International Journal of Engineering & Technology, 7 (4) (2018) 2422-2429

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i4.14681
Research paper

Empirical Evaluation of Design Level Metrics for Aspect Oriented Business Process Execution Language in *SOA*

SENTHIL VELAN S[1*] and SAM JAFFRAY M[2]

[1]*Department of Computer Science and Engineering, Amity University, Dubai, UAE*
[2]*SSN College of Engineering, Chennai, Tamil Nadu, INDIA*
[*]*Corresponding author E-mail: svelan@amityuniversity.ae, svsugana@gmail.com*

Abstract

Service Oriented Architecture (*SOA*) facilitates seamless application integration through standards-based predefined web services. During integration, Business Process Execution Language (*BPEL*) plays a vital role in composing existing Web Services thereby achieving a service based workflow model. Due to frequently changing business requirements, it becomes very much essential for an *SOA* application to have the capability to dynamically bind with an alternate service rather than statically fixing the services in a given composition. However, *BPEL* lacks support for the run-time inclusion of a new Web Service or functionality. Aspects overcome this limitation by providing support to independently encapsulate the cross-cutting functionalities by separating them from the core business logic. Using *AOP*, it is possible to achieve dynamic binding in web service composition. To illustrate the embedding of *AOP* constructs into a *BPEL* process, this paper implements a case study on distributed e-HealthCare system. Further, two core design level properties namely, cohesion and coupling have been measured and the impact of introduction of *AO* into a composed *BPEL* process has also been discussed. Empirical evaluation of the design level properties shows that cohesion improves by the introduction of *AOP* in *BPEL*.

Keywords: *AOSD; SOA; BPEL; Software Design Metrics; Cohesion; Coupling*

1. Introduction

Enterprise software needs to be developed to cater to the frequently changing business requirements. Service Oriented Architecture (*SOA*) helps in achieving such agility and interoperability. Web Services are the primary technology for realizing a *SOA* [1–5]. Web Services [6] are reusable knowledge building blocks offered by an organization to their clients across the available network infrastructure. Web Services provided by organizations enable better communication in terms of Business to Business (*B2B*) and Business to Consumer interactions.

SOA is a framework of services that communicate and cooperate with each other to achieve a significant task [6]. In any business domain, there will be various complex business processes which involve orchestrating a lot of activities in order to achieve the desired functionality. Every activity can be realized by a single service or by a combination of multiple services.

Business Process Execution Language (*BPEL*) is the de facto standard that is used for composing multiple component services by specifying relevant composition rules. Web Services communicate with each other using the *SOAP* protocol messaging framework [7,8]. In *SOA*, a component web service can assume a role of a service provider or a broker or a service requester and the way it functions depend primarily on its role [9]. The composition of web services involves the creation of an orchestration from an existing set of services along with a collection of composition rules.

Aspect Oriented Programming (*AOP*) [10] is a unique methodology that helps in overcoming the problems arising out of inefficient encapsulation of cross-cutting (tangled and scattered) concerns. Using *AOP*, the software designer can neatly encapsulate these concerns into independent functional units called as *aspects*. By doing so, it can directly improve the modularization of software [11] and can also have a potential impact on the core quality attributes such as maintainability, reusability and evolution. Aspects of *AOP* can model the cross-cutting functionalities that need to be dynamically introduced into an *SOA* application without altering the deployment of base code. The capability of dynamic introduction of functionalities is possible through the constructs specified and supported by *AOP*.

Separation of cross-cutting behavior can be achieved at relative ease using the constructs of *AspectJ* programming language. However, introducing the same kind of behavior in the *BPEL* workflow language is still at its nascent stage. Charfi et al. [12] have proposed as well as implemented *AO4BPEL* and have tested through simple case studies [12–15]. The primary reason for the limited use of *AO4BPEL* is because of the need to understand the impact of using this in developing SOA based applications. Hence, it is vital to quantitatively assess the design level properties in an *AO* based *BPEL* environment. Cohesion and Coupling are two important design properties that quantify the internal strength of a given module and the external relationships between software modules respectively. Empirical evaluation can provide an improved insight towards understanding the impact of using any new methodology for software design. Towards these objectives, this paper explains the application of a set of standard metrics to measure the cohesion and coupling of *AO* software design. The overall flow of the research work explained through this paper is shown in Figure 1. The main contributions of our work are:

- Measurement of two core design level properties, namely cohesion and coupling, to provide insight on the impact of modelling an *SOA* application using aspectized *BPEL*.
- Development of an *SOA* based e-Healthcare software as a case study application with dynamic weaving and aspectized web services in a measurable *SOA* based TestBed environment.

