Run-Time Adaptation of Non-functional Properties of Composite Web Services Using Aspect-Oriented Programming

N.C. Narendra[1], Karthikeyan Ponnalagu[1], Jayatheerthan Krishnamurthy[2], and R. Ramkumar[2]

[1] IBM India Research Lab, Bangalore, India
{narendra, karthik.ponnalagu}@in.ibm.com
[2] IBM India Software Lab, Bangalore, India
{jayatheerthan, ramkumar_rj}@in.ibm.com

Abstract. Existing web service composition and adaptation mechanisms are limited only to the scope of web service choreography in terms of web service selection/invocation vis-à-vis pre-specified Service Level Agreement constraints. Such a scope hardly leaves ground for a participating service in a choreographed flow to re-adjust itself in terms of changed non functional expectations and most often these services are discarded and new services discovered to get inducted into the flow. In this paper, we extend this idea by focusing on run-time adaptation of non-functional features of a composite Web service by modifying the non-functional features of its component Web services. We use aspect-oriented programming (AOP) technology for specifying and relating non-functional properties of the Web services as aspects at both levels of component and composite. This is done via a specification language for representing non-functional properties, and a formally specifiable relation function between the aspects of the component Web services and those of the composite Web service. From the end users' viewpoint, such up-front aspect-oriented modeling of non-functional properties enables on-demand composite Web service adaptation with minimal disruption in quality of service. We demonstrate the applicability and merits of our approach via an implementation of a simple yet real-life example.

1 Introduction and Motivation

Web services have emerged as a major technology for deploying automated interactions between heterogeneous systems. They possess certain key properties [8, 12], viz., independent from specific platforms and computing paradigms, developed primarily for inter-organizational situations, and composable into composite Web services. Web service composition primarily concerns requests of users that cannot be satisfied by any atomically available Web service, but satisfied by a composite service obtained by combining a set of available Web services [13]. The dynamic nature of the business world highlights the continuous pressure to reduce expenses, to increase revenues, to generate profits, and to remain competitive. This requires Web services to be highly reactive and adaptive to business centric changes. In particular, composite Web services should be equipped with mechanisms to ensure that their constituent component Web services are able to adapt to meet changing requirements.

In this paper[1], we consider the important research issue of engineering adaptations on component Web services based on changed non-functional requirements imposed on the composite Web service, such as improved security, better scalability, etc. In particular, we focus on how non-functional requirements changes in the composite Web service can be met via appropriate pre declared modifications to the component Web services code, without affecting their core functionality. Our approach uses distributed aspect-oriented programming (AOP) technology [1, 4, 5] to dictate these modifications in component Web services in a non-intrusive manner. In addition, from the viewpoint of the users of the composite Web service, such an approach enables on-demand adaptation with minimal disruption. To the best of our knowledge, this is the first non-intrusive distributed AOP mechanism, especially applied to Web services. Hence our main contributions are the following: a distributed system architecture for non-functional adaptation of Web services via AOP (implemented on top of PROSE [2,3], a well-known AOP implementation environment[2]), a specification language for specifying non-functional properties of Web services, a formally specifiable relation function between the non-functional properties of the component and composite Web services, and a non-intrusive concern extraction and manipulation implementation for component Web services based on the relation function.

Our paper is organized as follows. We review related work in section 2. Section 3 introduces our approach and conceptual architecture. We then describe our running example in Section 4, and then use it to explain our approach in detail. In Section 5, we describe our specification language for describing non-functional properties of component and composite Web services. In Section 6, we discuss how multiple aspects can be weaved together, via a discussion of their inter-relationships. The detailed implementation of our running example is presented in Section 7. Finally, Section 8 concludes the paper with suggestions for future work.

2 Related Work

Aspect-oriented programming (AOP) [1,4,5] is an extension of other software development paradigms; it allows capturing and modularizing concerns called aspects that crosscut a software system. AOP makes very powerful program transformations possible, through a composition process where aspect *advices* are *woven* into the core program at locations called *pointcuts*. Members and methods can also be inserted in classes through an aspect construct called introduction. Aspects have the ability to introduce functionality in a core program in a non-invasive way, making it possible to alter the behavior of a system a posteriori. This aspect weaving can be done at any time – compile time, load time or run time.

