Non-functional Concerns in Web Services: Requirements and State of the Art Analysis

Benjamin Schmeling
SAP Research Center Darmstadt
benjamin.schmeling@sap.com

Anis Charfi
SAP Research Center Darmstadt
anis.charfi@sap.com

Mira Mezini
Software Technology Group, TU Darmstadt
mezini@informatik.tu-darmstadt.de

ABSTRACT

For the success and adoption of web services it is essential that not only functional concerns (FC) are addressed but also non-functional concerns (NFC) such as security, reliable messaging, performance, and availability. The support for non-functional concerns encompasses two aspects: the specification and the enforcement and we focus thereby on two views: the black box view (only the interface) and the gray box view (the internal process of a composite service).

In this paper we identify the most important requirements for the specification and enforcement of NFCs whilst taking the two different views into account. Furthermore, we present a thorough analysis of the state-of-the-art works based on those requirements and identify areas where future work has to be conducted to fulfill all identified requirements.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*Methodologies*; D.2.12 [**Software Engineering**]: Interoperability—*Distributed objects*; C.2.0 [**Computer communication Networks**]: General—*Security and protection*

General Terms

Security, Reliability, Performance

1. INTRODUCTION

When developing web services two major aspects have to be addressed: functional concerns on the one hand (i.e., WHAT the service should do) and non-functional concerns (i.e., HOW the service should perform describing the characteristics or quality properties of the implementation of functional concerns) on the other hand. As examples for NFCs in web services we mention security, reliable messaging, performance, availability, and transactional behavior. Non-functional concerns are extremely important for the success and adoption of web services because enterprise web services cannot be exposed without appropriate support for security, reliability, or transactional behavior. E.g., an online payment service requires appropriate support for NFCs such as performance and availability because customers will not be willing to wait 5 minutes for a payment confirmation. In such cases non-functional attributes become an important criterion for the selection of a service from a set of services implementing similar functional capabilities.

The support for non-functional concerns encompasses two aspects: the specification of NFCs on the one hand and the enforcement of the specified NFCs on the other hand. For the specification there must be some means to express which NFC should be applied to what extent and with which FC it should be integrated. For the enforcement there must be some mechanisms to ensure the realization of the specified NFCs.

In addition, three different service perspectives can be taken when addressing non-functional concerns: the black box view (i.e., an external view where only the service interface is visible), the gray box view (i.e., the internal process of a composite service is visible in addition to the interface), and the white box view (i.e., all implementation artifacts are visible). We will focus especially on the first two views because tackling NFCs in the white box view is rather specific to the programming language and the platform used for the implementation.

In this paper, we present a meta-model for the NFC domain and based on that we identify the most important requirements for the specification and enforcement of NFCs in web services whilst taking the black box and the gray box views into account. We also present a thorough analysis and discussion of the state of the art based on those requirements. Furthermore, we identify areas where future work has to be conducted to fulfill all identified requirements.

The paper is structured as follows: Section 2 explains the terms that this paper builds on and introduces our meta-model for the NFC domain. Section 3 identifies the most important requirements for the specification and enforcement of non-functional concerns in web services. Section 4 shows how state of the art research works address those requirements. Section 5 concludes this work and prospects future research activities.

2. TERMINOLOGY

In this section, a brief explanation of the most important terms and definitions that are used throughout this paper are given[1] and visualized in Figure 1.

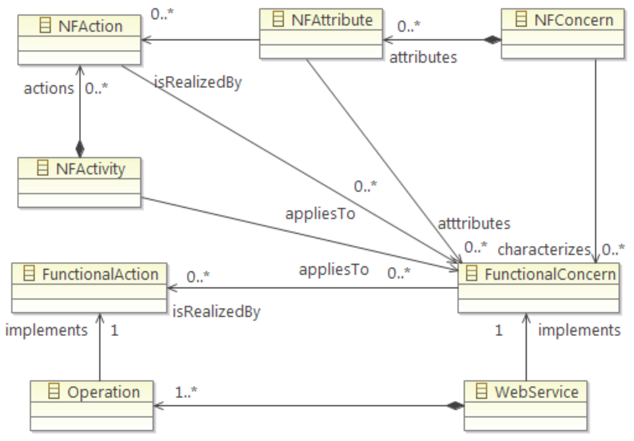

Figure 1: A meta-model for the NFC domain.

