Unified Compliance Modeling and Management using Compliance Descriptors

Falko Koetter[1], Maximilien Kintz[1], Monika Kochanowski[1], Christoph Fehling[2], Philipp Gildein[2], Frank Leymann[2] and Anette Weisbecker[1]
[1]*University of Stuttgart IAT and Fraunhofer IAO, Germany, Stuttgart, Germany*
[2]*University of Stuttgart, IAAS, Stuttgart, Germany*

Keywords: Business Process Management, Compliance Modeling, Model-Driven Architecture, Business Process Compliance.

Abstract: Due to innovations in the field of cloud computing business processes become distributed, encompassing a combination of services spanning multiple IT systems. Due to a growing number of regulations, managing business process compliance in this cloud environment is a necessary task for companies, leading to a growth in compliance management and compliance checking approaches. Compliance stems from laws and is implemented in all parts of enterprise IT. Thus, both a connection between business and IT as well as a broad coverage of compliance scenarios is necessary. To solve both challenges, we use an integrating compliance descriptor for conceptual compliance modeling. This descriptor is used to configure a compliance management architecture, integrating different types of compliance checking. For creating compliance descriptors, it proved necessary to introduce a formalism and a graphical notation, which is introduced and evaluated in a prototype and expert interviews.

1 INTRODUCTION

Cloud computing is both a chance and a challenge for many companies (Wei and Blake, 2010), especially in the field of security, privacy, (Takabi et al., 2010) and - consequently - compliance. The possibility to acquire services over the cloud in order to better perform their business processes gives companies new possibilities to manage their business. However, this adds to the complexity of the involved IT systems as well as poses new challenges in managing these. Due to growing regulatory requirements stemming from new laws like the Sarbanes-Oxley Act there is an increasing demand for business process compliance solutions in the industry (Sadiq et al., 2007).

However, there is a gap between compliance management and business process management, as one is driven by legal requirements and the other by business needs as well as the new cloud technologies. Additionally, compliance management spans not only business processes but also the process environment consisting of software systems, physical hardware and personnel, as described in (Kochanowski et al., 2014). Here, also compliance applications and their needs are mentioned. This requires communication between legal specialists, business users and IT personnel. All this makes keeping processes compliant a cumbersome task.

As business process models and their implementation increase in complexity (Bobrik et al., 2007), manual compliance checking is not feasible for large organizations. Existing IT-supported compliance management solutions focus on specific process execution environments (e.g. process engines) and only support specific kinds of compliance rules (Kharbili et al., 2008). As far as processes are executed within such an environment, they can support compliance enforcement with strict process models, transition rules, double-checks, etc. However, not all parts of a business process are usually contained within such solutions, and not all of the compliance requirements can be enforced in such a way. We found the factor preventing effective compliance management is not a lack of tools, but rather a lack of integration, between different kinds of compliance checking as well as between business and IT.

To alleviate this problem, in previous work we proposed an integrating compliance descriptor (Koetter et al., 2013), which bridges legal, business and IT levels by separating laws, compliance requirements

Koetter, F., Kintz, M., Kochanowski, M., Fehling, C., Gildein, P., Leymann, F. and Weisbecker, A.
Unified Compliance Modeling and Management using Compliance Descriptors.
In *Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) - Volume 2*, pages 159-170
ISBN: 978-989-758-182-3

that stem from them and compliance rules implementing these requirements. As the approach encapsulates compliance rules, multiple rule languages can be used, providing integration and coverage of all kinds of compliance requirements. We described how to use this compliance descriptor for compliance checking, gathering results from different rules and aggregating them to determine requirement fulfillment and law compliance (Koetter et al., 2014).

In this work, we will build on this approach by developing a conceptual modeling language for compliance descriptors and integrating it with other modeling languages for processes and compliance rules. This is the main contribution of this paper, as it proved necessary to develop a formalism and graphical modeling notation in order to facilitate creating compliance descriptors. Using a model-driven approach, we show how a conceptual model of compliance can be transformed to different artifacts necessary for compliance management in a reference architecture, which is extended from previous work. Evaluating the work using a prototype and real-life example, we show our approach is a feasible solution for bridging business and IT views of process compliance in a rule-language-independent and extensible fashion. Compared to our previous work, we evaluated the approach not only in a prototype but also with a real-life system and real users.

The remainder of this work is structured as follows. In Section 2 an overview of work in compliance modeling and compliance integration is given. In Section 3 we describe conceptual compliance modeling using the compliance descriptor. In Section 4 we describe the extensible architecture for compliance management as well as the model transformation. Section 5 describes the prototype and evaluation. Finally, Section 6 gives a conclusion and outlines future work.

2 RELATED WORK

Achieving Business Process Compliance is not a one-time task, but a continuous activity. Different compliance checks are performed at different phases in the business process lifecycle. (Kharbili et al., 2008) gives an overview of compliance checking methods and distinguishes design-time, run-time and ex-post compliance checking. In particular, (Kharbili et al., 2008) notes a lack of an universal approach, supporting all phases of the lifecycle as well as continuous change. To address this, we introduced an integrating compliance descriptor in (Koetter et al., 2013). Rather than designing yet another compliance rule modeling language, this compliance descriptor connects laws, compliance requirements and rules in different compliance rule languages. Thus, a link between the business and IT view of compliance requirements is preserved. Using this link, the impact of changes in laws, requirements or implementation to overall compliance can be assessed at any time, enabling maintenance of compliance in the face of change (Koetter et al., 2014).

Similarly to the structure of the compliance descriptor, in (El Kharbili et al., 2008a) three levels of regulatory compliance are defined. Regulations define measures and directives which are implemented by policies, internal controls, and procedures. Furthermore, eight requirements for a compliance management framework are defined, among others enforcement, change management, traceability, and impact analysis. As in other previous work, different types of compliance checking are identified. To tackle these challenges, an architecture for a compliance checking framework based on semantic business process models is proposed. In this architecture, regulations are modeled as semantic policies which are monitored by a policy monitoring component. From these, semantic business rules are generated to be enforced at design-time and run-time by an inference engine (El Kharbili et al., 2008b).

