Distributed aspect-oriented service composition for business compliance governance with public service processes

MingXue Wang, Kosala Yapa Bandara and Claus Pahl
Dublin City University, Ireland
$[mwang \| kyapa \| cpahl]@computing.dcu.ie$

***Abstract*—Service-Oriented Architecture (SOA) offers a technical foundation for Enterprise Application Integration and business collaboration through service-based business components. With increasing process outsourcing and cloud computing, enterprises need process-level integration and collaboration (process-oriented) to quickly launch new business processes for new customers and products. However, business processes that cross organisations' compliance regulation boundaries are still unaddressed. We introduce a distributed aspect-oriented service composition approach, which enables multiple process clients hot-plugging their business compliance models (business rules, fault handling policy, and execution monitor) to BPEL business processes.**

***Keywords*-Web service composition; Process-oriented; Aspect-oriented; Business compliance; Rule; Constraint; Fault handling policy;**

I. Introduction

The business service and process technology can enable the integration of service-based business components for complex business goals. Organisations often have internal (private) business processes, as adopting processes from outside organisations (public) often restricted by their business compliance concerns [15]. However, with demand of borderless enterprises and embracing of cloud computing, the 'process-oriented' paradigm is emerging as a key requirement [15], [17].

A business process describes an automated workflow of an organisation. The organisation as a business entity is obliged to conform to business regulations [21], [14], e.g. business practices, policies, agreements, etc. With private business processes, the organisation is in charge for the process development and deployment. The organisation's regulations will be considered during process development. The process ensures compliance with regulations.

With business process outsourcing and cloud computing [11], [15], organisations are looking for existing external processes (external process providers) to quickly adopt new business and also save costs on process development and maintenance. The same process from a process provider is expected to serve multiple clients, just same as a Web service can be invoked by multiple service clients. We call this process a public business process. It is decoupled from the process clients. However, it is problematic to address business compliance for multiple clients.

Due to the dynamic nature of businesses, their regulations are constantly changing. Frequently updating business regulations is a must for process clients to support quick adaption to real-time business situations. Therefore, to address business compliance restriction within public processes, we define the following key requirements to enable a process-oriented solution:

- Business processes are decoupled from organisation and application, accessible for multiple process clients.
- Business processes can deal with any clients' business regulations and comply with accepted regulations.
- Business regulations can be defined and updated by process clients without changing the business process.

In this paper, we introduce a novel approach - a distributed aspect-oriented service composition architecture, which allows process clients to address business compliance problems (business rule integration, fault handling, execution monitor) on public business process requests. We introduce a BPEL instrumentation mechanism which transforms the original process to an aspectual business process. Combined with a weaving mechanism, we allow clients hot-plugging their business compliance governance models through encapsulation as different aspects. In addition, our approach is BPEL engine-independent for process providers.

This paper is structured as following. In first section, we study the business compliance solution components for private business processes, then analyse the challenge with public process. In section two, we introduce our concept design with AOP and the architecture overview. We detail the prototype design and implementation in section four. Finally, we discuss related works and give conclusions.

II. A business scenario and problem analysis

Business compliance is a major concern today. Current research addresses applying business regulations for business processes [24], [5], [22], [6], [23], [25], though these approaches are designed for private process used in conversional SOA. To leverage these achievements, we have present the problem in different solution components: business rule and fault-handling policy components to manage business regulations; execution monitor component to capture process behaviours for validating business regulations.

Figure 1. Bill payment business process scenario

An example shall be based on a utility bill payment process for multiple process clients (post offices, convenience shops, etc.). Customers pay an amount of money for a bill at any of the process clients. The process clients charge a fee to the utility company for the customer service.

A business process can be modelled in terms of functional and non-functional requirements [21]. This includes capturing a set of business tasks that model the functional behaviour of business requirements, but also non-functional requirements referring to business regulations to be complied to. Figure. 1 shows a public process from a provider, where business regulations have to be addressed for each client. For example, a process client (convenience shop) policy requires a max amount for each payment (3000). Another process client (bank) want to use its own online transfer services.