- **Non-functional attribute:** An attribute that describes the quality or characteristics of a functional concern. For example confidentiality is an example for a non-functional attribute for the functional concern *customer registration.*
- **Non-functional concern:** A group of semantically correlated non-functional attributes. E.g., security is an NFC that comprises attributes such as confidentiality and integrity.
- **Non-functional action (NFA):** An NFA represents behavior that results in the enforcement of a non-functional attribute. E.g., encryption is an NFA that realizes the non-functional attribute confidentiality. An NFA cannot be executed without its functional context; i.e., it must apply to an FC. An NFA is not visible for the consumer of a service and hence cannot directly be invoked by the consumer but it is always executed in the context of a FC.
- **Subject of an NFA:** A well-defined point in the realization of an FC. An example is the execution of a functional action or the observation of a certain type of message.
- **Non-functional activity:** A composite structure that includes several NFAs that apply to the same subject and defines their ordering or control flow. A non-functional activity is not visible for the consumer of a service and hence cannot be directly invoked by the consumer.
- **Functional concern**[2] **(FC):** A coherent piece of functionality that is part of the software system, for example the *user registration* concern for a Web application. An FC can always be associated with a use case specified in the system requirements.
- **Functional action (FA):** A behavior that can be executed in order to realize (parts of) the FC. In terms of web services an FA is mapped to an operation.

[1]Remark that the terms activity and action are used in a similar fashion as in the UML2 specification [16].
[2]It is assumed that the system has been decomposed in such a way that one web service realizes exactly one FC. Theoretically, it is also possible that one service implements more than one FC.

3. REQUIREMENTS

In this section, we identify several requirements for specification and enforcement of NFCs in the context of web services.

3.1 NFC Specification

An obvious requirement is the ability to specify non-functional requirements (**Requirement S1**). A further requirement is the ability to associate the specified requirements with the actions that contribute to their realization (**Requirement S2**). The separation of the requirements and their realization is also found in popular software development processes like the IBM Rational Unified Process [21] that differentiates the Requirements Discipline and Analysis and Design Discipline (aiming at how the requirements can be realized). The requirements definition should be made by a requirements engineer, whereas the action definition should be made by an expert of the non-functional domain, e.g. a security expert. Satisfying the non-functional attributes confidentiality or response time to a certain extent is an example for non-functional requirements of a web service. Confidentiality might be realized by the NFA *Encrypt* or the response time could be optimized by the NFA *Cache.* The particular action has an impact on the quality and level of attribute enforcement. For example the chosen encryption algorithm is an indicator for the quality of the confidentiality.

An important requirement for the specification of NFCs is the existence of means for defining the subjects of non-functional requirements or actions (**Requirement S3**). Without information about the subject it is not clear to what the NFC applies. Different granularity levels can be distinguished for the subject specification. A subject can be coarse-grained such as a whole service (or even a set of services), or fine-grained at the level of a service operation, its input or output or fault message, or alternatively a certain type of message. These subjects are all part of the WSDL interface of a web service, i.e., the web service is considered a black box.

If there is more than one non-functional requirement for a service, a set of NFAs must be applied to the same subject. In that case the execution order of the NFAs has an impact on the result or effect the actions produce. For example we consider the composition of a caching action that caches frequent web service invocations (and thus enforces better performance) with an accounting action that tracks all invocations per consumer in order to charge her on a per-use basis. If caching is executed before accounting, already cached invocations will not be accounted, whereas in the reverse order all invocations no matter if cached or not will be accounted. As a consequence one needs appropriate means to define the execution order of NFAs depending on the particular use case (**Requirement S4**).

In case of composite web services there is more knowledge on the service than just its WSDL interface, for example,

ID	Requirement
S1	Non-functional Requirements Specification
S2	Non-functional Actions Specification
S3	Web Service Subjects Specification
S4	NFA Execution Order Specification
S5	Composite WS Subjects Specification
S6	Stateful NFC Specification

Table 1: Specification Requirements

ID	Requirement
E1	Separation of Concerns
E2	Transparent Integration of FCs and NFCs
E3	Quantification
E4	Superimposition
E5	Integration of NFCs with distributed WS
E6	Programming Language Independence
E7	WS Composition Support

Table 2: Enforcement Requirements

the composition logic of a service can be made explicit using WS-BPEL [2]. Taking the WS-BPEL process as additional information about the service, we get further subjects for the NFC specification namely the process elements such as activities (**Requirement S5**). Messaging activities (such as *invoke* and *reply*) and structured activities (such as *scope* and *flow*) are valid subjects for the specification of NFCs.

Composite web services as defined by WS-BPEL are often stateful, e.g., there are several interactions between the process and its partners (e.g., the consumer) and the state is changed by the interactions. The state in form of the currently executed activity and its corresponding variables must be preserved as long as the process is not terminated. Therefore, WS-BPEL provides the concept of correlation sets, which allows matching the right process instance with a received message (presuming that there is more than one process instance running at the same time). The correlation problem can be generally observed for sets of messages that must be associated with a stateful conversation instance (refer also to [32]).

