Compliance Modeling

Formal Descriptors and Tools

Christoph Fehling¹, Falko Kötter², Frank Leymann¹

Report 2014/02

March 28, 2014

¹ Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstr. 38
70569 Stuttgart
Germany

² Institut für Arbeitswissenschaft und Technologiemanagement IAT
University of Stuttgart
Nobelstr. 12, 70569 Stuttgart
70569 Stuttgart
Germany

CR: D.2.1; D.2.12; F.3.2; H.4.1
# Content

1. Introduction ........................................................................................................ 2
2. Technology Overview.......................................................................................... 3
   2.1 Oryx and Model Checkers for BPMN .......................................................... 3
   2.2 TOSCA .......................................................................................................... 4
   2.3 ProGoalML ................................................................................................... 4
   2.4 Variability Descriptors ................................................................................ 5
3. Compliance Descriptor ....................................................................................... 5
   3.1 XML Schema of the Compliance Descriptor ............................................... 6
   3.2 Compliance Expression Grammar .............................................................. 9
   3.3 Java Implementation ................................................................................. 12
4. Example ............................................................................................................ 13
   4.1 Compliance Requirements and Rules ........................................................... 13
   4.2 Exemplary Compliance Descriptor............................................................. 14
5. Tools .................................................................................................................. 16
   5.1 Modeling Compliance Rules for BPMN ..................................................... 16
   5.2 Modeling Compliance Rules in TOSCA ...................................................... 18
   5.3 Modeling Compliance Rules in ProGoalML .............................................. 19
6. References ......................................................................................................... 21
1 Introduction
Compliance, i.e., respecting laws and regulations affects multiple aspects of IT applications. We consider applications centered on a business process model described in BPMN [1]. The business process or multiples thereof supported by the application are described in a formal model, which is then executed by a process engine. Additional functionality is provided by application components, often realized as Web services, which are enacted by the process. In addition to the process engine, these components also rely on a hosting infrastructure, which may be constituted by additional middleware. This middleware can, for example, subsume application servers, such as Apache Tomcat [2] or JBoss [3], or database management systems, such as MySQL [4] or Oracle 11g [5]. The server infrastructure is then provided in data centers or cloud environments. All these artifacts make up the application stack that is supporting companies’ business processes.

Laws and regulations may result in manifold requirements regarding this application stack. We differentiate the following layers that may be affected:

- **Modeling:** laws and regulations may require a certain ordering of activities in a business process or a certain timeliness of activities. For example, an inquiry of customer data may have to be followed directly by an activity asking the customer how this data may be used for marketing purposes.
- **Deployment:** laws and regulations often impact how an application’s runtime environment is hosted. For example, data may have to be stored in certain countries or within close proximity of application users. Configured passwords have to ensure a certain strength etc.
- **Runtime:** when the application is reconfigured during runtime, laws and regulations have to be respected. Some information required to ensure compliance may also only be available during runtime. For example, a regulation may require that an order status has to be sent to a customer within 30 minutes after the order has been placed. This runtime information often has to be reported as laws and regulations demand that companies execute internal and external audits to ensure compliance.

Laws and regulations are not described with the respective layers of an application they affect. Laws, such as the GDV Code of Conduct [6] and the German Federal Data Protection Act [7], therefore, may impact multiple aspects of the business process model, the deployment of the application supporting this model, and the runtime behavior of the application. Management tasks executed by companies’ employees may also be affected.

Due to this manifold impact of laws and regulations, ensuring compliance involves many different technologies and tools. In this report, we present a tool chain for this purpose. Section 2 gives an introduction to used technologies. Section 3 covers an extension in the form of a generic compliance descriptor. This data model serves as “glue” between laws and regulation documents and multiple tools required to ensure them. As part of this descriptor, we cover compliance rules as a machine readable specification of requirements deducted from laws and regulations. These compliance rules may rely on arbitrary expression languages and data formats, for example, WS Policy [8] to describe hosting
locations of data centers or ProGoalML [9] to specify required monitoring information. Expressions are referenced by the compliance descriptor which is serving as a single integration point while allowing the use of specific languages to describe versatile requirements. As the human readable law and regulative documents are not written by IT experts or with their IT impact in mind, we argue that this enables to connect laws and regulations with their machine readable requirements while leaving the IT expert the freedom to use a fitting expression language. To express relationships between different compliance rules and to evaluate compliance on the level of the compliance descriptor, we expect each of the used languages for compliance rules to be evaluated to either true or false. This would respect that the compliance rule is fulfilled or violated by the application, respectively. To describe the relationships between multiple compliance rules, a compliance expression is introduced for which we give a formal grammar.