The *SeaFlows Toolset* (Ly et al., 2011) is a framework for compliance verification of business processes. Using *compliance rule graphs*, rules can be modeled by imposing patterns of process activities and/or conditions on process data at specific points in process execution. Patterns are then checked at design time, while data conditions are checked at run-time in a BPM suite. In further work, the resource perspective, i.e. who performs tasks, has been added in an extended rule graph (Semmelrodt et al., 2014). This approach is interesting both in combining run-time and design-time rules in the same rule graph as well as combining multiple rules in a single graph and separating it from the process model. However, due to the implementation techniques used, it is dependent of specific modeling and execution environments and doesn't explicitly offer extensibility for other types of compliance rules.

(Reichert and Weber, 2012, Chapter 10) investigates design-time, run-time and ex-post compliance rules based on process traces, on which Linear Temporal Logic (LTL) expression or compliance rule graphs are tested. Also the impact of process change on compliance in models and running process instances is tackled by investigating the changes in compliance rules and their effects. However, only process models and instances in a workflow engine are in the scope of compliance checking.

(Schleicher et al., 2011) defines a graphical modeling language for LTL rules which can be used for design-time compliance checking. A *compliance domain* can be used to attach an LTL rule to a process or a part of a process. In (Awad et al., 2008) BPMN-Q, a graphical query language for business process models is used for design time compliance checking. Using compliance patterns and anti-patterns of violations, this approach is used to visualize reasons for compliance violations in the process model (Awad and Weske, 2010).

Compliance of business processes reaches beyond the scope of process models and execution, encompassing aspects like hosting, maintenance, encryption and physical access control. Business process management builds on these aspects, which need to be covered in order to guarantee overall compliance. The Topology and Orchestration Specification for Cloud Applications (*TOSCA*[1]) is an OASIS standard which allows modeling the topology of an application, including the implementation and physical deployment of components like web services and databases. In (Waizenegger et al., 2013), *Policy4TOSCA* is described, extending TOSCA models by so-called policies to describe non-functional requirements, which may stem from compliance rules. These can be checked during process deployment in order to guarantee a compliant hardware and software stack (Koetter et al., 2014).

While checking compliance rules for a business process can be looked at in a single-system fashion, i.e. the process is executed in a single, homogenous environment, in which compliance is checked, in practice process compliance needs to be managed in a multi-system fashion. One reason for this are heterogeneous, grown IT infrastructures found in practice (Patig et al., 2010), another reason are the different scopes compliance rules may encompass (Kharbili et al., 2008).

(Knuplesch et al., 2013) describes an approach for integrating compliance checking across multiple companies in cross-organizational business processes. Challenges arise because parts of an organizational process model may not be globally known and because local compliance requirements may contradict global compliance requirements. Thus, a notion of *compliability* is defined, meaning the interaction between process partners conforms to global compliance rules, even though private processes are not known. While this approach allows integrated design-time compliance across multiple public and private process models, it does not address the challenge of heterogeneous process environments and other kinds of compliance checking, though this shall be addressed in future work.

[1]www.oasis-open.org/committees/tosca/ (accessed 12.3.2015)

(Ramezani et al., 2012) advocates a separation of compliance management and process management by introducing a separate *compliance engine*, which checks an implementation system for compliance and interrupts it in case of risks. Compliance rules are defined on a business vocabulary from a conceptual model of processes. By separating process and compliance, any information system may be checked, similar to the model-driven solution for process monitoring used in our approach (Koetter and Kochanowski, 2013). Similarly, any compliance rule language may be used. This in general solves the integration problem, but requires a high degree of manual implementation in the compliance engine and in the integrated information system. To lessen the required effort, (Ramezani et al., 2014) lists methods of assisting domain experts in the selection of applicable existing rules from a rule repository, e.g. by question trees.

In (Comuzzi, 2014) the problem of aligning compliance rules and their implementation across partners in a business network is investigated. Compliance rules are defined on a conceptual level and then concretized to a specific scenario and process modeling environment, which in turn are concretized in a specific implementing technology. This resembles a compliance descriptors separation of requirements and rules. However, even after concretization, manual implementation and integration needs to be performed to achieve compliance monitoring.

(Karagiannis et al., 2012) describes a generic compliance evaluation method extending existing enterprise modeling frameworks to encompass compliance evaluation. Similar to this work, partial compliance can be evaluated and is visualized in a heatmap. Compliance is evaluated for individual architecture artifacts aggregated through architecture levels. For example, the compliance of a business process is calculated form the compliance of its activities. In comparison, our approach has a specific process focus and allows definition of emergent compliance rules encompassing multiple activities. Additionally, (Karagiannis et al., 2012) gives no techniques how to automatically determine compliance at the bottom level, while our approach integrates different compliance rule languages.

(Weigand and Elsas, 2012) proposes a model-driven method for auditing using an ontology, determining accountability and authorization and in turn defining controls to address identified risks. While an approach like this provides the possibility to discover requirements and necessary controls, it does not cover the actual compliance checking. A gap between definition and implementation remains.

A framework for defining and managing compliance requirements is presented in (Papazoglou, 2011). Requirements are defined using a declarative language and LTL. Design-time compliance checking is supported while run-time compliance checking is only conceptually described. Similarly to the reusable compliance rules in this approach, patterns are used for easier implementation of common requirement types. Compared to this approach, extensibility of checks, e.g. for software deployments, is not covered.

Overall, the related work shows first steps for solving the problem of heterogeneous environments and the need for compliance checking to cover the whole business process lifecycle and associated artifacts. However, approaches try to fit all compliance checking rules in homogenous modeling languages or require manual rule implementation. In contrast, this work contributes an integrating compliance management approach, which separates business and IT view of process compliance and encapsulates compliance rules in any rule modeling language.