Not only FCs can be stateful but also NFCs like monitoring or transactions. A stateful NFC can be seen as an object instance with its own lifecycle living longer than just for a single service invocation. For example, the realization of a monitoring concern can be defined by a *startTimer* action followed by a *stopTimer* action. The *startTimer* action captures the current time that is then consumed by the *stopTimer* action for calculating the difference $dif = stopTime - startTime$. The state of this activity is represented by the *startTime* which must be stored until the *stopTimer* action is invoked. Whenever an NFC requires to be stateful also different subjects must be identified for the involved NFAs. For example, it does not make sense to apply the *startTime* and *stopTime* actions both to the input of a web service operation, but rather applying *startTime* to the input and *stopTime* to the output. In the web service as black box view, subjects that can be correlated to the same instance are the input and the output of an operation that can be correlated through the synchronous request and response message exchange with that operation. In composite web services a richer set of subjects can be identified since a stateful NFC associated with a process has the semantics that each process instance is related to its own NFC instance. Hence, several *startTime*/*stopTime* actions can be applied to arbitrary activities of the process. It is important to be able to unambiguously define which instance of a stateful NFC applies to which FC (**Requirement S6**). Table 1 provides a summary of the requirements.

3.2 NFC Enforcement

For the sake of separation of concerns the code that is responsible to enforce the NFC should be separated from the code that realizes the FC (**Requirement E1**). Moreover, the integration of the NFC into the FC should be realized as transparent as possible in order to avoid the tangling of functional and non-functional code (**Requirement E2**). Another requirement is the capability to associate an NFC with multiple FCs (quantification [13]) (**Requirement E3**) and to associate one FC with multiple NFCs (superimposition [25, 5], **Requirement E4**).

Another group of requirements stems from the distribution and platform-independence characteristics of web services. One requirement for the enforcement of NFCs for web services is that the solution can cope with the distribution, e.g., it should be possible to apply an NFC to multiple web services at once even if they are running on different machines (**Requirement E5**). Another requirement is that the components that enforce the NFCs should be platform-independent and thus can be applied to the web services independently of the programming language that is used for its implementation (**Requirement E6**). A further requirement is that requirements specified on the process underlying a composite web service can be enforced (**Requirement E7**) and in particular that especially stateful NFCs (as stated in Section 3.1) are supported. The identified enforcement requirements are summarized in Table 2.

4. STATE OF THE ART

In this section, we present state of the art works on non-functional concerns in web services and discuss how they support the previously identified requirements.

4.1 NFC Specification

4.1.1 Specification for Web Services (Black Box View)

WS-Policy [34] is a declarative approach to specify non-functional requirements or capabilities for web services where each NFC is represented by its own domain-specific WS-Policy specification. For example, security is captured by the WS-SecurityPolicy [23] specification comprising a set of security related assertions. An assertion represents a requirement, a capability, or another property of a behavior. In case of security, there are—among others—protection assertions that define what is protected at what level, for example one can specify that the message body should be signed and encrypted. The subject of a policy is defined as an entity (e.g., an endpoint, message, resource, operation) which a policy can be associated with (see WS-PolicyAttachment [35]).

The WS-Policy framework realizes Requirement S2 and S3. It allows for the specification of NFAs in terms of assertions and facilitates different functional subjects as defined by WS-PolicyAttachment.

Requirement S1 is not in scope of WS-Policy as this language focuses on NFA definition not on the definition of non-functional requirements, concerns or attributes. WS-Policy cannot cope with S4 sufficiently, because the ordering of actions in a policy can only be expressed by the semantics of its declarative assertions. An example for this can be found in the WS-SecurityPolicy where two assertions *EncryptBeforeSigning* and *SignBeforeEncrypting* are defined which reflect the ordering of the non-functional actions encrypt and sign. Thus, for each possible reasonable combination of NFCs a new assertion must be invented. A similar problem occurs when regarding the interaction between different concerns. For advanced security requirements it is also necessary to secure the reliable messaging mechanism which is also solved by additional assertions. In order to combine security with reliable messaging the WS-ReliableMessaging Policy [12] defines a *SequenceTransportSecurity* assertion which indicates that also the messages produced by reliable messaging protocol will be secured. Another drawback with that approach is that the WS-ReliableMessaging Policy contains security related assertions breaking the separation of concerns principle. WS-Policy is a specification that has been designed for web services as black box and not for composite BPEL processes [7], hence it cannot cope with Req. S5 and S6.

In the SCA Policy Framework [4] an *intent* describes the non-functional requirements such as authentication or confidentiality of a component or an interaction between components. The information on how this requirement is realized must be known at deploy time. For example the confidentiality *intent* can be realized by attaching a WS-Policy that specifies encryption. Thus, SCA facilitates a simple mechanism to define non-functional requirements (NFR for short, Requirement S1). An *intent* and the policies can be attached either to a service or to its operations meaning that they apply to the communication between services. Alternatively, a policy can also be attached to the services implementation. Since the SCA Policy Framework uses WS-Policy there are the same issues not supporting S4, S5 and S6.

Soeiro et al. [31] propose a language based on XML that facilitates the specification and composition of concerns at the requirements level. Each concern can be described by properties like its name, classification and stakeholders (satisfying Requirement S1). How a particular concern is realized is not taken into account (Requirement S2 not satisfied). However, composition rules such as sequential, parallel or interrupted orderings of NFCs can be specified (Requirement S4 satisfied). Subjects (called match-points) for the NFC are arbitrary FCs. Since this work is general and on the requirement level no web service specific subjects are taken into consideration (Requirements S3, S5, S6 not addressed).

