Essential Aspects of Compliance Management with Focus on Business Process Automation

David Schumm, Tobias Anstett, Frank Leymann, Daniel Schleicher, Steve Strauch

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

Abstract: Compliance requirements coming from laws, regulations and internal policies constrain how a company may carry out its business. A company must take various different actions for preventing compliance violations and for detecting them. Business processes have to be changed accordingly in order to adhere to these requirements. Manual controls need to be installed in order to affect the work which is done outside of IT systems. Technical controls are required for assuring compliance within IT systems. In this paper, we present a compliance management model that captures the compliance problem from a holistic point of view. We elaborate on a technical control which is called compliance fragment and we position it in the compliance management model. A compliance fragment is a connected, possibly incomplete process graph that can be used as a reusable building block for ensuring a consistent specification and integration of compliance into a workflow. In particular, we propose language extensions to BPEL for representing compliance fragments. Furthermore, we introduce a methodology for integrating compliance fragments into given workflows.

1 Introduction

The term compliance denotes all measures that need to be taken in order to adhere to requirements derived from laws, regulations, internal policies etc. As non-compliance may lead to tangible punishments, immediate reactions to these requirements become a necessary part of business process management. Many of these requirements necessitate performing profound changes of the business processes and their technical implementations, referred to as workflows [LR00]. These changes have to be verifiable, traceable, and checkable for supporting governance and for providing evidence of compliance in case of an audit. Consequently, a company is compliant when it can proof that all requirements have been implemented and fulfilled in an appropriate manner. This includes that the requirements are accordingly monitored and adequately checked. Therefore, we need concepts for consistent representation of compliance requirements and methods for their reliable integration into business processes and workflows respectively.

In this paper, we build on and extend recent works related to compliance management in business processes. In [Da09] we described the research challenges for governing compliance in service-oriented architectures, spanning from design to execution and evaluation of concerns. As one part of an overall solution, we focus on workflows that have to be made compliant. In [Sc10a] we proposed to use process fragments as compliance controls and we have specified and described a life cycle for their usage. We discussed two main usage scenarios for the application of these *compliance fragments*. The first scenario is to use them during process design as reusable building blocks. They can either be glued into an existing process, or they can be used to completely build up a process from scratch. The second scenario is to annotate compliance fragments to a process in order to constrain the behavior of the process during its execution. We also investigate how to manage compliance in Software as a Service (SaaS) scenarios. In [Sc09] we introduced the concept of process templates that implicitly contain compliance controls as well as points of variability for customization. Furthermore, we presented an algorithm which ensures that these constraints are not violated. In this paper, we focus on the design time aspects of compliant workflow design. Aside from the compliance management model which we discuss in Section 3, the main contributions of this work comprise a way to represent reusable compliance controls for workflows, and a mechanism to integrate them into a given workflow.

This paper is structured as follows: the background of our approach and related work is discussed in Section 2. In Section 3 we present a general compliance management model and position the concept of compliance fragments in it. In Section 4 we specify language extensions to the Business Process Execution Language (BPEL) [Oa07] for representing compliance fragments. We also specify a methodology for integrating compliance fragments into a workflow. Finally, conclusions are provided in Section 5.

2 Background and Related Work

In the following we provide some background information on the usage of compliance fragments in process design. First of all, before a compliance fragment can be used it needs to be created. In [Sc10a] we described two manual ways. One way is that a domain expert analyzes existing processes and extracts meaningful structures, i.e., activities and control dependency among them. The other way is to design a fragment from scratch based on a specification of requirements. In order to utilize fragments for compliance both ways are applicable. The investigation of automatic techniques for identification and extraction of compliance fragments from existing processes is ongoing research in the COMPAS[1] project. However, independent of how the fragment has been created, it is important to have a proof that the process fragment is really implementing particular compliance requirements. To provide this proof, the compliance requirements need to be formalized in a language that can be understood by tools for verification and model checking. For instance, linear temporal logic (LTL) allows designers to encode requirements about the execution path in logical formulae.

