Compliant Business Process Design Using Renement Layers

Daniel Schleicher, Tobias Anstett, Frank Leymann, and David Schumm

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

BIBTEX:

```
@inproceedings {INPROC-2010-76,
   author = {Daniel Schleicher and Tobias Anstett and Frank Leymann and David
Schumm},
   title = {{Compliant Business Process Design Using Refinement Layers}},
   booktitle = {accepted for publication in OTM 2010 Conferences},
   editor = {T. Dillon et al. R. Meersman},
   publisher = {Springer Verlag},
   institution = {Universit{\"a}t Stuttgart, Fakult{\"a}t Informatik,
Elektrotechnik und Informationstechnik, Germany},
   type = {Konferenz-Beitrag},
   month = {Oktober},
   year = {2010},
   keywords = {Compliance, Refinement, BPEL, business process},
   language = {Englisch},
   cr-category = {H.4.1 Office Automation},
   department = {Universit{\"a}t Stuttgart, Institut f{\"u}r Architektur von
Anwendungssystemen}
}
```

Compliant Business Process Design Using Refinement Layers

Daniel Schleicher, Tobias Anstett, Frank Leymann, and David Schumm

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

Abstract. In recent years compliance has emerged as one of the big IT challenges enterprises are faced with. The management of a multitude of regulations and the complexity of current business processes are problems that need to be addressed.
In this paper we present an approach based on so-called *compliance templates* to develop and manage compliant business processes involving different stakeholders.
We introduce the concept of a *refinement process*. In the refinement process each compliance template is refined in a layered way to get an executable business process. The refinement steps are executed on *refinement layers* by different stakeholders. Compliance constraints are used to restrict the way a compliance template can be refined. Introduced in a certain refinement layer of the refinement process, compliance constraints are propagated to higher refinement layers.

1 Introduction

In the last years compliance has become one of the most important business-requirements for companies. Due to financial and other scandals, institutions responsible for controlling certain aspects of the market decided to define huge bodies of laws and regulations to protect the economy. Recent crises, such as the subprime crisis, clarify that existing regulations like the Sarbanes-Oxley Act [2], which regulates financial transactions, need to be adapted. New regulations are required to prevent new financial disasters from happening. Furthermore regulations may also be imposed by company-internal policies and social expectations such as realising a *green IT*.

From a company's point of view the requirement of being compliant introduces new challenges on both organisational as well as on a technical level. People have to be sensitised to compliance and new jobs or departments dealing with compliance have to be established. For example the Siemens AG increased the amount of compliance officers in the business years 2006 to 2008 by factor seven [5]. Business process development tools have to be adapted to meet new requirements. Examples for these requirements are the increasing complexity of real world business processes and the growing number of compliance constraints coming from new laws or enterprise-internal regulations.

In this paper we focus on compliance by design, meaning that design-tools support the process designer in developing compliant processes. To accomplish this, we are proposing a technique to design compliant business processes based on incremental refinements of *compliance templates* introduced in [14]. Therefore we depict a concept called to model compliant business processes in a layered way. The layers of this concept are mapped to different stakeholders with different areas of expertise. With this approach external consultants can for example be involved into the modelling process. These consultants may have only partial access to the process model. Thus, companies do not have to disclose the internal logic of their business processes. They only give access to the details necessary to complete certain constraint regions.

The approach of incremental refinement today is used to build cars, houses, and software. It is a divide and conquer approach that reduces complexity of the single decisions to make during the design phase of a business process. Thus, it is a logical way to design business processes with this approach in contrast to the brute force approach where the process is developed without preparations to improve the eventual design. Incremental refinement is no new approach but we want to introduce it in the field of business process design to develop solutions dealing with compliance. Thus, the contribution of this paper is the introduction of a refinement approach in business process design to conquer the manageability of complex processes that have to be compliant to a growing number of regulations. Along with that we present solutions on how to forward constraints between layers of refinement and show how constraints are merged.

Requirements engineering is the field of research, our work on regulatory compliance is based on. Requirements to a business process are desirable or undesirable. Analogous to that, requirements to a business process are compliant or non compliant to relevant law. Compliance of a requirement refers to its relationship to relevant law. Thus, compliance introduces law as a new dimension to be handled during the development of business processes. The question if a set of requirements is compliant or not is becoming the central question of requirements engineering in the future. A software project that fails to consider regulatory compliance when collecting requirements will not be realisable [7].

This paper is structured as follows: In Section 2 we present a motivational example. Section 3 provides a short overview of the concepts of a compliance template and process fragments. Section 4 presents the main contribution of this paper by describing the concept of a layered refinement process. Section 5 explains how compliance constraints are passed on between the layers of the refinement process. The satisfiability of these constraints is examined in Section 6. Section 7 describes a prototype implementing the concepts of this paper. Related work is presented in Section 8. And finally, conclusion and future work are provided in Section 9.

2 Motivating Example

When designing a business process, typically different stakeholders are involved. As discussed by Sadiq, et al. [13], there might be a clash between the business objectives and control objectives of a process. Control objectives are imposed by compliance requirements shown in Section 1. We illustrate the concepts introduced in this section by means of an fictive company named ACME.

