**Institute of Architecture of Application Systems**

# Maintaining Compliance in Customizable Process Models

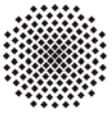
Daniel Schleicher, Tobias Anstett, Frank Leymann, and Ralph Mietzner

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{schleicher, anstett, leymann, mietzner}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings {INPROC-2009-70,
   author = {Daniel Schleicher and Tobias Anstett and Frank Leymann and Ralph
Mietzner},
   title = {{Maintaining Compliance in Customizable Process Models}},
   booktitle = {Proceedings of the 17th International Conference on COOPERATIVE
INFORMATION SYSTEMS (CoopIS 2009)},
   editor = {Robert Meersman and Tharam Dillon and Pilar Herrero},
   address = {Heidelberg},
   publisher = {Springer Verlag},
   institution = {Universit{\"a}t Stuttgart, Fakult{\"a}t Informatik,
Elektrotechnik und Informationstechnik, Germany},
   series = {Lecture Notes in Computer Science},
   volume = {5870},
   pages = {60--75},
   month = {November},
   year = {2009},
   isbn = {978-3-642-05147-0},
   keywords = {Compliance; Business process modeling},
   language = {Englisch},
   cr-category = {H.4.1 Office Automation}
}
```





Universität Stuttgart
Germany

# Maintaining Compliance in Customizable Process Models

Daniel Schleicher, Tobias Anstett, Frank Leymann, and Ralph Mietzner

Institute of Architecture of Application Systems,
University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{schleicher, anstett, leymann, mietzner}@iaas.uni-stuttgart.de

**Abstract.** Compliance of business processes has gained importance during the last years. The growing number of internal and external regulations that companies need to obey has led to this state. This paper presents a practical concept of ensuring compliance during design time of customizable business processes.
We introduce the concept of a business process template that implicitly contains compliance constraints as well as points of variability. We further present an algorithm that ensures that these constraints cannot be violated. We also show how these algorithms can be used to check whether a customization of this process template is valid regarding these compliance constraints. So the designer of a business process, in contrast to the template designer, does not have to worry about compliance of the eventual process.
In a final step we show how these general concepts can be applied to WS-BPEL.

## 1 Introduction

Today compliance is a more important quality of service (QoS) requirement for organizations than it was in the past. As a consequence from financial and other scandals, organizations are forced to ensure the compliance to more and more regulations. These regulations are not only imposed by laws such as the Sarbanes–Oxley Act (SOX) [15] which regulates financial transactions, but also corporate self-commitments and social expectations such as realizing a *green IT*.

Regulations apply to the way a company does business by means of how their business processes are executed. Thus being compliant requires rethinking of the way business processes are managed in respect to regulations. This may include the installation of new roles in the organizational structure such as compliance-officers responsible for analyzing compliance requirements to capture the ones relevant for the organization. It also may require changes to the IT infrastructure executing the business process to provide enough evidence to (external) auditors responsible for assessing the compliance. Furthermore there is a need for specialized process designers which are able to map the identified regulations to abstract business process models which allow further customizations by domain experts. During

these customizations the real business logic is added. These abstract business process models are further referred as *compliance templates* which basically realize reference process implementations of one or more compliance constraints.

The complexity of real world business processes and the poor tool support today makes it hard for process designers to build processes in a way that is common for traditional programming languages like Java. Thus humans should not need to bother with compliance constraints besides the business logic of a process. What is needed is support of the process designer during design-time to automatically check compliance concerns so that the designer could be absolutely sure, the eventual process will be compliant to certain compliance constraints. Further requirements are performance and scalability of the algorithms, which check the compliance of a process. It is not reasonable for the process designer to wait 5 minutes after inserting a new activity into the process for the validation to finish.

A company may employ their own compliance-officers and specialized process designers to develop compliant processes but also has the option to outsource these tasks. 3rd party standardization organizations as well as software as a service providers may offer compliance templates which may be further customized to execute them in their own infrastructure or in the cloud. One big problem of customizations of a predefined compliance template is that the implicitly contained compliance constraints might be compromised which would cause a violation of the identified regulations at execution time. Thus there is a need to detect those malicious customizations and restrict the modeler of being able to use them.

In this paper we present an algorithm that ensures that compliance constraints of (customizable) compliance templates can not be compromised. We introduce a motivating example in Section 2 which illustrates the problems we address and which is used throughout the paper. We then present an approach how to ensure compliance using compliance templates comprising an abstract business process, variability descriptors and compliance descriptors in Section 3. In Section 4 we present formal definitions of compliance assurance rules which are part of compliance descriptors to implement a compliance validation algorithm. The introduced concepts are then applied to WS-BPEL [9], the standard for modeling and executing business processes in service oriented environments, in Section 5. Finally we compare the presented approach with related work in this field and finish with a conclusion and an outlook to future work.

