Applying Aspect Oriented Programming in Distributed Application Engineering

Fatiha Khalifa[1]
Oran University of Science and Technology
Mohamed BOUDIAF USTO'MB, Oran, Algeria

Samira Chouraqui[2]
Oran University of Science and Technology
Mohamed BOUDIAF USTO'MB, Oran, Algeria

***Abstract*—Aspect-oriented programming is an emerging programming paradigm that stretches during the development phases in different domains. Many researchers have focused on the use of this paradigm in web service composition in different research axis. However, none of them use together aspect-oriented programming and design by contract to deal with the adaptation of the parameters in the web service composition process. This paper proposes a web service composition algorithm based on the planning graph using both Aspect-oriented programming and design by contract concept. The aspect-oriented Programming approach provides explicit support for separation of crosscutting concerns in web services composition whereas the design by contract approach allows the processing of parameters execution in pre-condition and post-condition mode by using contracts in order to ensure correct service execution with adaptation to external parameters without touching in properties which can be dealt with re-construction of the composite service. Future development of this planning graph will include the introduction of the dynamic way of aspect oriented programming and add comparison results.**

***Keywords*—Aspect oriented programming; design by contract; web service composition; parameters adaptation**

I. Introduction

Aspect-Oriented Programming (AOP), is a new programming paradigm introduced in information systems, presents a new element called aspect, in order to encapsulate the crosscutting concerns of the program. Instead of having one concern repetitive in multiple code blocks, the aspect can represent all these concerns in a single code block completely separate from the source code [8].

The aspect contains three main elements, a joinpoint, a pointcut, and advice. AOP also introduces the notion of a weaver. Weaving behavior is the process that allows weaving the program with these different aspects [3, 4].

Some researchers have focused on applying AOP technologies into the Web service composition domain. Their researches goals were situated around the increase in the adaptability of web service [17] or modularize crosscutting concerns in web service composition [10].

However, no one of them has treated the problem of parameter adaptation and conflict between the services during the composition phase by the mean of AOP and design by contract.

The Design by Contract (DbC) is an approach that uses a contract to specify and define the mutual obligations and expected parameters of the communication between services composite process, and use assertions to check whether an application complies with a contract. The failure of an assertion is typically a symptom of a bug in the software. There are three different kinds of assertions [5, 7]:

1) Pre-conditions: specifies parameters conditions that must hold before an operation executes.

2) Post-condition: specifies parameters conditions that must be hold after an operation completes, consequently, post-condition is evaluated after a method completes.

3) Invariant: specifies a parameters condition that must be hold anytime when a client invoke an object’s method.

The work in this paper proposes a web service composition algorithm based on the planning graph using both AOP and DbC to deal with the problem of parameter adaptation and conflict in web service composition using separation of crosscutting concerns.

Remind that web services are applications available on the internet, each of them performs a special task [1].

Except that, the requirements of the client always exceed the demand of a single request or a single task, for example, if the client wants to afford a holiday, he desires to find a web service that offers him in the same time, purchase of a plane ticket, hotel reservation, and car reservation, and other.

As no specific web service can meet all of these requirements at the same time, it should be possible to combine several existing services to fulfill one's needs. This is the composition of web services. However, one of the important issues to be addressed in the composition of web services is that some services impose certain input or output parameters that are defined by their suppliers and/or imposed by their clients. These constraints specify the conditions that must be met to ensure correct execution or appropriate interaction with the different services involved in the composition.

In this context, the main contributions of our research work are focused on:

Applying the AOP paradigm into web services composition to increase the adaptability of services and to modularize crosscutting concerns. When crosscutting concerns are separated from the code of each service, it becomes easy to modularize the crosscutting concern of the composite service and then monitoring these parameters as discussed by Sk. Riazu Rahemana et al. in [9, 20].

On the other hand, we have applied the DbC paradigm for safer interaction between input parameters of each new service which is added to the composition and the output parameters of the composite service belonging in web services composition to avoid conflicts and exceptions.

The remainder of this paper is structured as follows:

Section 2 reviews related work, Section 3 presents the conceptual architecture. In Section 4 an example is given. Finally, Section 5 concludes the paper.

II. Discussion and Related Work

Many types of research corresponding to the web service composition have been published in recent years. They revolve around different areas of research. We focus on those who used the AOP.

Various studies have been made concerned applying AOP in web services composition like those in [9, 10, 11, and 12].

