

AdaptiveBPEL: a Policy-Driven Middleware for Flexible Web Services Composition

Abdelkarim Erradi, Piyush Maheshwari
School of Computer Science and Engineering
University of New South Wales, Sydney, Australia
[aerradi,piyush@cse.unsw.edu.au](mailto:{aerradi,piyush}@cse.unsw.edu.au)

Abstract

The variation of contexts in which a Web service could be used and the resulting variation in functional and Quality of Service (QoS) requirements motivates extending Web services platforms to cater for differentiated service offerings and policy-driven dynamic adaptability. Adaptability is an important requirement in the context of Web services to cater for the need of diverse set of client applications requesting customized view of consumed Web services to fit their own contexts and preferences. Our research aims to devise a novel service composition framework, named AdaptiveBPEL, which leverages aspect-oriented software composition techniques to "open up" the service composition for dynamic change in order to provide a greater degree of configurability and dynamic adaptability of Web services. This framework can: 1) ease adapting to changes in collaboration policies and business rules governing service interactions 2) provide different levels of functional and QoS offerings. The adaptation process is policy-driven to declaratively define the adaptive service behavior and manage the dynamic injection of functional and non-functional extensions into a core service composition to allow on-demand and per-instance service refinement.

1. Introduction

Web services are becoming the de facto technology to interconnect and ease interoperability between heterogeneous and autonomous systems. It promises improved business agility and reduced integration costs through increased interoperability and reuse of loosely-coupled business-aligned services as well as standards-based Enterprise Application Integration (EAI) and B2B integration [9]. Web services vision is to enable dynamic integration by composing applications from replaceable and Internet accessible software components from various providers. The successful realization of this vision depends on satisfying three critical requirements:

- Interoperability requirements with respect to service interfaces, semantics, business protocols and collaboration policies. Using open standards in particular (WS-*) standards can help in addressing interoperability requirement but it is not enough [3].
- Quality of Service (QoS) requirements to ensure that the required services are reliable and timely.
- Adaptive services capable of self-adjusting, with minimal or no manual intervention, to accommodate policy and business rules changes as well as requirements and preferences of various clients accessing the service [5].

The push towards context-aware, adaptive and on-demand computing requires middleware infrastructures that support the delivery of adaptive services with varying functionality and QoS attributes [8] in order to meet the requirements of different clients accessing the service in different contexts and with different service levels expectations. For example, a financial analysis Web service could advertise different service classes to accommodate different clients' requirements by offering varying depth of the financial analysis, verbosity and/or formatting of the results, the rate of notification to clients, the guaranteed response time and the payment model.

Service realization often required the composition of various Web services, which is typically defined using a process description language like BPEL [14]. However, current platforms for Web services composition do not cleanly modularize and externalize context-dependant extensions to add value or to address QoS concerns. Thus the implementation of adaptability extensions and QoS aspects gets scattered and tangled with the core functional specification and implementation of the composition [7] and this in turn negatively impacts the system maintainability and adaptability. Addressing these limitations, calls for developing new architectural principles for building such systems, and extending composition middleware capabilities with mechanisms to ease addressing QoS aspects and to facilitate the development and delivery of differentiated and adaptive services.

Our approach aims to allow dynamic change of the process instance control flow path and dynamic manipulation of exchanged messages between the participating services. The adaptation process is driven by aspects weaving constructs generated based on a collaboration policy negotiated at runtime between the interacting endpoints.

The rest of this paper is structured as follows. Section 2 provides a background on Web services composition and QoS aspects. Section 3 describes the architecture and features of AdaptiveBPEL. Section 4 discusses related work while section 5 concludes the paper and provides some directions for future work.

2. Background and problem area

Increasingly, applications are being built by assembling services at a coarser-grain level from a range of internal and external systems (Figure 1). The main challenge of this model is to be able to offer differentiated and adaptive services that can quickly align the service behavior to varying functional and QoS requirements. This is motivated by the fact that Web services users (i.e., applications or individuals) can have varied profile, different requirements, and different willingness to pay for different service properties. However, the best-effort service delivery and ‘one size fits all’ is the prevailing model in Web services today, all requests are treated equally and no adaptable or customized service views can be easily provided. The main motivation for our research comes from the lack of suitable framework to facilitate the development and delivery of differentiated services with dynamic adaptability capabilities. Dynamic adaptation is also required to meet high availability requirements through hot fixes to long running processes where it might not be acceptable to take them offline to be extended [7].