Regarding Web services, existing web service composition and adaptation mechanisms are limited only to the process of web service choreography in terms of web service selection/invocation vis-à-vis pre-specified (Service Level Agreement) SLA constraints. Such a technique has many deficiencies, such as inability to manage ad-

[1] This is an expanded version of a paper that will appear in WS-Testing Workshop (co-located with SCC 2007).

[2] We have used version 1.3.0 of PROSE.

aptation, code duplication, inability to invoke an alternate Web service in case of failure, etc. To that end, several researchers are investigating AOP for improving the manageability of Web service compositions. For example, Cibrán and Verheecke propose a method for modularizing Web services management with AOP [11].

Charfi et. al. have approached this problem from a different direction [10]. They have proposed an extension to the BPEL language, which they called aspect-oriented BPEL (AO4BPEL). Their language brings in modular and dynamic adaptability to BPEL, since mid-flight adaptations can be implemented via advices in AO4BPEL. Ortiz et al. develop an aspect-oriented solution for Web services composition (of type orchestration) and for interaction patterns [6]. Orchestration is, here, defined as the process by which the Web services interactions are monitored and managed. The authors' work is motivated by the lack of standards associated with composition. More particularly, Ortiz et al. raised multiple questions related to the possibility of reusing interaction patterns previously implemented, and the efforts to put in for modularizing these patterns rather than scattering the code.

One recent approach towards service adaptation via AOP methods is described in [15]. In that paper, however, the authors have primarily focused on a template-based approach that enables the selection of the appropriate advice to be weaved into the Web service code based on mismatches with other participating Web services in the composite Web service. The focus in our paper, on the other hand, is on joint modeling and sharing of non-functional properties expressed as cross-cutting concerns via aspects. Hence we view the ideas in [15] as being complementary to our work. Similar to [15], our earlier work [14] proposes a method for decoupling security concerns in Web services via aspects, by expressing these concerns as contextual information separate from the core Web services functionality. This too, is complementary to the work reported in this paper.

One of the most well-known AOP implementations available today, is PROSE [2,3]. PROSE works by implementing methods – known as "hooks" – that intercept method calls in the Java Virtual Machine (JVM) at the point where the aspects are to be executed. Hence PROSE uses a modified version of the Java just-in-time compiler to insert code that checks for the presence of aspect advice at every possible join point, so as to implement system behavior modification at runtime. However, PROSE is not a distributed implementation, and works only to alter the behavior of a single component. Our system, therefore, seeks to extend PROSE for the distributed environment of web service composition and execution. Our system is also different from other distributed AOP systems [9], since it does not directly manipulate the source code of the individual component Web services; instead, it works by specifying advices to the individual component Web services so that they can change their functionality themselves.

Our solution approach uses WS-Policy and WSLA for implementing the negotiation of service requirements and capabilities between the service provider and the consumer, rather than using an enterprise service bus (ESB)-based approach. An ESB solution by itself does not provide native support for implementing negotiation of service requirements, but instead works in conjunction with WS-Policy and WSLA to implement the negotiation between the participating web services.

3 Solution Architecture and Approach

The composite web service model is extended to contain the list of its cross-cutting concerns that have a bi-directional mapping to those of the participating individual component web services. Each of these concerns, in turn, will have a mapping to the SLA constraints representing the different non-functional requirements. Hence there are two mappings to be established and maintained (These mappings need to be established between the composite and component Web services by prior agreement during the build time phase of the composite web service):

- The mapping between the non-functional requirements and the different cross cutting aspects of the composite web service
- The mapping (also known as relation function) between each aspect of the composite web service and the individual aspects of the component web services

The overall solution architecture is depicted in Figure 1.