In some works only a single concern is considered. Fox and Jürgens [14] use composition filters at the modeling level in order to specify the weaving order of a security concern to its base concern and thus fulfills Requirement S4.

Another group of works uses UML profiles for modeling non-functional attributes. For instance, the UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [27] describes a set of QoS characteristics such as throughput, latency, integrity, dependability, among others. The characteristics can also be parameterized with for example the type of keys or types of encryption algorithms in case of the security characteristics (Requirements S1 and S2 satisfied). For all UML profiles based approaches it is generally possible to apply the stereotypes and its tagged values to arbitrary UML meta classes, e.g. Class or Operation (and thus satisfying S3). The ordering of stereotypes cannot be expressed (S4) but composition-related requirements can theoretically be realized through behavioral diagrams like activity diagrams or state diagrams (hence, S5 and S6 are partly satisfied).

Ortiz and Hernandez [28] incorporate extra-functional properties into UML models by the use of UML stereotypes. E.g., there are stereotypes for encryption, decryption and logging which can be added to classes or operations that are representing services (Requirement S2 and S3). The non-functional requirements and attributes cannot be described (S1). In that work the term extra-functional properties corresponds more to our term of NFAs rather than to non-functional attributes. The work considers neither the ordering of actions nor composite web services (S4-S6 not fulfilled).

4.1.2 *Specification for composite Web Services (Gray Box View)*

The work of Chollet and Lalanda [8] presents a generative framework to address NFCs for web service compositions. As part of the framework they separate the orchestration meta-model from other non-functional meta-models. A non-functional meta-model for security is provided where concepts exist for requirements and how these requirements are ensured (Requirement S1 and S2 satisfied). The NFRs are then added to the composition model by adding annotations (Requirement S5 supported). For example one can specify that an email sending task must be confidential. Service subjects cannot be specified because only process tasks are valid targets for annotations (Requirement S3 not supported). Requirement S4 has not been addressed because one cannot model the execution order between actions that enforce a requirement specified by an annotation. Also S6 is not supported because with means of simple annotations only punctual requirements can be expressed (and not stateful ones).

Weber et al. [37] is more focused on the process level and thus allows for the composition of BPMN [38] processes with so called delivery components that are in our terms stateful NFCs. The authors describe the lifecycle of stateful NFCs in terms of state diagrams that can be composed with functional BPMN processes. The composition is realized by defining a certain state of a stateful NFC as precondition or postcondition of a BPMN task and the result of the composition is a new BPMN process that contains the transitions (represented by tasks) that are necessary to reach the state that has been specified. That approach supports only the requirements S5 and S6 but does not allow for a web service as black box definition of NFCs, nor does it support any requirements definitions.

The AO4BPEL [7] deployment descriptor (DD) allows for specifying non-functional requirements for BPEL processes. First the service (one of logging, security, transactions and reliable messaging) is selected. Then the class (e.g. authentication or confidentiality for security) can be specified. The type (e.g. encryption or decryption for confidentiality) is then the concrete action that must be executed in order to enforce the requirement. Then a selector can be defined such as for a certain BPEL activity. Each selector can be mapped to a requirement. The service concept of

	[34]	[4]	[31]	[14]	[27]	[28]	[8]	[37]	[7]	[6]
S1	−	+	+	−	+	−	+	−	+	−
S2	+	+	−	+	+	+	+	−	+	+
S3	+	+	−	−	+	+	−	−	−	+
S4	−	−	+	+	−	−	−	−	−	−
S5	−	−	−	−	(+)	−	+	+	+	+
S6	−	−	−	−	(+)	−	−	+	−	−

Table 3: State of the Art NFC Specification

the DD corresponds to the concern in our terminology and the different classes match with our non-functional attribute definition. Thus, with the DD requirements can be specified and in addition the concrete actions (corresponding to our NFAs) can be selected, hence Requirement S1 and S2 are solved. The selector does not allow to specify web service subjects for the selected requirements (S3 not satisfied) and it is not possible to customize the ordering of the NFAs (S4 not supported). However, S5 is supported because selectors can select arbitrary process activities. S6 is not fully supported by the DD because the details of the NFC states are hidden. For example a transaction can be started and committed during a scope activity is executing and all nested activities will automatically participate in that transaction. It is not possible to arbitrarily define which states of the transaction apply to which activity.

In Table 3 an overview of the works and the grade (+ supported, (+) partly supported, − not supported) of the specification requirements fulfillment is given.

4.2 NFC Enforcement

In academia and industry there are two prominent strategies to deal with the enforcement of NFCs:

- Use Variations of the Interceptor Pattern
- Aspect-Oriented Programming (AOP)

Both strategies are mainly motivated by the challenge to deal with the crosscutting nature of NFCs. Thus, they both provide an answer to the question how NFCs can be modularized and integrated with FCs in a transparent way, so that the FCs do not have to be changed at all.