Business rules A business rule is a statement that defines or constrains some aspect of a business; The intent of a business rule is to control or influence some aspect of a business through the imposition of structure [10]. Business rules formalize business regulations in a rule language [24], [14] and are managed by business rule management systems separately from the application logic. [24], [22] categorise business rules. These different types of rules allow that business regulations within in the safe boundary of the business be expressed as if-then business decisions.

Constraint rules represent assertions that must be satisfied in all states, considered as 'exception rules'. It defines the safe boundary of the business to restrict business behaviours. Violating any constraint indicates a business fault situation, e.g., *If amount of payment $>$ 3000, then violation of constraint.*

Derivation rules are statements of knowledge that is derived from other knowledge by inference or calculation for on-demand needs. For example, for a service fee agreement, *If payment amount $<=$ 500 , then service fee for utility company is 3%, else is fixed to 20.*

Reaction rules trigger actions when certain conditions are met. It defines additional actions needed in specified conditions, e.g., *If a payment more than 1000, then log customer information* is required for a client's policy.

With a private process development, business rule components of an organisation are fixed in the process. This approach only allows rule components integration during process development and is not suitable for a public process. The process needs to be dynamically integratde with rule components of an arbitrary process client.

Fault handling policy integration Faults are defective states of a system and root causes of failures. Unrepaired fault may lead to subsequence failure. For business process, faults are not just technical (runtime) faults, but also business-related faults caused by violations of constraint rules. It is up to the business to decide what remedies need to be applied when the faults occurs, which are defined in a fault handling policy as a type of business regulation for the business outside a safe boundary (constraint rules define the safe boundary). This policy defines the business actions be taken outside the boundary.

With a private process, static remedial strategies defined in the fault handling policy are embedded in the process during development. With a public process, it is impossible to pre-embed any static remedial strategies. Since different process clients may have different remedial strategies for the same fault scenarios. The process is expected to dynamically apply a remedial strategy for its clients.

Execution monitoring Constraint rules are commonly used as safeguards, e.g. QoS constraints for service-level agreements (SLA). In business processwa, these constraints are not only (global) process-level, but also local-level constraints on a single service [4], e.g., high reliability is wanted for the online transfer service. Since Web services are executed in dynamic Web environments, behaviour of service is changes frequently. It has to been monitored at runtime for accurate constraint validation.

Applying monitors on private processes is not difficult. Since public processes are executed at the process provider side, it is impossible to apply the monitors for single services from process clients. The process provider may be able to provide his own monitoring, but trust and accuracy issues arise [18]. Additionally, with multiple process clients, the client needs client-specific and not averaged data. Hence, process clients expect to be able to apply their own monitoring mechanism for any service of the process to give real-time accurate service profile data.

III. A DISTRIBUTED ASPECT-ORIENTED SERVICE COMPOSITION ARCHITECTURE

Aspect-Oriented Programming (AOP) is a programming paradigm that increases modularity by allowing the separation of cross-cutting concerns. An aspect is a modularization of a concern that cuts across multiple objects. Future modifications of these concerns are within the module only. Using AOP, we encapsulate our problems (business rules, fault handling, execution monitoring) as crosscutting concerns of a business process. We allow process clients to implement the aspects as they want to bring the maximum process governability to clients without requiring the process to change with changing aspects.

A. Aspectual business process

Join pont, *Pointcut* and *Advice* are key concepts introduced by AOP to enable modularisation of crosscutting concerns. Join points are well defined points during execution where crosscutting code can be applied, e.g. calling a method or reading a field. A pointcut is declared to specify where and when to apply the advice; it selects a set of join points. An advice is the implementing crosscutting code, applied to a declared pointcut. We apply these concepts in a business

process as follows. To define a join point model of a business process, we break the process down into different basic business elements, which represens the business information of the process.