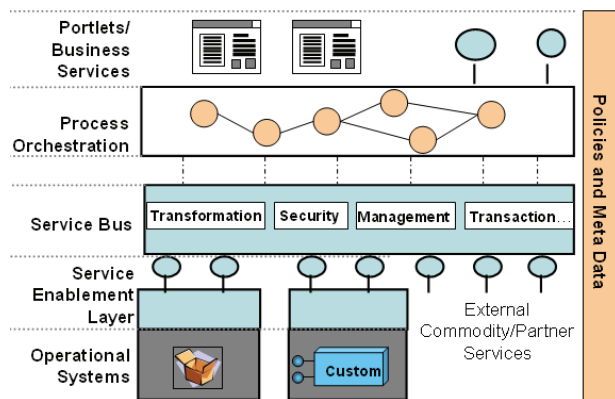


Figure 1: Service Oriented Architecture typical core Layers

2.1. QoS in Web services composition

Web services introduce a layer of abstraction above the operational systems layer to ease interoperability between heterogeneous systems running on different platforms, implemented in different programming languages, and may be managed by different providers. The true power of Web Services is leveraged through the combination of Web services possibly from different providers in order to create more value-added and feature rich integrated services. For example, an airline booking service, a hotel service and a credit card service can be composed into a travel booking service.

The composition specification typically uses a process description language like BPEL [14] to specify the process participants, the control flow (i.e., the order and the rules controlling conversations between participating Web services), the data-flow (i.e., the messages exchanged between the composed services) and the process fault handling mechanisms. Composition languages draw heavily on business process modeling and workflow systems. These compositions are usually executed using centralized BPEL engines that orchestrate and control interactions and message exchanges between participating services. However, Web services composition languages are silent in regards to the specification and handling of crosscutting concerns like QoS aspects while orchestration engines have insufficient support for enforcing collaboration policies that define QoS requirements as well as adaptability and extensibility needs. Hence, it is difficult to define, modularize and manage non-functional concerns and context-sensitive behaviors. The adaptation and QoS handling logic often ends up being implemented in proprietary manner and tangled with the core functional logic.

The problem is how to specify and address adaptability and QoS concerns in Web services composition without complicating the composition logic or breaking the loose-coupling requirements of service-oriented architectures. For example, how to make sure that the interactions generated by the composition are secure (authentication, message encryption and signing), and reliable (exactly-once message delivery) while insuring transactional integrity. Embedding the specification and the handling of non-functional concerns within the specification of the core logic of the composition would make the composition logic too complex and hard to maintain and evolve [6]. To address these limitations, we have first explored handling QoS concerns, like security, by routing all exchanged messages through a Web services messaging

middleware, named wsBus [11]. wsBus leverages Web services late-binding capabilities and introduces the concept of a Virtual Endpoint (VEP), on which multiple equivalent services, acting as a recovery block, can be configured. A VEP may have a policy (e.g., encrypt all outgoing messages using particular algorithm) that the inbound/outbound messages must adhere to. Handlers bound to the VEP intercept and manipulate both request and response messages (e.g., encrypt outgoing messages and decrypt incoming ones). wsBus acts as a Web services gateway, through which travel all messages exchanged during the process enactment. All request messages are sent to the VEP, and wsBus transparently redirect messages to real services at runtime. The selection of services is based on monitoring data and QoS metrics gathered from prior interactions or from trusted peers. Before dispatch, the message passes through a series of handlers to enforce QoS policies. Responses are routed back through wsBus where further processing such as correlation and persistence can take place before the response is routed back to the orchestration engine. Although wsBus transparently added valuable services while avoiding the complication of Web services composition with QoS handling logic, it could become a bottleneck unless multiple clustered instances are deployed.