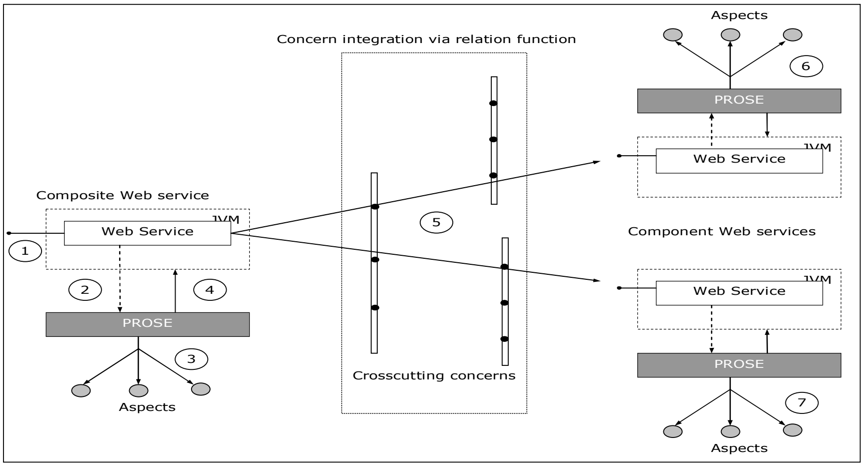


Fig. 1. Solution Architecture & Approach

Briefly, our system works as follows: at run time, in response to a change in a non-functional requirement imposed on the composite Web service by its user (1), the composite web service determines the appropriate aspect changes needed (2) to meet the change. The composite Web service makes this determination using the relation function (depicted via the "concern integration" rectangle in Figure 1) which it maintains. The composite Web service will therefore invoke the relation function (3, 4) to determine those aspects of the individual web services that need to be modified. It will then send messages (5) to the component web services, asking them to "re-weave" their functionalities in order to meet the changed requirements (6, 7). In case a component web service is not able to do so, it will send a reply to the message, upon which the composite web service will need to implement the appropriate exception handling mechanisms, for responding back to the initial user request.

We model the composite web service comprising aspects A_1 through A_n. Each component Web service W_i also possesses aspects a_{i1} through a_{in}. Each aspect A_i in the composite web service is related to the aspects a_{ij}, via the relation function:

$$A_i = f_i(a_{ij},\ 1 <= i <= m,\ 1 <= j <= n)$$

Of course, not all aspects a_{ij} will be affected by A_i, hence the relation function for each A_i would be different. Indeed, at its most elementary level, the relation function f_i is merely a mapping between A_i and the individual a_{ij} aspects, where each mapping could be suitably annotated with machine-readable information encoded in an XML formatted file.

It is to be stressed that our solution approach is *not* dependent on the choice of PROSE as an implementation mechanism. Different component Web services can have their own aspect-oriented implementation mechanisms (of which several exist in the literature[3]), as long as they can interoperate on sending/receiving advices for non-functional adaptation.

4 Illustrative Example

Figure 2 shows a simple example of a learning service. The application consists of a Learning Network Manager modeled as a composite web service, with the typical methods for clients to read/create the books that the Learning Network Manager offers from various publishers (each modeled as a component web service) offering books for various subjects according to the grade of the user. The Learning Network Manager allows the user to carry out various operations like reading books online, pick-and-choose topics from various publishers, maintain user history etc.

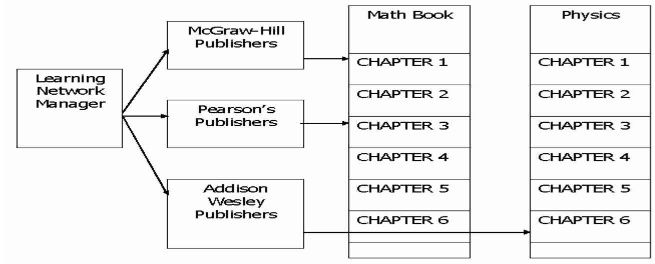


Fig. 2. Learning Network Manager and Associated Component Web Services

In this distributed environment, the Publishers and Learning Network Manager are required to customize their code to accommodate various kinds of customers with dynamic requirements. We have identified some of the important cross-cutting concerns. For example, the Publisher and Learning Network Manager need to accommodate the *timing* property in order to calculate the time taken by the customer to access the Web services. This function is important for tracking the performance of the system, especially under heavy load conditions. The second important concern is

[3] http://en.wikipedia.org/wiki/Aspect-oriented_programming#Implementations

scalability - in order for the Learning Network Manager to support a larger user base, it may expect the publishers also to support the same. Also, maintaining the *history* of each user, for future reference, is another important cross-cutting concern, which needs to be maintained by each publisher. Finally, *security* is needed for ensuring access to only authorized users.

