processors for the *Cache* pattern. We use the Synapse mediator extension framework to provide the cache mediator as an external *Processor* with its associated *CacheConfigurator* that loads and instantiates the processor from the mediation configuration. The cache algorithm is provided in *CacheProcessor* class.



Snippet 7

```
<synapse
xmlns="http://ws.apache.org/ns/synapse">
   <!-- request from the service requester -->
   <in>
     <cache
     cache_svr="au.edu.ceebi.cache.CacheServer
                    " />
     <!-- this could be sendOn or sendBack -->
     <send/>
   </in>
   <!-- response from the service provider -->
   <out>
     <cache
   cache_server="au.edu.ceebi.cache.CacheServer
   " />
     <!-- sendBack -->
     <send/>
   </out>
</synapse>
```

[1] http://logging.apache.org/log4j

4. Synapse Implementation

Synapse relies on fundamental service provided by Axis2. It uses the lightweighted Axis2 object model for message processing which is more extensible, faster and developer-convenient. More importantly, it uses Axis2 as an underlying transport for message interactions which supports varieties of MEPs including the Asynchronous Web Services. The rationale here, as we see it, is to treat the mediator itself as a common web service hosted in the Axis2 service container. In particular, mediations are carried out in a pipe-uniform-filter style[10], where mediators are independent on each other to incrementally mediate the incoming and outgoing messages. Moreover, the composite pattern is also used in the mediators design so that certain relationships between mediators are maintained without complicating the processing flow.

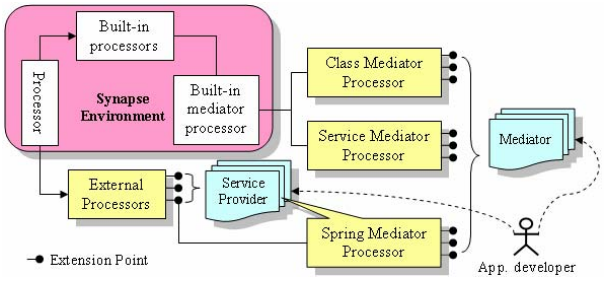


Figure 2. Synapse mediator extension framework

A crucial advantage of Synapse lies in its flexible mediator extension framework depicted in Figure 2. It enables mediation administrators to develop and deploy mediators in a loosely-coupled way. Synapse provides two fundamental extension instruments. First, Synapse API defines a generic Mediator interface, from which users can develop the mediation that fulfils specific customer requirements. The user defined mediator class is then. Alternatively, the mediator can also be deployed as an Axis web service that is then loaded into *Synapse Environment*. This extension mechanism is easy to implement and deploy. However, the *Process Configurator* cannot be customized. As a result, parameterizing the mediators becomes very difficult in this case and the mediator is thus unable to conduct mediation according to its enclosing contexts as well as to acquire constantly changing data from outside the Synapse. To overcome this difficulty,

Synapse provides external processor mechanism so that application developers can directly create the mediation processor with its associated configurator based on the J2SE Service Provider Interface (SPI) model[11].

5. Summary

To summarize, we provide a comparison in Table 1.

Table 1. Summary in Synapse.

Pattern	*Level*	*Direct Support*	*Workaround*
Transform	Data	√	
Protocol Switch	Message		√
Enrich	Data		√
Route	Message	√	
Distribute	Message		√
Monitor	Data	√	
Cache	Message		√

Due to its complexity and difficulty, process level mediation is not tackled in current Synapse, which aims to mediate message in the first place. Four out of seven mediation patterns relate to message mediation, and only one – *Route* – is directly supported by Synapse. It is also worth noting that the inability to '*support'* certain patterns (especially data mediation) in Synapse can be improved by adding new capable mediators, which are independent on the evolution of Synapse itself. However, the message level mediation needs more support from the way Synapse processes messages. Moreover, the WSDL Message Exchange Pattern mediation is not tacked by Synapse.

6. Conclusion

Mediation is largely acknowledged as one of the most important components for realizing the SOA and Web services. However, fundamental characteristics of mediation in Web services contexts are not formally explored. Commercial ESB products using proprietary mediation solutions cannot lead to comprehensive knowledge of service mediation in a broader sense such as mediation levels, requirements, and mediator architecture. Following this observation, the analysis of mediation patterns and corresponding Synapse implementation in this paper intends to fill this gap. We also examine the Synapse architecture, which facilitates these mediation solutions.

For our future work, two requirements that are not supported by Synapse are related to the distribution of the mediator architecture. In current version of Synapse [5], all the mediations are carried out on a dedicated Synapse server. Intuitively, such a centralized architecture has serious scalability problems when the number of requesters and providers increases across the Internet. Another reason driving mediator distribution is the maintenance issues [6]. Each mediator represents special knowledge from a specific domain. Mixing them together into a single "*Synapse.xml*" apparently can cause the messy situation when the mediation becomes very complex in production environment. We believe that our evaluation and analysis can be of valuable importance to define new empirical solutions for service mediation (e.g. the Synapse-based distributed mediation networks) and their mapping to the mediation patterns and architecture for our ongoing work on distributed WS mediation architecture.

7. References

[1] C. Herault, G. Thomas, and P. Lalanda, "Mediation and Enterprise Service Bus - A position paper," presented at First International Workshop Mediation in Semantic Web Services, 2005.

[2] B. Hutchison, M.-T. Schmidt, D. Wolfson, and M. L. Stockton, "SOA programming model for implementing Web services, Part 4: An introduction to the IBM Enterprise Service Bus," in *IBM Developerworks*: IBM, 2005.

[3] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen, "The enterprise service bus: Making service-oriented architecture real," *IBM Systems Journal*, vol. 44, pp. 781, 2005.

[4] G. Wiederhold and M. Genesereth, "The Conceptual Basis for Mediation Services," *IEEE Expert*, vol. 12, pp. 38 - 47, 1997.

[5] Synapse, "http://incubator.apache.org/synapse/."

[6] G. Wiederhold, "Mediators in the Architecture of Future Information Systems," *Computer*, vol. 25, pp. 38 - 49, 1992.

[7] M. Gudgin, A. Lewis, and J. Schlimmer, "Web Services Description (WSDL) Version 2.0: Message Exchange Patterns," World Wide Web Consortium, 2003.

[8] E. Cimpian and A. Mocan, "D13.7 v0.1 Process Mediation in WSMX," in *WSMX working draft*, 2005.

[9] S. Shrivastava, "Contract-Mediated Interorganizational Interactions," *IEEE Distributed Systems Online*, vol. 6, 2005.

[10] M. Shaw and D. Garlan, "Software Architecture: Perspectives on an Emerging Discipline," *Prentice-Hall, ISBN 0-13-182957-2*, 1996.

[11] SUN, "JAR File Specification - http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html#Service%20Provider," 1999.