In Section 4 we give a real-life example of a business process and show how the compliance descriptor can be used to model its compliance requirements.

Section 5 gives an overview of the tools used for compliance modeling. We cover modeling tools on the level of BPMN processes, for describing the application stack supporting these processes, and for monitoring an application. Therefore, these tools relate to the phases of the application: modeling of the business process, deployment of the application stack, and monitoring the runtime of the application.

2 Technology Overview

The following technologies have been used for compliance modeling. Oryx [11] has been used as a BPMN editor. It has been extended for specifying compliance rules on the level of business processes. These rules are evaluated using model checkers. The Topology and Orchestration Specification for Cloud Applications (TOSCA) [17] is an industry standard for modeling of application stacks and their management tasks. It has been used for capturing deployment aspects of compliance. ProGoalML [9] has been used to describe monitoring information that has to be captured during runtime as well as the values to be seen in this information in order to be compliant. A variability descriptor [10] was used to describe configuration points of compliance rules in order to enable their reuse in multiple compliance expressions within one compliance descriptor or even their reuse in multiple compliance descriptors.

2.1 Oryx and Model Checkers for BPMN

Oryx [11] is a Web-based graphical modeling editor for Business Process Model and Notation (BPMN) [1]. Schleicher [12] extended this tool to support compliance rules expressed in Linear Temporal Logic (LTL) [13]. LTL rules may be specified in their own graphical notation and are then transformed into the Process and Protocol Meta Language [14]. This language forms the input to the SPIN model checker [15] which is integrated into Oryx to validate the LTL rules associated with business processes. Graphical extensions to BPMN to describe the association of compliance rules with certain parts of the business process model have been described in [16]. The approach is to define graphical regions in the process model, so called compliance domains. Each compliance domain has a set of compliance rules associated with it. The corresponding LTL rules
implementing the compliance rules are then validated with respect to the elements contained in the compliance domains. Section 5.1 gives examples of the compliance rule specification in a graphical format and its integration into Oryx.

### 2.2 TOSCA

The Topology and Orchestration Specification for Cloud Applications (TOSCA) [17] as an industry standard aims at homogenizing how application can be described and managed across different cloud providers. It, therefore, can be considered as a standardized description language for the components of an application as well as management plans associated with it. The application model describes dependencies between components, for example, how they are hosted on each other, where accesses take place, or what kind of data is exchanged. The management plan captures at least how to provision the application (build plan) and how to decommission it (termination plan). Further management plans may be described, for example, to scale the application, handle component failures etc. Due to the standardization, an application once described in TOSCA may be deployed in various hosting environments supporting the standard.

To capture non-functional requirements, policies have been introduced to TOSCA models [18] [19] [20]. This enables the specification of hosting locations for certain application components, required password strengths etc. The values specified in these policies are then respected by the build plan and other management plans during the application runtime. For the purpose of compliance management, such TOSCA policies can, therefore, be considered compliance rules that have to be respected during the deployment and runtime of an application. Section 5.2 shows the tooling supporting the modeling of TOSCA policies to be referenced by a compliance descriptor.

### 2.3 ProGoalML

The Process Goal Modeling Language ProGoalML [9] is a language to specify key performance indicators (KPIs) and goals of a business process as well as the measurements necessary to calculate them. It can be used to describe how a process is to be monitored at run-time.

ProGoalML offers the following elements:

A *measuring point* is used to measure one or multiple *parameters* at a process element. Whenever the associated process element is reached during process execution, a measurement is taken, evaluating each parameter of the measuring point.

Based on measurements *key performance indicators* can be calculated from parameters. If parameters from multiple measurements are part of the calculated, these measurements are correlated by a process id, which needs to be measured at the respective measuring points.