[1] COMPAS: Compliance-driven Models, Languages, and Architectures for Services, www.compas-ict.eu.

Such formal rules can be used in model checkers to verify if the requirements are satisfied in a process [KA09]. The formalization of compliance requirements and their checking against a process or against a single compliance fragment is an important step for workflow compliance as we discuss in [Sc10b], but this is not the focus of this paper.

There exist different concepts for mapping compliance requirements to technical controls that can be installed in a workflow. One possibility for integrating additional steps related to compliance is the usage of sub processes or similar concepts such as Worklets [Ad05]. In this way modular requirements can be implemented, for example the retrieval and validation of a trusted timestamp from a certified timestamp provider can be defined as a sub process. Beside sub processes, there are other concepts that can be used to make a workflow compliant. Business processes can be annotated with constraints which have to be checked during design time or runtime in order to prevent violation of compliance rules. A variable can for example be annotated with a constraint allowing numbers between one and ten to be assigned. The case-handling approach [VA97] allows the ad-hoc definition of processes that adhere to certain rules like: "Activity A must be executed at least once until the process is terminated." In addition, approaches that address the detection of compliance violations in a process are currently discussed [Go09]. The concept of compliance fragments we present in this paper takes advantage of the concepts listed above and combines their strengths to ensure compliant behavior of a workflow.

3 Compliance Management in Business Processes

In this section, we propose a general model for compliance management (see Figure 1). In particular we describe the various actions a company can take in order to establish a strategy for achieving overall compliance. The compliance management model gives an overview of the different kinds of technical and manual controls. We elaborate on compliance fragments that can be installed as technical controls in workflow-based applications.

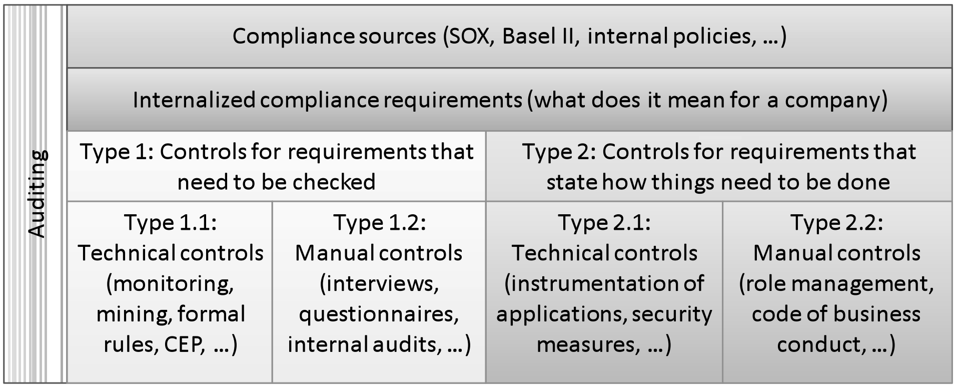

Auditing
Compliance sources (SOX, Basel II, internal policies, …)
Internalized compliance requirements (what does it mean for a company)
Type 1: Controls for requirements that need to be checked
Type 2: Controls for requirements that state how things need to be done
Type 1.1: Technical controls (monitoring, mining, formal rules, CEP, …)
Type 1.2: Manual controls (interviews, questionnaires, internal audits, …)
Type 2.1: Technical controls (instrumentation of applications, security measures, …)
Type 2.2: Manual controls (role management, code of business conduct, …)

Figure 1: Compliance management model

3.1 Compliance Management Model

An initial step of compliance management in a company is to perform a compliance assessment. In this step business experts (e.g., lawyers, consultants) collect all relevant *compliance sources* (e.g., laws, regulations, internal policies). The sources are then interpreted by these experts in order to define compliance requirements for the company and its business processes, i.e., *internalized* [Da09]. Compliance requirements basically involve any aspect of the processes in a company: requirements are related to control flow, information usage, location (of an actor in a process, data or resources), security, quality of service, monitoring, privacy, trust, and licensing [HPO08]. As compliance sources possibly give leeway in the interpretation, it is unlikely that two companies have exactly the same set of compliance requirements, or the same implementation thereof. Some requirements are generally like "use appropriate encryption methods". How such a requirement will be treated, depends on the particular interpretation of the compliance experts and on the technical infrastructure that is available in a company. In addition, as the degree of automation of the business processes differs from company to company, the choice of *controls* also differs. We distinguish between technical controls which are the measures that can be taken within IT systems, and manual controls which are measures that can be taken outside IT systems.