The above cross-cutting concerns can also be used to identify their respective pointcuts, viz., Timing, System Tuning (for scalability), User History and Security.

5 Specification Language for Non-functional Properties

In order for a composite web service to locate a component web service at runtime, based on its '*capabilities*' to adapt to a changing non-functional requirement, the component web service must declare its capabilities in a program readable format, which could be interpreted by the composite web service to make a decision at run-time whether or not to invoke a particular component web service. Hence in this section, we discuss a specification language for representing non-functional properties that is built on existing standards such as WS-Policy[4] and WSLA[5]. While we use the WS-Policy framework to represent the capabilities and requirements of a Web service, WSLA is used to publish the QoS and related parameters that a web service can offer to its clients.

5.1 WS-Policy Based Specification of Non-functional Properties

WS-Policy provides a general purpose XML model to define capabilities and requirements of a Web Service. A policy is a collection of policy alternatives, which are in turn a collection of policy assertions. A policy assertion represents an individual requirement, capability or other property of behaviour. In our model, we extend the WS-Policy grammar by adding the `policy-assertion` and `NonFunctional-Property` tag elements as illustrated below.

```
<wsp:Policy
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:nfp="http://www.ibm.com/schemas/nfp">
 <wsp:All>  <!-- policy alternative -->
  <nfp:policy-assertion name="…" type="…" category="…">
   <nfp:NonFunctionalProperty name="…">
    <nfp:Attribute name="…" value="…"/>
     <nfp:Parameter name="" value="" mandatory="(true | false)"/>
     <nfp:custom-tags/>
   </nfp:NonFunctionalProperty>
  </nfp:policy-assertion>
 </wsp:All>
</ws:Policy>
```

Each `policy-assertion` has the following attributes:

[4] http://www.w3.org/Submission/WS-Policy/
[5] http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf/

- `name:` A qualified name for the policy assertion that could be referred to by another element in the policy document.
- `type:` Type specifies whether this policy assertion represents the ‘requirement’ or the ‘capability’ of the web services defining the policy.
- `category:` Domain specific category name of the assertion. Example: *Security*, *Performance* etc.

While the `<Attribute>` tag would be used to provide more description about the `<NonFunctionalProperty>`, the `<Parameter>` tag accepts input parameters to be passed to the target application (component service or an aspect implementation). For example, the non-functional property `‘ResponseTime’` may have an *attribute* called `‘units’` that may specify the unit of time that would be used to track the response time (like seconds, milliseconds etc), while at the same time, may accept a *parameter* called `‘round-off-digits=nnn’` with which a composite service may inform to the component service as to how many digits should the Response time output be rounded off to. The `<custom-tags/>` in the policy specification above provides flexibility for the participating web service to specify any domain specific custom tags that would be required to define the non-functional properties in a more detailed manner. However, please note that the XML schema definition and the interpretation of the `<custom-tags/>` is to be exchanged between the participating web services a priori.

Given below is an example of how the above mentioned model would be instantiated. The example below depicts a component web service that declares its *capability* of supporting ‘Authentication’ as a non-functional property. Please note that the `policy-assertion` type is ‘capability’ since the component web service exposes its ability to support ‘authentication’ as one of its non-functional properties.

```
<wsp:Policy
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:nfp="http://www.ibm.com/schemas/nfp">
 <wsp:All>  <!-- policy alternative -->
  <nfp:policy-assertion name="SecurityAuthSpec"
                 type="capability" category="Security">
    <nfp:NonFunctionalProperty name="authentication">
       <nfp:Parameter name="username" mandatory="true"/>
       <nfp:Parameter name="password" mandatory="true"/>
    </nfp:NonFunctionalProperty>
  </nfp:policy-assertion>
 </wsp:All>
</ws:Policy>
```

Please note that our specification language differs from WS-CoL [18] in that WS-CoL extends WS-Policy to define 'constraints' to be imposed during the execution of web services as well as to retrieve external data required to evaluate a constraint expression whereas our extension of WS-Policy provides a facility to specify the 'capabilities' and 'requirements' of a service. Our specification is domain independent due to the support of a generic `<NonFunctionalProperty>` tag as well as the `<custom-tags/>` place holder to support domain specific extensions and representations of non-functional properties.