4.2.1 Interceptor Pattern

The application of the interceptor pattern (aka Chain of Responsibility pattern [15]) is often employed in message-oriented middleware and allows for the augmentation of additional functionality in a transparent way. In web service frameworks the term handler (Apache Axis2 [1]) or filters (e.g. in the Java Servlet Specification [11]) is often used synonymously. The interceptor pattern has been adopted by many frameworks that are used in industry. The most prominent example from the Java world is the JEE Platform [29] where a user can specify interceptors in a declarative way by using annotations that can either be applied to Session or Message-driven Beans.

4.2.2 AOP for Web Services (Black Box View)

AOP on the other hand introduces new language concepts for the modularization of crosscutting concerns. In AOP, there are new means of modularization in form of aspects. An aspect can be used to encapsulate crosscutting behavior and thus is a perfect match to provide programs with a clear separation of functional and non-functional concerns (Requirement E1). The integration of aspect behavior into the functional behavior can be done without changing the functional code by means of pointcuts (Requirement E2). Pointcuts can be understood as a facility to query over join points. Join points on the other hand are well-defined points during the execution of a program (e.g. an execution of an operation). In a single query, multiple join points can be selected (quantification, Requirement E3) and multiple aspects can point to the same join point (superimposition, Requirement E4). Requirement E5 and E6 cannot be addressed by AOP in general it rather depends on the concrete implementation.

Henkel et al. [17] use AOP technology, namely AspectJ [22], in order to implement QoS monitoring without changing the existing service implementation. They have written a performance aspect that measures the execution time by defining two join points, one at the beginning and one at the end of a method call. The advice captures the time when the request arrives at the service and when the response is returned by the service. Further, they implemented a cost aspect that counts the types of service invocations and cumulates them to calculate the monetary total cost of usage. The reliability aspect logs every invocation that ends with an exception. In conclusion, they showed exemplary how QoS metrics can be measured when applying aspect technology to the implementation of the service. The approach since relying on AOP satisfies Requirements E1 - E4, but E5 and E6 are not sufficiently addressed because the solution is restricted to Java (AspectJ is a Java language extension) and AspectJ does not support remote pointcuts. Thus, one cannot quantify over a set of distributed web services. Also composite web services are not taken into account (Requirement E7).

Nishizawa et al. [26] propose the concept of remote pointcuts. With this approach it is possible to select join points on different hosts. Therefore, Nishizawa introduces a new language similar to AspectJ called DJcutter that is based on Java. The language allows for restricting pointcuts to specific hosts by introducing a new poincut designator `hosts(Host ....)` and additional reflection data about the host the pointcut matched with. DJcutter is restricted to Java and Nishizawa et al. verified it with Java RMI. Thus, the approach could also be transferred to web services (E1-E4 only partly supported). In summary, DJcutter supports Requirement E5, but E6 still remains open since DJcutter is based on Java and hence is not applicable to arbitrary programming platforms, e.g. web services implemented in other programming languages cannot use DJcutter.

Singh et al. [30] introduce the Aspect-Oriented Web Services (AOWS) model. In their approach an AOWebService Requester consumes a AOWebService using an AOConnector. When the consumer needs multiple services to fulfill his request and thus multiple AOWebService providers are involved, the relevant providers are selected and bundled by an AOComposite. An AOWebService provider registers his service by depositing the AOWSDL interface at the AOUDDI. AOWSDL comprises a technical and a human readable descriptor specifying functional and non-functional properties of web service interfaces. It is modeled as a set of AOComponents each containing a set of functional and non-functional aspects. The AOUDDI registry locates appropriate AOAdaptors that are used by the AOConnector to serve the request with the proper protocol (e.g. Business Transaction Protocol (BTP)). Hence, the AOConnector is the key role for separation of concerns. The AOWS subsystems have been modeled, analyzed, and verified in Alloy [20] which is a first-order logic based structural modeling language for expressing complex structural constraints and behaviors. In summary, Singh et al. provide a heavy-weight aspect-oriented extension to the well-known web service roles and specifications. Therefore, no specific programming language must be assumed (E6 satisfied and E1-E4 supported through use of AOP).

Nabor C. Mendonça et al. [24] introduce the Web Service Aspect Language (WSAL) which provides aspects that can be freely specified, implemented, deployed and executed as loosely coupled web services and thus are implementation technology-independent. Weaving is realized at network level by Web intermediary technology. Supported join point types are message part, service name, service operation, service location, client location and combinations of them. Advice are implemented by WS operations and the following types are allowed: `before request`, `upon request`, `after response`, `upon response`, `after exception`, `upon exception`, and `around`. Additional context information like the intercepted message or the service location can be passed to the advice implementation operation. The context information can be used to restrict the information that is transmitted to the advice WS. Some advice types, e.g., before and after can be executed in parallel with the join point. Aspects are specified by an XML syntax and contain a set of join point and advice elements. The aspect weaver is implemented by using IBM WBI [19] intermediary technology. In summary, Nabor C. Mendonça et al. provide a platform-independent (Requirement E6) solution that also supports distributed join points through the service location type (Requirement E5), but they do not consider composite web services (Requirement E7 not supported).

