Web Services Adaptation at Policy Layer

Margarita Velasco-Olvera, David While, Pathmeswaran Raju
Enterprise Systems Lab, Knowledge Based Engineering Lab, Birmingham City University, Birmingham, United Kingdom

Abstract

Web services technology is evolving as a base for implementing service-oriented architectures (SOA), which has become an important architectural style for Web applications development. As Web Services development has grown considerably, the need for Quality of Service (QoS) has become a substantial aspect for developing reliable service applications. In diverse business domains, Web services need to exhibit quality characteristics such as response time, availability, and reliability. However, a problem arises when a Web service is modified, updated or replaced by a new one. The new Web service may not fully match the QoS features provided from the previous one, this unavoidable problem is recognized as a mismatch. Mismatches can be found in different layers of the Web service interoperability stack and corresponding adapters are needed to deal with this issue. This paper proposes a study to investigate the feasibility of achieving interoperability in Web services by evaluating the functionality of adapters when mismatches appear at the policy layer, which includes non-functional properties like QoS.

1. Introduction

Web services are an essential part of the Internet because they enable software programs to interoperate seamlessly and efficiently, delivering platform independent systems with high flexibility [1]. They are evolving as a base for implementing service-oriented architectures (SOA), which has become an important architectural style for Web applications development. In essence, SOA is a group of services which interact with each other through their interfaces. The interaction involves two or more services by coordinating some activities. However, because of the rapid growing of Web service technology, the coordination of activities among them is getting more complex [2]. Current examples of Web services are included in applications for weather forecasting, credit checking, online tickets reservation, maps, banking and travel services. They are applications that use Extensible Markup Language (XML) based technology as a common language for representation and communication for exchanging data across the Internet. Moreover, Web services provide a means for wrapping software applications allowing developers to access them through standard languages (e.g., Java and C#) and protocols (e.g., SOAP and REST). A Web Service Description Language (WSDL) is used to describe the interface of a web service. This standard describes the functional properties of a service, which generally do not vary over time.

When users cannot be satisfied by any atomically available Web service, a composite service is needed by combining the functionality of heterogeneous Web services. Web service composition has to deal with services from diverse providers and sometimes they are not consistent. The compatibility of service interaction is an essential measure determined by the successful collaboration among a set of services. However, when a service is modified, mismatches can appear when the offers and demands do not match in the service interaction. To solve these incompatibilities, adapters can be used. Web services were born as a solution to the integration problem, bringing with them standardization as a main advantage [3]. Nevertheless, adapters are needed to solve some incompatibilities.

Web services have non-functional features which are often dynamic, such as quality of service (QoS) including response time of a service, which may vary over time depending on the current workload. Thus, the environment of a Web service composition is very dynamic because the integration of new services, others becoming updated or non-functional properties like QoS (e.g., response time and availability) requirements may change frequently. This may affect the performance of composite services and end in a downtime. To specify Web service non-functional information, policies may be created. Thus, QoS properties as non-functional information are candidates for policy specification using the Web Services Policy framework (WS-Policy) standard.

This work aims to investigate Web service interoperability by considering non-functional features like QoS described in a policy. The use of adapters would help mediate the information exchanged by the involved parties to solve mismatches at the policy layer among participant services.

This paper is structured as follows. Related work is discussed in section 2. Section 3 describes Interoperability in Web services, Policies in Web services and the basic characteristics of WS-Policy standard. QoS in Web services and mismatches in Web services are explained in Section 4. The following section highlights the importance of adapters. In section 6, we propose a study to evaluate the feasibility of achieving interoperability in Web services by evaluating the functionality of adapters when mismatches appear at the policy layer. Finally, section 7 presents conclusions and further work.

2. Related Work

In the development of Web services, policies provide the specification of who can use a Web service under certain terms and conditions, how information should be provided to a Web service and how that information will be used later. There are approaches based on policies to monitor Web services considering their functional and non-functional features identifying different types of policies such as service, server and supported and requested policies. Maamar *et al.* [4] focus on the constraints that Web services have and how they are related to non-functional details like response time, execution time and reliability. Another work related to using policies to manage Web services is reported in [5]. They present an approach to make easier the selection of a web service based on non-functional properties, especially QoS characteristics. Their approach considers WS-Policy and Ontology Web Language (OWL) to enable a semantics-enriched description of QoS properties.

