

# Process as a Service - Distributed Multi-tenant Policy-based Process Runtime Governance

MingXue Wang, Kosala Yapa Bandara and Claus Pahl  
School of Computing, Dublin City University  
Dublin 9, Ireland  
[mwang|kyapa|cpahl]@computing.dcu.ie

**Abstract**—With the emergence of Business Process Outsourcing and Cloud Computing, enterprises are looking for available business processes outside of their organizations to quickly adopt to new business requirements and also reduce process development and maintenance costs. The process execution needs to be governed as policy enforcement might differ between different clients. Since a process is deployed outside of the organizations and serves multiple process clients, distribution and multi-tenancy have become two requirements for runtime governance of service processes.

We address this problem by introducing a policy-oriented aspectual business process framework. The runtime governance from process clients are integrated as aspects through dynamic weaving into process execution.

**Keywords**-business process; runtime policy; runtime governance; cloud; AOP; process-oriented architecture

## I. INTRODUCTION

Service-oriented architecture (SOA) has become the standard for enterprise application development and integration. Service processes based on WS-BPEL orchestrate service-based business components as workflows to accomplishing complex goals [1], [2]. On the business side, policies are created throughout the enterprise to establish 'best practices' or standards, meet environmental or regulatory requirements, to increase efficiency and to help streamline business processes [3]. SOA governance defines and enforces the policies that are needed to manage a SOA accordingly [4], [3]. Based on the service development lifecycle, business process governance can be broken into design-time and runtime governance [4]. We aim at adopting existing processes, thus focusing on runtime policies and governance.

With the emergence of Business Process Outsourcing and Cloud Computing, enterprises are looking to adopt existing processes to quickly take on new business opportunities and also save cost on process development and maintenance [5], [6], [7]. Process level collaboration has been becoming a necessity for enterprises - c.f. the *process as a service* notion for the cloud. However, simple service request/response approaches in conventional SOA [8], [9] do not work for process-level collaboration. A business workflow as an automation of an organization's process is obligatorily enforced by organizations through the process

clients' policies. As a consequence, we note the following as a need for process services:

$$\text{process request} = (\text{service}) \text{ request} + \text{runtime governance}$$

While SOA governance is only starting to become mainstream practice, the advent of the cloud could reimpose the governance challenge. "Once we start using services from the cloud or putting services out on the cloud, it's going to add another complicated layer to what we're dealing with. We're not even starting to be ready to deal with it" [10]. To take this challenge, we first analyse the problem: identifying the unique governance requirements and features for processes in the new cloud environments compared to conventional SOA:

- 1) The process is deployed outside the organization/process client; it is executed by a process provider in the cloud.
- 2) The process is decoupled from the organization; it serves multiple process clients.

With above, two new requirements for runtime governance can be identified:

- 1) *Distributed* – the process allows itself to be governed by process clients remotely.
- 2) *Multi-tenant* – the process allows each client to govern its own process request instances without interfering with other clients.

We address above problems by introducing a policy-oriented aspectual business processes concept. We apply the AOP paradigm for distributed multi-tenant process runtime governance. We develop abstract policy function components in a business process as crosscuts, where the policy-based governance as crosscutting concerns can be applied as aspects. We outline our concept design in section 2. In sections 3 and 4, we introduce policy categories for runtime governance and provide abstract policy component designs for each policy category. We show an aspects and weaving design in section 5. The prototype and evaluation results are presented in section 6. At the end, we compare our approach with related work and give conclusions.

## II. AN AOP FRAMEWORK FOR POLICY-BASED RUNTIME GOVERNANCE

### A. AOP background

Aspect-oriented programming (AOP) is a programming paradigm that increases modularity by allowing the separation of crosscutting concerns. An aspect is a modularization of a concern that cuts across multiple objects. *Join point*, *Pointcut* and *Advice* are key concepts introduced by AOP. 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 a collection of related join points. An advice is the implementing crosscutting code, applied to a declared pointcut. The type of advice (such as before/after) indicates when to apply it. Central to AOP is weaving, which introduces the advice code at the captured join points of the target program. Regarding BPEL processes, AOP concept has been adopted for BPEL to support dynamic changes of business processes [11], [12].