## 2 Motivating Example

In this Section we show the motivation for this paper by means of a SaaS (Software as a Service) [14] scenario. We assume a fictional SaaS provider, *John Doe IT Service* offers a template for a loan approval process. This template will be completed by the customers of John Doe. The template implicitly comprises some compliance rules, like "two approve activities which must be executed in sequence" or "a credit check activity must be executed at the end of the process".

After the finalization the fully functional business process is deployed on John Does infrastructure. During the process of completing the template, activities, which are being inserted, could violate some of the contained compliance rules. For example an inserted activity could end the process before a credit check activity was executed.

In this paper we provide an approach to support the process designer in order to complete the template to a fully compliant business process.

## 3 Controlling Compliance with Process Templates

Compliance of a business process can be verified in numerous ways. There are approaches, which use monitoring to ensure the compliant execution of a business process [12]. Another approach [3] uses deontic logic to model obligations and permissions, which can then be used in the design phase of a business process to verify the compliance of the process. Our approach differs from the approaches above in two ways. First of all we provide an abstract business process, which already contains all activities important for compliance. This approach is very powerful because one can model a big set of compliance constraints within an abstract business process. This abstract process then will be completed by the process designer. Secondly to ensure that no activity, inserted by the process designer, violates the compliance of the abstract process, on the one side we constrain the set of activities, which can be inserted, and on the other side we provide a set of rules, which ensure, that an inserted activity cannot violate the compliance. In this section we show how to ensure compliance of customizable business processes by means of compliance templates (see Figure 1). A compliance template comprises three parts, (i) an *abstract business process*, (ii) *variability descriptors* and (iii) *compliance descriptors*. The abstract business process implicitly contains compliance rules. These are the rules the resulting business process should comply with after being completed by a process designer. The business activities contained within the abstract business process implement these rules. Additionally the abstract business process defines placeholders that need to be completed before the abstract process can be executed. Placeholders are described and restricted in more detail by the variability descriptors that are included in the compliance template. These variabilities are independent from the abstract business process and can differ for different application domains. Compliance descriptors describe additional compliance rules that customizations of the abstract process must comply with and that cannot be specified in the abstract process directly. In the following each of the parts of a compliance template are described in detail.

### 3.1 Abstract Business Processes

An abstract business process is a business process that contains so-called *opaque activities* and *compliance activities*. Opaque activities are used as placeholders that need to be replaced with concrete constructs during the customization to fill

the abstract business process with concrete business logic and make the process executable. Compliance activities are activities in the business process that are not allowed to be removed or altered otherwise. They implement the compliance requirements which the compliance template should fulfill.

Figure 1 shows an abstract business process as a template for a loan approval process, a compliance descriptor, and a variability descriptor of a compliance template. The dotted arrows show exemplarily what alternative, compliance link, and compliance assurance rules is applied to which activity. The activities with label *1st decision* and *2nd decision* are compliance activities. Opaque activities are labeled *Opaque*.

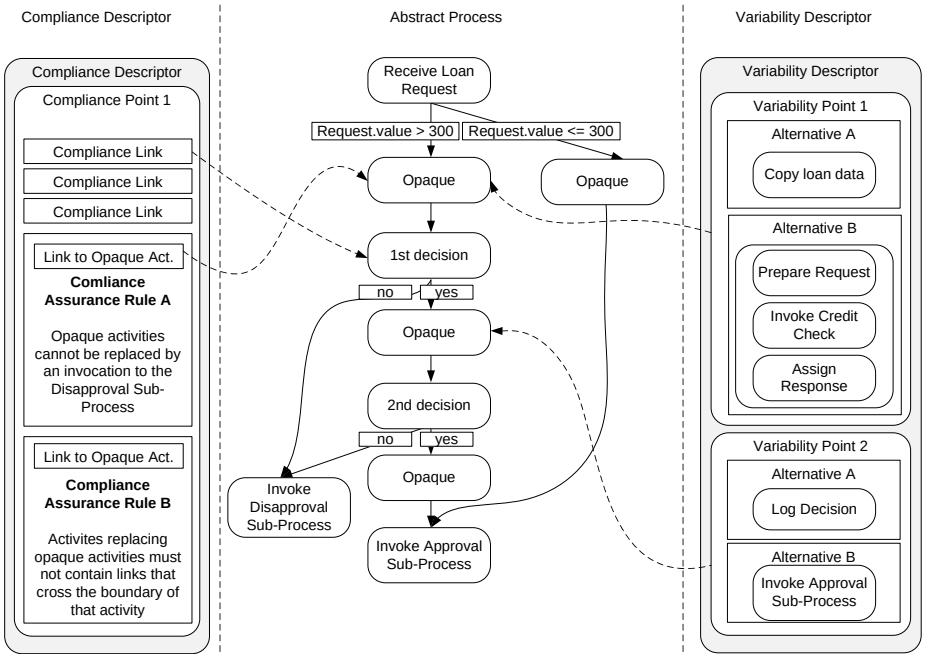



**Fig. 1.** Loan approval process with compliance descriptor and variability descriptor

### 3.2 Variability Descriptors

Variability descriptors [7, 8] are a means to describe variability for applications. Unlike other approaches [13, 5, 11] they are not focused on process models alone but whole applications. Variability descriptors can be used to describe variability across all layers of the application ranging from the GUI over the workflow layer down to individual services and database schemes.