5.2 Service Level Agreement

A service level is used to define the expected performance behavior of a deployed Web service, where the performance metrics are, for example, average response time, supported throughput, service availability, etc. During deployment of a Web service, the resources of an underlying Web service container can be reconfigured to provide a certain service level. Even the same Web service can be offered at different service levels to different clients by dynamically allocating resources for execution of individual Web service requests. Hence, to receive assurances on the service level, a client creates a priori a service level agreement (SLA) associated with this Web service with the service provider.

In our running example, a Publisher's web service would have an SLA defined for 'ResponseTime' using WSLA, representing the Timing pointcut introduced in Section 4. Given below is a sample SLA document defining an SLAParameter called 'ResponseTime' and the metric used to measure it.

```
<OperationGroup name="ReadOperations">
   <Operation name="WSDLSOAPGetChapter">
      <SLAParameter name="ResponseTime" type="float"
                    unit="seconds">
          <Metric>AverageResponseTime</Metric>
      </SLAParameter>
   </Operation>
</OperationGroup>
```

An SLAObligation for the above mentioned example may be defined as below:

```
  <ServiceLevelObjective name="SLO_for_ResponseTime">
    <Obliged>McGrawHillPublisher</Obliged>
    <Expression>
      <Predicate xsi:type="Less">
         <SLAParameter>ResponseTime</SLAParameter>
         <Value>2</Value> <!-- 2 seconds -->
      </Predicate>
    </Expression>
  </ServiceLevelObjective>
```

6 Aspects and Their Relationships

We relate each of the above identified point-cuts to an aspect. An aspect of a component web service is affected by zero or more aspects of the composite web service. Aspect interactions can be complex, subtle and very difficult to identify. Please note that finding such interactions is outside the scope of our research. In our work we assume a fixed ontology of aspects, with all interactions explicitly identified ahead of time. We provide an XML file representation for specifying the aspect interaction and conflicts. Our model is extensible and hence we can contain any level of complex relationships here. Our model of aspect interactions is leveraged from [7]), and features the following: *Orthogonal* – if the combined contribution of both aspects is equal to the sum of their individual contributions (e.g., user-history aspect of component & composite Web services); *Complementary* – if their combined contribution is greater than the sum of their individual contributions (e.g., authentication and timing aspect), *Depends* – if they can only be deployed along with each other (e.g., timing aspect of composite Web service and timing aspects of component Web service); *Conflict* – if their combination has a negative effect on the behavior of the composite Web service (e.g., timing and user-history aspect may conflict, especially if the policy of charging the customer is based on the content accessed); *Prevents* – if the application of one aspect prevents the application of the other (e.g., if one aspect measures the response time with respect to a threshold, which would deactivate other aspects such as caching, security and logging); *Equivalent* – if their individual effects are the same (e.g., different logging types such as CBELogging, JTraceLogging, etc.)

7 Implementation Details

Our learning network manager is modeled as a composite web service, with operations such as authenticateUser, showBooks, showSubjects, showTopics, showContents and createBook exposed for clients to read/create the books that the Learning Network Manager offers from various publishers (see Figure 3). Each publisher is modeled as a component web service offering books for various subjects through their exposed operations. (We have not displayed the details of the WSDL-based interfaces of the Web services in this Section, since we have chosen to focus on the non-functional property modeling aspects.)