In the department for business process design of ACME the employees have two roles. There are process owners and compliance officers, like shown in Figure 1. The process owners are aware of the actual work that is performed on a business process. While the compliance officers have to manage compliance requirements imposed on the business processes. This can easily result in conflicts, inconsistencies and redundancies that need to be resolved.

Let us assume the ACME company has been ordered to develop a back office solution based on business process technology. The customer provided a set of compliance requirements to be obeyed. In the following we introduce a high level methodology to be used with the concept of a refinement process. The refinement process is a human-driven, non-IT process it is run by humans during the design phase of a business process.

The first step for ACME is to develop so-called compliance templates for the business processes that later form the back office application. These compliance templates are the basis for the new business processes that to be used in the back office application. The compliance templates for these processes are stored in a repository shown in Figure 1. The process owner checks out a suitable compliance template from the repository and does a first refinement. This compliance template is used as a starting point for the refinement process. Compliance-aware process design tools are needed to support the process designers to build compliant processes.

The process owner acts on layer one of the refinement process. Then, other stakeholders (e.g. compliance officers) are assigned for refinement and completion of the compliance template. One of these stakeholders can for example be the legal department of the ACME company. The legal department acts on layer two of the refinement process. It does some further refinements. Afterwards it passes the compliance template on to another department of the ACME company opening another layer of the refinement process. After several refinements are done the completed business process is returned to the process owner. The process owner verifies the collaboratively created business process and deploys it for execution. The refinement process is explained in detail in Section 4. In this section we provide answers to questions, that arise when passing a compliance template around, like: How are new constraints introduced on each layer of the refinement process passed on between the layers? And, how are constraints of different layers merged?

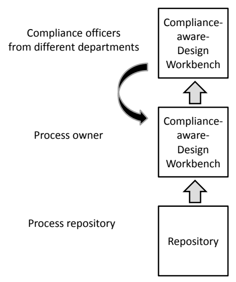

Compliance officers from different departments
Compliance-aware-Design Workbench
Process owner
Compliance-aware-Design Workbench
Process repository
Repository

Fig. 1. The refinement process: Concrete View

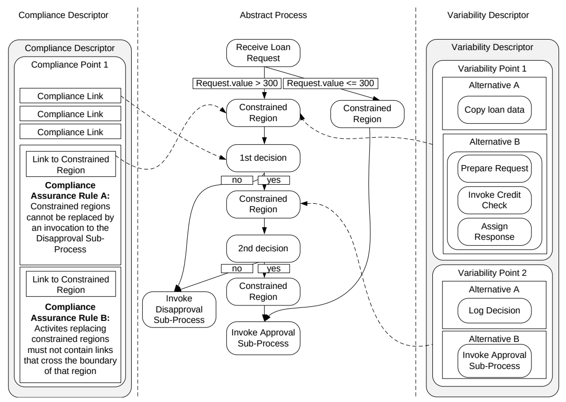


Fig. 2. Compliance Template: Basis for compliant process

3 Compliance Template

In [14] concepts and corresponding algorithms, which support the process designer in modelling compliant business processes, have been introduced. As a core concept we introduced *compliance templates*. Compliance templates consist of an *abstract business process*, *compliance descriptors* and *variability descriptors*. Figure 2 shows the abstract business process in the middle, the compliance descriptor to the left, and the variability descriptor to the right. These constituent parts of a compliance template are explained in the following.

3.1 Abstract Process

The abstract business process is the basis for the process designer to start modelling a new business process. It is called abstract, because it does not contain all information required to be automatically executed by a business process execution engine. The abstract process roughly defines how the eventual business process should behave to be compliant. It contains so-called constrained regions which need to be completed (filled with activities) in order to get an executable process. It is not allowed to change the abstract process in any other way, as this could violate compliance rules implicitly contained within the abstract process. Take a look at Figure 2 for an example. The abstract process starts with an activity receiving a loan request. Afterwards a decision is made if the value of the loan request is above or below \$300. It depends on the result of this decision if the control flow of the process takes the right path or the left path. The abstract process of this compliance template implicitly implements a number of compliance requirements. One requirement is that the activities labelled 1st decision and 2nd decision must have been executed when the process is finished compliantly and the request value is above \$300. If a process instance is finished not having executed these two activities the instance is considered as non-compliant. Another requirement is that the disapproval sub-process can only be invoked after these activities have ended. Thus, it should not be possible to delete these activities from the abstract process.

A constrained region can be filled with a single activity, with further compliance templates, or with *process fragments*. Process fragments are an approach to facilitate the reuse of certain areas of a business process as described in [15]. In this work process fragments have been introduced as connected sub graph of a process graph, whereas some parts of a fragment may be subject to parametrisation and regions may be explicitly stated as variable in order to increase the freedom for reuse.