Besides documenting variability, variability descriptors can be transformed into workflows [8] that can be used to guide users through the customization of

applications. They present the user with choices and abstract from the concrete implementation making it easier for non-developers to customize the applications.

In short a variability descriptor consists of a set of *variability points*. A variability point consists of a set of *locators* and a set of *alternatives*. Locators point into artifacts of the application such as a BPEL file or an HTML file which should be affected by that variability point. Alternatives describe alternative values for that variability point. Alternatives can be one of the following types.

- *explicit*, meaning that the concrete value is specified,
- *empty*, meaning that the code the locator points to is deleted,
- *free*, meaning that a user is prompted to enter any text that should go where the locator points to, or
- *expression*, meaning that the value where the expression points to or that is calculated during the evaluation of the expression, is inserted at the location the locator points to. An expression could be for example "insert the value inserted at another point in the document times 2".

Variability points are connected by *dependencies*. Dependencies describe the order in which variability points must be bound. Additionally to describe more advanced dependencies, so-called *enabling conditions* can be used to enable certain activities based on a condition. Using dependencies and enabling conditions it is possible to describe complex dependencies such as "after alternative A of variability point V1 has been bound only alternative C and not alternative D of variability point V2 can be bound".

### 3.3 Compliance Descriptors

Similar to variability descriptors explained in Section 3.2 *compliance descriptors* are defined. Compliance descriptors are specified independently from the abstract business process they belong to. This facilitates the reuse of compliance descriptors. For example an abstract business process, which implements a certain separation of duty constraint, can be reused several times in different compliance templates because the compliance descriptors are not tied to this abstract business process.

Compliance descriptors (see Figure 1) consist of so-called *compliance points*. Every compliance point expresses one compliance requirement for the business process. A compliance point consists of one or more *compliance links* and *compliance assurance rules*. Compliance links are used to mark the activities of an abstract process, which belong to a compliance point. These activities are called *compliance activities*. Activities marked this way should not be altered in any way in order to preserve the compliance of the abstract business process. When editing a process with a graphical tool, compliance activities, with a compliance link pointing on them, can for example be coloured red to show that these activities can not be altered in any way. Listing 1.1 shows the XML representation of a compliance link. The *document*-function of XSLT is used to point to the file where the corresponding activity is located. In this case it is an XML document that describes the abstract business process (e.g., a BPEL file). Additional XPath functionality is used to navigate inside the document to the designated activity.

**Listing 1.1.** Compliance Link

```
<ComplianceLink>
    document("loanApproval.bpel")//bpel:opaqueActivity[@name="payment"]
</ComplianceLink>
```

A compliance assurance rule consists of the rule itself and a set of links to opaque activities of the abstract business process contained in the compliance template. In this way the rule is applied to a number of opaque activities. Compliance assurance rules describe which alternatives are not allowed to be put into an opaque activity. This ensures that the corresponding abstract process template stays compliant after inserting an alternative from a variability descriptor.

A compliance descriptor has one compliance point. The duty of this compliance point is to make sure, that a loan request is approved by two different persons. So there are two compliance links to two activities in the abstract business process, which implement the approve functionality of the process. These activities are now marked as unchangeable. Further there is a set of compliance assurance rules. These rules are written to make sure, that no activity inserted into an opaque activity can bypass one of the approve activities. For example one of these rules would permit that links, from activities replacing opaque activities, bypass activities marked by compliance links. Further examples for compliance assurance rules follow in Section 4.

## 4 Checking Variability Against Compliance Rules

There are two rule sets. Compliance rules implicitly defined in the abstract business process of a compliance template and *compliance assurance rules* which are part of a compliance descriptor. Compliance assurance rules ensure that a compliance template could not be completed in a way that destroys one or many compliance rules which are implicitly implemented in the abstract business process of a compliance template. The following Section introduces formal definitions which are then used to compose a formal representation of each compliance assurance rule. This formal representation is then used in Section 4.1 to implement a compliance validation algorithm.

### 4.1 Formal Definitions

In this section we define sets and functions for use in the compliance validation algorithm presented in Section 4.2. We also show a list of compliance assurance rules, which are essential to preserve compliance of a compliance template. For every compliance assurance rule a formal description is provided. This formal description facilitates the implementation of the rule in an algorithm.

In [4] the abstract syntax of WS-BPEL 2.0 is described. In the following we use some definitions from this document shown in short in the list below. Although the formal representation of a business process used in this paper is based on BPEL it is generic enough to be adapted for other modelling languages.