Charfi and. al. has approached this problem from a different direction. 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 [15] However, they do not pay attention on the issue of the crosscutting concerns consisted in service compositions.

Both of [10], [11] propose 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.

Authors in [12] have proposed a formal method through a Petri net-based algebra for aspect-oriented web service composition. The formal semantics of the composition operation including composition operation for modeling basic compositions and crosscutting operation for modeling aspects is expressed in terms of Petri nets.

In [16] the authors used distributed aspect-oriented programming (AOP) technology to model an adaptive architecture for Web services composition, by representing the non-functional properties of each Web service - composite and component - via AOP. They make a relation function between the aspects of the composite web service and the individual aspects of the component Web services.

In [17] authors proposed a method to increase the adaptability of web service by using the main AOP agreed semantics.

In [18] an approach that have brings design by contract to Web services has been presented. Authors have elaborated generic solution architecture, and define its components and have investigated the foundations such as important guidelines for applying design by contract.

In [2, 19] authors have proposed a graph plane based approach model and detect composition conflicts related to introduction (structural composition).

In [13] authors have applying design by contract an Aspect Composition.

However, none of these approaches have been applying both AOP and DbC in the same context of web service composition. Thereby this paper is the first attempt at using both the AOP approach and DbC benefit in web service composition focused on parameter adaptation.

III. Conceptual Architecture

A. *Concepts and definitions*

When crosscutting concerns are separated from each service in a web services composition, a service composition can be seen as a result of a composite web service weaved with aspects and contracts.

This section of the paper will describe web service composition algorithms based on the planning graph construction. On giving first certain definitions below.

- Definition 1

L is the set of different available services participating in the composition of web services. L= {S1, S2, S3… Sn}

And Si is a service number i defined by Si=<Pi.I, Pi.O, Cc>

- Pi.I is the input parameters of the service i
- Pi.O is the output parameters of the service i
- Cc is a list of the crosscutting concerns (scattered or tangled codes) requirements of the service.

- Definition 2

R is the set of different Aspect, R= {A1, A2, 3… An}

A is an Aspect defined by

A=<Cc, Joinpoint, Pointcut, Advice>

Where,

- Cc: crosscutting concern functionality.
- Advice: is a workflow code that encapsulates Cc.
- Joinpoint some points in the program of the service related to pointcuts of the aspect.
- A pointcut is a function that relates a joinpoint to a set of advice.

There are three sorts of pointcuts:

- A before pointcut Si.Cc⤵A.advice, represent that advice is executed before the execution of the service i.
- An after pointcut Si.Cc⤴A.advice , represent that advice executed after the execution of the service i.
- An around pointcut Si.Cc⤴⤵ A.advice , represent that advice executed around execution of the service i.

If an aspect A.advice crosscuts a crosscutting concern of a service S, it gives us: Si' = Si ◁ A, which represents that the service Si, is weaved with aspect A.

- Definition 3

We define Db as a Boolean contract relationship between the output and input parameters of two layers successive in the Graph, given by:

Db{type, Si.Output parameters, Si+1.Input parameters}.

Where type can take three formats:

- @Pré (a precondition of the contract): specify a contract that must hold before the execution of the input parameters of the service Si.PI.
- @post (postcondition of the contract): specify a contract that must hold before the execution of the input parameters of the service Si.PI.
- @Inv (invariant): specifies a contract that must behold any time when service features are invoked.

Several cases are treated:

- If (Si.PI∩Si+1.PO = Ø) then

  Db (@Pré, Si.PI, Si+1.PO) = true

If this condition is satisfied we have: $S = S1 \perp S2$

$S1 \perp S2$ represents a composite service S that results from performing the service S1 followed by the service S2, S1 must be completed before S2 can start.

- if (Si.PI∩Si+1.PO = Ø) then

  Db (@Post, Si.PI, Si+1.PO) = true

  In this case, we have: $S = S1 \vdash S2$

$S1 \vdash S2$ represents a composite service S that results from performing unordered between S1 and S2, the service S1 followed by the service S2 or S2 followed by S1

- if (Si.PI∩Si+1.PO = Ø) then

  Db (Inv, Si.PI, Si+1.PO = true

In this case, we have: S1|| S2 represents a composite service S, which results from performing in parallel service S1 and/or service S2.

- Definition 4

We define the composite web service request as a tuple:

REQ= <L, R, Db> where:

- PI: is the set of the input parameters that the client can provide.
- PO: is the set the output parameters expected by the client.
- PE: is the set of constraints representing required limitations on input and output parameters, as required by the client.