3 CONCEPTUAL COMPLIANCE MODELING

As business process compliance needs to cover many aspects of the enterprise, multiple artifacts need to be modeled for compliance management. On the business level, *laws* impose compliance *requirements* on a business *process*. On the IT level, requirements are implemented in *rules* which are checked throughout the process lifecycle, e.g. at design-time, run-time and during deployment.

Figure 1 gives an overview of compliance-related models. A *process model* describes the process on a conceptual level. This process model may be prescriptive or descriptive, i.e. it may be executed or describe the execution by other IT systems. On an implementation level, different types of compliance rules can be used for compliance checking. LTL rules (Schleicher et al., 2011) are used for model checking at design-time, e.g. for verifying the order of activities. ProGoalmML rules (Koetter and Kochanowski, 2013) are used for compliance monitoring at run-time, e.g. to check if timing restrictions are kept. TOSCA policies (Waizenegger et al., 2013) are used to verify the physical deployment of process infrastructure, e.g. to satisfy data protection requirements.

For connecting the business and IT levels the *compliance descriptor* is used. It contains all *laws* (and other regulatory documents) applicable to the process. Compliance requirements from these laws are contained as well, described in natural speech as well as in an expression referencing implementing compliance rules. As the compliance descriptor serves to integrate the business and IT levels of compliance management, a modeling language needs to be understood both by business and IT experts. In previous work we defined the structure of the compliance descriptor on an implementation level using XML schema (Fehling et al., 2014). While this proved sufficient for implementing compliance checking, creating compliance descriptors proved to be difficult. To close this gap, we will define a graphical modeling language for compliance descriptors.

3.1 Laws

In the compliance descriptor, a *law* represents a regulatory document from which compliance requirements stem. A law $l \in L$ is defined as a tuple:

$l \in L := (n_l, P_l, v_l)$

where n_l is the name of the law, v_l identifies the version of the law and P_l is a set of paragraphs the law consists of. On a technical level the law is stored as an XHTML document, providing both human readability due to html formatting as well as a well-defined structure for automatic processing due to xml validity. Laws need to be structured in order to make references to paragraphs of the law possible.

Paragraphs $p \in P_l$ are defined as a tuple:

$p \in P_l := (n_p, c_p)$

where n_p is the number or name of the paragraph and c_p is its content. Paragraph numbers have to be unique within the law, but may be repeated among different laws. Thus, to uniquely reference a paragraph $p \in P_l$, a combination of its number n_p and the law's name n_l can be used.

We define this reference as a law url $u \in U$ as follows:

$u \in U := (n_{l_u}, n_{p_u})$

with n_{l_u} as the name of a law $l \in L$ and n_{p_u} as the name of a paragraph $p \in P_l$ within that law. Note that if a law url refers to a law, it may only refer to a paragraph within that same law:

$(n_{l_u}, n_{p_u}) \in U \rightarrow p \in P_l$

3.2 Entity

Entities within the process context (e.g. a process activity or a server) are modeled as *entities*, so they can be referenced by the compliance descriptor. An entity $e \in E$ is defined as follows:

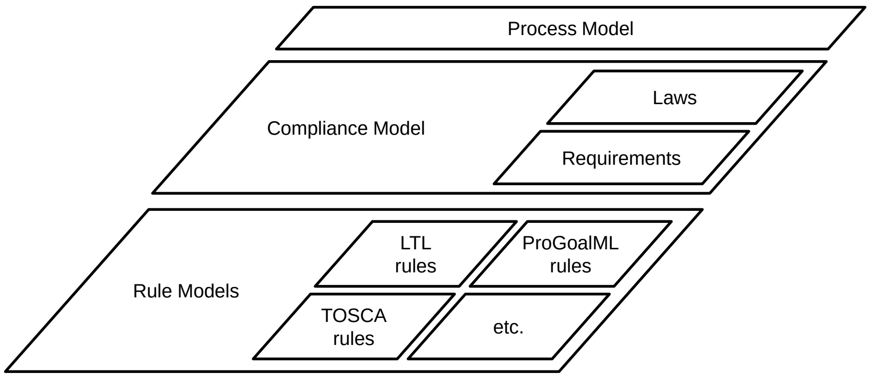

Figure 1: Compliance-related models.

$e \in E := (n_e, ref_e)$

where n_e is the name of the referenced entity and ref_e a unique reference to the entity (e.g. a reference to an activity in a process model).

3.3 Compliance Rules

A compliance rule is an implementation level artifact used to enforce or monitor a fact, e.g. if a variable has a certain value, if an activity is always followed by another activity or if a database is encrypted. Thus, the concrete implementation of a rule is not stipulated by the compliance descriptor. To provide a wide support in regards to compliance rule languages, a rule is described with generic attributes, allowing handling it during compliance management without knowledge of implementation semantics. Only during model transformation is the concrete implementation used.

A compliance rule is described as follows:

$r \in R := (n_r, d_r, il_r, ph_r, rex_r, VD_r, B_r)$

where n_r is the name of the compliance rule, d_r its description in natural language. These are written by IT personnel to give business users an understanding of the semantics of the rule. il_r is a unique identifier for the implementation language of the rule, e.g. *LTL*. ph_r identifies the phase of the process lifecycle in which the rule is applied, e.g. design-time or run-time.

The concrete implementation is called a *rule expression* and stored in rex_r. Depending on the type of rule this may either be a formal description of the rule (e.g. an XML file) or a reference to the full implementation (e.g. a reference to a graphical rule model). In any case, a compliance rule must provide a suitable result for evaluation. Depending on the lifecycle phase, a compliance rule may either be fulfilled (*true*), not fulfilled (*false*) or not yet evaluated (*unknown*). Each rule must provide these results.