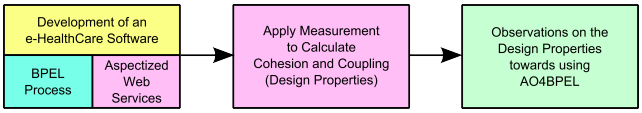

Figure 1: Overall Flow of the Research Work

The rest of the paper is organized as follows. Section 2 explains the design properties and their importance towards measurement. Section 3.1 explains the case study application. Section 3.2 provides the background on the standard Business Process Execution Language (*BPEL*). Section 3.3 expands on the method of introducing Aspect Oriented Programming in *BPEL*. A working model of the proposed system is illustrated using a system architecture in Section 4. The case study application namely service oriented e-Healthcare System is explained in the Section 5. Section 6 explains the need for an Aspect Service in the application taken as case study. Generally, an Aspect Service is a Web Service, which is invoked based on the Aspect logic and is woven into the *BPEL* language at run-time. Section 7 explains the evaluation of design properties namely, coupling and cohesion. Section 8 discusses the observation of the empirical evaluations. Section 9 concludes the paper and provides directions for further research.

2. Measurement of Design Properties

All software exhibits design properties that reflect on its units of encapsulation and the development model. Cohesion and coupling are two prominent properties of design considered as indicators of various quality attributes of a software. In fact in the literature, the first set of metrics for *AOP* [16, 17] that have been proposed were to measure these two important properties. A new set of metrics for the measurement of different design properties have also been introduced by Senthil et al. [31–33]. Similarly, introduction of *AOP* in *BPEL* requires quantitative measures to infer on the impact and hence cohesion and coupling are considered for measurement with the case study application.

3. Related Work

This section provides an explanation of the case study application, the *BPEL* language and also on the introduction of *AOP* in *BPEL* business process language.

3.1. Case Study Application

Mezini et al. [12], have clearly brought out the inherent deficiencies related to the separation of concerns in *BPEL* language. Business process oriented languages like *BPEL* do not provide mechanisms to encapsulate the cross-cutting functionalities such as logging, caching, persistence and auditing. It is also not possible to dynamically compose and adapt web services during run-time. Hence, an extension to *BPEL* language, named as *AO4BPEL* has been proposed by Mezini et al. with necessary constructs. This language provides mechanisms to define aspects and can be composed with an existing set of web services. However, this work did not quantitatively evaluate the design properties for applications developed using *AO4BPEL*.
Towards this objective, an *SOA* based distributed *e-Healthcare* system has been taken as a case study in the present work to evaluate the effectiveness of using *AO4BPEL*. A non-*AO* version of this system was developed by Kart [18] to overcome the issues related to communication of medical information between the doctors, nurses, pharmacists, and the primary stakeholder, namely the patients [19–21]. Paper records and handwritten transcriptions are major causes of mistakes in a predominantly manual healthcare system. In order to overcome this issue, an *SOA* based distributed e-Healthcare system has been designed and developed for a hospital which can effectively minimize the number of mistakes. An *AO* version of this system is not currently available. In the present work, it has been developed by modeling the non-functional requirements like *logging* and *transaction* as cross-cutting concerns and appropriately modularizing those concerns using aspects.

3.2. Business Process Execution Language (BPEL)

The flexibility in providing compositional web services plays a vital role in effective reusability of functionalities. *BPEL* is a workflow based composition language with an *XML* based workflow model and can be used to represent simple or complex business process(es). It consists of a set of basic and structured activities with a well defined order of execution. Two important constructs in the definition of *BPEL* language are variables and partners. Variables are used for data exchange between activities whereas partners represent the parties in the process of interaction and are called as *Partner Links*. The BPEL engine orchestrates the invocation of partner web services defined in the process specification. This work has been carried out using the open source Apache Orchestration Director Engine (*ODE*) [22, 23]. *WS-BPEL* utilizes several *XML* specifications, while *WSDL* plays the major role of describing the partner Web Services and also provides an interface for communication between the involved partners. Even *WS-BPEL* process has a corresponding *WSDL* document describing it to its consumers.

3.3. Introduction to Aspect Oriented Programming (*AOP*) in *BPEL*

The focus of this research work is the introduction of Aspect Oriented Programming (*AOP*) in *BPEL* and to quantify the impact by measuring the design level metrics namely, cohesion and coupling. The cross-cutting functionalities can be encapsulated using the aspects of *AOP* and embedded in a web service. The aim of *AOP* is to enhance the modularity and maintainability of the software and to provide mechanisms for the clear separation of multidimensional concerns [24–26]. Based on the business logic modeled in an *AO* software, concerns or functionalities are classified as core and cross-cutting concerns [27]. Core concerns are functionalities that address the core business logic or primary functions in the domain. When modeling a business application using an *AO* software, a software architect starts with the identification of the primary functionalities in the business. For example, considering a banking software, the core business logic can be managing the transactions of each customer in a bank, whereas, system-wide concerns involve features such as logging, authorization, persistence and other elements, which cross-cut the core business logic in multiple ways [27]. Aspectization involves the identification and encapsulation of the core and cross-cutting concerns into separate entities, thereby achieving higher modularity and better maintainability for the software.