- *Business activity* is an implementation of a business task. A process contains a set of business tasks invoked in a specific sequence to achieve a complex goal. In service processes, a business activity is accomplished by invoking an operation of a Web service.
- *Data/Business object* is a collection of attributes for a business entity, e.g. order or bill. Data objects, processed by business activities, form the information flow of the process. For example, a data object (Bill) is getting updated via bill request activity in the process.
- *Business faults* are business events where a fault situation occurs during the process execution. This includes both runtime and business aspect faults.

The following table shows basic pointcut declarations for a business process. The signature is used to match a range of services operations, message elements, and faults.

Pointcut	**Description**
Invoke (Web service operation signature)	Select joint points whenever the specified business activity is executed.
Process(message element signature)	Select joint points whenever the specified data object is processed.
Handle (fault signature)	Select joint points whenever the specified business fault is occured.

There are two different types of advice for business activity and data object pointcuts: *Before* and *After*. It tells when to apply the advice, i.e. executing the advice code before or after the pointcut. In other well-known AOP frameworks for Java programming, such as AspectJ [1] and SprintAOP [2], they also include a third advice type: *Around*, which we do not support, but can be achieved by a combination of before and after advice here. For replacing, we allow business services to be replaced in extreme conditions (business faults). This can be achieved with fault-related aspects. In fact, skipping is only function which we can not achieve currently. However, in service processes, the pre-executed Web services generally provide data required as input for subsequent services. Skipping is feasible only very rarely.

B. Architecture overview

With distributed aspect-oriented service composition (Table I left), the process provider offers the business process, where the process client is responsible for aspect design and implementation. The business compliance solution components are encapsulated as aspects of the process. These aspects will be dynamically integrated with the process when the process is requested by the client.

A process instrumentation component is a core component on the provider side to enable this distributed collaboration. The instrumentation transforms the original process to an AOP-enabled BPEL process before deploying it in any execution engine. The instrumented process is able to communicate with the weaving component (weaver) on the client side, which enables dynamic aspect integration.

With this design, the business compliance model is completely separated out from (functional) process workflow. The core concept of our architecture is to enable a *hot plugging* of the business compliance model with the workflow model. There is no interruption to process execution by changing or replacing business compliance models. The architecture supports three different implementation patterns (Table I right) for meeting various business scenario needs:

- *Client driven pattern* The business compliance models are implemented by process clients. Each process client freely defines their own business regulations for the business process. We have detailed this pattern as our business scenario - section 2.
- *Provider driven pattern* The compliance models are implemented by the process provider. The process provider defines regulations for different process clients. For example, in a scenario with internal clients of a large organisation, several compliance models for different regional branches have different regulations.
- *Hybrid (Client&Provider) driven pattern* The business compliance models are implemented by both clients and provider. A sample scenario: additional of organisation self-using, it also provides excrescent computation resource for external customers.

IV. Design and implementation

With distributed AOP on a public process, the process does not know *where* (pointcut), *when* (before, after), and *who* (client) would has *what* (advice). The advice could be everywhere from different clients. One purpose of process instrumentation is exposing every crosscut point of the process to the clients during the execution. The client side's weaver is responsible for matching aspects with these crosscut points. There three different types of crosscut service are instrumented in the original process to expose all crosscut points.

To establishing a communication between the processes and weaver. The process clients need to provide the weaver Web interface. This means in each process request, the client needs to give request message data additional with the weaver interface data (Table below). The serviceReference contains the communication data allows the crosscut services dynamicly invoking the weaver. In case of the serviveReference is empty, or the weaver is inaccessible, the crosscut services will be simply 'ignored'. This means the instrumented processes are still workable for conversional service approach, which do not have compliance requirements and implementations.