In this paper, we explore alternative techniques to address QoS aspects at the process level, our proposed framework, named AdaptiveBPEL, aims to augment a Web services composition engine, like ActiveBPEL [1], with policy-driven aspects weaving mechanisms to transparently enforce QoS policies and dynamically adapt the composition instance through the ability to weave predefined extensions as Web service calls either before, after or instead of an activity instance. Some examples of such extensions are:

- Check client-specific constraints defined in an SLA contract
- Add debugging and auditing features through persisting any update to given variables
- Persisting and retrieving the process state between activities
- Encrypt outgoing messages and decrypt incoming ones
- Trap all send/receive messages and redirect them to a different service using a different protocol or buffer messages for later replay.

2.2. Fundamental aspects of adaptability

Adaptability is an important requirement that needs to be taken into consideration in the context of Web services based applications. Adaptation can be static or

dynamic, manual or automatic, and proactive or reactive [8]. Static adaptation requires software code modifications while dynamic one only modifies the software run-time behavior. Manual adaptation can be contrasted with automatic one which is performed by the software itself when a certain condition occurs. Proactive adaptation is triggered before a change occurs in the environment whereas reactive one is performed as a consequence [8]. Our work focuses on runtime adaptations of composite service instances since they are more challenging and insufficiently addressed in related work.

Current Web services platforms provide no mechanisms that allow developers to create policy-driven adaptable Web Services, and this impedes the flexibility and loose-coupling promised by Web services. Also, Web services composition engines do not have any built-in mechanism to enforce policies during process enactment, and there is no clean separation of concerns between the functional and the non-functional code within SOAP handlers. Because of their capability to separate concerns, aspect-oriented software composition techniques could be leveraged to help alleviate these limitations.

2.3. Adaptability using aspects weaving

Aspect-Oriented Software Development (AOSD) is a paradigm that allows capturing and modularizing concerns that crosscut a software system into modules called aspects. Furthermore, application's behavior can be dynamically adapted using dynamic weaving [4] as aspects can be activated/deactivated at runtime. An aspect may contain different code fragments (called advice), and location descriptions (called pointcuts) to identify where to plug the code fragments in. The well-defined points in the program flow that can be selected by the pointcuts to inject additional behavior are called join points [17]. The activity of integrating aspects into a base component is called weaving. A weaver is a mechanism that integrates a base program with aspects to alter the behavior of a system either by source code modification or by use of hooks during compilation or at runtime time [17].

2.4. Motivating Example

Consider the following example of a Loan Process that is deployed at a typical loan broker. The loan broker accepts a request from a client, performs a credit check with an external service and then routes the application to different loan agencies. After receiving the offers, the best offer is selected and the customer is notified. The scenario consists of the following participants: LoanBroker, CreditRating service, ABCLoanService, StarLoanService and the

customer. The Loan process (Figure 2) orchestrates interactions between these services. This scenario illustrates that for this process to execute successfully the policies of the Loan Broker and those of the participating services need to be consistent and compatible. Also at the process enactment time middleware mechanisms are required to compute consistent policies (in our example between the Loan Broker local policies and the policies of the participating services) and enforce them. This can take place either at the messaging bus (Figure 1) usually deployed beneath the business process layer or the BPEL engine itself can be augmented with mechanisms like aspects weaving capabilities to take the responsibility of enforcing the policies declared by each process participant. The latter option allows greater flexibility to quickly adapt to changing collaboration policies without redeploying the business process.