In the field of adaptation, Kongdenfha *et al.* [6] identifies and classifies different scenarios in Web services and presents mismatch patterns. They use an aspect-oriented approach for service adaptation due to interface and protocol mismatches by developing stand-alone adapters using pattern methods. However, the approach requires developers to manually define the mismatch before performing the adaptation. There are some works that propose adaptation methods for dynamically identifying and composing services by using process schema, which is a process template applied to adjust services found on user's environment and exceptions. Others consider adaptation methods for potential service faults and environment changes in dynamic service composition. Finally, dynamic adapters have been implemented to solve QoS inconsistencies, but in most cases they are considered as faults.

It seems that research on dynamic service adaptation has addressed only a little of the fundamental issues in dynamic adaptation, which includes QoS policies in a typical process for invoking services. Many of the works are limited to treating either functionality-related faults or QoS-related faults, and they mainly focus on adapting service interfaces and business processes. Strategies for designing and developing dynamic adapters with service-oriented architecture (SOA) standards are missing. For example, Leite *et al.* [7] shows that adaptation to functional characteristics changes is having more attention with 92% of the reviewed studies, just 8% deal with non-functional requirements and 46% deal with both. The key difference between our work and previous contributions relies on the fact that this work focuses at the policy layer using WS-Policy standard for adaptation of non-functional mismatches in Web service composition.

In summary, adaptation techniques for identification of mismatches at the policy and non-functional layers are limited. Therefore, we aim to propose an automatic approach that characterizes mismatch patterns at that layer, to propose a solution to the adaptation problem, identifying and capturing possible differences that are not detected by manual and semi-automatic approaches.

3. Interoperability in Web Services

Interoperability is defined as the facility among two or more systems, networks, devices, components or applications to exchange and use information between themselves. Increased interoperability is the most important advantage of SOA, especially when Web services technology is considered.

A Web service can be defined generally as a service offered via the Internet that can be accessed by a Uniform Resource Locator (URL). The main purpose of a Web service is transporting unified and real communication and also is considered as an independent software component which provides a specific functionality to be discovered and invoked over the Internet. According to Nezhad *et al.* [8], is useful to study Web services interoperability issues in terms of layers to represent various parts of the problem at different levels of abstractions and specifications of services integration based on layer concepts for better investigation of the relevant problems in different layers.

The policies layer provides a grammar for representing the non-functional attributes of entities in Web services based XML environment [8]. Web services include policies (e.g., privacy, trust, authentications and authorization policies) and other non-functional aspects such as Quality of Service (QoS) descriptions including response time, availability, performance or reliability that are valuable for 'service clients' to decide if they will be able to interact with the service [8].

Most of the current work in web services interoperability has been dealing with finding mismatches in the interfaces and protocols layer. Interoperability in lower layers need to define properties in almost all applications, contrary with higher layers, which specification does not require to be defined.

However, this study is focused in the policy layer including policies and non-functional properties. This would have great importance because a service description can include policies like privacy policies and other non-functional properties like QoS requirements that clients interacting with a service should have clear understanding [8].

3.1 Policies in Web services

Web services need to explain their requirements and capabilities to their environment and users. Policies can be considered as information that developers can use to modify a system's behavior. A policy is a collection of capabilities and requirements. Policies might be dynamically, external, adaptable rules and parameters that are input into a system [9]. If a Web service offers policies, service users have to meet the terms with the declarations found in the policy document. Furthermore, policies generally define security considerations and QoS characteristics which are necessary for service communication. Therefore, policies are helpful in following actions that origin unanticipated exceptions and in identifying how these exceptions are to be repaired without the need to alter the composition specification [9]. There are different ways to associate policies with Web services. They can be functional at different levels, including operation, service and process levels. Maamar, Benslimane & Anderson [9] propose six possible scenarios of Web services policies that might be used by services and administrators to classify and detect mismatches, namely, announcement, selection, compatibility, agreement, verification and composition. During runtime, consumers can generate policies connected with partner services. These policies specify consumer capabilities and properties that should be offered by required Web services. From the Web services perspective, a policy can be defined statically or dynamically. Basically, the former is attached in the WSDL (service contract) and the latter is generated to service a consumer that interacts with a service.