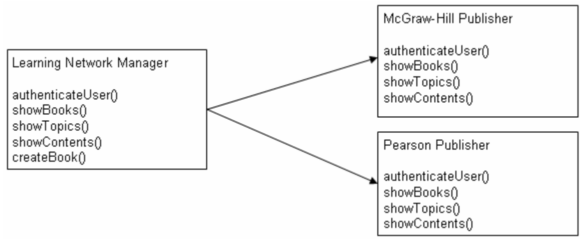


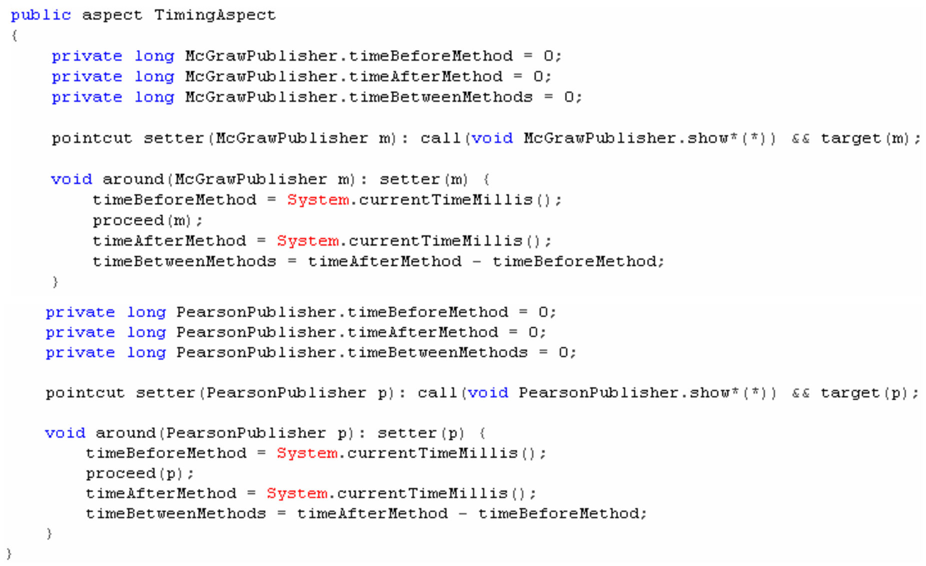
Fig. 3. Service Operations

The timing aspect relationship between the Learning Network Manager and any of the publishers is depicted in Figure 4.

```
<ar:aspect-relation xmlns:ar="http://www.ibm.com/aspectrelationship">
    <ar:source-aspect id="a1" component="ws1">
        <ar:aspect class="class1">
            <pointcut name="methodAround" expression="execution(* class1.methodAround(..))"/>
            <advice name="advice1" type="around" bind-to="methodAround"/>
            <parameters>
                <parameter name="p1" value="v1"/>
                <parameter name="p2" value="v2"/>
            </parameters>
        </ar:aspect>

        <ar:relation type="depends">
            <ar:target-aspect id="a2" component="ws2"/>
                <ar:aspect class="class2">
                <pointcut name="methodAround" expression="execution(* class2.methodAround(..))"/>
                <advice name="advice1" type="around" bind-to="methodAround"/>
                <parameters>
                    <parameter name="p1" value="v1"/>
                    <parameter name="p2" value="v2"/>
                </parameters>
                </ar:aspect>
            </ar:target-aspect>
        </ar:relation>
    </ar:source-aspect>
</ar:aspect-relation>
```

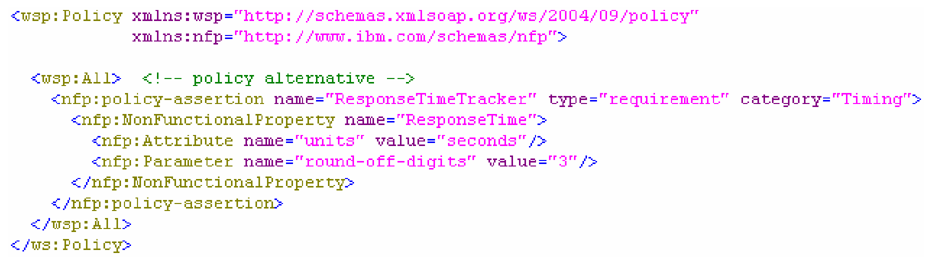
Fig. 4. Timing aspect relationship between component and composite services

Given below is the Timing Aspect implementation for McGrawPublisher service in PROSE.



```
public aspect TimingAspect
{
    private long McGrawPublisher.timeBeforeMethod = 0;
    private long McGrawPublisher.timeAfterMethod = 0;
    private long McGrawPublisher.timeBetweenMethods = 0;

    pointcut setter(McGrawPublisher m): call(void McGrawPublisher.show*(*)) && target(m);

    void around(McGrawPublisher m): setter(m) {
        timeBeforeMethod = System.currentTimeMillis();
        proceed(m);
        timeAfterMethod = System.currentTimeMillis();
        timeBetweenMethods = timeAfterMethod - timeBeforeMethod;
    }

   private long PearsonPublisher.timeBeforeMethod = 0;
   private long PearsonPublisher.timeAfterMethod = 0;
   private long PearsonPublisher.timeBetweenMethods = 0;

   pointcut setter(PearsonPublisher p): call(void PearsonPublisher.show*(*)) && target(p);

   void around(PearsonPublisher p): setter(p) {
       timeBeforeMethod = System.currentTimeMillis();
       proceed(p);
       timeAfterMethod = System.currentTimeMillis();
       timeBetweenMethods = timeAfterMethod - timeBeforeMethod;
   }
}
```