3.3.1. Symptoms of Non-Modularity

During the design of a software, if a clear separation of concerns is not achievable, then it results in code tangling and scattering. Code tangling is caused by multiple and simultaneous implementations of various concerns in a single module. i.e., one module encapsulates parts of multiple concerns [27]. Code scattering occurs due to the duplication of code, and is the consequence of a single feature im-

plemented in multiple modules. *AOP* is able to provide a solution for the neat encapsulation of concerns with the help of the following four major concepts, namely:

Join points are well defined points in the execution of the *BPEL* language, where cross-cutting functionalities can be woven.
Pointcut is a programming construct used for the selection of the join points and also to specify the weaving rules.
Advices are functionalities that need to be optionally executed when a join point is reached in the base code (can be *before*(), *after*() or *around*() the join point).
Aspects are entities that encapsulate the definition of pointcut and the corresponding advices. During compilation, the advices of aspects are woven at the respective join points of the base code, and this process is shown in Figure 2.

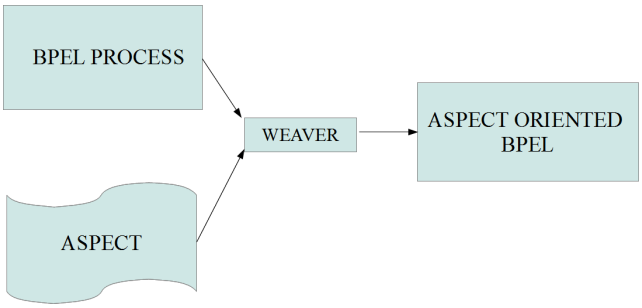

Figure 2: Weaving of aspect and base code during execution

4. Generic Working Model of the Proposed System

The introduction of Aspect registry into the system plays a vital role in promoting the runtime change in a *BPEL* process. The Service binding resulting from the inclusion and replacement of Web Services within the existing *BPEL* process can be achieved on-the-fly. A parameter set can be defined and registered in the Aspect registry for its invocation into the composition. The architecture of the proposed *AO4BPEL* system is shown in Figure 3.

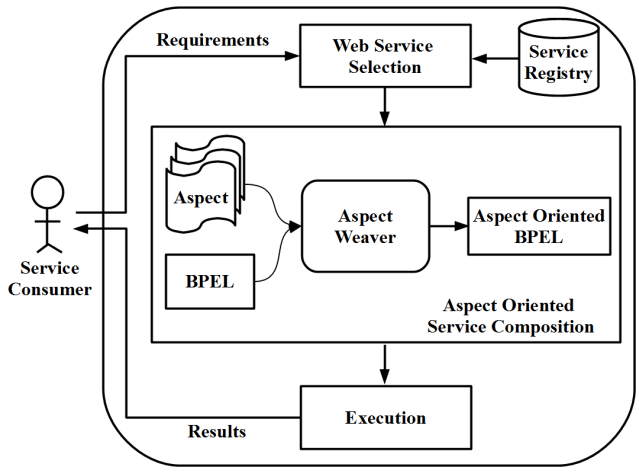

Figure 3: Generic Architecture of proposed *AO4BPEL* System

The service consumers specify their needs as a list of requirements. Based on these requirements a set of web services are selected from the service registry and composed using the *BPEL* language. Later, the aspects are woven to *BPEL* to satisfy the cross-cutting concerns specified by the consumers. This process is called as Aspect Oriented Service Composition. Later, the composed and woven *BPEL* process is executed as per the defined work flow and the results are sent back to the service consumers.

5. Service Oriented e-Healthcare System

In a healthcare centre such as a hospital, the effective communication between the stakeholders namely, the patient, physician and pharmacist plays a vital role in providing accurate and effective service to its patients. Being in the age of electronic bookkeeping, the communication mechanism is based on a large quantity of paper work [19–21]. Situations of ineffectiveness while prescribing on paper include: (*i*) A pharmacist may misunderstand the prescription written by the physician, (*ii*) The patient may even forge the prescription, and (*iii*) The physician might be unaware of the medication provided by other physicians for a patient.A Washington Post article indicates that as much as 5 percent of the three billion prescriptions filled during each year are incorrect [18]. In order to provide an effective communication between the trios and to introduce the concept of e-Healthcare, the functionalities of registration, appointment, prescription, availability of medicine and billing are performed electronically.

5.1. Component Web Services

A set of web services is proposed, designed and implemented for the e-Healthcare system as shown in Figure 4 and 5. These services have been identified in order to capture the functionalities involved in the process of providing an efficient health care to the patients of an hospital. The services created for the e-Healthcare system are given below with suitable explanation regarding their functionalities.

RegistrationWS Registers the information collected from a patient and provides a unique ID to the patient associating him/her to the correct physician.
PrescribeMedicineWS Helps the physician to prescribe medicines to the patient and further sends the control to the Pharmacy. It also returns the fees structure for the physician based on the patient and his/her illness.
SurveyMedicine A particular formula of medicine can be manufactured by different pharmaceutical companies. Encapsulating this functionality, this service returns the best manufacturer for a given medicine.
StockVerificationWS Checks for the availability of the prescribed medicine and returns the status to the corresponding requester.
OrderIDWS A service which returns the order ID and helps in transferring control to the new branch.
PharmacyInWS This service computes the price for each medicine prescribed by the physician.
PrepareInvoiceWS The *PrepareInvoiceWS* service obtains the payment information from the physician and the medicine invoice from the pharmacy. Based on the contents of the invoice and the information provided in the prescription, a bill is generated and calculated as Net Pay.
InsuranceWS This service receives the Net Pay, Date and Insurance Identification Number as input, and returns the total amount mentioned as Net Pay (as a result of claiming amount from Insurance) to the *Hospital_AdminWS*.
Hospital_AdminWS A service that displays notification upon the receival of Net Pay.
CreateLogWS Records a log to track information about the services created through BPEL, which gets triggered for a given input from the requester.
ProvideDiscountWS This service provides the functionality of calculating the discount based upon the value obtained for the Net Pay.