### B. Policy-oriented aspects

We apply the AOP concept for policy-based runtime governance of business processes. The process development and runtime governance module development is separated for process provider and clients (Figure 1). The provider focuses on capturing a set of business tasks that model the functional behavior of the process workflow. E.g., an order/check out process includes payment, post services, etc tasks. The policy-based runtime governance as non-functional requirements are applied to the business process as aspects. In this case, the non-functional requirements are not limited to quality concerns, but also the functional behavior configuration (e.g. options of the post method), compliance, and recovery – which are all concerned with the policies – can be included.

- *Policy (function) components* are Web services of business processes, as connectors between processes and policy engines as implementations. They allow a business process, given an input, to provide an output based on the policy to govern the process execution.
- *Crosscuts* are available points crosscut a process workflow where policies can be executed. E.g., A trust constraint policy involves a crosscut before a payment service. When policy components of a business process have been identified, they need to be developed or instrumented in the process before deployment. These integrated policy components are abstract policy services as crosscuts of the process workflow. These crosscuts are where the possible concrete policy function implementations can be executed in the process runtime. This transforms the business process to a policy-oriented aspectual BPEL process where the policy-based runtime governance is considered as crosscutting concerns of process execution.

- *Join points* The business workflow can be broken into three types of policy-related process elements (business activity, data/business object, business fault) to model the join points. All crosscuts are associated with join points, which give identities to crosscuts.
- *Pointcuts* These are predicates that match join points. They allow any crosscut of a business process to be queried to apply concrete policy implementations. E.g., to apply the trust payment constraint can query the crosscut associated with the payment service. Table I shows the basic pointcut declarations.
- *Advices* are implementations of the abstract policy components. They are implemented by process clients and provide concrete policy functions by executing the code. E.g., A code decides if the payment service is met the trust policy requirement.
- *Aspects* are packages coupling specified abstract policy components (pointcuts) with the concrete policy implementations (advices). For each client, the valid range of its aspects are only process instances that are created for its own requests.
- *Weaving* is matching of aspects with integrated abstract policy components of a process during process execution. Once the pointcut of an aspect is matched with the identity (weavingRequest) of an abstract policy component, the advice of the aspect will be executed. The responses of all executed advices (weavingResponse) will be returned by the abstract policy component.

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

Table I  
BASIC POINTCUT DECLARATIONS

In following section, we first identify different categories of business policy, then define abstract policy components for these policies, i.e. determining all crosscuts of a business process.

## III. A CLASSIFICATION OF BUSINESS POLICY

Business policies are widely implemented in business process in industry, especially in finance and insurance sectors. However, business policies change frequently and policies that apply to multiple services and processes can introduce redundancy and inconsistency within service logic and contracts [13]. Hence, runtime policies have been separated from processes in development, formalized as business rules [14]. They express the policies as business decisions and are centralized in a rule management system. Rule or policy components are integrated in the processes as runtime