*Aggregated KPIs* are calculated among multiple process instances limited by a time or length window (e.g. calculate over last hour or last 50 process instances). *Goals* can be imposed on parameters, KPIs and aggregated KPIs. A goal is a Boolean function, where true indicates the goals is fulfilled and false indicates the goal is violated. A special kind of goal is a *timing goal*, which imposes a maximum time between to process elements. To model run-time compliance rules, goals are used to detect violations.
Based on a ProGoalML model, a model-driven monitoring architecture (aPro) can be automatically configured and deployed as a web application [9]. This web application provides monitoring data gathering, monitoring data processing (e.g. calculation of KPIs and goals), monitoring data storage, monitoring data presentation in a dashboard as well as alerting if a goal is violated. Using ProGoalML to model run-time compliance requirements, each compliance rule can be modeled as a goal, either on a KPI if the compliance rule is specific to each instance or on an aggregated KPI if the compliance rule is global. Section 5.3 gives an overview of the tooling to model in ProGoalML.

2.4 Variability Descriptors

Variability descriptors have been introduced by Mietzner [10]. As part of the Composite Application Framework (Cafe) [21], this descriptive model captures variability points of software. Such variability points are, for example, user interface languages, hosting locations, used middleware etc. The variability captured in these descriptors is interpreted by Cafe during a user-driven provisioning of applications via a self-service interface. Customers can interactively bind variability points to alternatives meeting their requirements, thus, influencing the provisioning of the application. We use variability descriptors to capture variability of compliance rules. For example, the expression of a compliance rule checking that an activity is executed within a certain amount of time after another activity essentially remains the same even if the amount of time changes. Analogous, the compliance rule remains the same if it shall be verified for two different activities. Using a variability descriptor, the LTL specification may expose these variability points, thus, it can be used in different scopes, i.e., validate different activities and time frames. Furthermore, the complexity of rule adjustments is significantly reduced. Users do not have to rewrite LTL rules to check for similar conditions, thus, the reuse of once defined compliance rules is simplified. Adjustments necessary due to changing laws and regulations can also be handled easier by adjusting the values to which variability points are set.

3 Compliance Descriptor

The central document managing compliance for a certain application and the supported business process is the compliance descriptor. It contains the following elements or references them:

- **Legal documents**: laws and regulations are captured in XHTML [22]. This enables the referencing to individual paragraphs and sentences in order to connect them with compliance rules.
- **Compliance rules**: this is the machine-readable specification of an expression to be verified. Compliance rules are reusable building blocks to check compliance-related conditions and can be configured. For example, there can be reusable rules to check for activity ordering, time elapsed between activities, the human employees handling activities etc.
- **Compliance requirements**: each law and regulatory document may result in multiple compliance requirements. These requirements may
combine multiple compliance rules which are linked using a compliance expression.

The remainder of this section covers the XML schema of the compliance descriptor document and the formal grammar of compliance expressions.