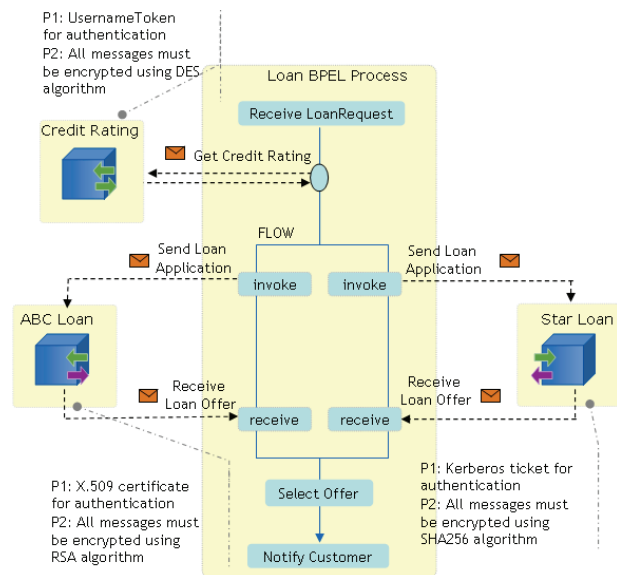


Figure 2: Loan BPEL process

In our scenario runtime policy negotiation could be required if the Loan Broker policy stipulates that messages for loan requests greater than a certain threshold or coming from corporate customers should be strongly encrypted while for other requests a weaker encryption could be sufficient. Runtime policy negotiation and enforcement is also justified if the Loan Broker policies are sensitive to runtime conditions, for example a policy could stipulate that when the server CPU usage or the pending messages

queue length is greater than a certain threshold, the system should switch to using weaker encryption policies to meet performance SLAs.

3. AdaptiveBPEL architecture

AdaptiveBPEL (Figure 3) aims to investigate, define, implement and experimentally evaluate a novel approach based on AOSD [2] mechanisms to ease the development of differentiated and adaptive Web services composition. AOSD seems to be a promising paradigm to address service adaptability and non-functional concerns [7]. Nevertheless, AOSD is currently mostly focused on low-level language extensions [17]. In order to exploit aspect-oriented software composition techniques at the process level, AOSD techniques need to be improved to support:

- run-time composition of aspects to enable dynamic customization of process instances
- policy-driven selection of aspects to enable client-specific customization of services

The core of the framework is a run-time aspects weaving middleware that can be integrated on top of existing composition platforms, like [1], to achieve adaptable executions and extensible business processes. The adaptation process is policy driven to declaratively define the adaptive service behavior through selective and dynamic injection of functional and non-functional extensions into a core service to allow on-demand service instance adaptation and to be able to provide differentiated service levels from the same service composition.

For example, to implement authentication logic, execution points (i.e., point-cuts) need to be defined along the process control flow path, to allow intercepting sent/received SOAP messages, to be able to add security headers like adding the caller's username/password pairs as requested by the security policy declarations of the partner Web service.

First, components addressing service adaptability and non-functional concerns need to be encapsulated into a set of aspects. The second step is to identify and declaratively describe the join points in the service composition where possible adaptations or extensions may be required. Then, depending on a deployment descriptor or a run-time negotiated policy between the service client and the service provider, AdaptiveBPEL will drive the plug-in and removal of pre-defined aspects into the abstract process instance.