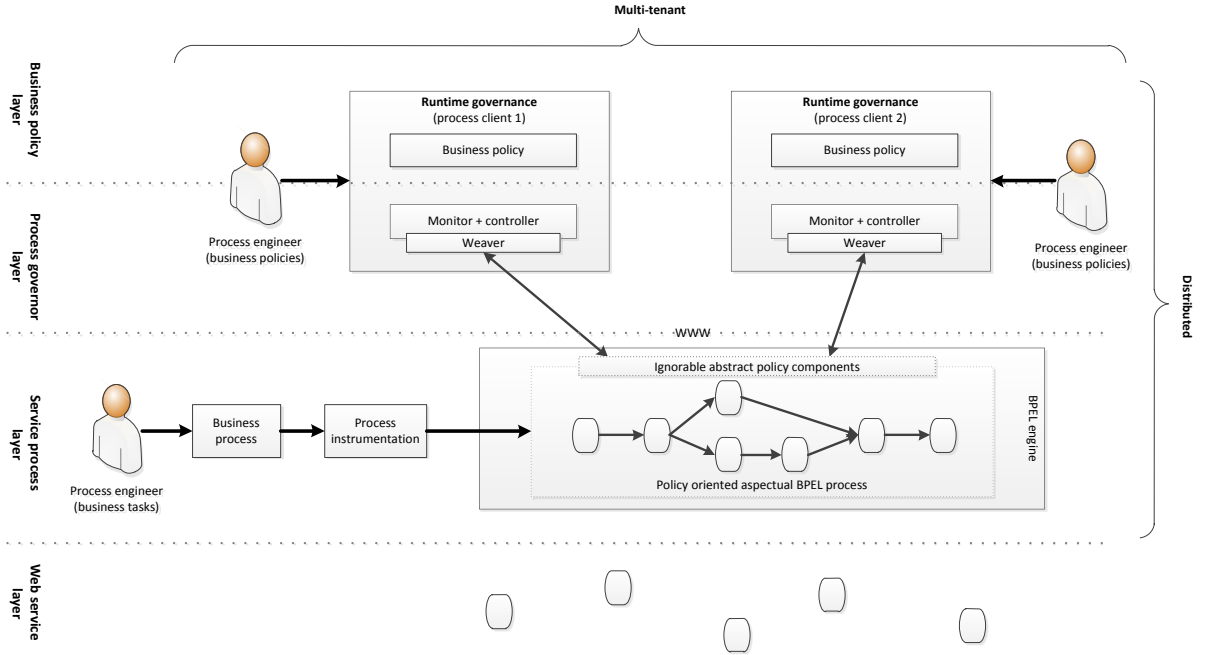


Figure 1. Distributed multi-tenant service process runtime governance

governance for policy enforcement. All processes can be automatically updated by changing business rules in one central location without redeveloping and redeploing the processes.

In current research, business rules have been categorised into three types [15]: integration rule, derivation rule and reaction rule. While this shows different types of formal expression of the business policies, it does not give any concrete meaning to a business process. It is hard to establish a common connection between policies and the process. Based on the action areas of the process execution, we categorise the policies for runtime governance in three different categories:

- 1) *Within the safe boundary* – this business policy category expresses the business decisions within the safe boundary of the process execution. The business steps continue after the decisions are made. It is used to specify variable business decision logic for various expected business scenarios such as different customer types, different types of post method use or frequently changing agreements (e.g. different discount rate over times). The business dynamics is the driving force.
- 2) *On the safe boundary* – this business policy category defines the safe boundary of the process execution to restrict business behavior. Policies are expressed as integration or constraint rules to specify assertions that must be satisfied in all states of the process execution such as Services Level Agreements (SLA). This rules make sure the business complies with regulations.
- 3) *Outside the safe boundary* – this type of business policy defines the business reaction when the process

crosses the safe boundary, i.e., the constraints are violated. The business needs to decide what remedial strategy is required to avoid potential subsequential failure of the business goal. Since the constraint violations are viewed as 'faults' of process executions, this policy category is also known as fault policy. Examples are Oracle's fault policy management [16].

Policy components for different policy categories are used for different governing actions. Based on [17], [18], [19], [20], [21], [14], we can correlate process governing actions – see table II. For these different governing actions, policy components have different interface and can be integrated at different locations of the process workflow. These interfaces and locations will be identified in next section.

Policy category	Correlated governing actions
Within the safe boundary	Business flow and data control
On the safe boundary	Constraint validation
Outside the safe boundary	Remedy determination

Table II  
CONNECTIONS BETWEEN BUSINESS POLICY AND PROCESS

#### IV. INTEGRATING ABSTRACT POLICY COMPONENTS INTO BUSINESS PROCESSES

With the process as a service notion, the provider expects the process to be available serviceable for more process clients. Hence, the maximum governability is an important goal in process design. More policies can be applied to the process to meet the various requirements of clients – for example, adding a policy component (getPostage) rather than assuming the postage is fixed by the courier as a client may

offer free postage for its customers. While we are not going to discuss process design here, we identify different types of policy component and integration in our approach. We discuss how and where they can be integrated in a business process for different policy categories.

#### A. Ignorable abstract policy components

All policy components integrated in a business process are 'ignorable' and 'abstract'. Ignorable means the policy component has a default output. It will not halt the process execution, if the client does not have a concrete implementation for it. Abstract means the real output is based on the policies, which are implemented and executed by the client. These integrated abstract components are key to enable client-side runtime governance. The following shows the basic logic inside the abstract policy component. During process execution, each encountered abstract policy component sends the current crosscut information weaving (Request) to the weaver of the client and waits for output in (weavingResponse).

```

Output = default value;
//try to get the result from the client
weavingResponse = call weaving(weavingRequest);
if weavingResponse != empty then
    output = weavingResponse;
return output

```

The 'weavingRequest' is a complexType, which consists of following information: processReference - current process identity (includes a unique process instance id created for each process request). serviceReference - identity of the associated business activity service. dataObject - input/output SOAP message of the business service. adviceType - advice type (before/after/replace/handle). violationData - details of constraint violations. The 'weavingResponse' consists of following: dataObject - input/output SOAP message of the business service. violationData - details of constraint violations. remedialStrategy - selected remedial strategy to be applied to current process.