### 3.1 XML Schema of the Compliance Descriptor

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema targetNamespace="http://comb.iao.fraunhofer.de"
  version="1.0"
  xmlns:tns="http://comb.iao.fraunhofer.de"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="bindingStrategy" type="tns:bindingStrategy"/>
  <xs:element name="compliancedescriptor" type="tns:complianceDescriptor"/>
  <xs:element name="compliancerequirement" type="tns:complianceRequirement"/>
  <xs:element name="compliancerule" type="tns:complianceRule"/>
  <xs:element name="entity" type="tns:entity"/>
  <xs:element name="law" type="tns:law"/>
  <xs:element name="laws" type="tns:lawsMapElement"/>
  <xs:element name="lawurl" type="tns:lawURL"/>
  <xs:element name="paragraphreference" type="tns:paragraphReference"/>
  <xs:element name="requirements" type="tns:requirementsMapElement"/>
  <xs:element name="ruleExpression" type="tns:ruleExpression"/>
  <xs:element name="rules" type="tns:rulesMapElement"/>
  <xs:element name="stringExpression" type="tns:stringRuleExpression"/>
  <xs:element name="urlExpression" type="tns:urlRuleExpression"/>
  <xs:element name="variabilitybinding" type="tns:variabilityBinding"/>
  <xs:complexType name="urlRuleExpression">
    <xs:complexContent>
      <xs:extension base="tns:ruleExpression">
        <xs:sequence minOccurs="0" name="referencedExpression" type="xs:anyURI"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexType>
  <xs:complexType abstract="true" name="ruleExpression">
    <xs:sequence/>
  </xs:complexType>
  <xs:complexType name="lawURL">
    <xs:sequence>
      <xs:element minOccurs="0" name="law" type="xs:string"/>
      <xs:element minOccurs="0" name="paragraphReference" type="tns:paragraphReference"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="paragraphReference">
    <xs:sequence minOccurs="0" name="xpath" type="xs:string"/>
  </xs:sequence>
  <xs:complexType name="stringRuleExpression">
    <xs:complexContent>
      <xs:extension base="tns:ruleExpression">
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```
<xs:sequence>
  <xs:element minOccurs="0" name="expression" type="xs:string"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="complianceRule">
  <xs:sequence>
    <xs:element minOccurs="0" name="description" type="xs:string"/>
    <xs:element minOccurs="0" name="language" type="xs:string"/>
    <xs:element minOccurs="0" name="phase" type="xs:string"/>
    <xs:choice minOccurs="0">
      <xs:element name="ruleExpression" type="tns:stringRuleExpression"/>
      <xs:element name="ruleExpressionURL" type="tns:urlRuleExpression"/>
    </xs:choice>
    <xs:element minOccurs="0" name="variabilityDescriptor" type="xs:anyType"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="bindings" type="tns:variabilityBinding"/>
    <xs:element minOccurs="0" name="bindingStrategy" type="tns:bindingStrategy"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string"/>
</xs:complexType>

<xs:complexType name="variabilityBinding">
  <xs:sequence/>
  <xs:attribute name="variablityPoint" type="xs:string"/>
  <xs:attribute name="parameter" type="xs:string"/>
  <xs:attribute name="constant" type="xs:string"/>
</xs:complexType>

<xs:complexType name="entity">
  <xs:sequence/>
  <xs:attribute name="name" type="xs:string"/>
  <xs:attribute name="type" type="xs:string"/>
  <xs:attribute name="description" type="xs:string"/>
</xs:complexType>

<xs:complexType name="law">
  <xs:sequence>
    <xs:any namespace="##other" processContents="skip"/>
  </xs:sequence>
  <xs:attribute name="title" type="xs:string"/>
</xs:complexType>

<xs:complexType name="complianceDescriptor">
  <xs:sequence>
    <xs:element minOccurs="0" name="rules" type="tns:rulesMapElement"/>
    <xs:element minOccurs="0" name="requirements" type="tns:requirementsMapElement"/>
    <xs:element minOccurs="0" name="laws" type="tns:lawsMapElement"/>
    <xs:element name="entities" type="xs:complexType">
      <xs:sequence>
    </xs:sequence>
  </xs:element>
</xs:complexType>
<xs:element name="entities" type="tns:entity"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:complexType>

<xs:complexType name="rulesMapElement">
  <xs:sequence>
    <xs:element name="rule" type="tns:complianceRule" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="requirementsMapElement">
  <xs:sequence>
    <xs:element name="requirement" type="tns:complianceRequirement" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="complianceRequirement">
  <xs:sequence>
    <xs:element minOccurs="0" name="complianceExpression" type="xs:string" />
    <xs:element minOccurs="0" name="description" type="xs:string" />
    <xs:element maxOccurs="unbounded" minOccurs="0" name="lawURLs" type="tns:lawURL" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" />
</xs:complexType>

<xs:complexType name="lawsMapElement">
  <xs:sequence>
    <xs:element name="law" type="tns:law" />
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="bindingStrategy">
  <xs:restriction base="xs:string">
    <xs:enumeration value="SINGLETON"/>
    <xs:enumeration value="CLONE"/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>
3.2 Compliance Expression Grammar

The compliance expression is used to interconnect multiple compliance rules to a
more complex compliance requirement. As the individual languages for
compliance rules are arbitrary, this enables a generic compliance language on the
level of the compliance descriptor. While the most fitting languages may be used
to verify compliance during modeling, deployment, and runtime, the compliance
expression is used to validate the overall compliance for compliance requirements
and the overall compliance descriptor. The compliance expressions follow a
formal grammar. We specify this grammar by W3C-style and railroad diagrams.