To facilitate reuse of compliance rules among different process models and requirements, rules are variable. This means certain parts of the rule can be modified in order to adjust the rule to its concrete use case. For example the name and location of a database can be chosen. For this purpose, a so-called variability descriptor is used (Mietzner et al., 2009), an XML format which allows variability in arbitrary documents by referencing variability points and possible values. The variability descriptor of a rule is stored or referenced in VD_r and is a set of variability points vp_r:

$VD_r := \{vp_r\}$

A set of *bindings* B_r describes which concrete values are chosen for each variability point. A binding $b \in B_r$ is defined as follows:

$b \in B_r := (vp_b, val_b, type_b)$

where vp_b identifies the variability point that is bound, val_b is the value the variability point is bound to, which depends on the type of the binding $type_b$.

There are three types $type \in BTYPE$ of bindings (Mietzner et al., 2009). A *constant* value val_b is bound to the variability point. Depending on the variability point, this may for example be an integer or string value. An *entity e* in the compliance descriptor is bound to the variability point. val_b identifies the entity by its name n_e. A *parameter* indicates the variability point is not yet bound in the rule, but will be bound later. val_b indicates the number vp_b shall have in the order of parameters.

The bindings B_r are called a *complete binding*, if:

$complete(B_r) := \forall vp \in VD_r : \exists b \in B_r : vp_b = vp \wedge type_b \neq parameter$

A complete binding thus provides a concrete value for each variability point. No further information is necessary to create a concrete rule.

On the other hand the bindings B_r are called a *partial binding*, if:

$partial(B_r) := (\forall vp \in VD_r : \exists b \in B_r : vp_b = vp) \wedge (\exists b \in B_r : type_b = parameter)$

Note that even a partial binding must provide a binding for each variability point. Bindings which fail both criteria are invalid.

3.4 Compliance Requirements

A compliance requirement is a single compliance-related requirement to the business process or the process environment stemming from a law. It is used to link laws and implementation. A requirement $q \in Q$ is defined as follows:

$q \in Q := (n_q, d_q, u_q, cex_r)$

where n_q is the unique name of the requirement, d_q is a description of the requirement in natural language, provided by business users. $u_q \in U$ is a law url referencing the paragraph the requirement stems from. Note multiple requirements may stem from the same law or even paragraph.

Aside from a description d_q in natural language, it contains a formal compliance expression cex_r, which describes the requirement by referencing compliance rules. The compliance expression $cex \in CEX$ is defined as follows:

$cex \in CEX := (f_{cex}, R_{cex}, B_{cex})$

A compliance expression links multiple rules $r \in R_{cex}$ in a formula f_{cex}. This formula uses Boolean operators to relate rules to each other. As rules may be variable, additional binding information may be necessary to create the rule. This is the case if $partial(B_r)$ is true. Then, additional bindings $B_{r_{add}}$ need to be specified for each binding $b \in B_r$, where $type_b = parameter$. These additional bindings are stored in B_{cex} and are defined as follows:

$B_{r_{add}} \in B_{cex} := \{(vp_b, val_b, type_b) | type_b \neq parameter \wedge (\exists b_r \in B_r : type_{b_r} = parameter \wedge vp_{b_r} = vp_b)\}$

The bindings are then combined to create are so-called final binding:

$B_{r_{final}} := B_{r_{add}} \cup \{b \in B_r | type_b \neq parameter\}$

For the final binding $complete(B_{r_{final}})$ must be true, as otherwise no concrete rule may be created. Then no deployment of rules can take place. Note that there may be multiple final bindings if a rule is used multiple times within compliance expressions.

The formula f_x defines a Boolean expression linking the rules R_{cex}. The additional bindings B_{cex} are specified using parentheses and parameter order. Quotation marks are used to bind constants, names without quotation marks are used to bind entities. The formula language provides the operators *AND* and *OR*, as well as defining precedence using parentheses. Note the absence of a negation. The reason for this is twofold. First, the negation of a rule may be counterintuitive. For example, the negation of a rule imposing activity A is always followed by activity B is not that B never follows A, but rather that in at least one possible case B does not follow A. We found that safely using these negations requires proficiency in predicate logic as well as in the implementation language which average business users do not process. Second, as rules may evaluate to *unknown*, ternary Kleene logic (Kleene, 1952) is used to evaluate compliance expressions rather than binary Boolean logic. Thus, negations are not used to approximate the behavior of the rule language to the intuitive understanding.

3.5 Graphical Model

Based on the formal description of the compliance descriptor, a graphical modeling language is designed. The modeling elements are shown in Figure 2, corresponding to the elements defined above.

A *law* is modeled as a rounded square with a section sign indicating it contains multiple paragraphs. An *entity* is modeled as an oval. All entities referenced in bindings must be explicitly added to the compliance descriptor. Modeled entities may reference their counterparts in other models, e.g. an activity in a process model.

Rules are modeled as rounded squares without any further decoration. For each entity bound in B_r, a reference to the entity has to be modeled. This explicit modeling of entities serves to visualize the impact of compliance rules and requirements on the process, which would otherwise be hidden in an attribute.

A *requirement* is modeled as a double rounded square, indicating it may consist of multiple rules. Like the rule it has to reference each entity bound in B_{cex}. Additionally, requirements have to reference all rules R_{cex} used in the compliance expression. Requirements reference the law they stem from using their *law url* u_q. In comparison to the other references, a *law url* contains an XPATH expression indicating the law l_u and the paragraph p_u. A law url is decorated with a section sign to distinguish it from the other edges.

3.6 Example

Figure 3 shows a simplified example process from our work with insurance companies (for more detail see (Koetter et al., 2014)). This claim management is used to automatically process damage claims from a customer, e.g. in case of car damages. A claim is received either digitally or in paper and stored in a customer database. The claim is then processed to find additional information, e.g. if the claim is covered by the insurance policy. Based on this data, the claim is decided to be either accepted in full, partially ac-

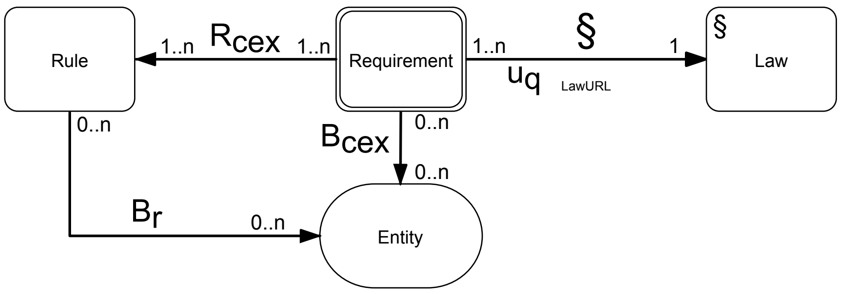

Figure 2: Modeling elements of the compliance descriptor with cardinalities and formal equivalents for all relationships.