However, not all the above information has to be provided and be the same in each weaving call. Depending on the governing actions used for different policy categories, we have identified three types of abstract policy components (see Table III).

#### B. Policies within the safe boundary

The policy within the safe boundary is able to control both business flow and data of the process. Flow control is made by branch selection, the different business actions as different branches are developed in process workflows for possible results of the business decision. The policy component as decisions point (e.g. getPostMethod in Figure 2) is integrated into the workflow before a control flow structure, such as BPEL-if or switch structures. The corresponding branch is selected after is decision is made based on the policy – for example *if the item value is less than 20 euro,*

*then delivery as normal post, else delivery as registered post.* This type of policy component is designed and integrated in the process by the process developer. The default value is also provided by the developer. Still, the default value must be careful chosen for loop structures, as infinite loops can be caused.

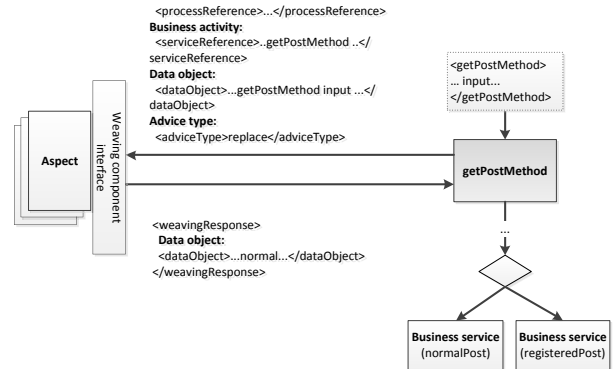


Figure 2. Policy component as business activity

There are two different scenarios with data control. The first scenario is the same as flow control. The policy components (e.g. getPostage) are designed and integrated by the process developer, e.g., *if the item value is more than 100 euro, then the postage is free.* The default value is also provided by the developer. In this example, it might be the standard postage from courier.

There are two types of abstract policy component integration for a business process. The above two examples (getPostMethod and getPostage) are of the first type, where the policy component itself is a business activity, which is designed and integrated by the developer (first row of Table III). The serviceReference.operation of weavingRequest is getPostMethod. The AdviceType is replace, since it is itself a business activity. The expected information of weavingResponse is a dataObject – a decided post method or a postage fee. The default value of this type policy component is assigned by the developer.

The second category of policy components are instrumented in the business process for other business activities. In the second scenario with data control, the policy component could be applied at any point of the process data flow (i.e., before and after each business service) to modify the business data object passing through the component. This could be used to apply the policy which was not considered during the process development. For example, with a policy made for a business promotion: *if the payment amount is more than 200 euro, then give 5% discount on the payment.* The process client could apply this policy by modifying the amount value before the payment business service is executed. In this case, the serviceReference.operation of

Integration	Governing actions	weavingRequest	Expected weavingResponse	Associated join points
By process developer (e.g. getPostMethod service)	Business flow and data control	processReference serviceReference dataObject adviceType=replace	dataObject	business activity data object
Instrumentation (before/after service)	Business data control and constraint validation	processReference serviceReference dataObject adviceType=before/after	dataObject violationData	business activity data object
Instrumentation (handler service)	Remedy determination	violationData adviceType=handle	remedialStrategy	business fault

Table III  
THREE TYPES OF ABSTRACT POLICY COMPONENT

weavingRequest is the payment. The dataObject is the input message of the payment service, which is also the default value of the output. The adviceType is before. The expected weavingResponse is a dataObject – the modified payment amount.

### C. Policies on the safe boundary

Policies on the business safe boundary can be divided as two types of constraints for each business activity of the process – pre-condition and post-condition constraints. The pre-condition validation is inserted before each business service (adviceType=before); the post-condition is inserted after each business service (adviceType=after). Constraints can be made for the business data which the business activity processes, e.g., (*if the amount payment > 5000 euro, then violation of syntax constraint.*), but also as properties of business activity profiles, e.g., (*if the trust of the payment service < 3, then violation of trust constraint.*). In this case, the expected weavingResponse is a violationData, which might contain a set of violation types or is empty. If it is empty, then the payment service will be executed; otherwise the policy component should throw the violation as a business fault. The default value is an empty violation variable.