B. *Proposed Algorithm*

This section describes the algorithms for constructing web services composition based in a planning graph (see Fig. 1), applying aspect-oriented programming and contracts techniques.

Our planning graph is a horizontal directed layered graph in which the jump to the next node is permitted only from one node layer to the next.

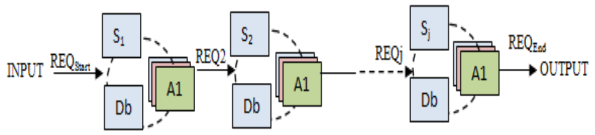

Fig. 1. A Planning Graph of the Web Services Compositions.

The node in level 0 corresponds to the REQStart

- REQStart is the node input of the graph which includes specifications and parameters given by the client in the composite web service; it's the Composition start request.

REQStart <L, R, Db> = INPUT request= {specification set of the client}

The node in level i depends on the composite service got from on the result of the layer i-1, who will be in turn submitted to the action of the REQi by weaving the aspects required and by applying the necessary contracts.

REQi <{Si-1,Si}, R, Db>

- REQEnd is the last node of the graph which give the composition plan (algorithm 3) result that must accomplish the client requirements as specified in the INPUT request.

REQEnd is the OUTPUT request which gives us as a result the composition plan.

Algorithm1. Services composition algorithm

INPUT: REQstart{ Composition request start }, L{ Set of available services }, R{Set of Aspects}, Db{ relation of Contract }, n (maximum numbers Service available in L)

OUTPUT: GraphPlan (REQEnd or failure)

```
 1: Graphplan=null;InputParameters =REQstart.PI
 2: n = ∑service ∈ L
 3: Si' =null
 4: for i=1 to n do
 5: L=i
 6: for each Service Si ∈ L do
 7: if (Si.PI⊂ Inputparameters) and (Si ∉ grapheplan) then
 8: for each A ∈ R do
 9: if (Si.Cc = A.Cc) then
10: addAspect(Weaved, Si, A)
11: end if
12: end for
13: OutputParameters=OutputParameters ∪ Si.PO
14: AddService(REQ L, Si',Si)
15: Graphplan= Graphplan.proceed
16: REQi = REQi+1
17: end if
18: i=i+1
19: end for
22: Graphplan= Graphplan.Completed
21: return failure
```

The Algorithm 1 gives the composition service model based on the graph plan, the expected result is the service composite that accomplish all specifications given by the client.

Our service composition approach use aspect oriented programming method to solve problems of crosscutting concerns that target services, and design by contract to give order of performing parameters between services.

The composition service model begins with REQstart, which gives input parameters according to the specifications required by the client. , in the composition process graph, each service belongs to a layer level inside the graph where a new REQ of the same level is generated (line 16).

Each service in the level layer will be woven with all crosscutting concerns which are separated and encapsulated in aspects (line 10) with function addAspect given in algorithm 2.

A contract relation is done between the output and input parameters of services in two layer.

The service will itself be inserted afterwards in the graph (line 14) given in algorithm 3.

Algorithm 2. AddAspect

```
INPUT Weaved(service weaved with aspect),
      Si (Service Si), A (Aspect)

OUTPUTSi (Si′; Si weaved with the Aspect A)

1: A.jointput → Si.Cc (Si.Cc is advised jointput)
2: if A.advice related to Si.Cc befor then
3: Si.Cc⤵A.advice (relates A.advice to an
 advised joinpoint)
4: else if A.advice related to S.Cc after
 then Si.Cc⤴A.advice
5: else Si.Cc ⤵⤴A.advice
6: end if
7: end if
8: Si′ =Si ◁ A
```

In AddAspect (algorithme 2) an advised jointput in a service will be weaved with the advice of the aspect (line1).

Since we have working with the aspectJ the advice can be executed before the pointCut (line 2-3) or after (line 4) or around (line 5). In the end we have a new service generated from the weaving.