Verheecke et al. [36] propose the Web Services Management Layer (WSML) for the dynamic integration, selection, composition, and client-side management of web services in Service-Oriented Architectures (SOA). There are different aspect types. Service Redirection Aspects specify the communication logic to swap from a web service to an alternative one or to a web service composition implementing the same logic by calling multiple services. Service Selection Aspects enforce a certain selection policy which can be of one of the types Client-initiated selection, Non-functional service property based selection, Service behavior-based selection and Service-initiated selection. Service Management Aspects modularize concerns like encryption, billing, reliable messaging, transactions, etc. which can be enforced per service type, per composition or per web service. The aspects are implemented by JAsCo [33], an adaptive programming approach which supports Superimposition (E4) by Combination Strategies. Combination Strategies is an imperative approach that allows for employing the full expressiveness of Java.

Some works combine the AOP and handler approach. For example, Baligand and Monfort [3, 18] use a single global handler that is restrained to SOAP logic. In addition, they use a bytecode modification of the web service stubs to select appropriate aspects that enforce the requirements specified by WS-Policy. In this work, policy and aspects are used to increase web service (WS) adaptability. It is stated that existing WS technology, exemplified by handlers, is not policy-adaptable which is a requirement when policies have to be changed at runtime or the same WS must handle different policies from different clients. The answer to this problem given by Baligand and Monfort is to replace the multiple specific handlers by a single global handler whose role is restrained to SOAP logic. Weaving of aspects is applied at load time in the WS stub. A policy engine selects the appropriate aspects to enforce the requirements defined by the policy. The solution is implemented with Tomcat and Apache Axis and modifies the bytecode of the WS stub through a modification of Tomcat's class loader.

The work of Baligand and Monfort integrates declarative policies by using handler technology. With this approach the implementation of the web services does not have to be changed, thus adaptability at runtime is facilitated. The handlers are written in Java and are integrated tightly with the SOAP framework making their solution technology dependent. The solution cannot be reused for different technology setups (e.g., Java, .NET) and hence Requirement E5 is not satisfied.

4.2.3 Enforcement for composite Web Services (Gray Box View)

Other works focus solely on composite web services using WS-BPEL and hence address Requirement E7.

Wohlstaedter et al. [39] introduce a new service-oriented middleware architecture for runtime web services interoperability facilitating clients to use middleware as a service. In their architecture, which is called Cumulus, the interoperability requirements of a web service are expressed by WS-Policy (describes which interoperability protocol should be used, e.g., WS-BA) and WS-PolicyAttachment (describes at which service or which BPEL activity the requirement occurs). These requirements are enforced by middleware (mw) services that are described by metadata (for example the implemented protocols) that facilitates mw service selection. Therefore, a mw service registry matches effective policies of a client with mw service metadata. When a matching mw service has been discovered, it will be applied using the so-called mw service injection protocol (MIP) which is provided through an extra port type by each mw service. The bind operation defined by the MIP takes as an input the policy assertion that was used to discover the mw service and returns a session identifier, a specific message context and information if the mw service should be added as intermediary in the incoming or outgoing message flow. A corresponding unbind operation unregisters the client from the mw service. Additionally, there are two operations to deliver incoming or outgoing messages to the mw service. If several mw services are bound then a static ordering of policy assertion types is used. The architecture has been applied prototypical to BPEL (Cumulus4BPEL). In this implementation, the BPEL engine must interpret additional local handler wrappers for activities (e.g. the scope activity) and hence supports Requirement E6 partly. Distributed join points (Requirement E5) are also not supported.

AO4BPEL [7] is an aspect-oriented extension to WS-BPEL which allows to define aspects that enforce NFCs such as security, reliable messaging and transactions. An aspect in AO4BPEL is a container for multiple pointcuts and advice constructs and can be activated at runtime. The pointcut language is XPath [10] which allows for selecting arbitrary

	[17]	[26]	[30]	[24]	[36]	[3]	[39]	[7]	[6]
E1	+	(+)	+	+	+	+	+	+	+
E2	+	(+)	+	+	+	+	−	+	+
E3	+	(+)	+	+	+	+	−	+	+
E4	+	(+)	+	+	+	+	−	(+)	−
E5	−	+	−	+	−	−	−	−	−
E6	−	−	+	+	−	−	(+)	(+)	+
E7	−	−	−	−	−	−	+	+	+

Table 4: State of the Art NFC Enforcement

BPEL activities. When a pointcut matches a certain join point its corresponding advice is executed. An advice is defined in BPEL and can be used to call dedicated middleware services in order to enforce NFRs. A security, reliable messaging and transaction middleware service are provided and predefined aspects for its integration into BPEL processes exist that are generated from an deployment descriptor.