W3C-style grammar of compliance expressions:

```
parse    ::= complianceexpression EOF
complianceexpression
 ::= or
 or      ::= and ( 'or' and )* 
 and     ::= rel ( 'and' rel )* 
 rel     ::= add ( ( '=' | '>' | '<' ) add )* 
 add     ::= mult ( ( '+' | '-' ) mult )* 
 mult    ::= unary ( ( '*' | '/' ) unary )* 
 unary   ::= '-' term
        | '+' term
        | 'not' term
        | term
 term    ::= INTEGER
        | IDENT
        | '(' complianceexpression ')' 
        | WS 
EOF      ::= $
```

/* ws: definition */

```
IDENT    ::= LETTER LETTER*
INTEGER  ::= DIGIT+
WS       ::= ( ' ' | #xA | #xD | #x9 )+
DIGIT    ::= [0-9]
LETTER   ::= [a-z]
        | [A-Z]
EOF      ::= $
```

Railroad diagrams of compliance expressions:

```
parse:
```

```
parse    ::= complianceexpression EOF
```

no references

```
complianceexpression:
```

```
complianceexpression ::= or
```

referenced by: parse, term
or:

\[
\text{or} ::= \text{and} ( 'or' \text{ and} )^*
\]

referenced by: \text{complianceexpression}

and:

\[
\text{and} ::= \text{rel} ( 'and' \text{ rel} )^*
\]

referenced by: \text{or}

rel:

\[
\text{rel} ::= \text{add} ( ( '=' | '>' | '<' ) \text{ add} )^*
\]

referenced by: \text{and}

add:

\[
\text{add} ::= \text{mult} ( ( '+' | '-' ) \text{ mult} )^*
\]

referenced by: \text{rel}

mult:

\[
\text{mult} ::= \text{unary} ( ( '*' | '/' ) \text{ unary} )^*
\]

referenced by: \text{add}
unary:

unary ::= '‐' term
       | '+' term
       | 'not' term
       | term

referenced by: mult

term:

term ::= INTEGER
       | IDENT
       | '(' complianceexpression ')'

referenced by: unary

_: _ ::= WS
     /
     /* ws: definition */

no references

IDENT:

IDENT ::= LETTER LETTER*

referenced by: term

INTEGER:

INTEGER ::= DIGIT+

referenced by: term
3.3 Java Implementation

The compliance descriptor has been implemented in Java using JaXB to allow marshalling and unmarshalling of compliance descriptor XML files. Using the unmarshalled compliance descriptor, compliance expressions can be parsed and compliance rules can be created as well as deployed.
4 Example

Figure 1: example process model in Oryx

Figure 1 shows an example process modeled in BPMN 2.0 in the Oryx editor. This is a simplified insurance claim management process handling damage claims. The process has the following steps:

- Receive claim: a claim is received via mail, fax etc., converted in a structured data format and stored in a customer database
- Process claim: the claim is automatically examined for consistency, plausibility etc.
- Decide claim: based on the processing result the claim is accepted or rejected.
- Send claim: a letter is sent to the claimant informing him or her of the claim management result.

4.1 Compliance Requirements and Rules

Two exemplary compliance requirements for this process arise from different laws:

- **R1**: the GDV Code Of Conduct [6] stipulates a claimant is to be notified in a timely fashion, if his or her data is used for marketing purposes, which may be the case as soon as it is in the customer database. This notification can be sent with the claim result. A timely fashion is further specified as 14 days. Therefore, the requirement is as follows:
  \[
  \text{At most 14 days after the claim is received the claim result has to be sent.}
  \]

- **R2**: the Federal Data Protection Act (Bundesdatenschutzgesetz) [7] in Germany regulates data storage. To make sure German law is upheld, the requirement is as follows:
  \[
  \text{The customer database must be hosted within Germany.}
  \]

On a technical level, these compliance requirements need to be implemented by compliance rules during different times of the process lifecycle.