In our research policies are used to represent QoS mismatches, considering measurable quality attributes such as response time, reliability, availability and accessibility as non-functional attributes. Our research aims is to develop methods for automatic generation of adapters in Web services to resolve mismatches at the policy layer considering QoS attributes in a policy by using WS-Policy standard.

3.2 WS-Policy standard

The WS-Policy standard comprises a model to denote Web service policies, presenting their capabilities and general characteristics. Policies represent and match the functional and non-functional attributes of entities in a Web service based on XML technology [10]. According to the WS-Policy model, a policy is a set of alternatives, which are a collection of policy assertions. A policy assertion is a condition or rule which describes Web Services behavior. Moreover, a WS-Policy can be represented in XML using the following tags:

- The "Policy" tag is used to start and end a policy. All the child assertions must be satisfied.
- The "ExactlyOne" tag is used to contain a set of alternatives. Exactly one of the child assertions must be satisfied.
- The "All" tag contains all the assertions in an alternative. All the child assertions must be satisfied

As WS-Policy offers a means to specify QoS attributes, providers may specify their services using WSDL and WS-Policy. Policies can be embedded in the WSDL or can be attached as an external document as depicted in Figure 1. Service policies should be specified at different levels in the WSDL service definitions to detect mismatches more efficiently. Although WS-Policy is sometimes considered a general framework and it does not make clear how QoS policies, prices or other information will be managed and developed, it has very positive expectations in applying non-functional attributes to Web services due to its flexibility and robustness [10].

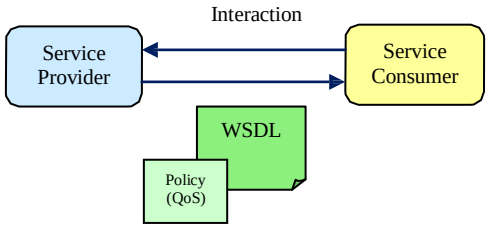

Figure 1. Web service policy interaction

4. QoS in Web Services

QoS is an arrangement of several non-functional characteristics such as the response time of a request. The introduction of QoS into the Web service architecture offers diverse benefits. QoS publication helps selecting among services with the same functionality, service composition based on QoS and estimation of alternative execution paths for process adaptation. QoS characteristic vary in diverse applications and they can consist of some non-functional properties that can be present in a WS-policy framework standard [11]. Table 2 shows quality of Service attributes found in the literature. For instance, response time, availability, reliability, accuracy and scalability. These QoS attributes can cause mismatches. For example, when an invoked service does not send a response to a consumer service within a certain amount of time.

Table 1. QoS characteristics

QoS characteristic	Description
Response time	It is the time elapsed from the submission of a request to the time the response is received.
Availability	Represents the percentage of time that a service is operating.
Reliability	It is the probability of successfully executing a Web service.
Accessibility	Represents the capability of serving a web service request.
Successability	Is the number of responded request messages

4.1 Mismatches in Web services

Mismatch is an interoperability problem that occurs when a service consumer does not match the features offered by a service provider. In consequence, mismatches amongst web services can occur at different levels of the interoperability stack: behavioral, technical, signature, quality of service (non-functional properties), and semantic level [12]. For instance, mismatches on functionality, interface or non-functional properties can be recognized by comparing the features of the service provider with the consumer.

Non-functional aspects include QoS properties which consist of a single or more quality properties and their estimated values at runtime, which is normally specified in the Service Level Agreement. Unacceptable values of QoS properties will end in a mismatch. Considering mismatches in the policy layer, the compatibility of two service policies must be matched with requirement attributes of another service policy contained in a set of services.

Our work will represent mismatches of non-functional properties using WS-Policy standard. For example, a QoS mismatch may be found in reliability, a common quality attribute. In this case adapters would be needed to solve this incompatibility.

5. Adapters

An adapter is a fragment of code that sits between two components and balances the differences between their interfaces. Some authors consider that when one component provides a service that another component requires, it is regularly impossible to connect two components if they were programmed with different collaboration specifications. Moreover, Mateescu [13] affirms that an adapter is a process that helps to avoid deadlock situations when a collection of services interact together. Adaptation can be static or dynamic. It is static when a composition instance is changed before it is started. It is dynamic when a running composition instance is restarted from the beginning and is changed without being stopped. Since static adaptation occurs at design time, static adapters are relatively straightforward to design and implement, compared to dynamic adapters.