- $\mathcal{A}$ is the set of all activities in a process.
- $\mathcal{A}_{type}$ is the set of all activities of a certain type.
- $\mathcal{V}$ is the set of all variables of a process.
- $\mathcal{L}$ is the set of control links in a process.
- $\mathcal{C}$ is the transition condition of a link.
- $\mathsf{LR}$ denotes a link between two activities, a link can contain a transition condition:

$$\mathsf{LR} \subset \mathcal{A} \times \mathcal{L} \times \mathcal{C} \times \mathcal{A} \tag{1}$$

  For further information see [4].
- The function $\mathsf{descendants} : \mathcal{A} \rightarrow 2^{\mathcal{A}}$ returns the set of all descendants of an activity. Descendants of an activity $a \in \mathcal{A}$ are the activities that are nested within this activity.
- The function $\mathsf{partnerLink}_{\mathcal{CO}}$ assigns a partner link to a message construct.

**Definition 1 (Set of Replacement Activities).**
*The set of replacement activities $\mathcal{A}_{replacement}$ consists of business activities, which replace an opaque activity in the process of gaining an executable completion [9] out of the compliance template. An opaque activity can only be replaced by* one *replacement activity. This is necessary because an opaque activity can have one or more incoming and one or more outgoing links. These links need to be mapped to the activity replacing the opaque activity. However $\mathcal{A}_{replacement}$ could contain several other activities. For example, in figure 1 $\mathcal{A}_{replacement}$ in alternative A consists only of the activity* Copy loan data.

**Definition 2 (Replace Function).**
*$\mathcal{A}_{opaque}$ is the set of all opaque activities in a BPEL process. $\mathcal{E}$ is the set of all activities in a BPEL process except the opaque activities. The function $\mathsf{replace} : \mathcal{A}_{opaque} \rightarrow \mathcal{E}$ with $\mathcal{E} = \mathcal{A} \setminus \{\mathcal{A}_{opaque}\}$ replaces an opaque activity with a non opaque activity in order to become an executable completion of the abstract business process of a compliance template. The variability descriptor, shown in figure 1, shows all activities, which are available for replacement of opaque activities of the abstract business process. If an opaque activity can be replaced by a particular alternative of a variability descriptor, is verified by the algorithm presented in section 4.2.*

**Definition 3 (Variable Read Relation).**
*The relation $VR \subseteq \mathcal{A} \times \mathcal{V}$ shows that there is a read relation between an activity $a \in \mathcal{A}$ and a variable $v \in \mathcal{V}$.*

*In other words it shows that a variable $v$ is read by an activity $A$.*

**Definition 4 (Variable Write Relation).**
*The relation* $VW \subseteq \mathcal{A} \times \mathcal{V}$ *shows that there is a write relation between an activity* $a \in \mathcal{A}$ *and a variable* $v \in \mathcal{V}$.
*In other words it shows that a variable* $v$ *is written by an activity* $A$.

**Definition 5 (Set of Compliance Activities).**
*The set of compliance activities* $\mathcal{A}_{compliance}$ *comprises all activities with a compliance link pointing to. In figure 1 the activity labeled* 1st decision *is such a compliance activity because a compliance link from compliance point 1 is pointing to it.*

**Definition 6 (Compliance Template).**
*A compliance template* $\mathcal{T}$ *comprises three parts, (i) an* abstract business process*, (ii)* variability descriptors *and (iii)* compliance descriptors *as described in section 3.3 and shown in figure 1.*

Compliance assurance rules can be divided into two kinds. Rules which have to be applied to all compliance templates and rules which have to be applied to specific compliance templates to ensure certain requirements. Compliance Assurance rule 1 is an example for the first kind of rules.

The following list describes the compliance assurance rules which are applied to all compliance templates and provides a formal description of each rule. We do not argue that the provided rules are complete as compliance assurance rules can differ strongly for any application domain. However, these rules are standard rules applied to every compliance template. A violation of these rules would allow to circumvent or deactivate compliance activities at run-time.

1. **Opaque activities must not be replaced with exit activities.** This rule ensures that the execution of a business process could not be stopped by replacement activities and thus compliance rules could not be violated. This rule applies for every compliance template because a compliance rule, implicitly defined in an abstract business process of a compliance template, could easily be violated by inserting an exit-activity like the *exit*-activity in WS-BPEL.

$$\forall o \in \mathcal{A}_{opaque} : \mathsf{replace}(o) = \mathcal{A} \setminus \{\mathcal{A}_{exit}\} \tag{2}$$

   For instance, if any opaque activity in figure 1 would be replaced by an exit activity the loan approval process could be terminated abnormally. This is not intended because the compliance rules, implicitly contained within the abstract process, could then be violated.