Since we can see two types of abstract policy components appearing twice in same location (second scenario of data control and constraint validation) before each business activity. We could merge two components as one *before-crosscut service* (see second row of Table III and Figure 3) to reduce the number of abstract policy components in a process. The idea is that the violations (violationData) will not be thrown by the policy component itself. A following BPEL if structure checks if the violationData is empty. If it is not empty, the violation data is copied into a defined BPEL exception (constraintViolation) and is thrown by a BPEL throw activity.

A corresponding *after-crosscut service* is also instrumented after each business activity. It is almost the same as the before-crosscut service, except it has an after-advice type in the weavingRequest. It is used for business service output modification and post-condition validation. Both before/after

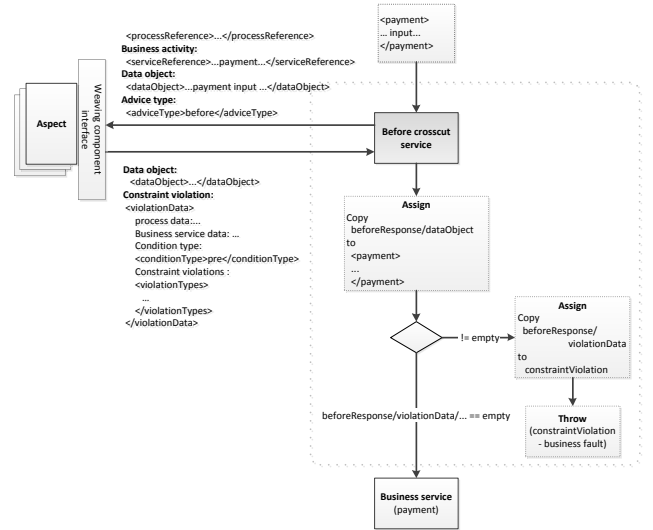


Figure 3. Policy component before business activity

services can also be used to capture the temporary data required for the working memory of the rule engine.

### D. Policies outside the safe boundary

When the constraintViolation exception is thrown, it indicates that process execution has crossed the safe business boundary. In this case, the fault policy is in charge of the process governance. "A critical dimension of your SOA Governance model is how you will anticipate and deal with governance exceptions. Exceptions to your governance model are not only to be expected, but they are important for evolving your policies and overall governance model" [4]. The abstract policy component is inside a BPEL catch handler, which is responsible for catching the exception (third row of the table and Figure 4). The fault policy specifies a remedial strategy for the violation. For example, *if trust pre-condition violation with payment service, then replace with payment-2 service.* The input of this policy component is the constraintViolation caught by the BPEL

fault handler; the output is a remedial strategy based on the fault policy. Depending on the systems, various remedial strategies are supported. We briefly introduce four remedial strategies supported by our prototype. (*Ignore* the fault, *Retry* the fault business service, *Replace* the fault service with an alternative service. *Abort* the current process execution.) The default value is *abort*. The advice type is *handle*.

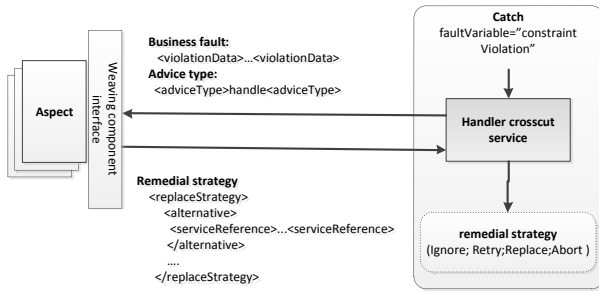


Figure 4. Policy component for business fault

## V. RUNTIME GOVERNANCE BY ASPECT

During process execution, each encountered abstract policy component sends the current crosscut identity information (weaverRequest) to the weaver. The weaver is responsible for matching aspects (pointcut and advice type) with the crosscut identity. To allow distributed runtime governance, the weaver is deployed on the client side and has a service interface, which is able to establish a communication with the process. To allow multi-tenant runtime governance, the process clients provide the weaver interface reference to govern their process instance for each request. In each process request, it includes message data (service request) additionally to the weaver interface (for runtime governance), see Table IV. The serviceReference contains the weaver endpoint data that is assigned for each encountered abstract policy component of the process instance, to allow the abstract policy components to dynamically invoke the weaver. In case the weaver is inaccessible, the abstract policy components are 'ignored' and the default value is returned.