Type 1: Controls for requirements that need to be checked.

Type 1.1. Controls of type 1.1 provide functionality for compliance checking within IT systems. Controls of this type include checking of constraints during design time, at runtime, or offline (i.e., after-the-fact). Depending on the particular application that has to be checked, constraints can be specified in machine-readable languages like logical languages (LTL, CTL, Deontic Logic etc.), in domain-specific languages or in graph-based languages like annotation fragments as we proposed in [Sc10a]. Appropriate software components need to be installed for checking these constraints, for instance a model checker is required for checking logical constraints at design time and complex event processing (CEP) engines are often used for runtime checking. Additionally, offline monitoring is useful in many scenarios (e.g., process analysis, process mining). Controls of this type in general do not change the behavior of the IT system, but they have the prerequisite that the information which is required for checking is available. For this reason some changes of the involved systems might be necessary, in order to make the required information available. For instance, keeping records of electronic communication is crucial for providing later evidence of compliance. This might require modifications of the involved systems to support the extraction, collection, and storage of sent and received emails, committed transactions, and other data (e.g., program execution events or audit trails).

Type 1.2. Controls of type 1.2 refer to manual checking of compliance. The usage of this kind of controls is necessary when the requirement cannot be checked within an IT system in a (semi-) automated fashion. For instance, interviews and questionnaires can be used for checking processes within a company which are not automated. Recently, anonymous complaint boxes (e.g., for customers) and so-called "Whistleblowing hotlines" are also attracting interest of companies for achieving compliance.

Several forms of audits represent a supplementary control. Audits include sample checks performed for instance by a compliance officer, a technical audit of a software framework, a financial audit, or an audit with a particular subject like “customer data protection”. In addition to that, certification of a software system according to particular requirements can also be a workable measure.

Type 2: Controls for requirements that state how things need to be done.

Type 2.1. Controls of type 2.1 have an impact on the behavior of an IT system. Many compliance requirements refer to security, privacy or trust. Thus, hardening the software environment is a basic control. This includes installation and proper configuration of rights and role management (like access control lists), firewalls, anti-virus, browser security, license management etc. Custom settings of the applications and their data storage may also be necessary, e.g., configuring a database to store particular data in an encrypted manner, or installing a trigger that customer data may only be deleted after ten years. Some applications can be configured with annotations or with a deployment descriptor to use particular settings, e.g., related to security. As applications can invoke services provided by a business partner, a company must ensure that the applied service also adheres to the internalized requirements. Policies and Service Level Agreements (SLA) are used for this purpose. In our research, we focus on the process structures that allow an augmentation of the technical implementation of a business process with activities related to compliance.

Type 2.2. Controls of type 2.2 are the measures outside of IT systems. More and more companies are performing organizational changes for compliance management, e.g., the Siemens AG dramatically increased the number of employees involved in compliance programs [Si08]. This includes for instance the installation of one or more compliance officers and role management for preventing violations of segregation of duty or binding of duty requirements. Employees are instructed to adhere to the compliance requirements with particular guidelines or a “code of business conduct”. Management of the security of the company’s facility is also important. Analogous to workflows, business processes can be inter-organizational as well (e.g., involve subcontractors). Therefore, a company must ensure that its business partners also follow particular requirements. According contracts are required for this. A necessary action for enabling compliance checking and for supporting an audit is to keep records of documents, e.g., in tax consultancy most documents typically have to be stored for at least ten years. Keeping records of talks is also important, e.g., the minutes from a product advice talk in a bank are required for providing proof that the risks of a product have been properly explained.