2. **An alternative replacing an opaque activity must not contain links that cross the boundary of that opaque activity.** This rule ensures that compliance activities could not be bypassed. Links are only allowed between sub activities of replacement activities. This rule must also be applied to every compliance template because any compliance activity could be bypassed with a link from an inserted activity.

$$\forall r \in \mathcal{A}_{replacement}, \forall a \in \mathsf{descendants}(r), \forall a' \in \mathcal{A} \setminus \mathsf{descendants}(r) : \\ \neg\exists l \in \mathcal{L} : (a, l, c, a') \in \mathsf{LR} \vee (a', l, c, a) \in \mathsf{LR} \tag{3}$$

In figure 2 the irrelevant parts of the loan approval process are drawn transparent. As you can see an opaque activity has been replaced by the replacement activity *Copy loan data*. We assume that the alternative A containing this activity also contains a link to another activity. This link is labeled *Cross Boundary Link* in figure 2. It is not allowed to insert this link into the abstract process because it bypasses the activity labeled *2nd decision* and another opaque activity. Thus the compliant execution of the process template could no longer be guaranteed.

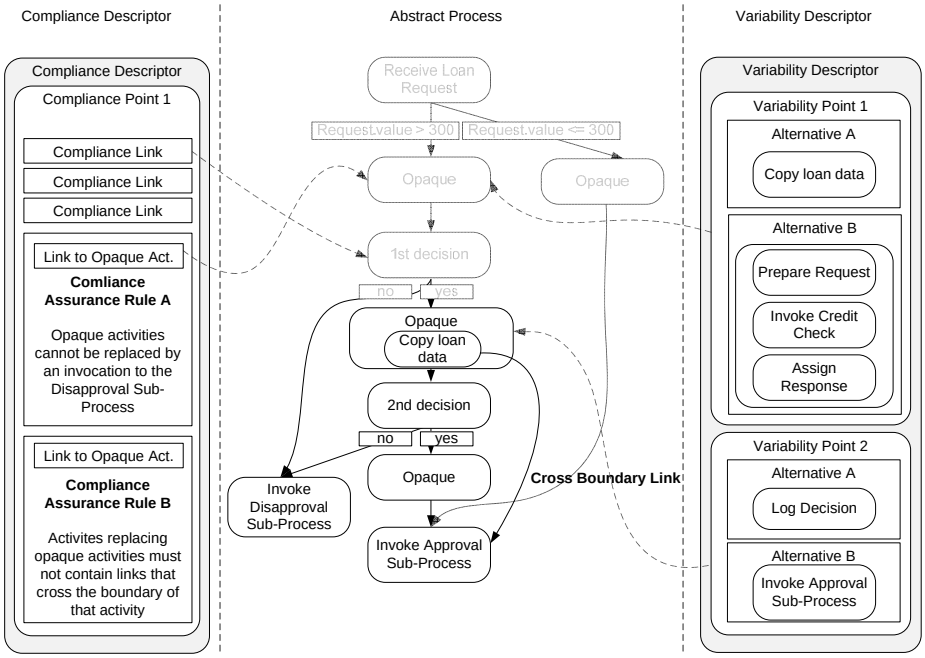



**Fig. 2.** Example for the use of links crossing the boundary of an opaque activity

3. **Variables that are read by one of the compliance activities must not be modified by replacement activities.** The possibility of data corruption through replacement activities is prevented with this rule. $\mathcal{V}_t$ denotes all variables which are part of the abstract business process of a compliance template. This rule must not necessarily be included in a compliance template. Structural compliance rules, like *activity A must be executed before activity B*, can not be violated by any write access of business activities to process variables.

The fact that write access to such variables is denied, is a small restriction for the process designer. In most processes it won't be necessary to write to a variable belonging to compliance activities, because then the process designer would want to change the execution behavior of the resulting process. It will be more likely that an extra set of variables is introduced, which is used by the replacement activities. However, it is possible to read from a Variable in $\mathcal{V}_t$.

$$\forall a \in \mathcal{A}_{compliance}, \forall r \in \mathcal{A}_{replacement}, \forall v \in \mathcal{V}_t : \exists VR \Rightarrow \neg\exists VW \tag{4}$$

As an example, in mind we add a variable with name *loan amount* to the abstract process shown in figure 1. This variable is written by the activity *Receive Loan Request*. If this variable is later in the process modified by a replacement activity (e.g. from the value 200 to value 500) some compliance rules, implicitly defined in the compliance template, could be violated. The process execution would take another branch than intended.

4. **Services which are being invoked from compliance activities must not be invoked from replacement activities.** This rule prevents violation of compliance rules through invocation of services which are used by compliance activities. This rule also must not be included in every compliance template because not every business process invokes other services.