In following sub sections, we will detail aspects associate with three defined types of business element through process execution. Section 4.1 details both business activities and data objects are governed by business rules where within

(1) Client driven pattern
(2) Provider driven pattern
(3) Hybrid driven pattern

Table I
ARCHITECTURE OVERVIEW (LEFT) , IMPLEMENTATION PATTERNS (RIGHT)

<utilityBillPayRequest> **Message data:** <bill>...</bill> </utilityBillPayRequest>	<utilityBillPayRequest> **Weaver interface:** <serviceReference>...<serviceReference> **Message data:** <bill>...</bill> </utilityBillPayRequest>
Conversional process request	Process request with business compliance

Figure 2. Instrumentation template of before crosscut service

the safe boundary of the business, additional with service monitor. Section 4.2 details that the fault handling policy takes control of the process when it is outside the safe boundary, i.e., a business fault occurs.

A. Aspects associate with business activity and data object join points

1) BPEL instrumentation with before/after crosscut service: A *before crosscut service* is instrumented in the crosscut point before each business service of the process. It is responsible for before advice related with business activity and data object joint points, which are interested for business rules. Figure. 2 shows the instrumentation template of before crosscut service for each business service of the process.

The before crosscut service takes the current crosscut information and client's weaver interface as input, then sent the crosscut information to the weaver. The crosscut information includes the joint points data (business activity, data object) and advice type data. In this case Figure. 2, the business activity is a bill request service in the Process, the message billRequest is a data object *bill*, and the advice type is before. The response of weaver is a possible updated data object, and a possible list of constraint violations (*violationData*). The violationData is consisted of the current business activity data, and a list of violated constraint types. For a complete list of business constraint type in [25].

A BPEL *Assign* is the first activity after the crosscut service. It copys the updated data object back to the business service request parameter (billRequest), which is ready for the business service (bill request service) execution.

After the *Assign* is a BPEL *If* control structure. It checks if the violation message data is empty, i.e., if any constraint is violated in this crosscut. If it is not empty, the violation data will be copied to a defined BPEL *Exception* (*CosntraintViolation*), and be thrown by a BPEL *Throw* activity - A business fault occurs. We will detail this in later section.

A corresponding *after crosscut service* is instrumented after the business service. It is responsible for after advice with business activity and data object. It is almost same as before crosscut service, except it has after advice type in weaver request.

2) Aspects defining and weaving: One key step in AOP is weaving. After a set of aspects are defined for the target program, the weaving process introduces the advice codes at the captured join points of the target program. With our approach, the weaving is performed at runtime during the process execution by the weaver. With each request from the crosscut service, the weaver matches received crosscut point information with defined aspects' information (point cut, and advice type). If any aspect is matched, the contained advice code will be executed.

Following code are aspect examples show three types of business rule implementations. With our framework, each aspect is implemented as a Java class. The pointcut and advice type metadata of a aspect is defined in Java annotation, which retrieved by Java reflection. An additional XML file defines aspects associate with a process. The advice method has following interface standard. It takes defined point cut (service operation and/or message element) as input, and return an adviceReponse. The adviceReponse includes message element and a set constraint violations.

```
@Aspect
public class BillRequestServcieSecurityConstraint{
  @Pointcut("invoke(billRequest)")
  @Before
  public AdviceResponse checkServcieSecurity(ServiceReference sr){
    // ------  Constraint rule example --------
    //(defrule billRequestService-security
    //  (ServiceReference (url "http://localhost:8080/businessService/
    //                                              utilityBillPayService?wsdl"))
    //  (ServiceReference (operation "billRequest"))
    //  (ServiceProfile {security < 3})
    //   => add (new ConstraintViolationType "SECURITY")) )
    return adviceResponse;  // return maybe include a violation type
   }
}

@Aspect
Public class ServiceFeeAgreement{
  @Pointcut("invoke(billRequest) && process(bill)")
  @After
  public AdviceResponse getServiceFee(ServiceReference sr, Bill bill){
    // --------- Derivation rule example ---------
    //(defrule serviceFee-calculator
    //   ?b <- (Bill) =>
    //   (if (<= ?b.amount 500)
    //       then (add (new ServiceFee (* ?b.amount 0.03)))
    //   else (add (new ServiceFee 20))) )
    bill.setAmount(bill.getAmount - serviceFee);
    return adviceResponse; // return includes a updated bill message
  }
}

@Aspect
Public class LogLargeAmountRule {
  @Pointcut ("invoke(billRequest) && process(bill)")
  @Before
  Public AdviceResponse logLargeAmount(ServiceReference sr, Bill bill){
    // --------- Reaction rule example ----------
    // If bill.amount >= 1000, then call log(bill+currentTime)
    return adviceResponse; // return always empty(null)
  }
}
```

