Process Views to Support Compliance Management in Business Processes

David Schumm, Frank Leymann, Alexander Streule

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{schumm, leymann, streule}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{SchummLS10,
  author    = {David Schumm and Frank Leymann and Alexander Streule},
  title     = {Process Views to Support Compliance Management
               in Business Processes},
  booktitle = {Proceedings of the 11th International Conference on
               Electronic Commerce and Web Technologies (EC-Web 2010),
               30 August – 3 September 2010, Bilbao, Spain},
  year      = {2010},
  pages     = {131--142},
  doi       = {DOI: 10.1007/978-3-642-15208-5_12},
  series    = {Lecture Notes in Business Information Processing (LNBIP)},
  volume    = {61},
  publisher = {Springer-Verlag}
}
```

Process Views to Support Compliance Management in Business Processes

David Schumm, Frank Leymann, Alexander Streule

Institute of Architecture of Application Systems, University of Stuttgart,
Universitätsstraße 38, 70569 Stuttgart, Germany.
{Schumm, Leymann, Streule}@iaas.uni-stuttgart.de

Abstract. Compliance has become an important driver in business process management, as it requires profound and traceable changes of the processes. Besides the increasing demand for security, privacy and trust, compliance also needs consistent integration and management of process structures related to compliance. We use the notion of compliance fragments to refer to such structures. In this paper, we discuss the challenges of managing compliance fragments in business processes. Extraction, integration, highlighting and hiding of compliance fragments represent the challenges we refer to. For extraction and hiding of compliance fragments we present an implementation for the process execution language BPEL, based on process view transformation concepts.

Keywords: Process View, Model Transformation, Compliance Fragment.

1 Introduction

From a high level perspective, business process management (BPM) basically consists of three tasks. Process modeling is the first task in the life cycle of a process. In this task a process is designed or changes to an existing process are made. The result is a new or modified process. A process comprises a set of activities which have to be executed in order to achieve a business goal. So-called control flow defines the order in which the activities have to be executed. The second task is the execution of the process. The execution is supervised in process monitoring, which may run parallel to the execution. Process monitoring closes the loop and leads back to process modeling and redesign respectively.

Although, seen from a more technical perspective, there are some more steps in this life cycle. A business process is typically modeled on a high level of abstraction in a language near to business, for instance by using the Business Process Modeling Notation (BPMN) [6]. Technical refinement is a step in between process modeling and execution, here a process is prepared for execution by technical personnel. Possibly, the process also needs to be transformed into a different language for execution, for instance to the Business Process Execution Language (BPEL) [7]. Further steps cover verification, validation, and technical monitoring.

We interpret the term *compliance* as conforming to particular requirements originating from the interpretation of compliance sources [3]. We assume that a compliance assessment is made by experts (e.g. lawyers) who interpret the compliance sources and break them down to concrete requirements. Compliance sources can be laws from the executive, regulations like Basel II [14], internal policies, industrial standards and also business agreements. Non-compliance can mean significant punishments to a company. Hence, companies are in urgent need to prevent violations by detecting them and reacting accordingly. Besides the installation of one or more compliance officers who take care that all compliance requirements are met, the business processes that drive the business are affected as well. This also has an impact on the tasks related to the process life cycle described above.

Amongst other things, compliance needs to be addressed in process modeling. Some requirements occur frequently and their realization thus is feasible for reuse. We proposed the notion of process fragments for compliance, abbreviated as *compliance fragments* [11], to represent the realization of compliance requirements concerning a process. A compliance fragment can be understood as a connected sub-graph of a process graph which addresses requirements related to compliance. Such a fragment has significantly relaxed completeness and consistency criteria compared to an executable process graph. A process graph consists of nodes which represent the activities of a process, and edges which represent control dependencies.

For the management of these fragments we need several techniques. In order to create reusable compliance fragments we need a technique for extraction of process structures which realize a compliance requirement in terms of activities and control flow. Then, we need a technique for integrating such reusable compliance fragments into other business processes that have to be augmented with compliance. In order to proof compliance to an auditor, we also have to define a mechanism for highlighting integrated compliance fragments. Despite integration and highlighting we also need a method for fading those structures out. In other words we need a way to hide those steps which do not represent the actual "work" in the business process. In [11] we have denoted this as process pollution problem. In summary, compliance fragments serve as reusable process structures which realize particular compliance requirements related to a business process. Process views for compliance, on the other hand, provide a means to work with such structures.