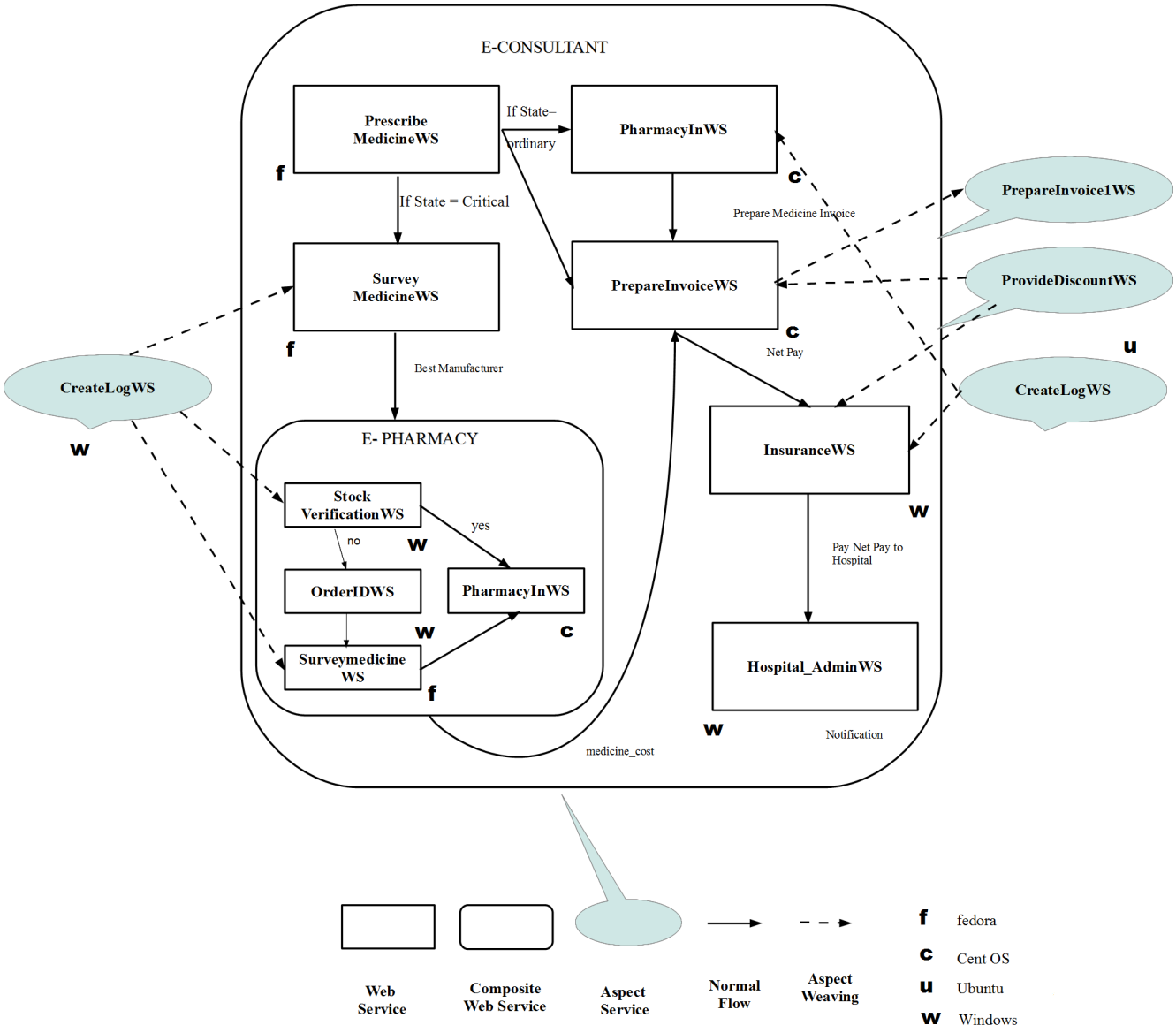

Figure 4: Web Service Composition in E-Healthcare Software

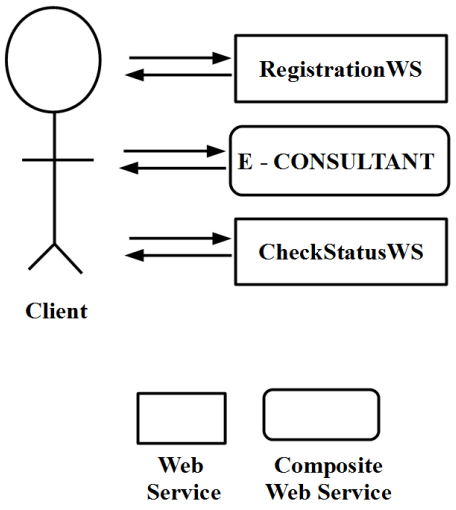

Figure 5: Aspect based Service-oriented e-Healthcare Software

6. Scope of Aspect Enabled Service in Application Scenario

Aspects of *AOP* are entities that can neatly encapsulate the cross-cutting concerns into modular units of functionality. Non-functional requirements like logging, persistence and caching are candidate functionalities to be encapsulated as aspects since their implementation can affect appreciable number of modules in the base code. In the E-CONSULTANT scenario, a generic log file is created, which captures the invocation details of all the Web Services involved in the composition. Considering the scenario where for every invocation of *SurveyMedicineWS* web service, the requester is charged US$ 100 (e-business) and the provider of *SurveyMedicineWS* web service claims a bill of US$ 1000 at the end of the month to the promoter. In order to cross verify the correctness of the bill, a log file is created for every invocation of *SurveyMedicineWS* web service using the *CreateLogWS* web service, which is modeled as an aspect. In order to introduce a discount during the bill processing dynamically, the discount calculation can be modeled as an aspect. *ProvideDiscountWS* web service can be weaved into *BPEL* by intercepting the output of *PrepareInvoiceWS*. In order to achieve this, the aspect encapsulated web services involved are listed below:

- *CreateLogWS*
- *ProvideDiscountWS*
- *PrepareInvoice1WS*

One more dynamic functionality is also encapsulated as an aspect and uses the powerful construct of *around*() advice. This advice is used to bypass the existing Web Service and replace with a new one in the *BPEL* process. In the composition logic, a condition may arise in which switching of services may be needed, ie., instead of executing Service A, based on some criteria an another service namely Service B need to be executed. This type of replacement is needed when a service involved in the composition throws a faulty result or because of the shutdown of a system or even due to network congestion which results in the unavailability of the required service. Any good application needs to make its services available 24x7 in addition to meeting the requirements of the customer. This can be achieved by the dynamic binding of services present in the Aspect registry, thereby making the application more flexible and adaptable. By using *AOP*, swapping of services is easily achieved, where instead of running the *PrepareInvoiceWS* web service, *PrepareInvoice1WS* web service is executed(from Application Scenario). Thus, the dynamic replacement of Web Service is achieved through the Aspect registry.