The first aspect example shows a pre-condition constraint rule applied for a business service. The rule is described by Jess rule language, as Jess rule engine [3] is used in our prototype development. If the service fails to meet the minimal constraint requirement, a constraint violation type will be returned. Second aspect is a derivation rule example; the service fee for the payment is deducted. The reaction rule is in lasted example, which triggers a log action.

Although, the last example we have demonstrated the possibility of integrating reaction rule. But the reacted actions

must be executed within client side; it is Not a business activity will be included in the process on the provider side. It is hard to allow reacting a business activity on a public process. Where, in a private business process, possible reacting activities are pre-embedded into the process during the development, by *If* or *Switch* BPEL control structures. For public process, it is unfeasible to pre-embed all business services.

Regarding with execution monitoring, by same strategy, the service client can trigger monitoring action before and after for any Web services. For example, get service performance data by collecting start and stop time.

Following code shows the weaving algorithm for befor/after crosscut service. For each matched aspect, i.e., both pointcut and advice type are marched, the advice method of the aspect will be executed. If the adviceResponse of a executed advice is not null, then the weaver response will be updated. After all matched aspects are executed, the weaver response will be sent back to crosscut service.

```
beforeAfterCrosscutWeaving(weaverRequest){
 weaverResponse.dataObject = weaverRequest.dataObject;
 For each aspect defined for current process
  If (pointcutMatching(aspect.pointcut, weaverRequest.pointcut) &&
              adviceTypeMatching(aspect.adviceType, weaverRequest.adviceType))
   adviceReponse = aspect.advice(pointcut);
  end if
  If adviceReponse.dataObject !=null
   MessageUpdate(weaverResponse.dataObject, adviceReponse.dataObject);
  end if
  If adviceReponse.violationData != null
   addNewViolationType(weaverResponse.violationData,adviceReponse.violationType);
  end if
 end for
 return weaverResponse
}
```

B. Aspects associate with business fault join points

1) BPEL instrumentation with handler crosscut service: The *handler crosscut service* is quite different with before/after crosscut service. It is instrumented inside a BPEL fault handler *Catch* for exposing the crosscut point, where a business fault (constraintViolation) occurs (Figure. 3). The BPEL fault handler is responsible for catching the fault, which is interested for the fault handling policy.

Figure 3. Partial instrument template of handler crosscut service

Unlike the before/after crosscut service, the handler crosscut service requires receive complex actions rather than updated message and violation data. This because that various remedial strategies defined by a fault handling policy are required to be applied directly on process execution. In this case, the response of weaver for handler crosscut service is a selected remedial strategy. A complete instrumentation temple is able to apply any return remedial strategy on the process execution, we have shown this in our previous work [25]. In this work, we only offer three types of remedial strategy: *Ignore* the fault, *Retry* the fault service, *Replace* the fault service with an alternative service. In [25], we also include a *Recompose* strategy, which discard the current fault process and establish an alternative process. However, this is not available in case, as a public process is shared with other process clients.