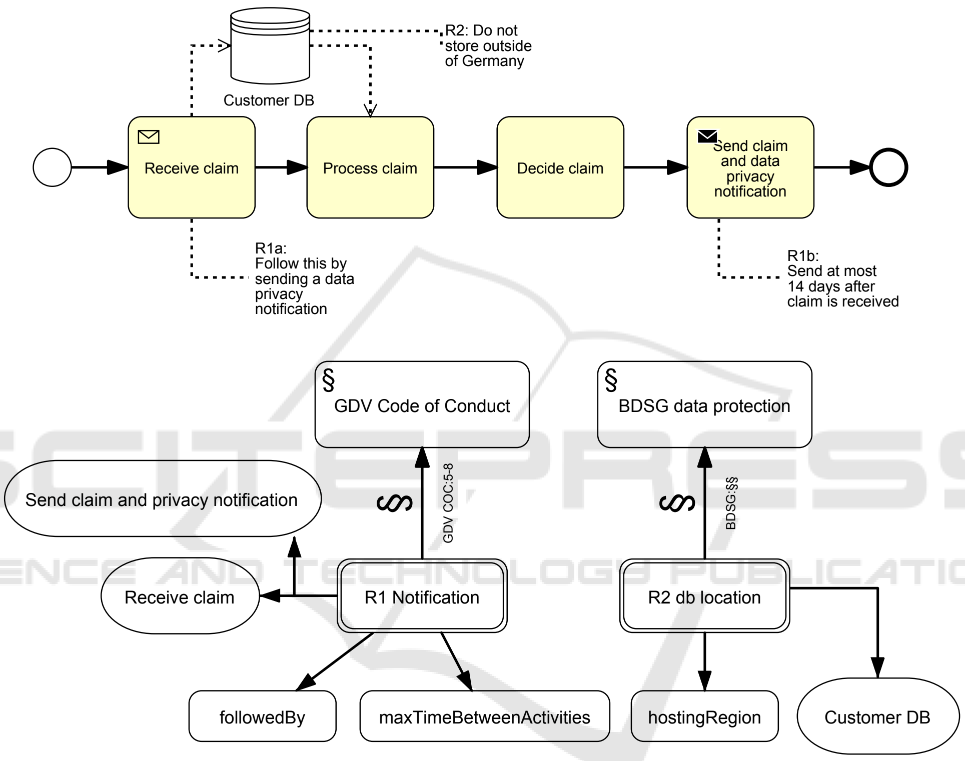
Customer DB
R2: Do not store outside of Germany
Receive claim
Process claim
Decide claim
Send claim and data privacy notification
R1a: Follow this by sending a data privacy notification
R1b: Send at most 14 days after claim is received
GDV Code of Conduct
BDSG data protection
Send claim and privacy notification
GDV COC:5-8
BDSG:§§
Receive claim
R1 Notification
R2 db location
followedBy
maxTimeBetweenActivities
hostingRegion
Customer DB

Figure 3: Example claim management process (top) and compliance descriptor (bottom).

cepted or rejected. Finally, a notification of the result is sent to the customer as a letter.

Two laws relevant for this process are the Code of Conduct of the German Insurance Association (GDV) (German Insurance Association (GDV), 2012) and the German Federal Data Protection Act (BDSG) (Bundesdatenschutzgesetz (BDSG), 1990). From these laws two requirements are derived as an example. The GDV Code of Conduct states a customer who provides personal data has to be asked in a timely fashion if this data can be used for marketing purposes. Requirement R1 is thus that such a notification takes place after data is received. This requirement is realized using two rules, *followedBy*, an LTL rule checking during design-time if an activity is always followed by another activity, and *maxTimeBetweenActivities*, a ProGoalML rule which checks at run-time that at most fourteen days pass between receipt and notification. The BDSG regulates storage, processing, and exposure of data. To comply with it, a suitable hosting provider within Germany has to be found. Requirement R2 thus states that the customer database is hosted within Germany. To check this, the rule *hostingRegion* is used, which is a TOSCA policy applied during deployment.

The graphical compliance descriptor for these processes is shown in Figure 3. It references activities of the process model and external rules (for a full specification see (Fehling et al., 2014)).

4 COMPLIANCE MANAGEMENT ARCHITECTURE

Figure 4 gives an overview of the compliance management reference architecture. An *editor* in the frontend allows the user to graphically model a compliance descriptor. Additionally, existing capabilities for process and rule modeling can be used within the editor. All models are stored in a *model repository* in the backend. From this repository, a compliance descriptor in XML can be exported. To use a compliance descriptor for compliance management, it needs to be transformed into implementation specific artifacts. For this, the XML compliance descriptor is read by the *model transformation*, which creates rule expressions as well as a so-called VisML file (Kintz, 2012), a dashboard description language which describes how rule checking results are to be visualized.

The created rule expressions are deployed to their specific *rule checking implementations* by a *rule deployment* component. For each deployed rule a *deployment descriptor* is created. The details of the deployment descriptor depend on the type of rule, but contain a unique identifier of the rule as well as all details necessary to undeploy it. During *results gathering* the deployment descriptor is used to get all results from the rule checking implementations. These results are then aggregated to determine requirement fulfillment and law compliance, thus translating compliance checking results from an IT level to a business level. All kinds of results are provided in an implementation-independent way to reporting and to a dashboard (via a *value provider*). Encapsulating rule implementations makes the architecture extensible, as only interfaces for rule deployment and result gathering need to be added for each rule language. The deployment descriptor handles implementation specific data in a generic fashion throughout the process lifecycle.