6.1. Component Web Services

The pointcut is defined using the syntax of *XPATH* as *BPEL* uses the notations of the *XML* language. Extending this to the definition of join point, activities in the *BPEL* processes are captured as join points. The syntax of using a single pointcut which captures multiple join points are analyzed and shown in the code segment given in Listing 1. The *XPATH Union* operator is used in case of capturing more number of join points with the same advice that needs to be woven.

Listing 1: A sample *testPointct*

```
<aspect : pointcutname="testPointcut"
          language="xpath">
/  bpel : process [@name='ehealth']
// bpel : invoke  [@name='service1'] |
// bpel : invoke  [@name='service2'] | ...
</aspect : pointcut>
```

The interconnection between the web service that encapsulates an aspect and the *BPEL* process is made with the help of context variables [28]. In this way, the level of content coupling increases as the value from the *BPEL* process is passed as a parameter to the remote aspect service and is executed based on the procedure defined in the aspect service. A code segment that defines the usage of the context variable is shown in Listing 2.

Listing 2: Code segment for Context Variable

```
<copy>
<from> <![CDATA[$ContextVariable.parameters/
                                    inv1:return]]>
</from>
<to> <![CDATA[$incrementopRequest.parameters/
                                    inc:args0]]>
</to>
</copy>
```

The Context variables [28] are used to access the meta-data of the join point. The consequence of this method of passing arguments could result in having a high or low level content or control coupling. Even though this tight coupling might not be desirable, since the aspects can be dynamically plugged and unplugged, the functionalities modeled in the aspects can be modified without affecting the base code.
AOP was introduced to increase the modularity of software which can lead to highly cohesive modules. However, the effect of modularizing cross-cutting functionalities can lead to increase in coupling between aspects and the base code. As explained before, these highly cohesive units can be independently modified and hence the ill effect of the higher degree coupling will not necessarily affect the quality of software.