AO4BPEL strictly follows the separation of concerns principle because the code for the enforcement of NFCs is modularized into middleware services (Requirement E1). The integration of FCs (the BPEL process) and NFCs is achieved by pointcuts (Requirement E2) and the code is modularized by aspects. Quantification is supported by the pointcut language which allows also to quantify over more than one process (Requirement E3). Superimposition is supported by a static ordering mechanism similar to AspectJ so that each concern has a predefined priority. It is not applicable to change the ordering individually. This can only be achieved by adapting the generated middleware aspects (Requirement E4 partly solved). Requirement E6 is partly solved. On one hand the middleware services are implemented as web services and thus are per se independent of the programming language, on the other hand it is presumed that the web service exposes its composition logic as BPEL (Requirement E7 satisfied). Hence, AO4BPEL is not able to enforce NFCs for black box web services. The integration of aspects with distributed web services is not possible in AO4BPEL (E5 not supported).

In [6] Charfi et al. present an approach to NFCs in BPEL processes based on external policy attachment. An XPath based selector is used to select activities that share a common non-functional requirement and a policy is associated with that selector. The enforcement of the requirements is done by a SOAP middleware developed in the context of the project Colombo.

In Table 4 an overview of the works and the grade (+ supported, (+) partly supported, − not supported) of the enforcement requirements fulfillment is given.

5. CONCLUSIONS AND OUTLOOK

For the composition of web services and NFCs both the specification and enforcement problems have to be addressed. In this work we identified requirements for both problem domains, analyzed the state of the art and came to the following results.

There are many works that address the NFC specification problem in general. One cluster of these works considers web services as black boxes (the interface view). Another cluster looks at web services as gray boxes (where the composition is visible).

In the first cluster, some works focus only on requirements engineering ([31, 9]), whilst others consider the actions by which those requirements are enforced ([34, 35, 28]). In addition, there are some works regarding both requirements and actions ([4, 27]).

In the second cluster, the two most relevant works are those of Chollet and Lalanda [8] and Weber et al. [37]. The former presumes a technical BPEL orchestration, whereas the latter operates on BPMN level and considers stateful NFCs from the business level.

With respect to enforcement most works are based either on AOP or variations of the interceptor pattern. Works in the field of AOP can be categorized into those works that apply AOP at the implementation level ([17]), those that extend the web service framework (e.g., its roles and technical specifications) with aspect-oriented concepts ([30]), and those that introduce new aspect languages ([24, 7]).

In summary, none of the investigated works provides a sufficient solution to most of the specification and enforcement requirements. Thus, there is a need for a holistic solution for both the specification and enforcement, which supports the identified requirements and copes with both the black box and the gray box views of the service. We already started to work on such an approach, which supports the modeling of concerns and their attributes as well as the composition of actions and the definitions of their interdependencies. Furthermore, this approach supports the composition of NFAs with services and the subsequent generation of enforcement code.

6. ACKNOWLEDGEMENTS

This work has been partially supported by the German Federal Ministry of Education and Research (BMBF) in the project Premium Services (01IA08003A).

7. REFERENCES

[1] Apache Axis2. http://ws.apache.org/axis2/, 2010.

[2] Alves et al. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, April 2007.

[3] F. Baligand and V. Monfort. A concrete solution for web services adaptability using policies and aspects. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 134–142, New York, NY, USA, 2004. ACM.

[4] M. Beisiegel, N. Kavantzas, A. Malhotra, G. Pavlik, and C. Sharp. SCA Policy Association Framework. In A. Dan and W. Lamersdorf, editors, *ICSOC*, volume 4294 of *Lecture Notes in Computer Science*, pages 613–623. Springer, 2006.

[5] L. Bergmans and M. Aksit. Composing Multiple Concerns Using Composition Filters. In *Communications of the ACM*, 2001.

[6] A. Charfi, R. Khalaf, and N. Mukhi. QoS-Aware Web Service Compositions Using Non-intrusive Policy Attachment to BPEL. In *ICSOC*, pages 582–593, 2007.

[7] A. Charfi, B. Schmeling, A. Heizenreder, and M. Mezini. Reliable, Secure, and Transacted Web Service Compositions with AO4BPEL. In *ECOWS*, pages 23–34. IEEE Computer Society, 2006.

[8] S. Chollet and P. Lalanda. An Extensible Abstract Service Orchestration Framework. In *Proceedings of*

the 2009 IEEE International Conference on Web Services, 2009.

[9] L. Chung and J. C. Prado Leite. On non-functional requirements in software engineering. pages 363–379, 2009.

[10] J. Clark and S. DeRose. XML Path Language (XPath)Version 1.0, November 1999.