The need for adapters in Web services essentially appears from heterogeneity at the higher levels of the interoperability stack or because a high number of clients, every of which can support diverse interfaces and protocols, thus generating the need for providing multiple interfaces to the same service [3]. Mateescu, *et al.* [13] explains how to build a service adapter, a non-trivial task. First, is necessary to design a model by considering diverse constrains. Second, the service adaptor will contain service specifications. The adapter should be able to send and receive messages to the service. Finally, the adapter has to consider all constrains in the adaptation contract.

Taher *et al.* [14] proposed an adaptation approach to adapt automatically service interface and business protocol incompatibilities. Thus, they develop operators for each transformation pattern associated to mismatches such as split, merge and aggregate. They use automata representations to model these operators where each operator automata should be converted to a continuous query to generate a Complex Event Processing (CEP) adapter. Benatallah, Casati, and Grigori [3] introduced the concept of mismatch patterns. Mismatch patterns are design patterns that can be used to detect the possible mismatches among services to make the development of adapters easier. Their work develops adapters to resolve differences at the interface and business protocol level. Kongdenfha *et al.* [6] identified mismatches among services. They proposed a taxonomy of common mismatches at service interfaces and business protocols layer and developed methods for patterns considering stand-alone adapters. We aim to extend the work of Kongdenfha *et al*. [6] in order to develop adapters from mismatches found at the policy layer including policies and non-functional aspects such as QoS. Our work will be developed using the standard WS-Policy by representing QoS attributes in a policy. The use of policies may resolve mismatches more efficiently and could contribute to fulfill the missing piece of the adapter's life cycle, developing common governance layer for policies based on abstraction, using on-demand ruled based systems to define the relationships and patterns for policies based on standards.

5.1 Adaptation Methods in Web services

Service adaptation refers to the creation of new services as adapters to bridge interactions in a service composition. Web service composition has the advantage to save time and costs, but when adaptation occurs frequently in runtime, the service composition performance will be affected.

Kongdenfha *et al.* [6] consider two adaptation approaches, the first one is called stand-alone adapter that consist basically in develop a new service to make possible the interaction between two or more services. The other approach consists of modifying a service that will be, later, compatible with the other services. Essentially, adaptation approaches suggest using adapters by common logic involving diverse activities like: intercept, store, transform, create and send a new message when necessary. Moreover, adaptation can be anticipated in two approaches, proactive and reactive. On the one hand, proactive adaptation happens before a specific event, for instance, when the system perceives that there is an unavailable service. On the other hand, proactive adaptation occurs after a specific event. For example, when there is an unsuccessful call.

Table 2 shows different reviewed methods and techniques to develop adapters in Web service composition.

Table 2. Adaptation Methods in Web services

Author	Adaptation method
Hung *et al.*[15]	An Interface Pushdown System (IPS)
Seguell *et al.* [16]	Interaction Analysis Matrix (IAM)
Motahari Nezhad *et al*. [17]	Adapter simulation algorithm
Taher *et al*. [14]	Complex Event Processing (CEP)
Kongdenfha *et al*. [6]	Stand-alone adapter and modifying a service.
Mateescu *et al*. [13]	Process algebra and on-the-fly-technique
Wang *et al.* [18]	Service Adaptation Machine
Shan *et al.* [19]	Flexible, dynamic and on-the-fly adapter generation