R1 can be partially checked by the sequence of activities during process design. “Receive claim” must be followed by “Send claim and Data privacy notification”.
But this alone is insufficient. The timing constraint of 14 days can only be monitored at run-time.
R2 can be checked during deployment of the process infrastructure.
To model these rules and requirements, the compliance descriptor is used.

4.2 Exemplary Compliance Descriptor

Below the compliance descriptor for the example process is listed.
The compliance descriptor contains the following:

- Laws: both laws from which compliance requirements stem, GDV Code Of Conduct and Bundesdatenschutzgesetz are contained as XHTML.
  (Lines 70 – 77 in the exemplary descriptor)
- Compliance requirements: both compliance requirements R1 and R2 are contained, referencing the sections of the law they stem from with an XPATH expression.
  - Compliance expressions: each compliance requirement contains an expression in the compliance expression language detailed in Section 3.2, referencing the compliance rules of the descriptor.
- Compliance rules: three compliance rules for implementing the compliance requirements:
  - The rule followedBy is an LTL rule to be checked at design time.
    (Lines 16 – 27 in the exemplary descriptor)
  - The rule hostingLocation is a TOSCA policy to be checked during deployment.
    (Lines 28 – 39 in the exemplary descriptor)
  - The rule maxTimeBetweenActivities is a ProGoalML rule to be monitored at run-time.
    (Lines 3 – 15 in the exemplary descriptor)

Note that compliance rule expressions as well as variability descriptors and full law texts are omitted for space reasons. The compliance rule expressions are detailed in Section 5.

```xml
<compliancedescriptor name="simplified claim management">
  <rules>
    <rule id="maxTimeBetweenActivities">
      <description>A second activity must follow a first activity within a specified time.</description>
      <language>ProgoalML</language>
      <phase>run-time</phase>
      <ruleExpression>
        <expression>...</expression>
      </ruleExpression>
      <variabilityDescriptor>...</variabilityDescriptor>
      <bindings variablityPoint="$firstActivity" parameter="1"/>
      <bindings variablityPoint="$secondActivity" parameter="2"/>
      <bindings variablityPoint="$maxTime" parameter="3"/>
    </rule>
    <rule id="followedBy">
      <description>A first activity must be followed by a second activity.</description>
      <ruleExpression>
        <expression>...</expression>
      </ruleExpression>
      <variabilityDescriptor>...</variabilityDescriptor>
      <bindings variablityPoint="$firstActivity" parameter="1"/>
      <bindings variablityPoint="$secondActivity" parameter="2"/>
      <bindings variablityPoint="$maxTime" parameter="3"/>
    </rule>
  </rules>
</compliancedescriptor>
```
<ruleExpression>
  <expression>([]($A -> <> $B))</expression>
</ruleExpression>

<variabilityDescriptor>...</variabilityDescriptor>

<bindings variablityPoint="$A" parameter="1"/>
<bindings variablityPoint="$B" parameter="2"/>

</rule>

<rule id="hostingLocation">
  <description>A component has to be hosted at a specific region.</description>
  <language>TOSCA</language>
  <phase>deployment</phase>
  <ruleExpression>
    <expression>...</expression>
  </ruleExpression>
  <variabilityDescriptor>...</variabilityDescriptor>
  <bindings variablityPoint="$component" parameter="1"/>
  <bindings variablityPoint="$region" parameter="2"/>
</rule>

<requirements>
  <requirement id="R1">
    <lawURLs>
      <law>GDV - Code of Conduct</law>
      <paragraphReference>
        <xpath>/html/body/p[1]</xpath>
      </paragraphReference>
    </lawURLs>
    <description>After a claim is received the claimant has to be asked for agreement within 14 days.</description>
    <complianceExpression>followedBy("Receive claim", "Send claim and data privacy notification") AND maxTimeBetweenActivities("Receive claim", "Send claim and data privacy notification", "14 days")</complianceExpression>
  </requirement>
  <requirement id="R2">
    <lawURLs>
      <law>Bundesdatenschutzgesetz</law>
      <paragraphReference>
        <xpath>/html/body/p[17]</xpath>
      </paragraphReference>
    </lawURLs>
    <description>The Customer DB shall not be hosted outside of Germany.</description>
    <complianceExpression>hostingRegion("Customer DB", "Germany")</complianceExpression>
  </requirement>
</requirements>