Controls of different types can be combined to more advanced instruments [An09]. For example, software components of type 1.1 for checking for compliance violations can be combined with instruments of type 2.1 for dynamically changing the behavior of a running system (so-called compliance enforcement). Finally, for companies it is desirable to have a way to flexibly react on changes of requirements, for instance when a law is changed. Therefore, a connection of the compliance sources, the internalized requirements and the controls which are used to implement them, needs to be documented and maintained. This is also important to provide transparency to an auditor.

3.2 Compliance Fragments

A fundamental insight in our research is that compliance requirements related to workflows do not necessarily require new types of activities or completely new language constructs. The compliance requirements on which we base this work are provided by case studies which are defined by industry partners of the COMPAS research project: Thales Services SAS (France) and Telcordia Poland, in cooperation with PriceWaterhouseCoopers Accountants N.V. (the Netherlands). These requirements are mainly related to checking (*Type 1.1*) and enabling (*Type 2.1*) of security, role management, traceability, audit trails, quality of service, and licensing. Most of the security requirements related to workflows can be realized with available security frameworks. Role management in a workflow based on BPEL can be implemented with BPEL4People [Kl07] or using annotations and according tools for checking. Traceability requires the extension of a workflow engine in order to emit execution events that are augmented with unique identifiers, as our project partners showed in [Ho10]. However, what is missing for achieving compliance in service-based applications is a concept for reusable process structures that allows a consistent management of compliance requirements within workflows (*Type 2.1*, with focus on business process automation).

To illustrate our approach, we take a simplified loan approval process as an example. The activity labeled "Approval by Manager" is the step where a manger can approve a loan request (see Figure 2, left). We assume a changed internal policy, stating that a loan request can be approved without involving a manager if the risk is low. Whether a risk is high or low can be decided by a clerk. To implement this requirement, the process has to be adjusted. By reusing a compliance fragment for decisions (which someone created before) we can add the functionality to check a loan request a second time (see the center part of Figure 2). The right part of Figure 2 shows the process that is augmented with the new compliance fragment (light grey).

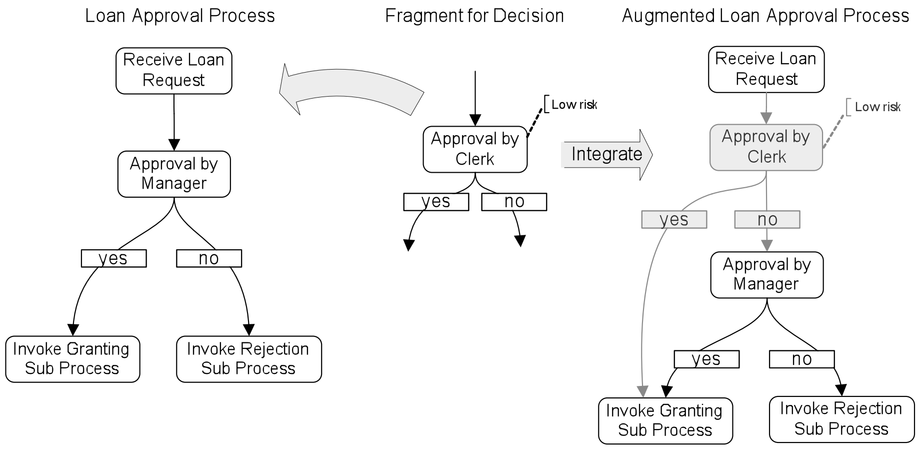


Figure 2: A loan approval process is being augmented with a compliance fragment for an additional decision