7. Empirical Evaluation

In order to quantitatively assess the effect of aspectizing *BPEL*, design level properties need to be measured with suitable metrics. In this evaluation, existing metrics have been used to quantify two important design level properties namely cohesion and coupling. Cohesion is a property of a software component that involves in combining the related functionalities into a single module. Coupling quantifies the degree of dependency between the modules of a software. In order to achieve a good design, it is desired to have high cohesion and low coupling among the components of the software. While it is desirable to have a higher value of cohesion in the spectrum ranging between 0 and 1, a higher level of coupling leads to a software that is very difficult to maintain during its evolution.
By neatly encapsulating cross-cutting concerns into independent modular units, *AOP* can definitely improve the cohesive strength of the modules of software. The effect of aspectization has been measured by quantifying the cohesion and coupling levels of the *BPEL* process with and without using the constructs of *AOP*, i.e., measuring the aspectized and non-aspectized versions of the *SOA* based e-Healthcare Software. In any *BPEL* process, a good number of Web Services are involved during the composition, in order to satisfy the requirements of the workflow. In the composition, an output of one service can be taken as one of the parameters of the input variable for another service. When an aspect is plugged in, the coupling strength remains high, as the introduction of the new functionality depends upon the weaving of aspects. We name this type of dependency that exists between the *BPEL* process and an aspect as aspect-process dependencies. The dependency is measured using the Fenton and Melton formula [29] as given in Equation 1.

$$Coupling\ Factor, CpF(B_p, A) = i + \frac{P_j}{1+P_j} \tag{1}$$

where,
B_p, A represents *BPEL* process and aspects respectively,
P_j represents the interconnections between B_p and A. These interconnections happen through pointcuts and join points in Aspect Oriented *BPEL*, and
i is the level of coupling type found between B_p and A.

For example, the value of i is 5 for content coupling, 3 for control coupling and 0 for no coupling. The different values assigned to the levels of coupling, i, is shown in Table 1.
Based on experience, it can be said that a component with higher level of cohesion will exhibit characteristics of higher reusability and maintainability. An Aspect Oriented Business process encapsulates cross-cutting concerns into separate entities resulting into highly cohesive units, thereby increasing its modularity. Cohesion in the *BPEL* process can be measured by counting the number of operations performed by a service. The formula for calculating the *Cohesion Factor* of the *BPEL* process which is derived Fenton and Melton formula [29] is given in Equation 2.

$$Cohesion\ Factor, CF = \frac{\alpha}{\gamma} \tag{2}$$

Table 1: Levels of Coupling

S. No.	Coupling Type	Level of Coupling (i)
1	High Level Content Coupling	5
2	Low Level Content Coupling	4.5
3	Common Coupling	4
4	High Level Control Coupling	3
5	Low Level Control Coupling	2.5
6	Stamp Coupling	2
7	Data Coupling	1
8	No Coupling	0

where,
α is the number of unique functionalities (operations) performed within the services involved in the *BPEL* process, and
γ represents the total number of services involved in the *BPEL* process.
When an aspect functionality is woven into the *BPEL* process, the Cohesion Factor may vary considering the different types of *BPEL* processes. In general, constructs such as advices are used in aspects to weave the tangled and scattered functionalities in the *BPEL* process. Hence, a *Net Cohesion Factor* can be calculated after the introduction of aspect, for a *BPEL* process. The Net Cohesion Factor is computed using the formula given in Equation 3.

$$Net\ Cohesion\ Factor, NCF = \frac{\frac{\alpha}{\gamma} + \frac{\beta}{\theta}}{2} \tag{3}$$

where,
β is the number of aspect functionalities, and
θ represents the total number of advices in all aspect files.
In Figure 6, $a_1|a_2|...|a_n$ denotes a single pointcut which captures multiple activities in *BPEL* process. The input and output are web services in the *BPEL* process. New Aspect services are dynamically brought into the *BPEL* process as Web Services through the aspect registry. This will promote conceptualization of web services that are able to encapsulate the aspects.

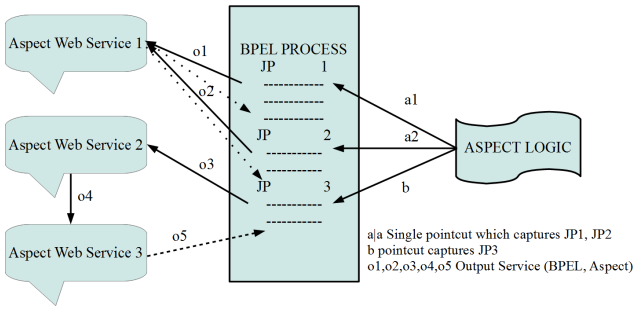

Figure 6: Flow of control between Aspect and BPEL promoting Content coupling

8. Discussions

Two important design properties of the software, cohesion and coupling have been measured using a set of available metrics for the e-HealthCare case study. An output from one of the Services (considered here as *Service A*) involved in the composition is taken as input by an Aspect service *Service B*. Further, *Service B* computes a result based on the input that is taken from *Service A* which results in a high value of content coupling. *ProvideDiscountWS* Aspect service calculates a final value based on the computed value of Net Pay in *PrepareInvoiceWS*. *PrepareInvoiceWS* service passes this computed value to another service in the *BPEL* composition.

In case of *CreateLogWS*, the values from various Web services (through multiple join points, in the composition) are captured dynamically and a log is maintained, to track the information regarding the service invocation details, time of the invocation, etc. Consequently, a low level of content coupling is maintained, and the Aspect service in no way affects the results which it interrupts through the respective pointcut. The primary purpose is to simply store the details of the join point, which is a Web Service and again set the flow back to *BPEL*, without manipulating the captured result.