<pre>&lt;order&gt;   Message data:   &lt;item&gt;...&lt;/item&gt;   ... &lt;/order&gt;</pre>	<pre>&lt;order&gt;   &lt;clientReference&gt;     Weaver interface:     &lt;serviceReference&gt;...&lt;/serviceReference&gt;     ...   &lt;/clientReference&gt;   Message data:   &lt;item&gt;...&lt;/item&gt;   ... &lt;/order&gt;</pre>
Conventional service request	Process request with runtime governance

Table IV  
PROCESS REQUEST WITH RUNTIME GOVERNANCE

We present an aspect code example applied to Figure 3 for constraint validation. With our framework, aspects are

implemented as Java classes. The pointcut and advice type metadata of an aspect are defined as Java annotation and retrieved by Java reflection. An additional XML file defines aspects' class name and association with business processes. These aspect classes will be dynamically loaded at runtime. The advice method has the following interface standard. It takes elements of weaverRequest (process reference, service reference, message element, violation) as input and returns a weaverResponse - a response of advice. For each matched aspect, the advice code will be executed and the returned advice response is updated to the final weaverResponse. After all matched aspects are executed, the final weaverResponse is sent back to the abstract policy component.

```
<process name="Order Process" operation="orderOperation" ... >
  <aspect className="onSafeBoundary.PaymentConstraint">
    <aspect ...
  </process>
```

```
@Aspect
public class paymentConstraint{

  @Pointcut(" invoke (payment)")
  @Before
  public WeaverResponse checkTrust(ServiceReference sr){

    // ----- Constraint by Jess rule -----
    //(defrule paymentService-trust
    // (ServiceReference
    //   (endpointUrl "http://localhost:8080/OrderE.../PaymentService?wsdl")
    // (ServiceReference (operation "payment"))
    // (ServiceProfile {trust < 3})
    // => add (new ConstraintViolationType "TRUST") )

    // Fire rule engine, return maybe include a violation type
    engine.run();
    Iterator currencies = engine.getObjects(
      new Filter.ByClass(ConstraintViolationType.class));
    // add the violation into weaverResponse
    return weaverResponse;
  }

  @Pointcut(" invoke (payment)&&process (amount)")
  @Before
  public WeaverResponse checkMaxAmount(ServiceReference sr, double amount){
    ...
  }
}
```

With aspect associated to fault join points and in case more than one constraint is violated, then the violationData includes more than one constraint violation type. The advices may return a bag of remedial strategies. But only one most severe remedial strategy (*Abort*>*Replace*>*Retry*>*Ignore*) is returned by the weaver and is applied to the business process.

## VI. IMPLEMENTATION AND EVALUATION

### A. Prototype

In previous work, we have developed an XML-based fault policy for handling constraint violations [19]. A BPEL instrumentation template can be applied to the four types of remedial strategy for the process. However, there are modifications to support the presented approach here, such as enabling instrumented services to communicate with the weaver, etc. The Jess rule engine (<http://www.jessrules.com/>) is used on the client side to develop a number of business rules for experiment. Additionally, some aspects are also developed as interceptors of process flow to collect data (e.g.

performance) for compliance monitoring purposes by using the before/after crosscut services. The ActiveBPEL engine (<http://www.activevos.com/>) is used for BPEL deployment. Our approach itself is not limited to any vendor-specific BPEL engine. Since we use XPath to handle XML messages in the BPEL process, BPEL engines with limited XPath support might be less suitable.

### B. Evaluation

An business process (the order/check out process information is presented in section 4) was developed as a service of a process provider. Two process clients (at the same machine, but with different weaver endpointUrls) have also been developed with a number of business policies that cover the three policy categories. The evaluation is focussed on runtime governance effectiveness and on dynamic weaving performance overhead.

*Effectiveness evaluation* We designed 24 test cases for each client. Each test case includes process input, aspects for governance, expected process execution flow by governance and expected output by governance. Based on an analysis of process outputs and process execution logs, we verified that all test cases are successful, i.e., our approach provides the distributed multi-tenant process runtime governance.

*Performance overhead evaluation* During process execution, each encountered ignorable abstract policy component needs to communicate with the weaver, which causes performance overhead. This can vary in different scenarios, e.g., large numbers of aspects defined could cause longer matching time or poor network speed could cause weaving delays. In our experiment scenario, the average time for a weaving call (in total 35 aspect methods) is an acceptable 174 ms.