After giving an overview of the architecture, we will describe model transformation used to create concrete rule expressions and a visualization schema.

To create deployable concrete rule expressions, rules R_{final} are created for each final binding $B_{r_{final}}$ resulting from a compliance expression *cex*.

$R_{final} = \{(n_r, d_r, il_r, ph_r, rex_r, VD_r, B_{r_{final}}) | B_{r_{final}} \in \bigcup_{cex \in CEX} \{B_{r_{final}} \in B_{cex}\}\}$

Using VD_r and $B_{r_{final}}$, a concrete rule expression $rex_{r_{final}}$ is created from rex_r. This rule expression can then be deployed to its implementation as indicated by il_r. The rule creation and deployment process is described in detail in (Koetter et al., 2014).

Automatically creating process monitoring dashboards using VisML has been described in previous work (Kintz, 2012). Using a ProGoalML file as input, for each KPI or goal the appropriate visualization is selected using a visualization mapping file, and configured with the necessary data source and parameters. For the visualization of compliance requirements, a new visualization for the monitoring of compliance requirements was designed and documented in VisML, and the mapping file was extended. On the back-end, a new data value provider type was implemented to provide the data structure required by the new visualization.

The new visualization for compliance (see example in Figure 5) was conceived as follows: The dashboard presents a box for each law. The box is labelled with the laws' name and colored in green if all underlying requirements are met, in red if one or more requirement is broken, and in yellow if the status is unknown.

A dashboard user has the possibility to click on a law box. The box is then replaced by a box for each underlying requirement, colored as mentioned above. Thus, the user can immediately see which requirements are fulfilled. The user can go one step further and click on a requirement box, to show the underlying rules in the compliance expression.

Tooltips provide additional information at every stage of the drill-down process from law to requirement to rules. The visualization mapping file was extended to indicate that a law mentioned in a compliance descriptor should be rendered as a "law box" compliance visualization. The VisML generation algorithm was extended as follows:

```
$L_{gen} := \emptyset$
for each $q \in Q$
  if $l_{u_q} \notin L_{gen}$
    $L_{gen} := L_{gen} \cup \{l_{u_q}\}$
  end if
end for
if $L_{gen} \neq \emptyset$
  add compliance value provider
      to data sources
  for each $l \in L_{gen}$
    add law box visualization for $l$
        to dashboard
    add law data set for $l$
        to data sets
  end for
end if
```

In this algorithm $q \in Q$ are the compliance requirements, L_{gen} the laws on the top level of visualization, which are found by following the law urls u_q of the requirements.

The visualized data is obtained querying a data

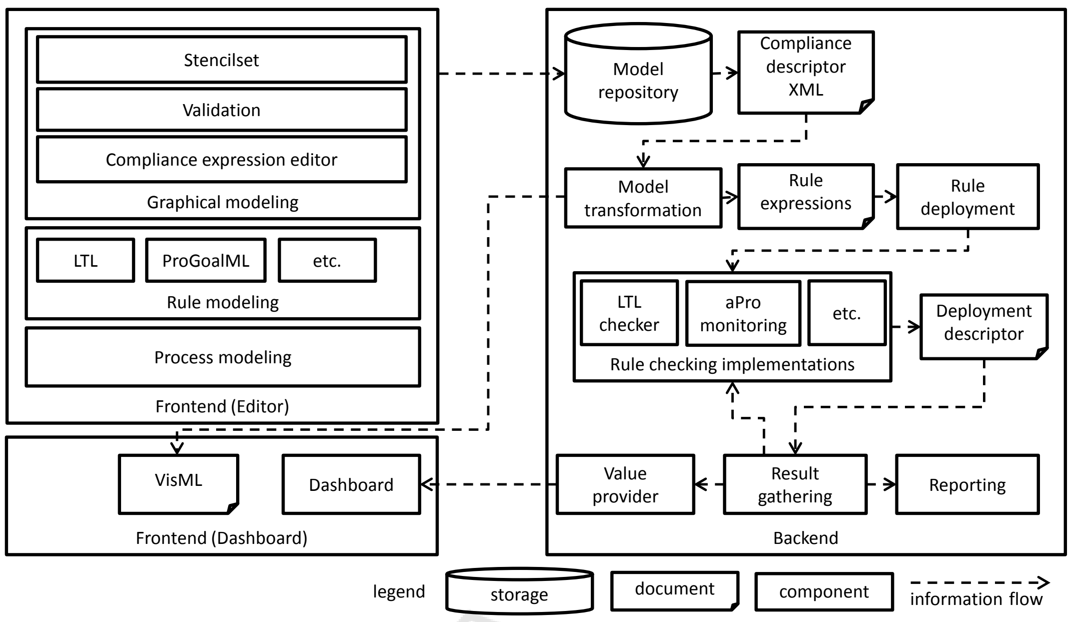

Figure 4: Compliance management architecture.

source with the appropriate parameters to obtain a data set. For compliance visualizations, a new data source *ComplianceValueProvider* was designed. It supports data sets using a parameter *law*, indicating the law for which the data is requested. It supports one or more parameters for each requirement, indicating the level of drill down.

5 PROTOTYPE AND EVALUATION

The architecture has been implemented in a prototype based on the Oryx editor[2], a web-based modeling tool. The prototype contains modeling capabilities for processes, compliance descriptors, and rules (ProGoalML and LTL).

When modeling a compliance descriptor, rules and entities may reference other models, which can be edited in another editor window. The compliance descriptor is automatically converted to XML, which is then used as a basis for model transformation. During model transformation, rules expressions are automatically created and deployed to their respective rule implementations. A VisML file is generated automatically and used with the configurable dashboard to visualize compliance checking results.

Figure 5 shows the modeling as well as the configured dashboard.