Figure 5 depicts a snapshot of WS-Policy declared by `LearningNetworkManager` Web Service stating that it *'expects'* the component services (Mc-GrawHill and Pearson publishers) to support the tracking of *'Response Time'* of `ShowBooks()` service. The component service would also weave an aspect code dynamically into its AOP runtime system to handle the change in non-functional requirement.

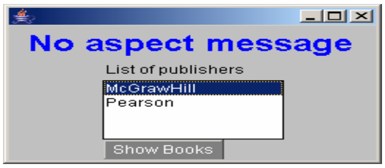


```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
            xmlns:nfp="http://www.ibm.com/schemas/nfp">

  <wsp:All>  <!-- policy alternative -->
    <nfp:policy-assertion name="ResponseTimeTracker" type="requirement" category="Timing">
      <nfp:NonFunctionalProperty name="ResponseTime">
        <nfp:Attribute name="units" value="seconds"/>
        <nfp:Parameter name="round-off-digits" value="3"/>
      </nfp:NonFunctionalProperty>
    </nfp:policy-assertion>
  </wsp:All>
</ws:Policy>
```

Fig. 5. WS-Policy defining composite service's Non Functional Property requirement

In response to the above mentioned requirement, the component service (eg., Mc-GrawHill publisher), would generate an SLA and publish it to the requesting service. Section 5.2 above shows a snapshot of the SLA published by the component service.

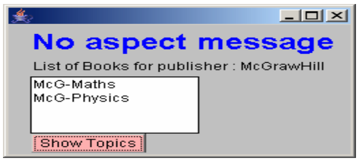
Given in Figure 6 below is a self-explanatory sequence of screen shots that explain the flow of the implementation. The Figure shows the ShowBooks() service of the McGrawHill and Pearson publisher services, *before* and *after* the invocation of the Timing aspect.


No aspect message

List of publishers

McGrawHill
Pearson

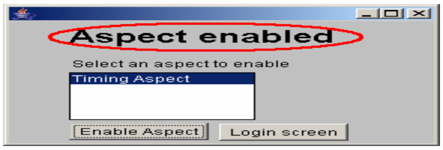
(a) Invocation of ShowBooks() operation


No aspect message

List of Books for publisher : McGrawHill

McG-Maths
McG-Physics

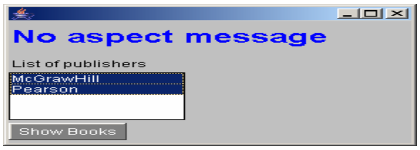
(b) ShowBooks() returns no aspect message


Aspect enabled

Select an aspect to enable

Timing Aspect

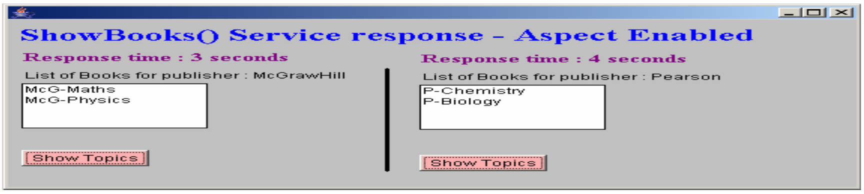
(c) Enabling of Timing Aspect


No aspect message

List of publishers

McGrawHill
Pearson

(d) Invoking ShowBooks() after enabling aspect


ShowBooks() Service response – Aspect Enabled

Response time : 3 seconds

List of Books for publisher : McGrawHill

McG-Maths
McG-Physics

Response time : 4 seconds

List of Books for publisher : Pearson

P-Chemistry
P-Biology

(e) Due to dynamic weaving of aspects, component service responds back to composite service with Response time details

Fig. 6. Screenshots of Implementation

8 Future Work

Future work will involve integrating our work with self-healing Web services environments as modeled in [16]. Additionally, we will also evaluate the recent work of a joinpoint inference technique based on behavioral specifications of state machine specifications [17], and investigate how it can be used to provide a formal specification of aspect interactions between the composite and component Web services.