Algorithm 3. AddService

```
INPUT REQ L (L is the number of the index layer in the GraphPlan)

Si (the new service to be added to the graph plan)

Si' (the product composite service by the previous layer)

OUTPUT REQ L+1(next request), Si″ (the product composite service by the current layer)

1: while (Si.PI ≠Ø) do
2: if Db(@pré, Si'.PO, Si.Pi)) =true then
 Si″ = Si'⊥ Si
3: else if Db(@pro, Si'.PO, Si.Pi)) =true then Si″ = Si'⊦ Si
4: else Si″ = Si'||Si
5: end if
6: end if
7: GraphPlan=GraphPlan ∪ Si
8: end while
9: Si' = Si″
10: L =L ∪ Si″
11: return GraphPlan
```

Algorithm 3 adds a new service Si to the set of services composite which are itself only a single service Si', these two services undergo a contract test based on their output and input parameters. This test defines the way to perform these two services in a given layer of the planning graph belonging to a given request line (line 6-10).

(Line 6) represents a new service composite Si″ that performs the previous service composite Si' followed by the service Si, Si must be completed before Si can start.

(Line 7) represents a new service composite Si″ , that performs unordered between the previous service composite Si' and the service Si, the service Si' followed by the service Si, or Si followed by Si'.

(Line 8) represents a new service composite Si″ that performs the previous services composite Si' and the service Si independently from each other.

At the end, the new service composite generated is included in the set of services L and added to the graph.

IV. An Example

This section, an example is given to better describe the proposed planning graph. Consider for example a basic version of shopping application that consists of the following sequence of tasks: Searching for products, submitting an order, Paying for the order, and shipping of the order (see Fig. 2).

The planning graph is assembled from the uses of these different available services:

- S1 offer the Search service,
- S2 offer the Order service,
- S3 offer the payment service
- S4 offer shipment service

Instead of the client using a single web service for each service they want to achieve (Search service, Order service, payment service, shipment service), it would be better to offer him a single service that meets all these requirements; it is the composition of Web services.

The Web service composition can be mapped to a planning graph (see Fig. 2) as follows:

The Search service, Order service, payment service, shipment service are the four Web services required by the client, which form the set L.

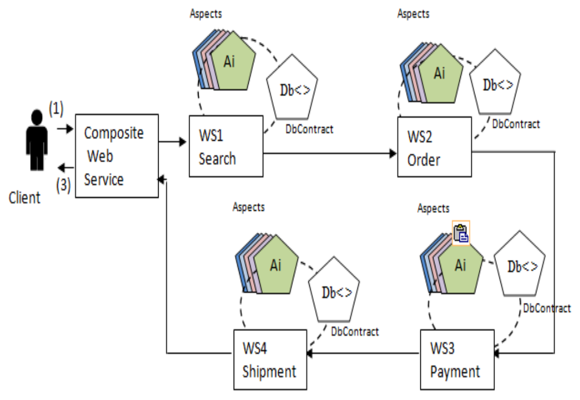

Fig. 2. The Planning Graph of the Example.

If we consider that each of the service mentioned before shares some crosscutting concerns, which will be defined as modules called aspects, cited below:

- Maintaining the history of the client, for future purchase {Aspect1 =History }
- Only authenticate client are allowed to effect the payment service

  {Aspect2 =Authentication}
- Ensuring the confidentiality of the client information about his bank account

  {Aspect3 =Security}
- Accommodate the timing property in order to calculate the time taken by the client to access the Web services, to be sure that the answer to the client request was not long {Aspect4 =Timing}

So we have:

L= {S'= null, S1=Search, S2=Order, S3=Payment, S4=Shipment}

And R= {A1=History, A2=Authentication, A3=Security, A4=Timing}

Supposing that input and output parameters for these services are:

- Search.PI={ProductNumber, DeliveryAddress }, Search.PO={ProductNumber, Product Address }
- Order.PI= {PaymentAmount, PaymentMethod}, Order.PO= {OrderNumber, PaymentAmount}
- Payment.PI ={ProductNumber }, Payment.PO ={PaymentConfirm}
- Shipment.PI= { PaymentConfirm, DeliveryAddress, Product Address, OrderNumber, ShippmentConfirm } , Shipment.PO= {ShippmentConfirm }

And the specification parameters required by the client are: PE = {C1, C2, C3} where:

C1 = ProductAdress ∈ {Europe}

C2= DeliveryAdress ∈ {Europe}

C3= PaymentMethod ∈ {visa, MasterCard}

The first request REQ1 is between S' (he is null because we haven't started the composition of the services yet) and S1 (search article service), the only aspects that crosscut these services are A1and A4 and they crosscut all the services, so we can wove him in the end of the composition processes. Let applying a contract relation:

Db (@Pré, S'.PO, S1.PI) = true

∅∩ {ProductNumber, Product Address} ∈ PE

S´1= S'⊥S1

S1'.PO= {ProductNumber, Product Address ∈ {Europe} }

RES1<L, PI, PO, PE> where , L=L ∪ S1', PO=

In the following request, S`1 will be perform with the service S2 and the contract relation is: Db (@Pré, S`1.PO, S2.PI) = true

{ProductNumber, Product Address ∈ {Europe}}∩ {PaymentAmount, PaymentMethod}, ∈ PE

S'2=S1' ⊥ S2

S2'.PO= { ProductNumber, Product Address ∈ {Europe}, PaymentAmount,

PaymentMethod ∈ {visa, MasterCard}}

S´2= S`1⊥S2 = (S'⊥S1) ⊥ S2

In the next request, S`2 will be performing with the service S3.The Aspect A2 and A3 crosscuts S3 and the contract relation is: Db (@Pré, S`2.PO, S3.PI) ∈ PE

S'3= S`2 ⊥ S3

S'3.PO= { ProductNumber, Product Address ∈ {Europe}, PaymentAmount,

PaymentMethod ∈ {visa, MasterCard}, PaymentConfirm}

S´3= S`2 ⊥ (S3 ◁ A2 ◁ A3)

And in the last request, S`3 will be performing with the service S4.The Aspect A2 and A3 crosscuts S4 and the contract relation is: Db (@Pré, S`3.PO, S4.PI) ∈ PE

S´4 = S´3 ||S4◁ A1, A2, A3, A4

S'4.PO= { ProductNumber, Product Address ∈ {Europe}, PaymentAmount,

PaymentMethod ∈ {visa, MasterCard}, PaymentConfirm, ShippmentConfirm}

This example can be regarded as a woven composition service

S= (((S'⊥S1)⊥ S2) ⊥ (S3◁ A2 ◁ A3))|| (S4◁A2◁ A3))) ◁ A1, A4

Where,

S=<PI, PO, Cc>

PI= {PI.S1, PI.S2, PI.S3, PI.S4}

PO= S4'.PO

Cc= {A1.Cc, A2.Cc, A3.Cc, A4.Cc}

The planning graph of the example is shown in Fig. 3.

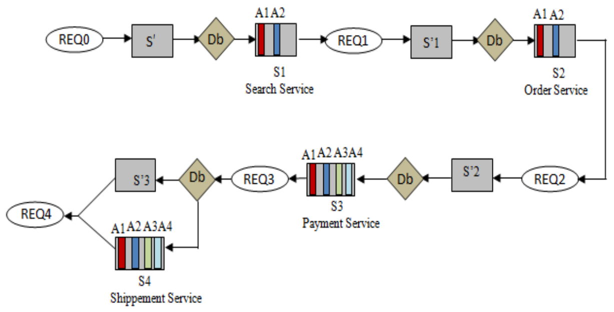

Fig. 3. The Composite Service of the Example.

V. Conclusion

The aim of this paper was the contribution of applying aspect-oriented programming in the web service composition domain with the use of the design by contract.

To this extent, we have proposed and illustrated algorithms based on the construction of a planning graph to eliminate the redundancy of the transversal codes of the crosscutting concerned in the various services belonging to the composition on the one hand and on the other to preclude conflict between parameters of the service composite generate from the web service composition process.

The planning graph using aspect-oriented programming and design by contract was introduced in our work to deliver a precise way to the web services composition without parameter conflict and without code redundancy.

We have shown that the proposed algorithms are suitable for the static detection of resolving conflict situations between parameters of services belonging to the composition.

We have implemented a web service composition prototype with eclipse and AspectJ [6] and a contract for java [14] that resolve conflict detection for each stage of the composition and for each service apart.

Using both AOP and Dbc as a planning graph technical for web services composition will certainly enhance web service composition quality in many ways including:

1) AOP offers better modularization in the web services composition domain, by gathering the crosscutting concerns of services that deals with the same aspect in one module avoiding the redundancy of crosscutting concerns in the composition.

2) AOP offers a consistent implementation in web services composition. Unlike traditional implementations of web services composition which are conspicuous in their inconsistency, AOP provides consistent implementation by having each aspect handled once and used in different web services sat the same time.

3) Moreover, AOP and Dbc are based on the same language and they are reusable and transferable. Therefore, developers don't need to learn more than one language.