2) Aspects defining and weaving: The aspect is also defined as a Java class (see below). In following example, the declared pointcut matches all types of faults, as we want all faults be handled. The advice method takes defined joint point (constraint violations) as input. It searches the fault handling policy and fault log database, returns a remedialStrategy for the type of constraint violation.

```
@Aspect
Public class FaultHandlingPolicyIntergation{
 @Pointcut ("handle(*)")
 Public remedialStrategy findARemedialStrategy(ServiceReference sr,
                                        conditionType ct, ViolationType vt){
   Default remedialStrategy = Ignore;
   Search fault policy and execution log
   return remedialStrategy;  //return a remedial strategy
 }
}
```

The aspect will be executed for each type of constraint violation. In case of more than one constraint are violated, i.e., the violationData includes more than one constraint violation type. The only most severity remedial strategy (Replace>Retry>Ignore) will be return by weaver finally. Below is the weaving algorithm for the handler crosscut service.

```
HandlerCrosscutWeaving(weaverRequest){
  Initial remedialStrategySet;
  For each violation in weaverRequest.ViolationData
    For each aspect defined for current process
      If pointcutMatching(aspect.pointcut, violation)
        remedialStrategy = aspect.advice(pointcut);
      end If
      remedialStrategySet.add (remedialStrategy);
    end For
  Return getMostSeverityRemedialstrategy(remedialStrategeSet);
}
```

V. Related work

Work in the context of business compliance mainly addresses the problems of private business processes. Although the ultimate goal of our work is different to the work we compare to, but we believe this provides considerable useful information from technical implementation aspects.

Business rules and fault handling [22] provide a distributed business rules architecture - VIDRE - implementing a service interface (following the JSR 94 standard) to allow a process client to manage the business rule remotely. It does not allow adding business rules to an arbitrary business process. [6], [23] allow developers to define fault handling policies. First, this implementation require a customized BPEL engine to enable this feature. Second, how multiple policies work for one process has not be addressed.

SOA and AOP [5], [9] implement AOP extensions for the ActiveBPEL engine, which offer a framework (Dynamo) to monitor constraints. The fault handling policies are required to be defined in a specific recovery language. However, the process requires be recompiled for any rule or policy change. [12], [13] implement an AOP extension with IBM's BPWS4J engine. The advice in the approach (AO4BPEL) is a BPEL activity defined by BPEL-like XML document. This work focuses on dynamic changes of the process workflow by adding/deleting BPEL actives (aspects) rather than business regulation centric concerns. In [20], an AOP feature is implemented for three BPEL engines through different engine adaptors, focussing on enhancing process

performance (service adaption) and flexibility (message mediator) specifically with regard to partner service interaction.

In the above AOP work, AOP features are implemented at BPEL engine level, limited to a particular BPEL engine. As our approach is designed on top of Web service and BPEL standards, we do not have such limitations. Different BPEL engine deployment descriptor generators are, however, required. In these works, aspects are only available be implemented at the provider side. In our approach, aspects of business compliance models can be implemented by both client and provider. Hence, our architecture supports three different implementation patterns (section 3).

VI. Conclusions

Business compliance problems in public business processes need a flexbile, process-oriented soluation. We have introduced a distributed AOP architecture that decouples the process from the organisation, enabling the process to work for multiple organisations' business compliance requirements. We have presented a BPEL instrumentation and a weaving mechanism, which allows distributed and platform-independent architectures to be implemented.

Our future work includes improving the prototype to support rich pointcut language. It includes more complex relation operators for join points, such as, Not Equal(!=), Or(||), etc. We currently only support the And (&&) relation. Also, the performance overhead of aspects with dynamic weaving still requires further study, although, this may be different for different scenarios.

Acknowledgment

The authors would like to thank the Science Foundation Ireland for their support for the CASCAR project.

References

[1] AspectJ from http://www.eclipse.org/aspectj/.

[2] Johnson, R., J. Hoeller, et al. Spring Framework from http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/.