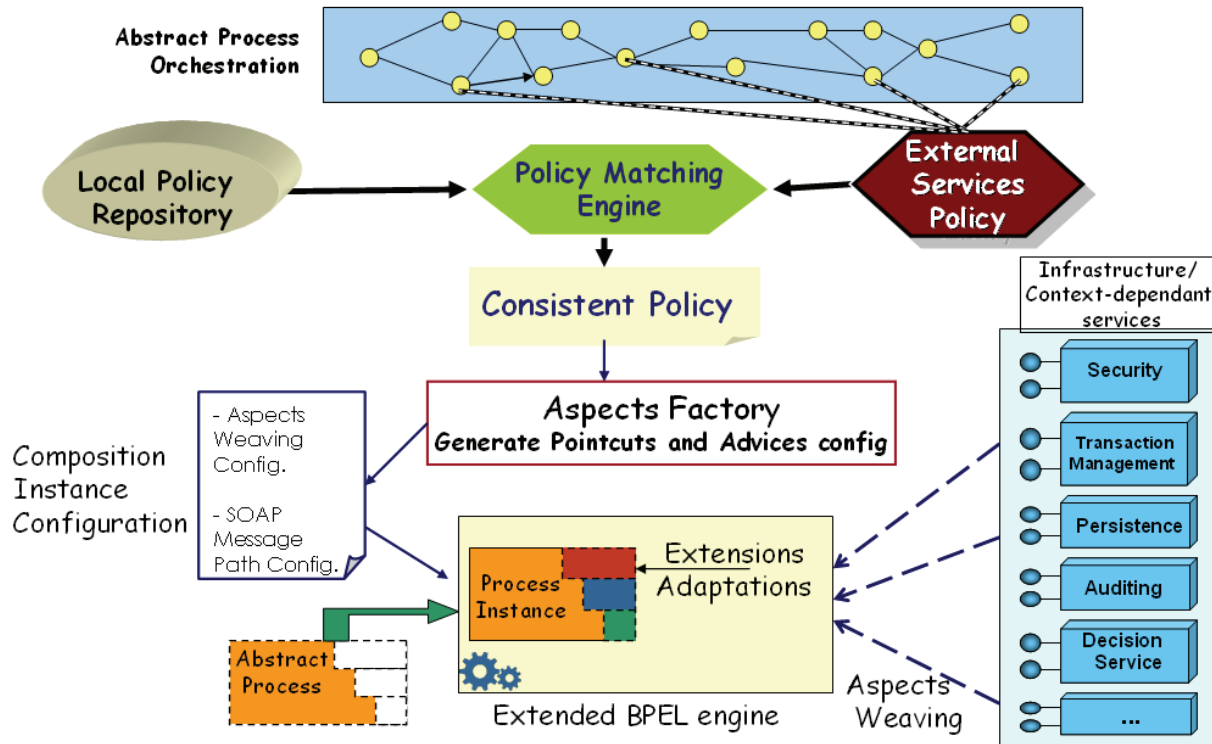


Figure 3: Conceptual AdaptiveBPEL architecture

3.1. Policy-driven adaptability

In AdaptiveBPEL, aspects weaving into a process instance can be driven either statically by a deployment descriptor manually crafted or dynamically by a negotiated policy at runtime.

To enable interoperation between composed services, their advertised conversation policies expressing requirements/capabilities and preferences, need to be consistent. The policies matching can take place during a pre-enactment phase or during enactment, particularly when policies change frequently or if they are sensitive to runtime conditions or when real Web services to be used are only known at runtime.

Given collaboration policies that are expressed in a declarative, machine-readable format and advertised by interacting services, using a language like WS-Policy [13], matching mechanisms are required to arrive at a mutually acceptable composite-policy satisfactory to interacting parties. For example, just knowing that a service supports WS-Security is not enough information to enable successful integration. The client endpoint needs to know what security tokens the provider is capable of processing (such as Username Token, Kerberos tickets, or certificates), and which one it prefers. The client must also determine if the service requires signed messages and what token type must be used. In addition, the client needs to

know when to encrypt the messages, which algorithm to use, and how to exchange a shared key with the service.

AdaptiveBPEL has a built-in policy mediator (PM) to negotiate a composite policy and oversee the aspects weaving to enforce the negotiated policy. In other words, as a pre-processing before service invocation, a handshake takes place to agree on policies that will drive the conversation. As illustrated in Figure 4, the process starts by the PM at the initiator endpoint issuing the consumer's policy to the service provider, then the PM at the service provider computes an intersection of the policies (i.e., policy matching) to find a consistent policy acceptable to both ends. The calculated policy is then communicated to the client. Next, the client endpoint verifies that the calculated policy is compatible with the local policy. If the policy matching fails, an exception is raised to alert that the policies of interacting services are incompatible.