4) Using DbC with AOP allows programmers to enforce a Boolean test of contracts and provide guidance in following best practices by creating reusable aspects without conflict and without exception.

We believe that this approach is general enough to be able to be used in all types of web service composition. We intend to use these two approaches together to explore the modeling and detection of constraints adaptation of parameters in the web services composition in our subsequent work.

References

[1] Krisada Sangsanit, Werasak Kurutach, Suronapee Phoomvuthisarn," REST web service composition: A survey of automation and techniques", ICOIN,Vol 1, pp. 116-121, 2018.

[2] Rihab Ben Lamine, Raoudha Ben Jemaaa, Ikram Amous Ben Amora, "Graph Planning Based Composition For Adaptable Semantic Web Services", Procedia Computer Science Elsevier, Vol 112, pp. 358-368, 2017.

[3] Omar Anwer Abdul Hameed, Ahmed Younus, Rasha Hassan Abbas, "Aspect oriented programming: Concepts, characteristics and implementation", Vol 7, pp.2022-2033,2019.

[4] G. A. S. Sheela, and A. Aloysius, "Aspect Oriented Programming - Cognitive Complexity Metric Analysis Tool", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), Vol 3, Issue (1), Jan. –Feb. 2018.

[5] George fairbanks, "Better Code Reviews With Design by Contrac", IEEE, Vol 36, pp.53-56,2019.

[6] AJDT for eclipse tools for AspectJ,accessed [23/07/2020] online available : https://www.eclipse.org/aspectj/.

[7] T. Thomas, I. Schaefer, M. Kuhlemann, "Applying Design by contract to Feature Oriented Programming". FASE, pp. 255-269, 2012.

[8] Anil Kumar, Arvind Kumar, M.Iyyappan, "Applying Separation of Concern for Developing Softwares Using Aspect Oriented Programming Concepts", Procedia Computer Sciecne, Vol 85, pp. 906-914, 2016.

[9] A. charfi, B. Schmeling, A. Heizenreder, M. Mezini, "Secure and Transacted Web Service Compositions with AO4BPEL". In Proceedings of The 2nd International Conference on Service Oriented Computing ICSOC, 2004.

[10] . Shanmuga Priya, K. Rajaram, "AOP Based QoS Monitoring of Dynamic Web Service Compositions". In: Conference: IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), pp. 1913-1917, 2014.

[11] F.Zaimer, M.yutao, H. Keping, P. Gong, "A Requirements-Driven and Aspect-Oriented Approach for Evolution of Web Services Composition". In: Conference: Web Mining and Web-based Application (WMWA), 2009.

[12] Liqiong Chen, Guisheng Fa, Huanhuan Zhang Lizhong Xiao," Petri nets-based method to model and analyse the self-healing web service composition", IJHPCN, Vol 9, 2016.

[13] H. Klaeren, E. Pulvermuller, A. Rashid, A. Speck, "Aspect Composition Applying the Design by Contract Principle". In: Proc of Generative and Component-based software-Enginering Second International Symposium GCSE , pp. 57-69, 2000.

[14] N. Minh Le, "Contracts for java: A practical framework for contract programming". http://code.google.com/p/cofoja/, last access on 08/08/2019.

[15] Charfi, M. Mezini. AO4BPEL: An Aspect-Oriented Extension to BPEL. Springer Netherlands, 309-344, 2007.

[16] M.M.B. Hmida, R. F. Tomaz, V. Monfort. Applying AOPconcepts to increase Web services flexibility", In Proceeding of International Conference on Next Generation Web Services Practices. 2005.

[17] HanineTout, AzzamMourad, Chamseddine Talhib, Hadi Otrok, "AOMD approach for context-adaptable and conflict-free Web services composition", Vol 44, pp. 200-217, 2015.

[18] Bernhard Hollunder, Matthias Herrmann, Andreas H¨ulzenbecher, Design by Contract for Web Services: Architecture, Guidelines, and Mappings", IJAS, Vol 5,2012.

[19] W. Havinga, I. Nagy, L. Bergmans, and M. Aksit. "A graph based approach to modeling and detecting composition conflicts related to introductions". Proceedings of 6th International. Conf. on Aspect-Oriented Software Development, pp. 85-95, March 2007.

[20] Sk. Riazu Rahemana, Hima Bindu Maringanti, Amiya KumarRath. Aspect oriented programs: Issues and perspective. Journal of Electrical Systems and Information Technology. Vol 5(2), 2018.