$$\begin{aligned} \forall a \in \mathcal{A}_{compliance,invoke}, r \in \mathcal{A}_{replacement,invoke} : \\ \mathsf{partnerLink}_{\mathcal{CO}}(a) \neq \mathsf{partnerLink}_{\mathcal{CO}}(r) \end{aligned} \tag{5}$$

In figure 1, the activity *Invoke Approval Sub-Process* invokes a sub-process to approve the loan request. This sub-process is only invoked once during the execution of the loan approval process. If a replacement activiy would invoke the *Approval Sub-Process* before the activity *Invoke Approval Sub-Process* is executed, it could bypass compliance rules implicitly defined in the compliance template.

### 4.2 Validation Algorithm

A valid replacement for an opaque activity must not violate compliance rules defined in a compliance template. In this section we present an algorithm to verify if a compliance assurance rule is being violated by replacing an opaque activity with a certain alternative. The algorithm is written in Java-style pseudo-code. It is divided into two parts to reduce complexity and support readability.

An enterprise might have a lot of variability descriptors with even more alternatives for all kinds of purposes. The goal of the algorithm is now to select the valid alternatives for each opaque activity which do not violate compliance rules implicitly defined in the process template. So the process designer does not need to think about the validity of alternatives to be inserted into an opaque activity. Thus with this algorithm the compliance of the resulting executable

completion is preserved by the design tool and must not be considered by the process designer at design time. The designer can then fully concentrate on the business objectives rather than struggling with compliance concerns.

Algorithm 1 shows how opaque activities, alternatives, and compliance assurance rules are linked in data structures. The prerequisite for the algorithm is a list of opaque activities. Furthermore it shows how these data structures are traversed in order to validate an alternative. Essentially it iterates over all opaque activities, gets the alternatives for each opaque activity and validates each alternative against the rules which apply for the current opaque activity. One opaque activity object contains a list of alternatives and a list of compliance assurance rules.

**Algorithm 1** Calculate valid alternatives

```
1: //An opaque activity contains a list of alternatives and compliance assurance rules
Require: List opaqueActivities;
2: for opaqueActivity in opaqueActivities do
3:    List alternatives = opaqueActivity.getAlternatives();
4:    List complAssurRules = opaqueActivity.getAssurRules();
5:    for alternative in alternatives do
6:       for complianceAssuranceRule in complianceAssuranceRules do
7:          if not (complianceAssuranceRule.validate(alternative)) then
8:             alternatives.remove(alternative);
9:          end if
10:      end for
11:   end for
12: end for
```

Algorithm 2 is the continuation of algorithm 1. It shows a sample implementation for the validation of compliance assurance rule 1 "'Opaque activities must not be replaced with exit and opaque activities"' from section 4.1. In order to check if a business activity is an exit activity, we have to iterate over all business activities of an alternative inserted into the abstract business process.

**Algorithm 2** Example implementation of method *validate* (algorithm 1) to check Compliance Assurance Rule 1.

```
Require: alternative;
1: Array activities = alternative.getActivities();
2: for activity in activities do
3:    if activity.getType == 'Exit' then
4:       return false;
5:    end if
6: end for
7: return true;
```

## 5 Application to WS-BPEL

In the specification of WS-BPEL so called *abstract BPEL processes* are defined. Abstract BPEL processes are used to define business process templates and thus can be used analogue to our definition of an abstract process. An abstract BPEL process can include so-called *opaque activities*, which during the customization will be filled with replacement activities from the provided alternatives.

To meet the constraint, that one opaque activity can only be replaced by exactly one replacement activity (see section 4.1), so called structured activities such as the scope or flow activity can be used. In BPEL a structured activity can contain several other activities. Besides this it has a number of other functions. The same is true for a sequence, while, repeatUntil, and forEach activity.

In the following we show how every compliance assurance rule can be applied to a business process written in BPEL.

- **Rule 1 (Opaque activities must not be replaced with exit and opaque activities):** Exit and opaque activities are included in the BPEL specification.
- **Rule 2 (An alternative replacing an opaque activities must not contain links that cross the boundary of that opaque activity):** This rule must be considered when using BPEL because in BPEL links for example can cross the boundary of certain activities like a Scope activity.
- **Rule 3 (Variables which are read by one of the compliance activities must not be modified by replacement activities):** Since with the BPEL language one can define variables, this rule can be applied.
- **Rule 4 (Services which are being invoked from compliance activities must not be invoked from replacement activities):** This rule must be considered, because BPEL is intended to be used above the Web service layer, so Web service invocations are an integral capability of BPEL.

The algorithms presented in section 4.2 can easily be applied when using BPEL, because they are written in an process language independent form.

## 6 Implementation Aspects

As a proof of concept we have implemented a prototype, which implements the algorithms presented in Section 4.2. It further shows how the algorithms and formal definitions from Section 4.1 could be used to verify alternatives being inserted into a business process. The prototype therefore implements the situation where a human wants to insert an alternative into a business process. The design of the prototype reflects two important aspects, extensibility and applicability in a graphical workbench.