This paper is structured as follows: Section 2 contains references to works related to our approach. Section 3 describes the challenges of managing of compliance fragments in business processes. In Section 4 the process view transformations for extraction and hiding of compliance fragments are elaborated. In Section 5 we discuss the limitations of our approach with respect to compliance management in general. Section 6 gives a short summary of the paper and characterizes future work.

2 Related Work

Due to the relevance of compliance management in business processes, an increasing number of works on this topic exists. Current approaches address all tasks related to the process life cycle, ranging from modeling of process constraints to their verification and checking for violations in monitoring. Most of the approaches are based on annotations that constrain the behavior of a process, preferably using domain-specific languages or formal rules as shown in [15]. Those approaches are very important to formally constrain a process and to formally proof compliance of a process and its execution respectively. However, these solutions do not address how to ensure a consistent specification of requirements in terms of activities and control flow in order to augment a process with compliance. We tackle this issue with the concept of compliance fragments [11], and together with our research partners we combined this concept with the formalization of requirements and process verification [17].

In our former work [11] we proposed two different methods for integrating compliance fragments into a process. The first method (which we called gluing) is to physically copy the fragment into the process. The second method is to make use of Aspect-Oriented Programming (AOP) techniques to augment a process with compliance fragments in a loosely coupled manner. This method is feasible for many compliance requirements, for instance related to auditing and logging. When aspect weaving is applied there is in fact no need for compliance fragment hiding, as the fragment is already separated from the process. However, the integration of some compliance fragments requires a physical redesign of the process, that is why gluing is not avoidable in any case. In particular, this is the case for compliance fragments with multiple entries or multiple exits, such as a compliance fragment for an approval which has one exit for acceptance, and another one for rejection.

Process views are a set of approaches addressing the increasing size of business processes, i.e. concerning the increasing number of activities which are contained in a process. Aside from the application for process abstraction, process views can also be used in other scenarios. In [1] an application of view transformations for extraction of reusable structures from Petri Nets is discussed. This mechanism is related to our approach of fragment extraction, though specified with a different purpose and for a different language. An approach for the generation of a public process for usage in outsourcing scenarios is presented in [5]. The mechanism in [5] is similar to the mechanism of hiding which we propose. However, it is also applied to a different language and limited in extensibility of the supported transformation functions. In [8] an overview on further application scenarios for process views is given. In general, most approaches make use of omission and aggregation of structures, for instance the above mentioned works. We argue that viewing concepts can be utilized as a means to support the management of compliance structures in business processes. To the best of our knowledge there is currently no comparable approach in this field.

3 Managing Compliance Fragments in Business Processes

In this section, we discuss the main challenges of managing compliance fragments in business processes, guided by a running example. A frequent compliance requirement is related to reviewing and assessing a particular situation. Let us assume for example an internal business process for approval of vacations. This process needs to be compliant with the requirements originating from internal policies. For this reason this process contains, among other steps, activities related to checking up on conflicts. To be more precise, it contains a fragment for checking up on conflicts concerning fixed appointments in the requested vacation time.

Extracting Compliance Fragments. To make the implementation of this compliance requirement reusable, the according structures need to be extracted. We propose to add a special tag on the activities that belong to the structures which should be extracted, see the illustration in Figure 1. This tag states that any kind of view transformation has to preserve those structures and all the artifacts (e.g. variables) which are related to them. With this method we need to define a view transformation that triggers the omission of all other activities. The implementation of the transformation needs to preserve the tagged structures and maintain control dependency. The result of the view transformation is a fragment for checking up on appointment conflicts. After a fragment has been extracted it can be stored in a fragment repository [13], ready for reuse, highlighting and hiding. Another possibility for the creation of a fragment is to design it from scratch as discussed in [11]. In Section 4 we present an implementation of this view transformation.