From Table 2, we could infer that the coupling factor is increased especially when there exists a high level content coupling between the aspects and the web services. The content coupling is tolerable since, the aspects can be dynamically plugged and unplugged and the functionalities modeled in the aspects can be modified without affecting the base code. Similarly, the level of security on *SOAP* must also be enhanced, before modifying the content that is passed from *BPEL* to the Aspect service. Ensuring Security in the phase of transaction from *BPEL* to Aspect level services plays a vital role in Banking and financial applications, since it checks for user authentication and thereby promoting secured transactions.

Cohesion which is another crucial design property was measured through Equation 3. As given in Table 3, the value of the Net Cohesion Factor is above 0.9 measured for the case study application. For the Test applications 1 and 2, when the number of advices for the same functionality is increased, its cohesion decreases. The Test applications are sample applications in which all the possible cross-cutting functionalities are modeled as aspects. In e-Healthcare software by modularizing the logging cross-cutting functionality in the *CreateLogWS* Aspect service, the metric value is improved and main-tained closer to 1.

In the same way, the discount functionality modeled using the *ProvideDiscountWS* web service is able to independently perform the criteria for discount calculation. Again the *PrepareInvoice1WS* web service is also able to capture the functionality in an independent web service and thereby strengthening the cohesiveness of the application. This is evident from Table 3, as the *NCF* of the case study is very high compared to the Test Applications 1 and 2.

Table 2: Measuring the Coupling Factor of BPEL with Aspect Services

Artifact	# of Join Points (P_j)	Coupling Type (i)	Coupling Factor ($CpF(B_p,A)$)
Aspect based e-Healthcare software without weaving of Aspects	0	No Coupling(0)	0
Weaving Discount functionality dynamically	1	High level Content Coupling (5)	5.5
Addressing cross-cutting concern	5	Low level Content Coupling (4.5)	5.3
By-Pass of *PrepareInvoiceWS* by *PrepareInvoice1WS*	1	High Level Content Coupling (5)	5.5
Test Application 1	7	Control Coupling (3)	3.875
Test Application 2	3	Low level Content Coupling (4.5)	5.25

Table 3: Measuring the Net Cohesion Factor of BPEL with Aspect Services

Artifact	Functionalities performed within th services involved in the *BPEL* process (α)	Total # of Web Services involved in the application (γ)	# of aspect function -alities (β)	Total number of advices in all aspect files (θ)	Net Cohesion Factor (NCF)
Aspect Based Service Oriented e-Healthcare software	8	9	3	3	0.944
Test Application 1	8	9	3	4	0.819
Test Application 2	8	9	2	4	0.694

9. Conclusion and Future Work

In this paper, an attempt has been made to quantitatively assess the effect of aspectizing *BPEL* by applying the measures of cohesion and coupling. As a case study, Service Oriented e-Healthcare software was designed and developed to run on different machines with different operating systems to achieve platform independence. Composition logic has been specified using the *BPEL* process and deployed in Apache ODE. Aspects are weaved into the existing *BPEL* through ODE Integration Layers. Dynamic inclusion was also achieved by bringing in a new Web Service into the existing *BPEL* process.

Cohesion and coupling were measured using the Net Cohesion Factor and Coupling Factor metrics defined by Fenton [29]. The metrics were applied to the case study and two test applications. The measurement showed an increase in cohesive strength of the case study software and positively affects the introduction of *AOP* in *BPEL*. Even though, the effect of modularizing cross-cutting functionalities increases the coupling factor; these highly cohesive units can be independently modified and will not necessarily affect the quality of software. This work can be extended further to other case study applications with more number of aspects to understand the impact of applying *AOP* in a broader sense. Also, security can be introduced in Web Services, by encapsulating it as an aspect.

References

[1] Wilde N., Bagui S., Coffey J., El-Sheikh E., Reichherzer T., White L., Goehring G., Terry C., and Baskin A., Interoperable Systems and Software Evolution: Issues and Approaches, in *Advances in Intelligent Systems and Computing*, Vol. 205, Springer Berlin Heidelberg, 2013.

[2] Herrero J. L., Sánchez F., Lucio F., and Toro M., Introducing separation of aspects at design time, in *Workshop on Aspects and Dimensions of Concerns*, (ECOOP 2000), June, 2000.

[3] Mens T. and Tourwe T., A survey of software refactoring, *IEEE Transactions on Software Engineering*, Vol. 30, No. 2, pp. 126–139, 2004.

[4] Quynh P. T. and Thang H. Q., Dynamic coupling metrics for service oriented software, *Journal of Science and Technology*, Vol. 3073, pp. 50–53, 2009.

[5] Weyuker E. J. and Avritzer A., A metric to predict software scalability, in *Eighth IEEE Symposium on Software Metrics*, pp. 152–158, 2002.

[6] Schmelzer R., Vandersypen T., Bloomberg J., Siddalingaiah M., Hunting S., Qualls M. D., Houlding D., Darby C., Kennedy D., Schmelzer R., Vandersypen T., and Bloomberg J., *XML and Web Services Unleashed*, Dorling Kinderslay Pvt. Ltd, 2008.

[7] SOAP Specifications, *W3C Simple Object Access Protocol (SOAP) 1.1*, 2004.