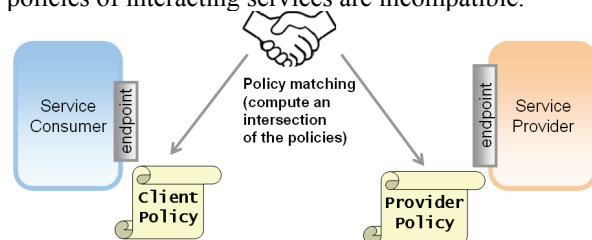


Figure 4: Policy mediation process

To enforce the computed policy, as well as any adaptation policies specific to the process initiator, the Aspects Factory component generates the required aspects weaving configuration to instruct the BPEL engine on how to intercept points in the process execution and where to plug-in calls to middleware or user-dependant services. The generated configuration file defines the XPath queries, based on the abstract process definition, to specify the *pointcuts* within the process instance where adaptability as well as QoS logic *advices* should be injected. In our framework, *advices* are specified as service calls to a pre-defined set of Web services that either encapsulate infrastructure services (like SOAP message signing), context dependent behavior services or services wrapping calls to enforce business rules managed by a Business Rules Management System (BRMS) like ILOG JRules [20].

4. Related work and discussion

Building applications that are easy to adapt and extend is a well-explored topic in software engineering. This section briefly highlights the research efforts closely related to our work. We also discuss how our work will differ and complement existing body of knowledge in the area of dynamic Web services adaptation.

4.1. Adaptability in traditional workflow systems

Workflow systems address the need for adaptability and flexibility using a variety of implementation approaches [5, 10, 12]. Their main contributions are formally founded methods to make the workflow process capable of efficiently adapting the tasks to be performed and their execution order (e.g., add or remove tasks, change control flow paths etc.) in reaction to changes in the environment conditions like changes in the types of the participants, their role in an organization and the infrastructure reconfiguration. The process adaptations can be classified according to whether they are performed at design-time or enactment-time either at the process schema or process instance level [12]. For example, eFlow [5] uses several constructs to achieve adaptability. These include dynamic service selection and binding, parallel execution of multiple equivalent services and the notion of generic service that can be replaced with a specific set of services at runtime. eFlow is also equipped with a migration manager that assists users to modify running process instance(s) without violating a predefined set of behavioral consistency rules. But adaptability support remains insufficient and vendor

specific [15]. Moreover, many adaptation triggers, like infrastructure changes, considered by workflow adaptation are not relevant for Web services composition because participating services hide all implementation details and only expose interfaces described in terms of types of exchanged messages and message exchange patterns. On the other hand, Web services composition need to be more adaptive to policies and business rules governing interactions between composed services.

4.2. Adaptation of distributed applications using aspect-oriented approaches

AOSD research explores non-conventional software composition approaches to provide flexible extension, adaptation and integration of components. AOSD techniques were first explored in the context of programming languages (e.g., AspectJ [17]), and currently extended to support the full software lifecycle [2]. Recently AOSD approaches have been explored in the context of distributed and component based applications [8, 22, 24]. For example, GlueQoS [24] propose an adaptive and AOSD approach to support policy-driven configuration of client and server middleware to ensure interoperability. QoS preferences are specified in the GlueQoS policy language. These policies are exchanged at binding time between systems interacting in an ad-hoc setting, using the GlueQoS mediation meta-protocol. The policies are then matched up, and resolved by the GlueQoS mediator. The resulting policy resolutions are deployed and executed.

Many projects such as [19] explore applying dynamic aspects to component architectures to enhance flexibility and adaptability. For example, the JAsCo infrastructure [21] enables the execution of Java Bean component based applications that can be dynamically adapted by aspects. The components when loaded into the infrastructure are modified to insert traps at every public method which invoke the aspect weaver when they are called. A management layer based on the JAsCo aspects [22] was developed to address client-side management of Web Services.

The above research efforts provide a good foundation for our work but their main focus is low-level object-oriented middleware extensions while our focus is on coarse-grained process-oriented interactions and composition adaptation. AdaptiveBPEL is also closely related to AO4BEPL [7] but our work adds the policy-driven adaptation and dynamic specification of aspects to enable instance-specific customization of the service composition.

4.3. Web services policy languages

Our work relates to formal representations of Web services policies. Policies can be seen as a decision-making guide specifying the rules governing choices in the behavior of a system [16]. Policy-driven systems provide the required flexibility in building and managing composite services by allowing a change of the system behavior without modifying the process specification. The authors in [16] identify three types of policies:

- Action Policy, specifies the action that should be taken when a condition is satisfied
- Goal Policy, specifies a desired resulting state that must be achieved and maintained, but does not specify how that is to be accomplished
- Utility Function Policy, where the system computes optimal action to maximize a utility function.