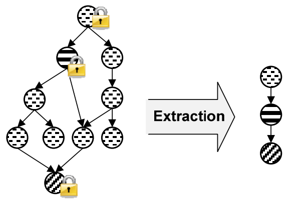

Fig. 1. Compliance Fragment Extraction

Integrating Compliance Fragments. Sometime later the internal business process for approval of business trips also needs to comply with the requirement for checking up on appointment conflicts. For this, we need to integrate the fragment which implements this requirement into the business trip approval process, see Figure 2. For the integration at first the entries and exits of a fragment have to be wired. This can be done by breaking existing control edges in the process and inserting the fragment in between existing structures, otherwise new control edges have to be inserted. To complete the integration, the context of the fragment (variables, etc.) has to be merged with the process context. During integration conflicts have to be resolved. For instance, parameter types used in the fragment possibly have to be adjusted to those used in the process (e.g. Boolean vs. String). Process view transformations are not applicable for this task. Therefore, we are currently working on a methodology for integration of compliance fragments and their related context into a given process.

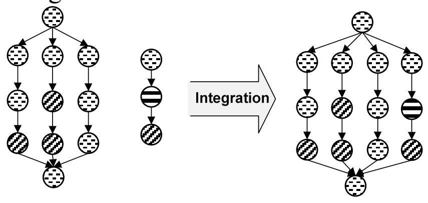

Fig. 2. Compliance Fragment Integration

Highlighting Compliance Fragments. During an audit a company has to provide all relevant information to proof compliance with laws and regulations. As described in [2], internal audits are an important measure, too. Applied to our running example, we need to provide information to an internal auditor on how we addressed the requirement for checking up on appointment conflicts. Process views also refer to the visualization of a process. Therefore, we propose using sub-graph matching algorithms [16] for the identification of known fragments related to compliance. The result of this fragment recognition step provides an input for according highlighting in graphical display of the process, as illustrated in Figure 3.

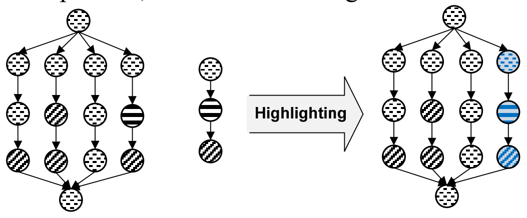

Fig. 3. Compliance Fragment Highlighting

We are currently extending our view transformation framework to support this application. The transformation component is based on Java/DOM, the visualization and modeling component is an extension of an open source process design tool [4]. In order to enable flexible highlighting we modify the predefined paint methods of the process constructs to adjust the visualization, in the same manner as shown in former work [9]. We identified several display properties of activities which can be customized to provide the highlighting (see Figure 4). For instance, using red border color with increased thickness already provides a straightforward solution. More advanced settings (e.g. involving shape size) are conceivable though.

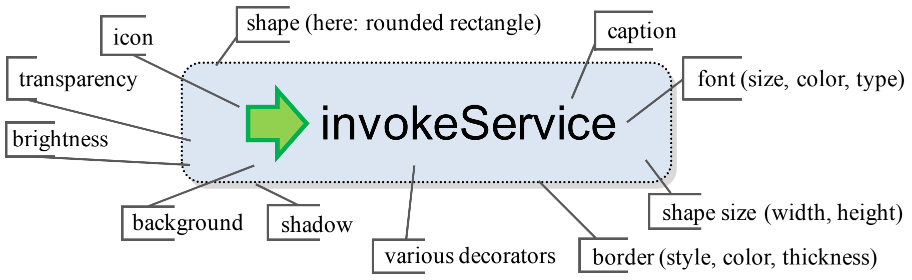

Fig. 4. Graphical Display Properties of Activities

Hiding Compliance Fragments. As mentioned in the introduction, process structures related to compliance sometimes do not represent the actual work that needs to be carried out in a process. If the number of compliance requirements that have to be addressed increases, the process becomes “polluted” and harder to understand. Therefore, we propose a view transformation for hiding those fragments in order to provide a clear view on an unpolluted process, see Figure 5. In Section 4 we discuss a technical implementation for this method.