There have been different adaptation approaches and techniques. Hung *et al.* [15] propose an adaptation approach for synchronous service interaction. The adapter model of this approach is characterized by an Interface Pushdown System (IPS). They use model-checking technique for verify the adapter during its generation. Seguell *et al.* [16] illustrate an automated approach of adapter to resolve behavioral mismatches that needs the adaptation only for a minimal set of messages exchanged. They built an Interaction Analysis Matrix (IAM) over the pairs of interaction for comparing and assessing the behavioral relation of their nodes. Motahari Nezhad *et al.* [17] describe an approach to solve interface mismatches in case of signature, split, and extra or missing mismatch type. They propose an adapter simulation algorithm that explores possible message exchanges between two service interfaces to evaluate related protocol mismatches. Taher *et al.* [14] present their approach based on the Complex Event Processing (CEP) to adapt both service interface and business protocol mismatches automatically. Kongdenfha *et al.* [6] propose an approach to modify the services via an aspect-oriented paradigm. Moreover, they develop both standalone and service modification adapters. Mateescu *et al.* [13] establish an adaptation technique that reduces the computational complexity of adapter generation by using on-the-fly exploration and reduction technique. Wang *et al.* [18] describe a runtime adaptation approach for service behavioral interface incompatibilities. They introduce service adaptation machine. Shan *et al.* [19] present a technique for a flexible, dynamic and on-the-fly adapter generation based on message and control flow adaptation. This approach can also combine control flow constraints into message adaptation.

6. Methodology

This paper proposes a study to investigate the feasibility of achieving interoperability in Web services by evaluating the functionality of adapters when mismatches appear at the policy layer. This will enable the production of adapters using an automated generation of code approach. Our study will include strategies of investigation such as experiments involving different scenarios [9] to evaluate Web services interoperability.

There are in the literature different non-functional properties [20], based on that we will be able to make a classification of mismatches based on non-functional properties considering the policy layer and common QoS. Table 3 shows the most common non-functional characteristics which are essential to detect mismatches at the policy layer. This research will focus in QoS properties because they represent a very significant aspect of non-functional characteristics for a Web service. These properties include response time, availability, reliability, accessibility and successability and others. Other important properties but peripheral to this work include security, business and environmental properties such as authentication, access control and encryption.

Table 3. Classification of Non-Functional Properties[20]

Non-functional properties			
QoS properties			ties
Execution	Security	Business	Environment
Response Time	Encryption	Cost	Temporal
Availability	Authentication	Organization arrangement	Location
Reliability	Access control	Monitoring	
Accessibility		Payment Method	
Successability		Reputation	

Web Service policy will be used to represent Web services non-functional attributes like QoS. Non-functional capabilities and requirements of a Web service have to be matched with those of a request to determine if a Web service is suitable for a particular use. Specifically, in matching the non-functional properties, the policy requirements of the entity invoking the Web Service must be compatible with the policies of the entity providing the Web service. Figure 2 shows an example where response time will be a decisive policy factor in order to choose the right supplier when multiple web services interact with each other. For example, two suppliers that compete for orders in a supply chain from a manufacturer.

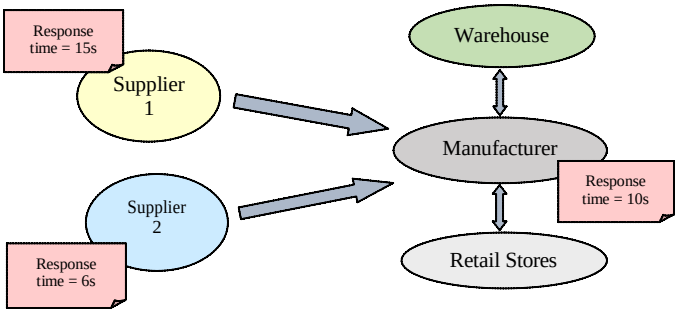

Figure 2. Web service policy interaction

Web service policy interaction considers a problem that arises when a Web service is modified, updated or replaced by a new one. The new Web service can be different from the previous one. Therefore, the QoS properties can be different. These differences have to be solved by the adapter service automatically at run-time. Thus, it is essential to determine if non-functional capabilities and requirements like QoS of a Web Service are the same as those of requested. The Web service will be utilized, for instance, when a specified response time is less or equal to the value specified by the requesting Web service, otherwise the process would stop.

A monitoring phase is essential to verify the functional properties like response time and availability of the invoked services in a business process. Figure 3 shows a screenshot of the JDeveloper composite editor that will be used to provide an integrated development environment of Web services.

In order to monitor Web services activity we used Oracle BPEL Process Manager's sensor framework, where business processes are collected by the Process Manager as a result of the service invocations. Business process sensors are used to monitor QoS attributes and to detect possible mismatches. For instance, considering measurable quality aspects for policy matching such as response time, availability, reliability, accuracy, capacity and scalability where mismatches may be identified.