Among the policy languages relevant for Web services, the most prominent one is OASIS eXtensible Access Control Markup Language (XACML) [18]. It defines a language for expressing access control policies. In XACML, conditions and effects are combined into rules, and rules are combined into policies. Typically, the result of an XACML policy is permit or deny access to the resource governed by the policy. XACML also includes the notion of obligations, which defines a requirement associated with the policy. For example, logging could be an obligation associated with permitting access to a resource.

For defining contracts to specify agreed upon non-functional characteristics, one can use Web Services Agreement Specification (WS-Agreement) [23]. It defines a template for agreement in the form of Service Level Objectives (SLOs). For example, provide a two second average response time for a Web service operation.

The Web Services Policy Framework (WS-Policy) [13] defines an extensible container to hold domain-specific policy assertions. For example, WS-SecurityPolicy defines semantics for security policies that are embedded within the WS-Policy wrapper.

In the context of Web services, policies can be defined and communicated either statically or dynamically. Static policies can be attached to the service contract (e.g., WSDL) while dynamic policies are created and communicated to service consumers during interactions with the service.

Policy languages space remains fragmented and a unifying framework is highly needed. Moreover, automated consistency checking of policies (expressing non-functional requirements such as security, reliability, and transactional integrity) is still

in its infancy [24]. The challenges arise from the fact that policies evolve over time and vary with a service's deployment context (which may change dynamically) and runtime environment (which is constantly changing). These factors complicate the task of automating policies interoperability and make a clear case for further research to address these needs. Two main problems that need to be addressed: 1) How can we ensure that composed services have compatible and consistent QoS policies? 2) How can we dynamically detect and resolve/mediate conflicts between policies of composed services? Our future work will investigate novel algorithms and a policy middleware to address these open issues.

5. Conclusion

Enforcing collaboration policies as well as supporting client-specific customization of Web services composition is required because 'one size fits all' model is not enough as there is often a need to adapt services to fit their invocation context and the user profile. This paper presented the initial architecture of a framework, named AdaptiveBPEL, to enable crosscutting and context-sensitive logic to be factored out of the service composition and modularized into aspects. AdaptiveBPEL manages the dynamic refinement of a service composition instance through policy-based aspects weaving to address QoS concerns and adaptability needs. In this paper, our goal has been to motivate and contextualize our framework. As future work we are working towards completing the development of the framework and experimentally evaluating it in realistic and advanced scenarios.

References

- [1] ActiveBPEL - Open Source BPEL Engine, <http://www.activebpel.org>.
- [2] AOSD, <http://aosd.net>.
- [3] Bentallah, B. and Nezhad, H. 2004, 'Service Oriented Computing: Opportunities and Challenges', in *Second International Workshop on Semantic Web and Databases (SWDB), held in conjunctions with VLDB '2004*, LNCS 3372, pp. 2-9.
- [4] Bockisch, C., Haupt, M., Mezini, M. and Ostermann, K., 'Virtual Machine Support for Dynamic Join points', in *3rd International Conference on Aspect-Oriented Software Development - AOSD 2004*, Lancaster, UK, pp. 83-92.
- [5] Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V. and Shan, M.-C., 'Adaptive and Dynamic Service Composition in eFlow', in *CAISE 2000*, Stockholm, Sweden. <http://www.hpl.hp.com/techreports/2000/HPL-2000-39.pdf>