## VII. RELATED WORK

How business process can be delivered in cloud environment is gaining an attention in academia recently. The ongoing Cafe project [22], [23] has proposed a end user application for the Software as a service (SaaS) approach. BPEL process and runtime governance is offered in a Web-based application. Each registered user (process client) is able to define his own business policy (SLA value in that case) as configuration data of the business process customization. Separate configuration data stored at the process provider is used to govern the process by the need of each separate client. While we propose a different approach, the process runtime governance is exposed as an API (weaving) to allow client-side control. We can identify four aspects of process governance of our solution that overcome limitations of the state-of-the-art.

*A pre-registered account is required for all process clients* To create separate configuration data for a process, each process client must be a registered user to keep a unique account ID in the configuration database. This registration

procedure ceases the open accessibility of the business process. In advanced scenarios, dynamic process discovery is required and invocation becomes impossible.

*Policy enforcement completely relies on process providers* After a process client sets the policy in the configuration database, the policy enforcement completely relies on the process provider, as the process client has no governability of the process anymore. This means the process provider has to be fully trusted without a satisfactory verification preferred by many business [24]. In our approach, the policy compliance is possible to be verified by process clients themselves.

*Business policy centralization and reuse* We introduced business policy centralized in a management system to avoid redundancy and inconsistency problems for policies with multiple service and process. However, saving policies at process providers reintroduces this problem. Especially, a company may have many different providers for different processes. In addition, each process provider may have different policy or rule specifications with configuration databases. This means that existing business rules of process clients might need to be reformatted before submitting to the process providers.

*Concerns over confidentiality of policies* Some business policies contain confidential information, which might concern the competitive advantage of a company. Storing these policies with process providers, i.e., outside the organisation, raises confidentiality concerns. Recent reports have shown confidential data being leaked from several process providers [25]. Also, the business partner relationships might change in dynamic business situations. A process provider as a business partner may be trusted today, but might become a competitor tomorrow. Some business policies may be restricted to be exposed to process providers.

Despite these benefits of our approach, some limitations remain. Beside performance overhead caused by abstract policy components, our approach does not support the functional service customization as e.g. the Cafe project, i.e., specifying a Web service for a business activity. We only allow a business service to be replaced in an exceptional condition (business fault) by the replace remedial strategy. However, the ignorable abstract component design technique could be used for business services to support functional customization, but this also will cause additional performance overhead as discussed.

## VIII. CONCLUSIONS

Business process management needs to automate policy enforcement and compliance monitoring as runtime governance. However, with process delivery in cloud environments, runtime governance needs a distributed, multi-tenant solution. We have introduced a policy-oriented aspectual business process framework that enables the clients to enforce their policies on process instances remotely. We have

given a classification for business policies and presented abstract policy component integration for each policy category. We also showed the runtime governance by process clients with aspects.

Distribution and multi-tenancy are requirements arising from the cloud computing paradigm. Within this context, flexible client-driven and client-controlled governance techniques are needed to manage cloud service processes dynamically and securely based on the individual policy enforcement needs of the clients.

Our future work includes investigating the performance overhead caused by the dynamic weaving with different scenarios and find possible optimal mechanisms to reduce it. For example, temporary weaving matching results might be kept on the provider side for each client weaver; the clients could notify the provider (e.g., provide a version number of weaver in clientReference) to update the temporary file once aspects are updated for a process. This could reduce the unwanted weaving calls from abstract components. In addition, the default value of abstract components is fixed in current work, but it might be dynamically assigned in certain situations to automatically select a remedial strategy or automatically select a process branch for load balancing.