In [Sc10a] we introduced a compliance fragment as a connected, possibly incomplete process graph that can be used as a reusable building block to realize compliance requirements in terms of process logic. Compliance fragments can be used to augment a process to make it compliant to requirements related to control structures and activities to be executed. Compared to a normal process, a compliance fragment has significantly relaxed completeness and consistency criteria. For instance, a process graph has to contain a process start and a process end node for consistency. A fragment may contain such nodes, but it is not required to. In addition, a fragment may also contain activity placeholders (so-called regions) that can be filled with activities or other fragments. The concept of compliance fragments we proposed in [Sc10a] allows us to specify various different kinds of fragments. An important differentiation is how many entries and exits a fragment has. A fragment with no entries has the meaning of a process start fragment, which is useful for modelling a process from scratch. However, when using more than one of such a fragment, a conflict might occur due to multiple start nodes. The fragments we focus on have at least one entry and one exit.

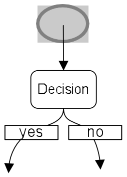
4 Managing Compliance Fragments

In this section, we propose extensions to BPEL to provide the capability to represent and serialize compliance fragments (Section 4.1). Subsequently, we present a methodology for integrating compliance fragments into a given workflow (Section 4.2).

4.1 BPEL Extensions for Compliance Fragments

We propose design time extensions to BPEL which do not require an extension of the runtime environment that executes the processes. All extensions except the extension for unique identifiers are replaced during integration of the fragment into the process (discussed in Section 4.2). We use the attribute `mustUnderstand="yes"` for the extension namespace so that a process engine will reject running a process in which not all placeholders have been replaced by standard BPEL constructs.

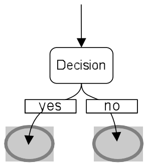
frg:fragmentEntry. The entry of a fragment is a placeholder for integration into a given process or for composition of multiple fragments. A `fragmentEntry` hast to have one or more leaving control links, and it must not have any incoming control links. An attribute (`type="mandatory|optional"`) specifies whether the entry has to be wired for assuring compliant behavior, or if it can be neglected and removed. Figure 3 illustrates the concept and the schema of the `fragmentEntry`.



```
<frg:fragmentEntry name="entryName" type="mandatory|optional">
   <bpws:sources>
      <bpws:source linkName="linkName" />+
   </bpws:sources>
</frg:fragmentEntry>
```

Figure 3: BPEL extension for fragment entry

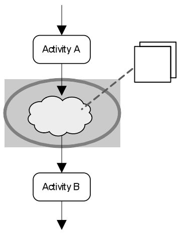
frg:fragmentExit. Analogously to an entry, a `fragmentExit` is a placeholder for a composition. It must have at least one incoming control link, and it must not have any leaving control links. A `fragmentExit` has the same attribute (`type`) as a `fragmentEntry` with analogous semantics. If a `fragmentEntry` has multiple incoming control links, then all of those control links have to have their target in one and the same activity in the final composition. Figure 4 illustrates the `fragmentExit` and its schema in BPEL.



```
<frg:fragmentExit name="exitName" type="mandatory|optional">
   <bpws:targets>
      <bpws:target linkName="linkName"/>+
   </bpws:targets>
</frg:fragmentExit>
```

Figure 4: BPEL extension for fragment exit

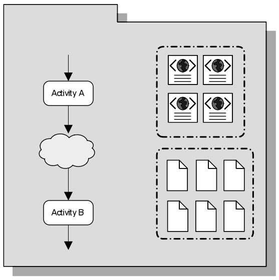
frg:fragmentRegion. A `fragmentRegion` is a placeholder that needs to be replaced with other BPEL structures. A region can be filled with standard activities, but it can also be used for composition of process fragments. Constraints can be imposed on a region for specifying how it may be filled. By using an annotation mechanism the constraints can be specified in an arbitrary format in a separate document. Figure 5 (left) shows a `fragmentRegion` (the cloud shape) on which constraints are imposed (the documents shape). We do not specify which language(s) to use for stating the constraints at this point in order to keep the overall concept modular and composable. Figure 5 (right) shows the schema for a fragment region.



```
<frg:fragmentRegion name="regionName">
   <bpws:targets>
      <bpws:target linkName="linkName"/>+
   </bpws:targets>
   <bpws:sources>
      <bpws:source linkName="linkName"/>+
   </bpws:sources>
</frg:fragmentRegion>
```