Figure 3 also illustrates that using BPEL Process Manager l helps to detect the response time of each activity involved in a business process including partner links as Web services.

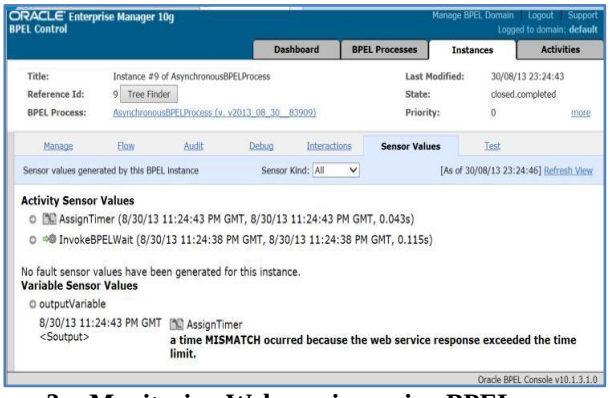

Figure 3. Monitoring Web services using BPEL manager

This also will facilitate the monitoring of different processes when many of them are running in parallel.

To characterize the problem of adaptation we propose to identify and classify different adaptation scenarios in Web services based on [17]. Adaptation will be required for one or more of the interoperability layers, since for two services to interact, compatibility must be achieved at all layers. Therefore, different examples of services will be used in this study to characterize possible mismatches at the level of service policies when at least two web services interact. We aim to use the mismatch structure suggested in [3] and presented in Table 4.

Table 4. The structure of a mismatch pattern [3]

Name	Pattern
Mismatch	Description of the captured incompatibility
Parameters	Information provided by the user to develop the adapter code.
Template	Description of the adapter's implementation to resolve the incompatibility captured.
Sample	Contains the evidence to help the developer generating the adapter.

In order to capture WS differences, mismatch patterns will be used as strategy to develop adapters and in a system to automatically produce the adapter code [5]. The system should be able to automatically produce the adaptation logic and deploy it to reconcile the differences.

The functionality of adapters will consist in mapping interaction with policy P1 into interactions with policy P2. This requires performing activities like receive messages, store messages, transform the message and invoke the message.

We will use the increasingly popular Web Services Business Process Execution Language (WS-BPEL) that describes Web service composition. However, the specification of the BPEL is static and it is not able to offer any mechanism to deal with non-functional properties like QoS including response time and availability or select services at runtime. Therefore, dynamic adaptation of composite services is needed when QoS requirements and runtime modifications need to be considered.

Currently, there are diverse languages to describe the behavior of Web services like Web Service Choreography Interface (WSCI), Web Services Choreography Description Language (WS-CDL) and Web Services Business Process Language (WS-BPEL), etc. They provide a definition model that is conceptually based on process models [2]. In this paper, we exemplify the use of Oracle BPEL execution engine as the specification of Web services. This engine will help for sending and receiving messages between composite service consumers and adapters. There are different BPEL engines for specification of composite services, such as Apache ODE (Orchestration Director Engine), which we have used in this research. Oracle-BPEL was preferred because it is free for

developers an open standard, it provides a native XML scripting environment that is perfectly appropriate to asynchronous document processing. Moreover, BPEL provides supplementary annotations to specify adaptation abstractions including model, details and expressions for this purpose. Furthermore, the activities depicted in Figure 3 can be represented by BPEL, which also can be used to define the adapter template to produce web services. For the development of Web services, we are using Eclipse as our interface development environment (IDE).

We will use an adaptation phase that will interact with a monitoring phase through changes in the service composition according to generated optimal policies. Thus, the adaptation methodology inputs two communicating Web services, S1 and S2, whose interaction may lock, and it builds (if possible) a Web service adapter D, which allows the two Web service to effectively interoperate. The adapter will be used to replace a Web service at runtime with another Web service which offers the same functionality. Service consumers will recognize mismatches by exploring service specifications; therefore, we will able to detect types of mismatches from the key elements of service specifications and identify mismatches in non-functional properties.