<laws>
  <law title="GDV - Code of Conduct">
    <html>...</html>
  </law>
  <law title="Bundesdatenschutzgesetz">
    <html>...</html>
  </law>
</laws>

<entities name="Receive claim" type="Activity"/>
5 Tools

5.1 Modeling Compliance Rules for BPMN

Compliance rule modeling in BPMN is centered on LTL rules. Figure 2 shows the graphical compliance rule editor. As specifying LTL textually Schleicher [12] introduced this graphical notation using the stencil set seen on the left. This specific rule followedBy stipulates that the use of a “receive claim” activity in a process implies that the “send claim and data privacy notification” activity must also be present in the business process and be executed if “receive claim” is executed. This graphical notation is then be transformed to LTL to be applied to a business process model.

The screenshot in Figure 3 shows a BPMN Process for which a compliance rule shall be used. For this purpose, the BPMN stencil set seen on the left has been extended to allow modeling of compliance domains. In this example, the compliance domain shall consider all activities in the process, thus, is encloses the complete business process. Then, the compliance rule is associated with this compliance domain.
During modeling the user may evaluate compliance rules associated with compliance domains in the business process by triggering the SPIN model checker as seen in Figure 4. The model checker retrieves the LTL rule from the compliance domain element and evaluates it with respect to all elements contained in the scope.
The result may also be shown in a more condensed fashion in the model checker window as seen in Figure 5.

![Figure 5: result windows of the SPIN model checker](image)

As the model checker results are only textual, the result is also used to visualize compliance in the process itself. This allows users to identify failing compliance domains quickly as seen in Figure 6.

![Figure 6: graphical visualization of compliance in the business process](image)

5.2 Modeling Compliance Rules in TOSCA

Figure 7 shows winery, an Open Source TOSCA editor proposed as an eclipse project [23]. Different application components are modeled connected via a “HostedOn” dependency. On the top is a customer database, which is installed on a MySQL DBMS. This middleware is installed on Ubuntu12.04 as operating system. On the very bottom, a blade center is used as runtime infrastructure. Two components have an associated data policy. The operating system has a “data location” policy associated with it to ensure that it is installed in a certain geographic region. The customer database has an additional “data location” policy to ensure that all management tasks handling this data have to respect where data may be stored. For example, data backup tasks would not have to consider policies affecting the application provisioning. Nevertheless, they have to respect the geographic location where data backups are stored. This model implements the *hostingLocation* rule in the example.
5.3 Modeling Compliance Rules in ProGoalML

ProGoalML is used to model run-time monitoring of a process, including measurements, KPIs and goals. Similarly to LTL, a ProGoalML model can be modeled in Oryx. The ProGoalML stencilset serves as an extension to BPMN 2.0, enabling the monitoring model to be modeled directly within the BPMN process model.

Figure 8 shows the example process extended with ProGoalML elements realizing a run-time compliance rule in the Oryx editor. Note the ProGoalML modeling stencils to the left. Monitoring data is measured at two measuring points, mp_a at the activity Receive claim and mp_b at the activity Send claim.
and data privacy notification. At both measuring points the process id of the current process instance as well as the current timestamp at the time of measurement is measured. Using these measurements, the compliance rule $maxTimeBetweenActivities$ is implemented as the timing goal $max_time_a_b$. This timing goal imposes that for each process instance, the time between the first and second measurement (and in turn between both associated activities) must be less than 14 days. If within 14 days after the first activity the second activity does not take place, the goal is violated and an alert is generated.

![Figure 9: example alert if timing goal is violated](image)

Figure 9 shows an example alert as it is displayed in the automatically generated dashboard. Optionally, an alert can be sent out as an e-mail as well to allow immediate reaction to compliance violations.
6 References