Figure 5: BPEL extension for fragment region

frg:fragmentScope. A process fragment may define a context (e.g., variables, data types). In BPEL, the `scope` construct is used for this purpose. However, we have to distinguish between the scope of a fragment and a BPEL `scope` to avoid confusion on the one hand, and to provide clear semantics on the other hand. A `fragmentScope` which is derived from the BPEL `scope` can be used as container for context constructs, such as `variables`, `partnerLinks`, `faultHandlers` etc. A `fragmentScope` has the same characteristics as a BPEL `scope` in terms of XML schema (see Figure 6).

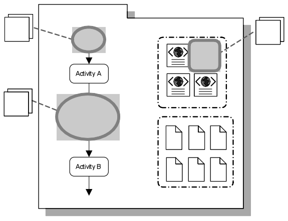


```
<bpws:extensionActivity>
   <frg:fragmentScope name="fragmentScopeName">
      <bpws:variables>
        <bpws:variable .../>+
      </bpws:variables>
      <!— Other context -->
      <frg:fragmentFlow name="fragmentFlowName">
        <bpws:links>
          <bpws:link name="linkName" />*
        </bpws:links>
        <frg:fragmentEntry ...
```

Figure 6: BPEL extension for fragment container

frg:fragmentFlow. BPEL is a hybrid language, both graph-based and block-structured [Ko09]. To support the concept of multiple entries and exits of a fragment we take the graph-based part, i.e., the BPEL `flow` construct as basis. Just like a `fragmentScope` is not the same as a BPEL `scope`, a `fragmentFlow` is not the same as a standard BPEL `flow`. When integrating the fragment into a process, the control links which are nested in the `fragmentFlow` have to be merged with the control links which are nested in the ancestor BPEL `flow`. Figure 6 shows how a `fragmentScope` and a `fragmentFlow` can be used for providing a container for a compliance fragment.

ext:id. In order to reference constructs from the outside, they must have a unique identifier. Using the name of a construct as identifier causes problems as the name might be changed often; this requires all references to be updated accordingly. Thus, we prefer a universally unique identifier (UUID) [LMS05] for this purpose. In order to keep the fragment clean from the constraints, we propose to extend all BPEL elements with an identifier attribute. An annotation mechanism which references this identifier allows the specification of constraints on regions and other BPEL constructs (e.g., impose constraints on variables) from the outside. For the identifier extension (see Figure 7) we have chosen a different namespace (with prefix `ext`) than for the other extensions as this extension can be ignored by an execution engine, i.e., `mustUnderstand="no"`.



```
<!-- for constrained entries -->
<frg:fragmentEntry name="entryName" ext:id="unique" />

<!-- for constrained regions -->
<frg:fragmentRegion name="regionName" ext:id="unique" />

<!—for constrained variables -->
<bpws:variable name="variableName" ext:id="unique" />
```

Figure 7: BPEL extension for unique identifiers

Using identifiers is also important for traceability and for maintenance reasons. Traceability in an execution environment can be supported by augmenting execution events with identifiers of the activities that are executed during runtime. The identifiers also allow distinguishing between the original parts of the process and integrated compliance fragments. The XML schema of the extensions and a concrete compliance fragment using these extensions can be found in COMPAS deliverable [Eu10].

4.2 Integrating Compliance Fragments into Workflows

During integration entries and exits of a fragment have to be “wired” with the process, i.e., control links between process activities and fragment activities are being established to obtain a complete and executable process. Placeholders are replaced with real functionality, i.e., the parameters are replaced with specific values, and regions are filled with activities or other fragments. In the following, we elaborate the different steps that have to be accomplished in order to incorporate a compliance fragment into an existing process. As illustrated in Figure 8, we use the concept of plugs to graphically represent a `fragmentEntry` and a `fragmentExit`. In order to wire a `fragmentEntry` (see Figure 8) we have to find and select a possible `fragmentExit` which can be plugged into it.