We propose to consider the case when the values of some quality aspects are less than the ones expected by the other Web service. For example, QoS (Service 1) provides additional quality aspects that are not required by QoS (Service 2) or QoS (Service 2) expects more qualities aspects than QoS (Service 1) is providing. QoS improving adapter is expected to remove mismatches between QoS assured by the service and one expected by the consumer.

The use of policies to represent mismatches in QoS attributes will be also taken into consideration. We will extend [17] work by adapting suggested scenarios in order to find different mismatches. For example, figure 4 shows the 'Verification' scenario, which represents the interaction of two web services, where the receiving WS2 verifies that the content of the message associated with this operation match with policies Pc and Pe.

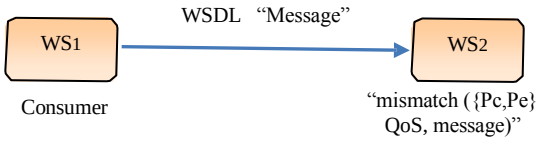

Figure 4. Verification Policy Scenario

In the "Composition" scenario, illustrated in figure 5, the interaction of three Web services that are not compatible, i.e., they do not meet the requirements of a consumer as represented.

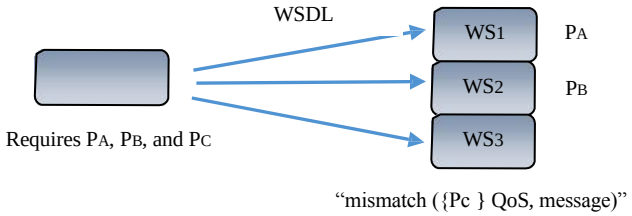

Figure 5. Composition Policy Scenario

Figure 5 illustrates that a QoS mismatch may appear if the attributes do not match the expectations of the consumer. From this scenario, the use of an adapter will be needed to make possible the interoperability among Web services WS1, WS2 and WS3, where the consumer is demanding for QoS attributes PA, PB, and PC. From this example, the QoS attributes provided (PA and PB) are less than the demanded.

Conversely, figure 6 represents the incompatibility scenario among services (verification policy), where the attributes provided are more than the requested.

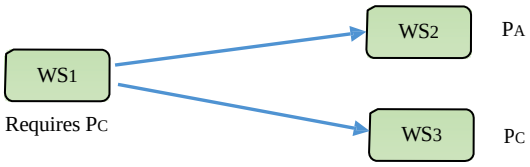

Figure 6. Verification Policy scenario

Finally, we propose to use a quantitative evaluation to assess the complexity of the adapters. The decision to use complexity is based on the fact that it is a common factor of performance in Web services [6]. It will allow measuring the development of adapters through a series of evaluation metrics. Table 5 describes the CK metrics [6] that will be adopted in this work to measure the complexity and coupling in Web services evaluation.

Table 5. Ck metrics in the evaluation [6]

CK Metric	Description
Line of code (LOC)	Number of lines of code used to measure the size of the development.
Number of Classes (NOC)	It is the number of classes or process in the context
Cyclomatic Complexity (CC)	It is the number of conditionals and loops statements to measure the complexity.
Weighted Method per Class (WMC)	It is the number of activities in BPEL associated with a process.
Coupling Between Objects CBO	It measures the coupling between adaptation logic and business.
External Classes NOEC	Number of external classes in the context.

7. Conclusions and Further Work

This paper proposed a Web service adaptation study based on non-functional properties like Quality of Service (QoS) to find mismatches in the Policy layer. The proposed work uses non-functional properties, in particular QoS policies as an additional input to the selection of the Web service. We are currently making some software experiments to design adapters and solve the QoS mismatches that have been identified in a policy containing the non-functional aspects of Web services at runtime. This may contribute to classifying mismatches in the policy layer and overview their impact on the Web service interoperability. In this paper, we also proposed to automatically resolve Web service differences through adapters. Mismatch patterns are viewed as a convenient means to capture the differences among two or more services at the policy layer and to encapsulate the solution to such differences. It is expected that our experiments can help draw conclusions on the development of adapters to solve non-functional aspects such as QoS mismatches to enhance interoperability in Web services. In the future we expect to carry out further studies about different incompatibilities to provide a better understanding of the adaptation process.

8. Acknowledgement

We would like to thank Birmingham City University for all the facilities given to carry out this research.