[11] D. Coward and Y. Yoshida. Java Servlet Specification, Version 2.4. Technical report, Sun Microsystems, 2003.

[12] D. Davis, A. Karmarkar, G. Pilz, and Ümit Yalçinalp (Eds.). Web Services Reliable Messaging Policy Assertion (WS-RM Policy) Version 1.2, February 2009.

[13] R. Filman and D. Friedman. Aspect-oriented programming is quantification and obliviousness. In *Workshop on Advanced Separation of Concerns in conjunction with OOPSLA*, pages 21–35, October 2000.

[14] J. Fox and J. Jürjens. A Framework for Analyzing Composition of Security Aspects. In *MMOSS*, 2006.

[15] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[16] O. M. Group. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. Technical report, November 2007.

[17] M. Henkel, G. Boström, and J. Wäyrynen. Moving from Internal to External Services Using Aspects. *Interoperability of Enterprise Software and Applications*, pages 301–310, 2006.

[18] M. Hmida, R. Tomaz, and V. Monfort. Applying AOP concepts to increase Web services flexibility. In *Next Generation Web Services Practices, 2005. NWeSP 2005. International Conference on*, pages 6 pp.–, 2005.

[19] IBM Research. Web Intermediaries (WBI). http://www.almaden.ibm.com/cs/wbi/.

[20] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.

[21] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2003.

[22] R. Laddad. *AspectJ in Action*. Manning Publications, 2003.

[23] K. Lawrence, C. Kaler, A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist. WS-SecurityPolicy 1.3. OASIS Standard, February 2009.

[24] N. C. Mendonça, C. F. Silva, I. G. Maia, M. A. F. Rodrigues, and M. T. de Oliveira Valente. A Loosely Coupled Aspect Language for SOA Applications. *International Journal of Software Engineering and Knowledge Engineering*, 18(2):243–262, 2008.

[25] I. Nagy, L. Bergmans, and M. Aksit. Composing Aspects at Shared Join Points. In R. Hirschfeld, R. Kowalczyk, A. Polze, and M. Weske, editors, *NODe/GSEM*, volume 69 of *LNI*, pages 19–38. GI, 2005.

[26] M. Nishizawa, S. Chiba, and M. Tatsubori. Remote pointcut: a language construct for distributed AOP. In *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 7–15, New York, NY, USA, 2004. ACM.

[27] OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, June 2004.

[28] G. Ortiz and J. Hernandez. A Case Study on Integrating Extra-Functional Properties in Web Service Model-Driven Development. In *ICIW '07: Proceedings of the Second International Conference on Internet and Web Applications and Services*, page 35, Washington, DC, USA, 2007. IEEE Computer Society.

[29] B. Shannon. Java Platform, Enterprise Edition (Java EE) Specification, v5. Technical report, Sun Microsystems, 2006.

[30] S. Singh, J. Grundy, J. Hosking, and J. Sun. An Architecture for Developing Aspect-Oriented Web Services. In *ECOWS '05: Proceedings of the Third European Conference on Web Services*, page 72, Washington, DC, USA, 2005. IEEE Computer Society.

[31] E. Soeiro, I. S. Brito, and A. Moreira. An XML-Based Language for Specification and Composition of Aspectual Concerns. In Y. Manolopoulos, J. Filipe, P. Constantopoulos, and J. Cordeiro, editors, *ICEIS (3)*, pages 410–419, 2006.

[32] W. M. P. van der Aalst, A. J. Mooij, C. Stahl, and K. Wolf. Service Interaction: Patterns, Formalization, and Analysis. In M. Bernardo, L. Padovani, and G. Zavattaro, editors, *SFM*, volume 5569 of *Lecture Notes in Computer Science*, pages 42–88. Springer, 2009.

[33] W. Vanderperren, D. Suvée, B. Verheecke, M. A. Cibrán, and V. Jonckers. Adaptive programming in JAsCo. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 75–86, New York, NY, USA, 2005. ACM.

[34] A. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and Ü. Yalçinalp. Web Services Policy 1.5 – Framework. Technical report, W3C, 2007.

[35] Vedamuthu et al. WS-PolicyAttachment. http://www.w3.org/TR/ws-policy-attach/, September 2007.

[36] B. Verheecke, W. Vanderperren, and V. Jonckers. Unraveling Crosscutting Concerns in Web Services Middleware. volume 23, pages 42–50, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[37] I. Weber, A. Barros, N. May, J. Hoffmann, and T. Kaczmarek. Composing Services for Third-party Service Delivery. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 823–830, Washington, DC, USA, 2009. IEEE Computer Society.

[38] S. A. White. Business Process Model and Notation (BPMN) 1.2. http://www.omg.org/spec/BPMN/1.2/, January 2009.

[39] E. Wohlstadter, S. Tai, T. Mikalsen, J. Diament, and I. Rouvellou. A Service-oriented Middleware for Runtime Web Services Interoperability. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 393–400, Washington, DC, USA, 2006. IEEE Computer Society.