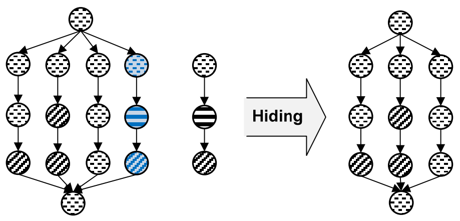

Fig. 5. Compliance Fragment Hiding

4 Extracting and Hiding of Compliance Fragments

In this section, we demonstrate how process viewing concepts can be used to extract and hide compliance fragments. In the work of [10], we have implemented a human-centric, model-driven framework for process view transformations based on the process language BPEL [7]. The BPEL standard provides a brief introduction to the notion of abstract processes. Abstract processes are either used to hide language elements of an executable process or they are not yet fully specified and serve as a process template. In our framework we currently use the metamodel of Abstract BPEL processes to represent compliance fragments. Originally, our framework has been designed to enable a semi-automatic generation of public processes to facilitate efficient process outsourcing as we discuss in [12]. However, in the following we show how to exploit and extend this framework to support also extraction and hiding of compliance fragments.

4.1 Principles of the Process Views

The process view transformation for fragment extraction which we propose requires a manual preparation step before the actual automated transformation can take place. In this preparation step the structures of the process that should be extracted have to be manually tagged for preservation. When we translate this to the example discussed in Section 3 then all structures related to checking up on appointment conflicts have to be tagged. In [10] we have shown how to extend a process design tool (Eclipse BPEL Designer [4]) to provide end-user support for this task. Figure 6 shows how this extension enables the user to add an annotation to selected activities via the context menu. The result of this step is a tagged process.

In addition to tagging, transformation rules which steer the transformation (described in detail in Section 4.2) have to be specified. These rules indicate which process constructs (targets) should be transformed, and which particular transformation operation should be performed (actions). The tagging step eases the definition of such rules, as targets can then be easily defined based on the annotations made. If no annotations are available, then construct attributes like activity name, portType etc. have to be used to select the targets.

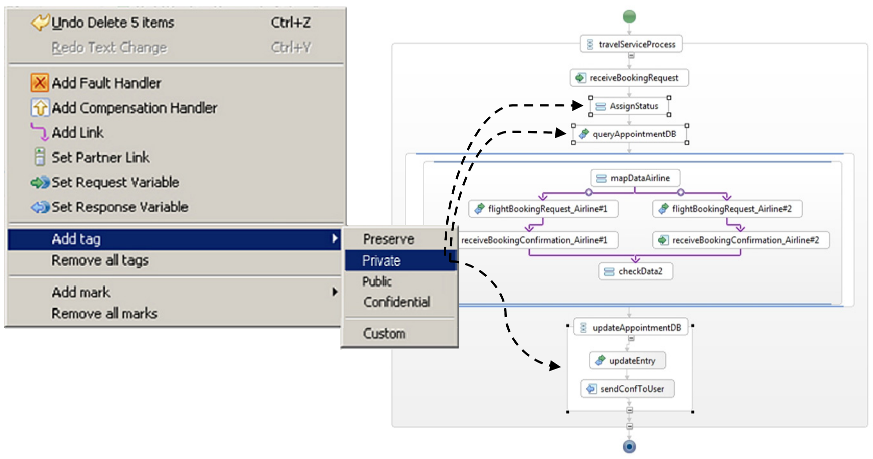

Fig. 6. Extension of the Eclipse BPEL Designer [4] for Annotation of Activities

Eventually, transformation actions are applied to the process, based on the transformation rules and the tagged input process. This results in a process view, i.e. in our extraction example in an abstract process containing only the preserved compliance fragment for checking up on appointment conflicts. The proceeding of extraction is depicted in Figure 7.

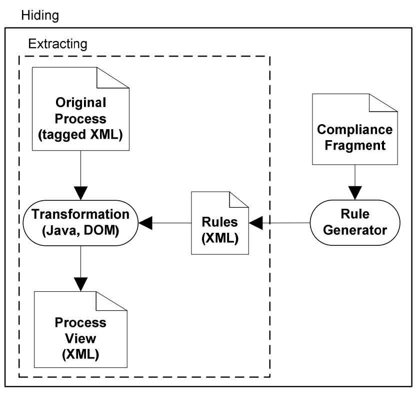

Fig. 7. Viewing Application Principle

For the implementation of the functionality for fragment hiding we have to extend this framework by another component, which we call rule generator, see the right part in Figure 7. The rule generator takes the fragment that should be hidden as input. Translated to our example, the compliance fragment for checking up on appointment conflicts would be such an input. For each activity and control structure in the fragment a rule for its omission is being generated. The generated rules are passed to the transformation component, which applies them to the input process. This transformation results in a process view that does not contain the input fragment anymore. Applied to our example, each activity of the checking up on appointment conflicts fragment would be omitted while preserving control dependencies.