- [6] Charfi, A. and Mezini, M., 'Hybrid web service composition: business processes meet business rules', in *Second International on Service-Oriented Computing - ICSOC 2004*, New York, NY, USA, pp. 30-38.
- [7] Charfi, A. and Mezini, M. 2004, 'Aspect Oriented Web Service Composition with AO4BPEL', in *The European Conference on Web Services (ECOWS'04)*, Erfurt, Germany, LNCS 3250, pp. 168-182.
- [8] Courbis, C. and Finkelstein, A., 'Towards Aspect Weaving Applications', in *27th Int. Conference on Software Engineering - ICSE 2005*, St. Louis, Missouri, USA.
<http://www.cs.ucl.ac.uk/staff/A.Finkelstein/papers/aspectise.pdf>
- [9] Crawford, C. H., Bate, G. P., Cherbakov, L., Holley, K. and Tsocanos, C. 2005, 'Toward an on demand service-oriented architecture', *IBM Systems Journal*, vol. 44, no. 1, pp. 81-107.
<http://www.research.ibm.com/journal/sj/441/crawford.pdf>
- [10] Edmond, D. and Hofstede, A. H. M. t. December 1999, 'Achieving Workflow Adaptability by means of Reflection', *ACM SIGGROUP Bulletin*, vol. 20, no. 3, pp. 10-10.
- [11] Erradi, A. and Maheshwari, P. 2005, 'A Broker-based Approach for Improving Web Services Reliability', in *IEEE International Conference on Web Services 2005 (ICWS'05)*, Orlando, Florida, USA.
- [12] Heintz, P., Horn, S., Jablonski, S., Neeb, J., Stein, K. and Teschke, M. 1999, 'A comprehensive approach to flexibility in workflow management systems', in *International joint conference on Work activities coordination and collaboration*, ACM Press, San Francisco, California, USA.
- [13] IBM, Microsoft et al., September 2004, *Web Services Policy Framework (WS-Policy)*. <http://www-106.ibm.com/developerworks/library/specification/ws-polfram>.
- [14] IBM et al. May 2003, *Business Process Execution Language for Web Services (BPEL4WS) v1.1*. <http://www-128.ibm.com/developerworks/library/specification/ws-bpel> [3 August 2005].
- [15] Karastoyanova, D. and Buchmann, A., 'Extending Web Service Flow Models to Provide for Adaptability', in *OOPSLA 2004 Workshop on Best Practices and Methodologies in Service-oriented Architectures: Paving the Way to Web-services Success*, Vancouver, Canada.
- [16] Kephart, J. O. and Walsh, W. E. 2004, 'An artificial intelligence perspective on autonomic computing policies', in *Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, Yorktown Heights, NY, USA, pp. 3-12.
<http://www.research.ibm.com/people/w/wwalsh1/Papers/policy04-acp.pdf>
- [17] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W., 'Overview of AspectJ', in *ECOOP 2001*, Budapest, Hungary.
- [18] OASIS eXtensible Access Control Markup Language (XACML) 2.0 February 2005.
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml [10 August 2005]
- [19] ObAsCo (Objects, Aspects, and Components) Research Group, <http://www.emn.fr/x-info/obasco>.
- [20] Rosenberg, F. and Dustdar, S. 2005, 'Business Rule Integration in BPEL - A Service-Oriented Approach', in *7th International IEEE Conference on E-Commerce Technology (CEC 2005)*, Munich, Germany.
- [21] Suvée, D., Vanderperren, W. and Jonckers, V. 2003, 'JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development', in *2nd international conference on Aspect-oriented software development*, Boston, USA, p. 21-29.
<http://ssel.vub.ac.be/jasco/papers/aosd2003.pdf>
- [22] Verhecke, B., Cibrán, M. A., Vanderperren, W., Suvée, D. and Jonckers, V. 2004, 'AOP for Dynamic Configuration and Management of Web services in Client-Applications', *International Journal on Web Services Research (JWSR)*, vol. 1, no. 3, pp. 25-41.
- [23] Web Services Agreement Specification (WS-Agreement) 1.1 August 2004.
<http://www.ggf.org/Meetings/GGF12/Documents/WS-AgreementSpecification.pdf> [10 August 2005]
- [24] Wohlstadter, E., Tai, S., Mikalsen, T., Rouvellou, I. and Devanbu, P., 'GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions', in *26th International Conference on Software Engineering - ICSE 2004*, Edinburgh, Scotland, UK.
http://www.research.ibm.com/AEM/pubs/GlueQoS_ics_e2004.pdf