[2]http://bpt.hpi.uni-potsdam.de/Oryx (accessed 18.3.2015)

We evaluated the prototype with the example (see 3.6) and synthetic execution data for run-time rules. The process was modeled together with domain experts of the example process. We found the compliance, process, and rules monitoring to work as an integrated workflow. However, cross-model validation and other usability features like a graphical compliance expression editor may further increase usability for business users. We found the concept of encapsulating implementing rules feasible. However, some checks require additional information of the process implementation, e.g. run-time data to check timings. Currently, these need to be supplied manually to the rule checking implementations. This could be partially automated by allowing IT users to give global configuration files for each type of rule. The created dashboard shows compliance on the rule, requirement, and law levels. Using drilldown, the cause for a lack of law compliance or legal fulfillment can easily be found. Further implementation details can be found in a technical report (Fehling et al., 2014).

Using these preliminary results, we further evaluated the prototype in a real-life process in the German insurance industry and in interviews with compliance experts. Implementing compliance checking with the real process encompassed design-time and real-time. For design-time checking, rules for a claim management process were created from laws and industry guidelines and validated within the process model, proving it to be compliant. For real-time checking, compliance goals on process KPIs and timing restrictions were modeled as compliance rules, which were

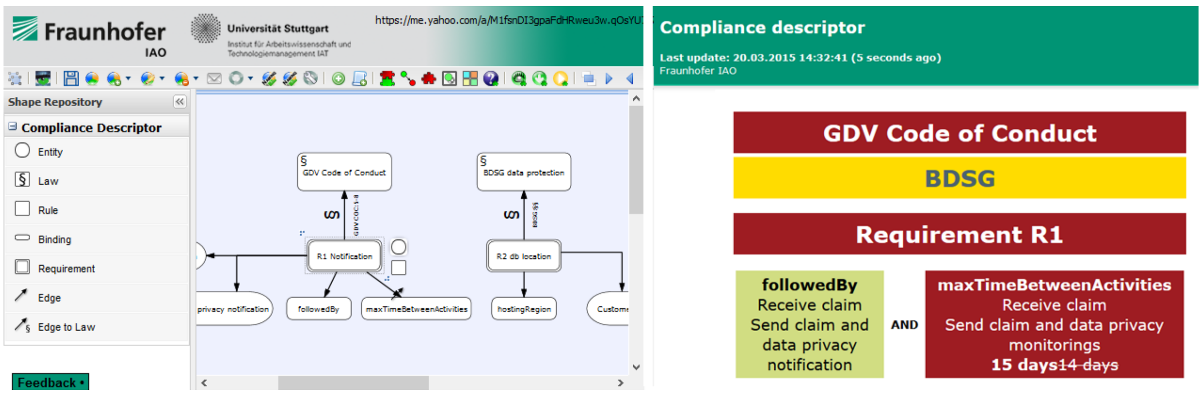

Figure 5: Prototype compliance descriptor modeling (left) and dashboard (right).

deployed to a model-driven process monitoring solution (Koetter and Kochanowski, 2013). Monitoring data was acquired from the live system using existing business monitoring information, which was manually integrated with the new monitoring solution. Run-time compliance checking was performed on a test system with real instance data. By modifying the running system, timing rules and business rules could be broken, resulting in a compliance violation, shown on the dashboard and as an alert. Deployment rules in TOSCA were not evaluated, as access to the deployment of real-life live systems was not possible. We discussed this solution with IT personnel as well as compliance specialists.

While IT personnel and specialists were generally interested in the approach, it was judged as only feasible with new or overhauled process. Efforts for integration were seen as high, even though partial automation is available, especially in existing infrastructures which cannot easily be changed and don't provide access to data in real-time. Additionally, even if rules are established, it is not possible to check past instances, as monitoring data is gathered only at runtime and not from existing sources (e.g. historical data). Especially for audits compliance checking of past process instances is necessary.

Another challenge in practice is the lack of process automation. Knowledge-intensive processes neither fully automated nor fully structured. Employees have a degree of freedom not captured in the process model. Compliance rules, even if communicated clearly to employees, are not always fulfilled. Examples named by interviewees are the omission of checks to serve customers faster and the use of non-approved or homegrown IT tools like Excel Sheets and cloud services. This suggests the need for further compliance rule types outside of the business process lifecycle not covered in this work yet. Examples could be process mining tools to compare the actual process to the prescribed process and infrastructure assessment tools to find unauthorized applications, files, and communications.

Compliance experts also noted that being notified about each compliance violation can be overwhelming in large processes. Compliance as a cross-sectional task depends on cooperation of all departments. Thus, investigating each violation independently will not be feasible, because it will require too many resources from other departments. Rather, compliance analytics should be able to find the root cause of a violation beyond the violated rule.

Considering this feedback, we find the general approach of using a compliance descriptor and different rule languages to be feasible for automated, new or overhauled business processes. To achieve compliance in real-life, large, partially automated business processes, the current range of functionality is not sufficient yet and needs to be extended considering the deficits outlined above.

6 CONCLUSION AND OUTLOOK

In this work described an integrating approach for compliance management. A compliance descriptor allows conceptual modeling of laws, requirements, and implementing rules. During model transformation the conceptual model is used to configure a reference architecture for compliance management, integrating different rule checking implementations. Using a prototype and real-life example, we have shown the practical application of these concepts, including compliance modeling, rule modeling, compliance checking, and result visualization.

In future work we will address the results of the practical evaluation, increasing usability of the prototype and adding further rule types, encompassing checks for partially automated processes and historical data.

ACKNOWLEDGEMENTS

The work published in this article was funded by the Co.M.B. project of the Deutsche Forschungsgemeinschaft (DFG) under the promotional reference SP 448/27-1.

REFERENCES

Awad, A., Decker, G., and Weske, M. (2008). Efficient compliance checking using bpmn-q and temporal logic. In *BPM '08*, pages 326–341. Springer.

Awad, A. and Weske, M. (2010). Visualization of compliance violation in business process models. In *Business process management workshops*, pages 182–193. Springer.

Bobrik, R., Reichert, M., and Bauer, T. (2007). View-based process visualization. In *Business Process Management*, pages 88–95. Springer.