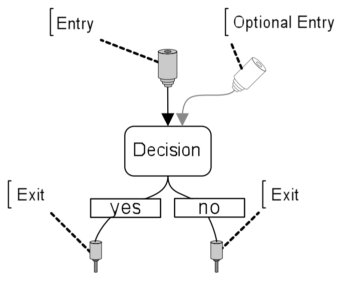


Figure 8: Compliance fragment for decisions where entries and exits are shown as plugs

If a former integration operation has not yet been completed, the available entries or exits can be chosen. Otherwise, either a new control link has to be created or an existing link has to be unplugged (i.e., broken) into a `fragmentExit` and a `fragmentEntry` as illustrated in Figure 9. Optional entries of a fragment are not required in any case. They can either be wired with the above described operations, or be removed. They are not required for the realization of the compliance feature that the fragment implements, but can be useful in process modeling. For instance, they can be used to define an additional control dependency to synchronize parallel paths. Wiring the exits of a fragment is done in an analogous manner. To wire an exit with an entry these constructs need to be plugged together: Transition conditions that possibly exist either need to be merged (using logical and) or one of them needs to be selected. Plugged connectors are transformed into one control link, i.e., a new link is inserted from the `source` of the `fragmentExit` to the `target` of the `fragmentEntry`. The helper constructs for the entry and the exit connectors are then no longer needed and can be removed.

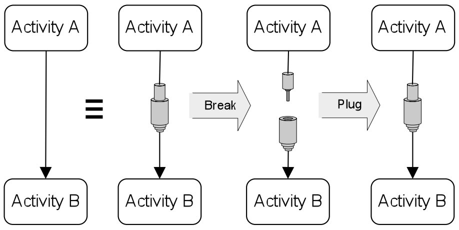


Figure 9: Breaking a control link and plugging an exit into an entry

The context defined in the `fragmentScope` (e.g., `variables`) has to be merged with the process context in order to complete the procedure of integration of a fragment into a process. Corresponding artifacts like variables can be matched based on their name, identifier, type or other attributes. Possibly, data types used in the fragment have to be adjusted to those used in the process for compatibility (e.g., Integer vs. Long). If any fault, compensation and termination handling is defined in the `fragmentScope`, this also needs to be taken into account. The main challenge when implementing design tool support for integration is how to design an easy-to-use wizard that assists the user in the wiring and in merging of the contexts. The integration is complete when all mandatory entries and exits have been wired, all optional ones are either wired or removed, all regions are filled or removed, and the context of the fragment and the process have been merged. In addition, the WSDL interface descriptions and policies which might be attached to a compliance fragment need to be combined with the WSDL and the policies of the process. After integration the process does not contain any fragment-related language extensions anymore, except for the identifiers used on the fragment constructs.

5 Conclusion and Future Work

In this paper, we have presented the essential aspects of compliance management in general and elaborated a workflow-based approach as one part of an overall solution for achieving compliance. We have discussed concrete language extensions to the process execution language BPEL for enabling compliant service composition with compliance fragments. Additionally, we have discussed a mechanism to integrate compliance fragments into workflows. With the concept of compliance fragments we can address requirements that concern the activities *within a workflow*, i.e., concerning the service invocations and human tasks defined therein, as well as their control dependency. The impact of compliance is however not limited to the workflow as we showed in the compliance management model. It also has an effect on the various applications and humans which are involved in the business processes. For instance, constraints on data storage fall into the category of requirements that can be tackled with technical controls, but not with the aid of compliance fragments. Further technical and non-technical concepts are needed for an overall approach to compliant business process automation and a compliant business.