#### ACKNOWLEDGMENT

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

#### REFERENCES

- [1] C. Pahl, "Layered ontological modelling for web service-oriented model-driven architecture," in *European Conference on Model-Driven Architecture Foundations and Applications ECMDA'2005*. Springer LNCS 3748, 2005, pp. 88–102.
- [2] R. Barrett, L. M. Patcas, C. Pahl, and J. Murphy, "Model driven distribution pattern design for dynamic web service compositions," in *International Conference on Web Engineering ICWE*. ACM Press, 2006, pp. 129–136.
- [3] "Policy based governance for the enterprise," *A WebLayers white paper*, 2005.
- [4] E. A. MARKS, *Service-Oriented Architecture Governance for the Services Driven Enterprise*. John Wiley & Sons, Inc., 2008.
- [5] T. Anstett, F. Leymann, R. Mietzner, and S. Strauch, "Towards bpm in the cloud: Exploiting different delivery models for the execution of business processes," in *IEEE Congress on Services - I*, 2009.
- [6] P. Fingar, "Cloud computing and the promise of on-demand business innovation," *Intelligent enterprise*, 2009.
- [7] M. Wang, K. Y. Bandara, and C. Pahl, "Distributed aspect-oriented service composition for business compliance governance with public service processes," in *International Conference on Internet and Web Applications and Services*, 2010.
- [8] C. Pahl, "A formal composition and interaction model for a web component platform," in *Proc. ICALP'2002 Workshop on Formal Methods and Component Interaction*, A. Brogi and E. Pimentel, Eds. Elsevier *Electronic Notes on Computer Science ENTCS Vol. 66 No. 4*, 2002.
- [9] C. Pahl, "A picalculus based framework for the composition and replacement of components," in *Workshop on Specification and Verification of ComponentBased Systems (OOPSLA)*. ACM Press, 2001.
- [10] J. Vaughan, "On the road to soa - part 2, governance is fundamental," 2009, [http://searchsoa.techtarget.com/tip/0,289483,sid26\\_gci1359534,00.html](http://searchsoa.techtarget.com/tip/0,289483,sid26_gci1359534,00.html).
- [11] D. Karastoyanova and F. Leymann, "Bpel'n' aspects: Adapting service orchestration logic," in *IEEE International Conference on Web Services*, 2009.
- [12] A. Charfi, T. Dinkelaker, and M. Mezini, "A plug-in architecture for self-adaptive web service compositions," in *IEEE International Conference on Web Services*, 2009.
- [13] P. d. Leusse, T. Dimitrakos, and D. Brossard, "A governance model for soa," in *IEEE International Conference on Web Services*, 2009.
- [14] S. Wang and M. A. M. Capretz, "A policy driven approach for service-oriented business rule management," in *IEEE International Conference on Industrial Informatics*, 2007.
- [15] H. Weigand, W.-J. v. d. Heuvel, and M. Hiel, "Rule-based service composition and service-oriented business rule management," in *Interdisciplinary Workshop Regulations Modelling and Deployment*, 2008.
- [16] "Oracle soa suite new features 10g(10.1.3.3) - fault management framework," <http://www.oracle.com/technology/products/ias/bpel/pdf/10133technotes.pdf>.
- [17] M. E. Kharbili and T. Keil, "Bringing agility to business process management: Rules deployment in an soa," in *6th IEEE European Conference on Web Services*, 2008.
- [18] F. Rosenberg and S. Dustdar, "Business rules integration in bpm - a service-oriented approach," in *7th IEEE International Conference on E-Commerce Technology*, 2005.
- [19] M. Wang, K. Y. Bandara, and C. Pahl, "Integrated constraint violation handling for dynamic service composition," in *IEEE International Conference on Services Computing*, 2009.
- [20] "Oracle business rules: Technical overview," *An Oracle White Paper*, 2007.
- [21] S. Subramanian, P. Thiran, N. C. Narendra, G. K. Mostefaoui, and Z. Maamar, "On the enhancement of bpm engines for self-healing composite web services," in *International Symposium on Applications and the Internet*, 2008, pp. 33–39.
- [22] R. Mietzner, T. Unger, R. Titze, and F. Leymann, "Combining different multi-tenancy patterns in service-oriented applications," in *IEEE International Enterprise Distributed Object Computing Conference*, 2009.
- [23] R. Mietzner, F. Leymann, and M. P. Papazoglou, "Defining composite configurable saas application packages using sca, variability descriptors and multi-tenancy patterns," in *International Conference on Internet and Web Applications and Services*, 2008.
- [24] C. Berndtson, "Interop: Cloud computing adopters ready to 'trust, but verify'," 2009, <http://www.crn.com/software/221900379;jsessionid=R1HY3YANN5EL1QE1GHOSKHWATMY32JVN>.
- [25] "Indian bpm and the ongoing struggle with data security issues," 2009, [http://searchsecurity.techtarget.in/news/article/0,289142,sid204\\_gci1374165,00.html#](http://searchsecurity.techtarget.in/news/article/0,289142,sid204_gci1374165,00.html#).