Acknowledgement. The authors wish to thank Sumanth Vepa, Zakaria Maamar, Dipayan Gangopadhyay and the anonymous ICSOC reviewers for their feedback.

References

[1] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J.: Aspect Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)

[2] Popovici, A., Gross, T., Alonso, G.: Dynamic Weaving for Aspect Oriented Programming. In: Proceedings of 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands (2002)

[3] Nicoara, A., Alonso, G.: Dynamic AOP with PROSE. Department of Computer Science, Swiss Federal Institute of Technology Zurich (ETH Zurich), CH-8092 Zurich, Switzerland, accessible from http://prose.ethz.ch/

[4] Popovici, A., Alonso, G., Gross, T.: Just in Time Aspects: Efficient Dynamic Weaving for Java. In: Proceedings of 2nd International Conference on Aspect-Oriented Software Development, Boston, USA (2003)

[5] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An Overview of AspectJ. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, pp. 18–22. Springer, Heidelberg (2001)

[6] Ortiz, G., Hernandez, J., Clemente, P.J.: Decoupling Non-Functional Properties in Web-Services: As Aspect-Oriented Approach. In: ICSOC'2004. Proceedings of The 2nd International Conference on Service Oriented Computing, New-York, USA (2004)

[7] Wohladter, E., Tai, S., Thomas, A., Rouvellou, I., Devanbu, P.: GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions. In: ICSE. Proceedings of International Conference on Software Engineering, Edinburgh, UK (2004)

[8] Ma, K.J.: Web Services: What's Real and What's Not. IEEE IT Professional 7(2) (2005)

[9] Nishizawa, M., Chiba, S., Tatsubori, M.: Remote Pointcut – A Language Construct for Distributed AOP. In: AOSD'04. Proceedings of International Conference on Aspect-Oriented Software Development, Lancaster, UK, March 22-26, pp. 7–15. ACM Press, New York (2004)

[10] Charfi, A., Schmeling, B., Heizenreder, A., Mezini, M.: Reliable, Secure and Transacted Web Service Compositions with AO4BPEL. In: ICSOC'2004. Proceedings of The 2nd International Conference on Service Oriented Computing, New-York, USA (2004)

[11] Cibrán, M.A., Verheecke, B.: Modularizing Web Services Management with AOP. In: Cardelli, L. (ed.) ECOOP 2003. LNCS, vol. 2743, Springer, Heidelberg (2003)

[12] Benatallah, B., Sheng, Q.Z., Ngu, A.H.H., Dumas, M.: Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In: ICDE. Proceedings of International Conference on Data Engineering (2002), also available from http://csdl.computer.org/comp/proceedings/icde/2002/1531/00/15310297abs.htm

[13] Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: A Foundational Vision for E-Services. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, Springer, Heidelberg (2003)

[14] Kouadri Mostefaoui, G., Maamar, Z., Narendra, N.C., Sattanathan, S.: Decoupliing Security Concerns in Web Services Using Aspects. In: ITNG 2006. Proceedings of Information Technology – New Generations, IEEE Computer Society Press, Los Alamitos (2006)

[15] Kongdenfha, W., Saint-Paul, R., Benatallah, B., Casati, F.: An Aspect-Oriented Framework for Service Adaptation. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, Springer, Heidelberg (2006)

[16] Kouadri Mostefaoui, G., Maamar, Z., Narendra, N.C., Thiran, Ph.: On Modeling and Developing Self-Healing Web Services Using Aspects. In: COMSWARE 2007. Proceedings of 2nd International Conference on Communication Software and Middleware, IEEE Communications Society, Los Alamitos (2007)

[17] Cottenier, T., van den Berg, A., Elrad, T.: Joinpoint Inference from Behavioral Specification to Implementation. In: ECOOP. Proceedings of European Conference on Object-Oriented Programming (2007) (to appear)

[18] Baresi, L., Guinea, S., Plebani, P.: WS-Policy for Service Monitoring. In: Proceedings of TES 2005 (September 2005)