Bundesdatenschutzgesetz (BDSG) (1990). Gesetze im Internet - Bundesdatenschutzgesetz (BDSG). http://www.gesetze-im-internet.de/bundesrecht/bdsg_1990/gesamt.pdf last accessed 19.01.2016.

Comuzzi, M. (2014). Aligning monitoring and compliance requirements in evolving business networks. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences*, pages 166–183. Springer.

El Kharbili, M., Stein, S., Markovic, I., and Pulvermüller, E. (2008a). Towards a framework for semantic business process compliance management. In *Processings of the 1st GRCIS*, pages 1–15.

El Kharbili, M., Stein, S., and Pulvermüller, E. (2008b). Policy-based semantic compliance checking for business process management. In *MobIS Workshops*, volume 420, pages 178–192. Citeseer.

Fehling, C., Koetter, F., and Leymann, F. (2014). Compliance Modeling - Formal Descriptors and Tools.

German Insurance Association (GDV) (2012). Verhaltensregeln fuer den Umgang mit personenbezogenen Daten durch die deutsche Versicherungswirtschaft. http://www.gdv.de/wp-content/uploads/2013/03/GDV_Code-of-Conduct_Datenschutz_2012.pdf last accessed 19.01.2016.

Karagiannis, D., Moser, C., and Mostashari, A. (2012). Compliance evaluation featuring heat maps (ce-hm): A meta-modeling-based approach. In Ralyt, J., Franch, X., Brinkkemper, S., and Wrycza, S., editors, *Advanced Information Systems Engineering*, volume 7328 of *Lecture Notes in Computer Science*, pages 414–428. Springer Berlin Heidelberg.

Kharbili, M. E., de Medeiros, A. K. A., Stein, S., and van der Aalst, W. M. P. (2008). Business process compliance checking: Current state and future challenges. In *MobIS*, volume 141 of *LNI*, pages 107–113. GI.

Kintz, M. (2012). A semantic dashboard description language for a process-oriented dashboard design methodology. In *Proceedings of 2nd MODIQUITOUS 2012*, Copenhagen, Denmark.

Kleene, S. C. (1952). Introduction to metamathematics.

Knuplesch, D., Reichert, M., Pryss, R., Fdhila, W., and Rinderle-Ma, S. (2013). Ensuring compliance of distributed and collaborative workflows. In *9th Collaboratecom*, pages 133–142. IEEE.

Kochanowski, M., Fehling, C., Koetter, F., Leymann, F., and Weisbecker, A. (2014). Compliance in bpm today - an insight into experts' views and industry challenges. In *Proceedings of INFORMATIK 2014*. GI.

Koetter, F. and Kochanowski, M. (2013). A model-driven approach for event-based business process monitoring. In *Business Process Management Workshops SE - 41*, volume 132, pages 378–389. Springer Berlin Heidelberg.

Koetter, F., Kochanowski, M., Renner, T., Fehling, C., and Leymann, F. (2013). Unifying compliance management in adaptive environments through variability descriptors (short paper). In *IEEE SOCA 2013*, pages 214–219. IEEE.

Koetter, F., Kochanowski, M., Weisbecker, A., Fehling, C., and Leymann, F. (2014). Integrating compliance requirements across business and it. In *18th EDOC*, pages 218–225. IEEE.

Ly, L. T., Knuplesch, D., Rinderle-Ma, S., Göser, K., Pfeifer, H., Reichert, M., and Dadam, P. (2011). Seaflows toolset–compliance verification made easy for process-aware information systems. In *Information Systems Evolution*, pages 76–91. Springer.

Mietzner, R., Metzger, A., Leymann, F., and Pohl, K. (2009). Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In *Proceedings of PESOS '09*, pages 18–25, Washington, DC, USA. IEEE Computer Society.

Papazoglou, M. (2011). Making business processes compliant to standards and regulations. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pages 3–13.

Patig, S., Casanova-Brito, V., and Vögeli, B. (2010). IT Requirements of Business Process Management in Practice - An Empirical Study. In *Proceedings of the 8th BPM*, pages 13–28, Heidelberg. Springer.

Ramezani, E., Fahland, D., and van der Aalst, W. M. (2014). Supporting domain experts to select and configure precise compliance rules. In *Business Process Management Workshops*, pages 498–512. Springer.

Ramezani, E., Fahland, D., van der Werf, J. M., and Mattheis, P. (2012). Separating compliance management and business process management. In *Business Process Management Workshops*, pages 459–464. Springer.

Reichert, M. and Weber, B. (2012). *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer Science & Business Media.

Sadiq, S., Governatori, G., and Namiri, K. (2007). Modeling control objectives for business process compliance. In *Business process management*, pages 149–164. Springer.

Schleicher, D., Fehling, C., Grohe, S., Leymann, F., Nowak, A., Schneider, P., and Schumm, D. (2011). Compliance domains: A means to model data-restrictions in cloud environments. In *15th EDOC*, pages 257–266. IEEE.

Semmelrodt, F., Knuplesch, D., and Reichert, M. (2014). Modeling the resource perspective of business process compliance rules with the extended compliance rule graph. In *Proceedings of the 15th BPMDS*, pages 48–63. Springer.

Takabi, H., Joshi, J. B., and Ahn, G.-J. (2010). Security and privacy challenges in cloud computing environments. *IEEE Security and Privacy*, 8(6):24–31.

Waizenegger, T., Wieland, M., Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Mitschang, B., Nowak, A., and Wagner, S. (2013). Policy4tosca: A policy-aware cloud service provisioning approach to enable secure cloud computing. In *OTM 2013*, pages 360–376. Springer.

Wei, Y. and Blake, M. B. (2010). Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing*, 14(6):72–75.

Weigand, H. and Elsas, P. (2012). Model-based auditing using {REA}. *International Journal of Accounting Information Systems*, 13(3):287 – 310. 2011 Research Symposium on Information Integrity & Information Systems Assurance.