4.2 Specification of the Process View Transformation

We have designed a rule language with a simple grammar which is on the one hand easy to handle, but on the other hand also capable of describing complex transformation statements. With this language a view transformation can be specified as a list of transformation rules. Each rule can target multiple constructs and trigger multiple transformation actions to be applied to these constructs. The language can be easily extended by new actions, parameterization options or new targeting possibilities. Instead of designing a new language from scratch, we could have also used (or extended) existing transformation languages like QVT (Query / View / Transformation) or ATL (ATLAS Transformation Language). However, these well-established languages are already very powerful and finding the minimal requirements a language for process view transformations has to meet was one of our research interests.

A transformation specification has the following structure:
```
<rules>            <!-- an ordered list -->
  <rule>*          <!-- multiple rules may be contained -->
      <actions/>   <!-- which actions have to be applied -->
      <targets/>   <!-- to which constructs -->
  </rule>
</rules>
```

The `<actions>` element denotes which actions have to be applied. It contains at least one `<action>` element. All nested child actions are executed in the order in which they are specified. If an action cannot be performed, e.g. if no construct is being addressed, the action will be skipped. Our framework currently supports three actions that can be used:

i. `actionOmit`: Omitting an arbitrary construct, i.e. the construct is removed from the process. If an activity is omitted, then existing control dependencies are being preserved.
ii. `actionOpaque`: Transforming an activity into an opaque activity [7], i.e. the activity is not removed completely but all implementation details are hidden.
iii. `actionSetAttributeTo`: Changing the value of an attribute of an arbitrary construct. If the value is set to NULL, the attribute is being removed.

The `<targets>` element is used to indicate the target constructs that should be affected by an action. Child elements can either be logical connectors (`<or>`, `<and>`, or `<not>`), or target elements. Logical connectors can be used to combine the different target elements. Our framework currently supports the following targeting possibilities:

i. `tag`: Targeting based on annotations, e.g. activities annotated with the tag "preserve".
ii. `attribute`: Targeting based on name-value pairs of XML attributes.
iii. `type`: Targeting based on the XML element type.

We can use the rule language to specify a general transformation rule for the extraction of compliance fragments, assuming that the parts of the compliance fragment are annotated with the tag “preserve”. The rule for extraction instructs the transformation to omit all structures which do not belong to the compliance fragment:

```
<rules>
  <rule name="extractFragment" apply="true">
      <actions>
          <actionOmit preserveChildren="true"/>
      </actions>
      <targets>
          <not>
            <tag tagName="preserve" />
          </not>
      </targets>
  </rule>
</rules>
```

For the hiding of a compliance fragment we use a rule generator to automatically create the required rules. For each construct contained in the input fragment, we generate a rule for its omission. The implementation of the omit action preserves consistency of the resulting view. The generated rules are based on the following scheme:

```
<rule name="omit%CONSTRUCT-NAME%" apply="true">
    <actions>
        <actionOmit preserveChildren="true"
                    preserveTransitionConditions="false"/>
    </actions>
    <targets>
        <attribute attributeName="name" value="%CONSTRUCT-NAME%" />
    </targets>
</rule>
```

We currently use a name-based matching for the hiding of fragments. Our framework also allows a matching based on unique identifiers though. Parameters in the elements allow refining the transformation, e.g. `preserveChildren` preserves nested structures from being removed while its parent construct is omitted. This is especially related to structured activities in BPEL like a `<forEach>` loop. `PreserveTransitionConditions` maintains all transition conditions on links in a flow (the graph-based component in BPEL) while only hiding the targeted activity itself.

4.3 Execution of the Process View Transformation

The specification and implementation of transformation actions on BPEL constructs is quite complex as the overall result of the transformation has to be consistent. The omission of activities with multiple control dependencies or the preservation of nested activities is one of the main challenges. For example, when omitting a `<sequence>` the `<sequence>` itself and all child activities are removed, except for the ones tagged with the preserve-tag. However, the result of such a transformation can be ambiguous without further information provided by the user. In this case, we decided to use a `<flow>` as a container because this construct can contain activities without any control dependencies. For this reason this construct is ideal to put the preserved activities “loosely” inside. An enhanced version could show different possible results to the user from which he may choose the one he actually intended. Furthermore, a new `<scope>` that encloses the `<flow>`-container is required. This is necessary because variables defined locally may need to be preserved as well, e.g. if the `<sequence>` contains a structure that defines variables locally.