[8] W3C, *Web Services Description Language (WSDL) 1.1*, March 2001.

[9] Vasiliev Y., SOA and WS-BPEL, *Composing Service-Oriented Solutions with PHP and ActiveBPEL*, PACKT Publishing, 2007.

[10] Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C. V., Loingtier J.-M., and Irwin J., Aspect-oriented programming, in *European Conference on Object Oriented Programming*, pp. 220–242, 1997.

[11] Laddad R., Aspect-oriented programming will improve quality, *IEEE Software Magazine*, Vol. 20, Iss. 6, pp. 90–91, Nov–Dec 2003.

[12] Charfi A. and Mezini M., Aspect-oriented web service composition with AO4BPEL, in *Web Services* (L.-J. Zhang and M. Jeckle, eds.), of *Lecture Notes in Computer Science*, pp. 168–182, Vol. 3250, Springer Berlin Heidelberg, 2004.

[13] Charfi A., Schmeling B., Heizenreder A., and Mezini M., Reliable, secure, and transacted web service compositions with AO4BPEL, in 4^{th} *European Conference on Web Services* (ECOWS 06), pp. 23–34, 2006.

[14] Charfi A. and Mezini M., AO4BPEL: An aspect-oriented extension to BPEL, *World Wide Web*, Vol. 10, No. 3, pp. 309–344, September 2007.

[15] Charfi A., Dinkelaker T., and Mezini M., A plug-in architecture for self-adaptive web service compositions, in *IEEE International Conference on Web Services*, (ICWS 2009), pp. 35–42, 2009.

[16] Zhao J. and Xu B., Measuring aspect cohesion, *Fundamental Approaches to Software Engineering*, LNCS, Springer Berlin Heidelberg, Vol. 2984, pp. 54–68, 2004.

[17] Zhao J., Measuring coupling in aspect-oriented systems, in *Information Processing Society of Japan* (IPSJ), pp. 14–16, 2004.

[18] Kart F., Moser L. E., and Melliar-Smith P. M., Building a distributed e-healthcare system using SOA, *IEEE IT Professional*, Vol. 10, pp. 24–30, March–April 2008.

[19] Weiss R., Medical errors blamed for many deaths: As many as 98,000 a year in the us linked to mistakes, *Washington Post*, pp. A01, 30 November 1999.

[20] Branigan T. A., Medication errors and board practice, *Colorado State Board of Pharmacy News, National Assoc. of Boards of Pharmacy*, pp. 1–4, February 2002.

[21] Kohn L. T., Corrigan J. M., and Donaldson M., To err is human: Building a safer health system, *Institute of Medicine, US National Academy Press*, pp. 1–8, February 1999.

[22] Apache ODE - Apache Orchestration Director Engine, January 2013.

[23] Apache Software Foundation Apache ODE (Orchestration Director Engine), January 2013.

[24] Tarr P., Ossher H., Harrison W., and Sutton S. M. Jr., N degrees of separation: multi-dimensional separation of concerns, in *Proceedings of the 21st International Conference on Software Engineering*, ICSE 99, New York, USA, pp. 107–119, ACM, 1999.

[25] Ossher H. and Tarr P., Using multidimensional separation of concerns to (re)shape evolving software, *Communications of the ACM*, Vol. 44, pp. 43–50, October 2001.

[26] Ossher H. and Tarr P., Multi-dimensional Separation of Concerns and the Hyperspace Aapproach, in *Software Architectures and Component Technology*, Vol. 648 of *The Springer International Series in Engineering and Computer Science*, pp. 293–323, Springer US, 2002.

[27] Ramnivas L., AspectJ in Action: Practical Aspect-Oriented Programming, *Manning Publications Co.*, Greenwich, CT, USA, 2010.

[28] D. von Alexander Look aus Dieburg, Expressive Scoping and Pointcut Mechanisms for Aspect-Oriented Web Service Composition, *PhD Dissertation*, Darmstadt University of Technology, Darmstadt, Germany, September 2011.

[29] Fenton N. and Melton A., Deriving structurally based software measures, *Journal of Systems and Software*, Vol. 12, Issue 3, pp. 177–187, July 1990.

[30] Jose H.S.S., Gonçalves F.E., Cappelli C., Santoro F.M., Providing Semantics to Implement Aspects in BPM, in Dumas M., Fantinato M. (eds) Business Process Management Workshops, BPM 2016, *Springer Lecture Notes in Business Information Processing*, Vol. 281, 2017.

[31] Senthil S., Chitra Babu, and Madhumitha R., A Quantitative Evaluation of Change Impact Reachability and Complexity across Versions of Aspect Oriented Software, in *The International Arab Journal of Information Technology*, Vol. 14, No. 1, pp. 41–52, January 2017.

[32] Parthipan S., Senthil S., and Chitra Babu, Design level metrics to measure the complexity across versions of AO software, in *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, pp. 1708–1714, May 2014.

[33] Vinobha A., Senthil S., and Chitra Babu, Evaluation of reusability in Aspect Oriented Software using inheritance metrics, in *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, pp. 1715–1722, May 2014.