Figure 3 shows design of the prototype in an UML class-diagram and the dependencies of the classes. For the reason of applicability in a graphical workbench, the init-method in the *VariabilityChecker*-class does the job the workbench later should do. It initializes the installed compliance assurance rules present in

the system and loads a Variability Descriptor from the file system. Everything is saved in a new object of the type *OpaqueActivity*. This object represents an opaque activity of an abstract process of a compliance template. Then one alternative included in the Variability Descriptor is loaded into an object of the type *Alternative*.

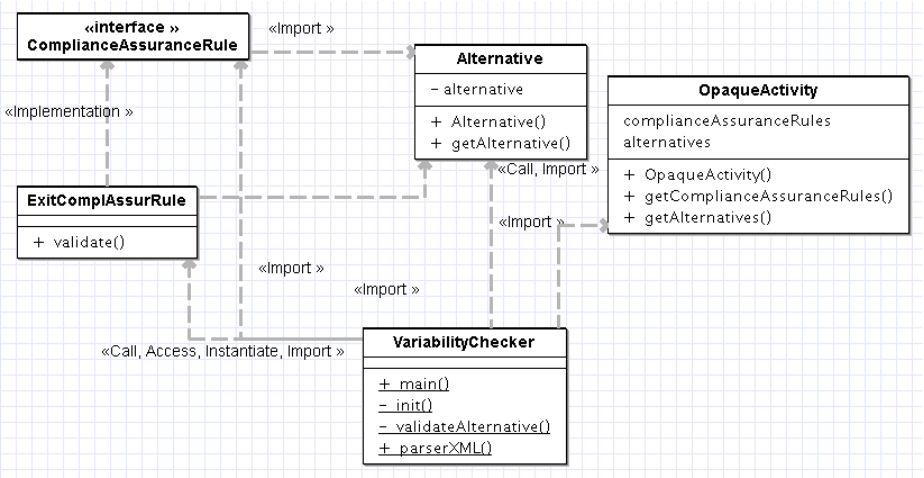



**Fig. 3.** Class diagram with dependencies of the prototype classes

After initialization, the validation of the loaded alternatives is triggered by invoking the *validateAlternative* method. In the future this method is the interface to the graphical process modeling tool. The implementation of this method is kept as similar as possible with the algorithms 1 and 2. Within this method the *validate*-method of the class *ExitComplAssurRule*, which implements compliance assurance rule 1, is invoked. At this point new compliance assurance rules can easily be added, because a single compliance assurance rule is represented by one Java class. Classes, which implement additional compliance assurance rules, have to implement the *ComplianceAssuranceRule* interface. After that the class can be added to the implementation of the validateAlternative method. Every class, which implements the ComplianceAssuranceRule interface, needs to implement the method *validate*. Thus there is no sole method where every compliance assurance rule is verified but a number of classes where this is done. This eases maintainability of core component, the implementation of the validation algorithms. Every class implements the validation algorithm for a single compliance assurance rule. With this approach it is easy to add, delete and modify implementations of compliance assurance rules.

## 7 Related Work

Our approach is different from related work as it combines the modelling of variability with the need for compliance in business processes. Other approaches such as configurable EPCs [11] focus on the modelling and description of variability in process models. The same approach is followed by [5] where points-of-variability in BPEL processes are introduced. However there is no means to verify whether the alternatives which are modelled do not contradict compliance rules which need to be followed by the process. Another approach, VXBPEL [13] extends BPEL with variability making it possible to model alternatives which need to be bound before execution.

Other approaches which deal with run-time adaptation of workflows (such as [10]) also need to take compliance violations into account as these might occur during the adaptation of the workflow. Therefore in [10] compliance rules are enforced during the adaptation which prevent the user to adapt the workflow in a way that these rules are violated. Our approach is different and does not require the extension of existing workflow middle-ware as it allows a modeller to check before the workflow is deployed, whether the variability the modeller has added to the process model is contradictory to the compliance rules. However the algorithms presented in this paper can also be applied in a run-time adaptation scenario because compliance descriptors and variability descriptors can be used to annotate a running BPEL process. In addition to that our approach allows the process designer to annotate different compliance requirements to the same process model thus making the process more flexible and adaptable to different requirements.

Business processes need to be consistent with internal regulations of enterprises and financial markets, laws and customer needs. This leads to a strong need for flexibility of business processes because of changing legislation, customer needs and market conditions [1]. But it is also crucial not to violate compliance rules which apply to a business process when adapting it at design time or at run-time.

There is a considerable amount of work done in annotation of business processes with compliance requirements. In [12] an approach is described which uses the formal contract language (FCL [2]) to describe compliance constraints on business processes. These compliance rules then are used as annotations for the business processes. Furthermore the notions of *compliance distance* and *control tags* are introduced. Compliance distance denotes the effort to make a process model compliant to certain rules. Control tags are used to model control objectives in a process model. In our work we use so called compliance descriptors to annotate business processes with compliance constraints. One advantage of compliance descriptors is, that you can reuse them for different business processes. This is made possible by the fact, that every compliance descriptor defines one atomic compliance constraint which then can be composed with other compliance descriptors to more complex compliance constraints.