Another example for ambiguity of the transformation is an activity which should be removed from a `<flow>` as depicted in Figure 8: Figure 8a depicts the original process, where activity X should be omitted. Figure 8b is a consistent solution to this as it maintains control dependency, though it increases complexity. Figure 8c and 8d suggest alternatives, however original semantics are changed compared to 8a. Figure 8e illustrates a solution which inhibits X from being completely removed, but provides simplicity.

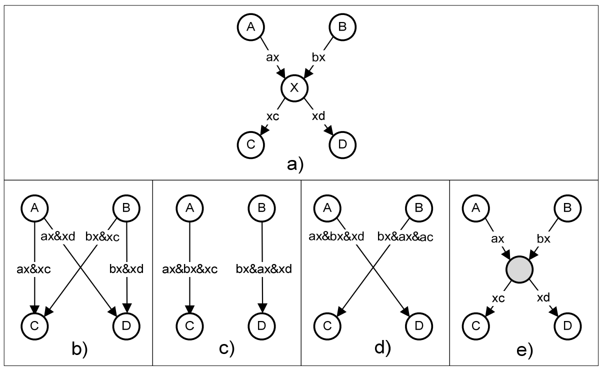

Fig. 8. Omission of Activities in a Flow

Transition conditions have to be handled in order to preserve the semantics of the process. To be consistent, omission can be realized by removing activity X and all adjoining links. Afterwards new links with appropriate transition conditions are inserted. To preserve full coherence of activities, new links containing properly concatenated transition conditions must be inserted to connect every activity to all subsequent activities (see Figure 8b). The dropping of non-needed links is performed similarly to accomplish more adequate visualization (see Figure 8c and 8d), even if semantics are changed and transition condition handling becomes quite complex. In our current implementation we have solved this problem as shown in Figure 8e. It states that X will not be removed completely when executing omission on this activity. X is transformed into an `<opaqueActivity>` automatically.

To speed up post-editing of the resulting process views, cleaning functions are implemented. It is likely that unnecessary constructs are still left in the process which should be removed automatically after the transformation rules have been applied. For instance, unused `<variables>` or `<partnerLinks>` may not longer be needed, because corresponding activities have been omitted or transformed into opaque activities. In addition, structured activities without any nested activities can be removed, e.g. removing an empty `<sequence>` is reasonable. The implementation offers a set of predefined functions to clean up the process.

5 Limitations of the Approach to Manage Compliance

The framework we presented can be extended by further transformation actions, e.g. an aggregation of multiple activities can be used for process abstraction which might ease the work of an auditor. Furthermore, many of the concepts we presented are not limited to the BPEL language and can thus be applied to other process languages as well. The process views which we proposed support the management of compliance fragments. Compliance fragments are capable of addressing compliance requirements which are related to control flow and activities within a process, but it has to be said that compliance management in general comprises many more aspects.

A fragment of a process can neither ensure compliance of the humans which are involved in that process, nor can it control the applications and services which it orchestrates. For instance, a requirement that demands storage of travel expense reports for at least ten years is related to a database which is external to the process. Besides, executing a process for over ten years would not be efficient. Even when focusing only on business process automation, compliance already has an impact on all involved components and related tasks: design, verification, technical refinement, execution, and monitoring. However, many compliance requirements (e.g. related to security, privacy or trust) have an impact on all of the components in the IT infrastructure of a company, which also includes ERP or CRM systems. In addition, compliance also has an impact on the business processes which run outside of the IT systems. This necessitates further methods to manage compliance. For instance, employees have to internalize the code of business conduct of a company - interviews can be used to check if the employees adhere to this code. Thus, compliance fragments are just one aspect of control in an overall solution to compliance management.