9. References

[1] S. Shetty and S. Vadivel, "Interoperability issues seen in Web Services". Journal of Computer Science and Network, vol. 9, no. 8, pp. 160–168, 2009.

[2] B. Benatallah, F. Casati and D. Grigori, "Developing adapters for web services integration". Advanced Information, 2005.

[3] Z. Maamar, Q. Z. Sheng, H. Yahyaoui, D. Benslimane, F. Liu and C. Bernard, "On Checking the Compatibility of Web Services' Policies" pp. 125–129, 2007.

[4] W. Kongdenfha, H. R. Motahari-Nezhad, B. Benatallah, F. Casati, and R. Saint-Paul, "Mismatch Patterns and Adaptation Aspects: A Foundation for Rapid Development of Web Service Adapters" IEEE Transactions on Services Computing, vol. 2, no. 2, pp. 94–107, Apr. 2009.

[5] Z. Maamar, D. Benslimane and A. Anderson, "Using Policies to Manage Composite Web Services". October, pp. 47–51, 2006.

[6] S. Wang, G. Zhang, X. Zhang and Y. Yang. "A Method for Detecting Behavioral Mismatching Web Services". Sixth Web Information Systems and Applications Conference, pp. 116–121, Sep. 2009.

[7] R. Mateescu, P. Poizat, and G. Salaun, "Adaptation of service protocols using process algebra and on-the-fly reduction techniques," *Softw. Eng. IEEE ...*, vol. 38, no. 4, pp. 755–777, 2012.

[8] Y. Taher, M. Parkin, M. Papazoglou, and W. J. van den Heuvel, "Adaptation of Web Service Interactions Using Complex Event Processing Patterns," Service-Oriented Computing, pp. 601– 609, 2011.

[9] H. H. Lin, T. Aoki, and T. Katayama, "Automated Adaptor Generation for Services Based on Pushdown Model Checking," in Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on, 2011, pp. 130–139.

[10] R. Seguel, R. Eshuis, and P. Grefen, "Constructing minimal protocol adaptors for service composition," in Proceedings of the 4th Workshop on Emerging Web Services Technology, 2009, pp. 29–38.

[11] H. R. Motahari Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati, "Semi-automated adaptation of service interactions," in Proceedings of the 16th international conference on World Wide Web, 2007, pp. 993–1002.

[12] K. Wang, M. Dumas, C. Ouyang, and J. Vayssiere, "The service adaptation machine," in on Web Services, 2008. ECOWS'08. IEEE Sixth European Conference, 2008, pp. 145–154.

[13] Z. Shan, A. Kumar, and P. Grefen, "Towards Integrated Service Adaptation A New Approach Combining Message and Control Flow Adaptation,"in Web Services (ICWS), 2010 IEEE International Conference on, 2010, pp. 385–392.

[14] L. Leite, G. Oliva, and G. Nogueira, "A systematic literature review of service choreography adaptation," Serv. Oriented …, vol. 3, pp. 199–216, 2012.

[15] M. Kamalabad. "Evaluating the similarity of web service policies using flexible parameter matching" Measurement, Information and Control (MIC), 2012 International Conference on. Vol. 2. IEEE, 2012.

[16] H. Nezhad, B. Benatallah, F. Casati and F. Toumani, "Web services interoperability specifications". Computer 39.5 (2006): 24-32.

[17] A. Al-Moayed and B. Hollunder, "Quality of Service Attributes in Web Services". Fifth International Conference on Software Engineering Advances, pp. 367–372, Aug. 2010.

[18] G. Zhang, S. Wang, M. Rong, and Q. Li, "A Model-Based Framework for Adapting Interaction Mismatches of Time- Aware Web Services," 2010 Int. Conf. Serv. Sci., pp. 76–81, 2010.

[19] D. Z. G. Garcia and M. B. F. de Toledo, "Semantics-enriched QoS policies for web service interactions" Proceedings of the 12th Brazilian symposium on Multimedia and the web - WebMedia '06, p. 35, 2006

[20] Y. Badr, A. Abraham, F. Biennier and C. Grosan, "Enhancing Web Service Selection by User Preferences of Non-functional Features". 4th International Conference on Next Generation Web Services Practices, pp. 60–65, Oct. 2008.