Another approach for integration of semantic constraints in process management systems is shown in [6]. The main focus lies here on the process management system and its capabilities to verify semantic constraints of business processes

when they are modified at run-time. Here only dependency and mutual exclusion constraints are considered. In our work the compliance constraints are implicitly defined by an abstract process template. In this template a infinite number of constraints can be defined.

## 8 Conclusions and Outlook

In this paper we presented an approach to prevent process designers from bothering with compliance constraints during design time. This approach is based upon the fact that in a compliance template most kinds of compliance constraints can be implicitly included. It is then the duty of the algorithms of the process design workbench to check if alternatives inserted into the template violate these constraints.

In the future we will work on the implementation of a design time tool with the concepts of this paper built in. Furthermore some aspects of these concepts can also be applied to a tool to adapt an already running business process. Here we want to extend a BPEL engine to support validation of run-time adaptations against compliance rules.

Our work will also include research on dependencies between alternatives and dynamic compliance rules. It could for example be possible that when a certain alternative is inserted into a process, another has to be inserted, too. By dynamic compliance rules we understand that a rule, which has not be satisfied during execution at the beginning of a process, must be satisfied at the end of a process. One example for this is that a credit check activity must be executed at least once during process execution. For example due to a Web service timeout a credit check activity could not be executed. For this reason an extra credit check activity has to be inserted at the end of the process in order to verify if the credit check has already been executed. If not, the credit check must be executed in order to fulfill the compliance requirement. This extra check could be inserted automatically if a credit check alternative is inserted into the process by a process designer.

## 9 Acknowledgments

The work published in this article was partially funded by the MASTER project[1] under the EU 7th Research Framework Programme Information and Communication Technologies Objective (FP7-216917).

## References

1. S. Goedertier and J. Vanthienen. Designing Compliant Business Processes from Obligations and Permissions, 2nd Workshop on Business Processes Design (BPD'06),

[1] `http://www.master-fp7.eu/`

Proceedings. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 5–14. Springer Verlag, 2006. Springer-Verlag Berlin Heidelberg 2006.
2. G. Governatori and Z. Milosevic. A Formal Analysis of a Business Contract Language. *Int. J. Cooperative Inf. Syst.*, 15(4):659–685, 2006.
3. M. E. Kharbili1, S. Stein, I. Markovic, and E. Pulvermüller. Towards a Framework for Semantic Business Process Compliance Management. In *GRCIS 2008*, June 2008.
4. O. Kopp, R. Mietzner, and F. Leymann. Abstract Syntax of WS-BPEL 2.0. Technical report, University of Stuttgart, IAAS, Germany, 2008.
5. A. Lazovik and H. Ludwig. Managing Process Customizability and Customization: Model, Language and Process. In B. Benatallah, F. Casati, D. Georgakopoulos, C. Bartolini, W. Sadiq, and C. Godart, editors, *WISE*, volume 4831 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2007.
6. L. T. Ly, S. Rinderle, and P. Dadam. Integration and Verification of Semantic Constraints in Adaptive Process Management Systems. *Data Knowl. Eng.*, 64(1):3–23, 2008.
7. R. Mietzner. Using Variability Descriptors to Describe Customizable SaaS Application Templates. Technical Report Computer Science 2008/01, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, University of Stuttgart, Institute of Architecture of Application Systems, January 2008.
8. R. Mietzner and F. Leymann. Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors. In *IEEE SCC*, pages 359–366. IEEE Computer Society, 2008.
9. OASIS. *Web Services Business Process Execution Language Version 2.0 – OASIS Standard*, 2007.
10. M. Reichert, S. Rinderle-Ma, and P. Dadam. Flexibility in Process-Aware Information Systems. *T. Petri Nets and Other Models of Concurrency*, 2:115–135, 2009.
11. M. Rosemann and W. M. P. van der Aalst. A Configurable Reference Modelling Language. *Inf. Syst.*, 32(1):1–23, 2007.
12. S. W. Sadiq, G. Governatori, and K. Namiri. Modeling Control Objectives for Business Process Compliance. In *BPM*, pages 149–164, 2007.
13. C.-A. Sun and M. Aiello. Towards Variable Service Compositions Using VxBPEL. In *ICSR '08: Proceedings of the 10th international conference on Software Reuse*, pages 257–261, Berlin, Heidelberg, 2008. Springer-Verlag.
14. M. Turner, D. Budgen, and P. Brereton. Turning Software Into a Service. *Computer*, 36(10):38–44, 2003.
15. United States Code. Sarbanes-Oxley Act of 2002, PL 107-204, 116 Stat 745. Codified in Sections 11, 15, 18, 28, and 29 USC, July 2002.