6 Conclusion and Future Work

In this paper we discussed the major challenges of managing compliance fragments in business processes. The main contributions of this work comprise a technique for extracting and hiding of compliance fragments based on process view transformations. Moreover, we have discussed a practical implementation of this technique for a language that is commonly used in industry for Web service orchestration. Currently we are extending our framework in order to enable automatic recognition of compliance fragments. Therefore, we utilize algorithms for sub-graph matching in order to recognize compliance fragments that are integrated in a process. With this technique we support an auditor with a tool to check if a particular compliance requirement is addressed and how it is integrated into a process. Furthermore, we are implementing support for compliance fragment integration in order to cover the whole life cycle of compliance fragment management: Extraction, integration, highlighting and hiding. As supporting infrastructure we are developing a view designer component that eases the specification of transformation rules, and a repository for storage and retrieval of processes and compliance fragments [13].

Acknowledgements
The work published in this article was partially funded by the COMPAS project (http://www.compas-ict.eu) under the EU 7th Framework Programme ICT Objective, contract no. FP7-215175.

References

1. D. Avrilionis, P.Y. Cunin, C. Fernström. OPSIS: a View Mechanism for Software Processes which Supports their Evolution and Reuse. Proc. of the 18th International Conference on Software Engineering (ICSE), IEEE Computer Society, 1996.
2. D. Caprasse, J. Laurent, W. Reed. Three Lines of Defence: How to take the Burden out of Compliance. In: European Insurance Digest, April 2008.
3. F. Daniel, F. Casati, V. D'Andrea, S. Strauch, D. Schumm, F. Leymann, E. Mulo, U. Zdun, S. Dustdar, S. Sebahi, F. de Marchi, M.S. Hacid. Business Compliance Governance in Service-Oriented Architectures. Proc. of the 23rd IEEE International Conference on Advanced Information Networking and Applications (AINA), IEEE Press, 2009.
4. Eclipse BPEL Project. Eclipse BPEL Designer, 2010. http://www.eclipse.org/bpel/
5. R. Eshuis, P. Grefen. Constructing Customized Process Views. In: Data & Knowledge Engineering, 64(2):419– 438, Elsevier, 2008.
6. Object Management Group (OMG). Business Process Modeling Notation Version 1.2. OMG Standard, 2009.
7. Organization for the Advancement of Structured Information Standards (OASIS). Web Services Business Process Execution Language Version 2.0. OASIS Standard, 2007.
8. S. Rinderle, R. Bobrik, M. Reichert, T. Bauer. Business Process Visualization – Use Cases, Challenges, Solutions. Proc. of the International Conference on Enterprise Information Systems (ICEIS), pages 204–211, INSTICC Press, 2006.
9. D. Schumm, D. Karastoyanova, F. Leymann, J. Nitzsche. On Visualizing and Modelling BPEL with BPMN. Grid and Pervasive Computing Workshops: 4th International Workshop on Workflow Management (ICWM), IEEE Press, 2009.
10. A. Streule. Abstract Views on BPEL Processes. Diploma thesis no. 2889, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2009. ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-2889/DIP-2889.pdf
11. D. Schumm, F. Leymann, Z. Ma, T. Scheibler, S. Strauch. Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. Proc. of the Multikonferenz Wirtschaftsinformatik (MKWI'10), 2010.
12. D. Schumm, F. Leymann, A. Streule. Process Viewing Patterns. Accepted for publication at the 14th IEEE International EDOC Conference (EDOC 2010), IEEE Computer Society Press, 2010.
13. Fragmento - Fragment-oriented Repository. Online Documentation, 2010. http://www.iaas.uni-stuttgart.de/forschung/projects/fragmento/start.htm
14. Basel Committee on Banking Supervision. International Convergence of Capital Measurement and Capital Standards: A Revised Framework, 2006.
15. S. Sackmann, M. Kähmer. ExPDT: A Policy-based Approach for Automating Compliance. In: Wirtschaftsinformatik (WI), 50(5), pp. 366-374, Gabler, 2008.
16. Z. Ma, W. Lu, F. Leymann. Query Structural Information of BPEL Processes. Proc. of the 4th International Conference on Internet and Web Applications and Services (ICIW), 2009.
17. D. Schumm, O. Turetken, N. Kokash, A. Elgammal, F. Leymann, W.v.d. Heuvel. Business Process Compliance through Reusable Units of Compliant Processes. Accepted for publication at the 1st Workshop on Engineering SOA and the Web (ESW'10), Springer, 2010.