Managing change of compliance requirements is another challenge that needs to be addressed. What is the methodology for updating a business process that has already been augmented with compliance fragments? As a first step to answer this question we investigate further techniques for managing compliance fragments (extraction, highlighting, and hiding). Furthermore, we examine the usage of different languages for stating constraints on the placeholders in a fragment, i.e., on the regions. Especially aggregation and combination of constraints in compositions is a challenge. Besides this, we are developing tools for compliance template and compliance fragment management.

Acknowledgements. The work published in this article was partially funded by the FP7 COMPAS project (www.compas-ict.eu, contract no. FP7-215175) and the FP7 MASTER project (www.master-fp7.eu, contract no. FP7-216917).

References

[Ad05] Adams, M. et al.: Facilitating Flexibility and Dynamic Exception Handling in Workflows through Worklets. Proceedings of the 17th Int. Conference on Advanced Information Systems Engineering (CAiSE'05), Springer, 2005.

[An09] Anstett, T. et al.: MC-Cube: Mastering Customizable Compliance in the Cloud. Proceedings of the 7th Int. Joint Conference on Service Oriented Computing (ICSOC'09), Springer, 2009.

[Da09] Daniel, F. et al.: Business Compliance Governance in Service-Oriented Architectures. Proceedings of the IEEE 23rd Int. Conference on Advanced Information Networking and Applications (AINA'09), IEEE, 2009.

[Eu10] European Project COMPAS: BPEL Extensions for Compliant Services. Project Deliverable D4.2, http://www.compas-ict.eu/results.php, 2010.

[Go09] Governatori, G. et al.: Detecting Regulatory Compliance for Business Process Models through Semantic Annotations. Proceedings of the Business Process Management Workshops, volume 17, chapter 2, Springer, 2009.

[Ho10] Holmes, T. et al.: Monitoring and Analyzing Service-based Internet Systems through a Model-Aware Service Environment. Proceedings of the 22nd Int. Conference on Advanced Information Systems Engineering (CAISE'10), Springer, 2010.

[HPO08] v.d. Heuvel, W.-J.; Papazoglou, M.; Orriens, B.: On the Risk Management and Auditing of SOA based Business Processes. Proceedings of the 3rd Int. Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), Springer, 2008.

[KA09] Kokash, N.; Arbab, F.: Formal Behavioral Modeling and Compliance Analysis for Service-Oriented Systems. Proceedings of the 7th Int. Symposium on Formal Methods for Components and Objects, FMCO 2008, Springer, 2009.

[Kl07] Kloppmann, M. et al.: WS-BPEL Extension for People (BPEL4People), Version 1.0, White Paper, 2007.

[Ko09] Kopp, O. et al.: The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. In: Enterprise Modelling and Information Systems. Vol. 4(1), Gesellschaft für Informatik e.V. (GI), 2009.

[LMS05] Leach, P.; Mealling, M.; Salz, R.: A Universally Unique Identifier (UUID) urn Namespace, RFC 4122, 2005.

[LR00] Leymann, F.; Roller, D.: Production Workflow. Prentice Hall PTR, 2000.

[Oa07] OASIS: Web Services Business Process Execution Language Version 2.0. OASIS Committee Specification, 2007.

[Sc09] Schleicher, D. et al.: Maintaining Compliance in Customizable Process Models. Proceedings of the 17th Int. Conference on Cooperative Information Systems (CoopIS), Springer, 2009.

[Sc10a] Schumm, D. et al.: Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI'10), Universitätsverlag Göttingen, 2010.

[Sc10b] Schumm, D. et al.: Business Process Compliance through Reusable Units of Compliant Processes. Proceedings of the 1st Workshop on Engineering SOA and the Web (ESW'10), in conjunction with ICWE'10, Springer, 2010.

[Si08] Siemens AG: Corporate Compliance Website, Report on Compliance 2008. Online, http://www.siemens.com/responsibility/report/08/en/key_figures/compliance.htm

[VA97] Voorhoeve, M.; v.d. Aalst, W.M.P.: Ad-hoc Workflow: Problems and Solutions. Proceedings of the 8th Int. Workshop on Database and Expert Systems Applications (DEXA'97), Springer, 1997.