[3] Jess, the Rule Engine for the Java Platform from http://www.jessrules.com/.

[4] Alrifai, M. and T. Risse Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition 18th International World Wide Web Conference, 2009.

[5] Baresi, L. and S. Guinea Dynamo: Dynamic Monitoring of WS-BPEL Processes 3rd International Conference In Service Oriented Computing, 2005.

[6] Baresi, L. and S. Guinea A Dynamic and Reactive Approach to the Supervision of BPEL Processes 1st India Software Engineering Conference, 2008.

[7] Daniel, F., F. Casati, et al. Business Compliance Governance in Service-Oriented Architectures International Conference on Advanced Information Networking and Applications, 2009.

[8] Taveter, K. and G. Wagner Agent-Oriented Enterprise Modeling Based on Business Rules the 20th International Conference on Conceptual Modeling: Conceptual Modeling, 2001.

[9] Baresi, L., S. Guinea, et al Self-Healing BPEL Processes with Dynamo and the JBoss Rule Engine International workshop on Engineering of software services for pervasive environments, 2009.

[10] Byrne, B. and G. Imirzian IBM Industry Models and ILOG business rules management systems, Part 1: Define business rules using IBM Industry Models from http://www.ibm.com/developerworks/data/library/techarticle/dm-0809byrne/index.html, 2008.

[11] Casati, F. Industry Trends in Business Process Management: Getting Ready for Prime Time 16th International Workshop on Database and Expert Systems Applications, 2005.

[12] Charfi, A. and M. Mezini AO4BPEL: An Aspect-oriented Extension to BPEL World Wide Web Journal, 2007.

[13] Charfi, A., B. Schmeling, et al. Transactional BPEL Processes with AO4BPEL Aspects Fifth European Conference on Web Services, 2007.

[14] Daniel, F., F. Casati, et al. Business Compliance Governance in Service-Oriented Architectures International Conference on Advanced Information Networking and Applications, 2009.

[15] Fingar, P. Cloud Computing and the Promise of On-Demand Business Innovation." Intelligent enterprise. from http://intelligent-enterprise.informationweek.com/print_article.jhtml;jsessionid=I0D53X4L1J0RRQE1GHPCKHWATMY32JVN?articleID=218500039, 2009.

[16] Gaur, H. and M. Zirn BPEL Cookbook Best Practices for SOA-based integration and composite applications development Packt Publishing, 2006.

[17] Hoyer, V., E. Bucherer, et al. Collaborative e-Business Process Modelling: Transforming Private EPC to Public BPMN Business Process Models 10th International Conference on Business Information Systems, 2007.

[18] Jurca, R., W. Binder, et al. Reliable QoS Monitoring Based on Client Feedback 16th International World Wide Web Conference, 2007.

[19] Lesiecki, N. Improve modularity with aspect-oriented programming from http://www.ibm.com/developerworks/java/library/j-aspectj/, 2002.

[20] Moser, O., F. Rosenberg, et al. Non-intrusive monitoring and service adaptation for WS-BPEL 17th international conference on World Wide Web, 2008.

[21] Pavlovski, C. J. and J. Zou Non-functional requirements in business process modeling 5th on Asia-Pacific conference on conceptual modelling, 2008.

[22] Rosenberg, F., C. Nagl, et al. Applying Distributed Business Rules - The VIDRE Approach IEEE International Conference on Services Computing, 2006.

[23] Subramanian, S., P. Thiran, et al. On the Enhancement of BPEL Engines for Self-Healing Composite Web Services International Symposium on Applications and the Internet, 2008.

[24] Taveter, K. and G. Wagner Agent-Oriented Enterprise Modeling Based on Business Rules the 20th International Conference on Conceptual Modeling: Conceptual Modeling, 2001.

[25] Wang, M., K. Y. Bandara, et al. Integrated Constraint Violation Handling for Dynamic Service Composition SCC, 2009.