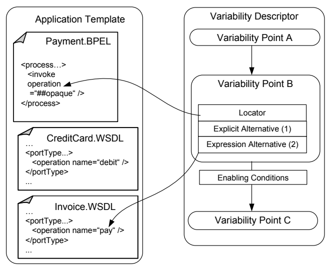
The fact, that only the constrained regions of the abstract process can be filled with activities, preserves the compliance of the process. The basic structure of the process cannot be changed and thus implicit compliance rules cannot be violated. One example for an implicit compliance rule is, activity A should always be executed before activity B.

3.2 Variability Descriptor

Variability descriptors [11] are used to describe variability of applications. For example a GUI written in HTML or in our case a business process can contain such points of variability. During the design-phase of a business process, variability descriptors provide the process designer with a choice of activities for each constrained region which can be used to fill the abstract process in order to get an executable process. Variability descriptors consist of *variability points* containing *locators* and *alternatives* (see Figure 3). Locators are used to point to artifacts of applications that are variable. The alternatives within a variability point describe the values that can be inserted at this point. Variability points can be dependent on each other. This means for example when a certain activity is used to fill a constrained region another activity from another variability point must be inserted, too. These dependencies describe the order in which variability points must be used.

Figure 3 shows a variability descriptor along with an application template. This template contains a BPEL process [12] and two corresponding WSDL-files. The locators of the variability descriptor point to the places where the BPEL process and the WSDL-files can be customised. Dependencies between the variability points A, B, and C are expressed by the arrows between them. Thus, variability point B depends on variability point A for example. That means, if variability point B is used then variability point A must be used, too.

As an example see Figure 4. Here variability point 2 is dependent on variability point 1. That means, if activity X is inserted into a constrained region, activity Y also has to be used in the process model. It depends on the constraints of the different constrained regions if activity Y can be inserted in the same constrained region as activity X.



```
<process…>
  <invoke
  operation
   ="##opaque" />
</process>
```

```
…
<portType...>
   <operation name="debit" />
</portType>
...
```

```
…
<portType...>
   <operation name="pay" />
</portType>
...
```

Fig. 3. Variability Descriptor

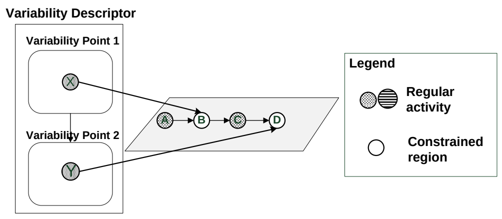


Fig. 4. Dependencies of variability points

3.3 Compliance Descriptor

Compliance descriptors (see left side of Figure 2) are defined following the design of variability descriptors. They consist of *compliance points* containing *compliance links* and *compliance assurance rules*. Compliance links point to certain activities in a business process where the corresponding compliance assurance rule should be applied. A formal representation of these rules can then be used to verify a certain replacement for a constrained region. An example for a compliance assurance rule is: "Constrained regions must not be replaced with exit activities." That means, the process designer is not allowed to insert an activity that immediately ends a process when it is executed. Without this rule a process designer could stop the execution of a process at a certain point, preventing the execution of activities important for compliant execution.

4 Refinement Layers

The abstract process contained in a compliance template provides the fundamentals to develop a compliant business process. In many cases different human process designers with different areas of expertise are involved in the design of a business process. These process designers need to be able to refine the process model according to their compliance and business requirements. For instance, dealing with quality assurance is handled differently from business to business. Some companies might have a single activity for a quality check, while others might have a multi-stage approval chain that spans over different departments.

In the following we describe a multilayer process design approach which allows such refinements, called *the refinement process*. The refinement layers of the refinement process are a concept being introduced to distinguish the work of the different process designers involved in the refinement process.

The purpose of the refinement process is to complete the abstract process of a compliance template to be executable. Many process designers of different departments of the enterprise can be involved into this process. The refinement process is started by a process designer on the layer of the abstract process. Figure 5 shows tree layers of the refinement process. The process designer takes a compliance template from the repository of the enterprise and begins to fill the constrained regions with sets of activities provided in the variability descriptor. It is not mandatory on any layer of the refinement process that all constrained regions are filled with a set of activities. It can also happen that a compliance template is further constrained by activities which are inserted. This comes from the fact, that constrained regions can be filled with a set of activities containing one ore more constrained regions. See refinement layer 2 of Figure 5 for an example. Here activity B is a constrained region.

This refinement process has the following steps:

1. A set of activities is inserted into constrained regions of the current layer. If this set of activities again contains one or more constrained regions a new refinement layer is created.
2. The inserted activities are validated against the constraints of the constrained region in which they were inserted.
3. All constraints of the constrained regions of the current layer are added to the constraints of possible new constrained regions of the higher layer.
4. Move one layer up.
5. Proceed from step one.

Each constrained region of an abstract process can be expanded into another modelling layer for refinement (see Figure 5). Within this layer, the person performing the refinement is free to place activities, as long as the result of the refinement complies with the given constraints. In Figure 5 the constrained region in the abstract process of refinement layer 1 are again refined with a process fragment.

The introduction of refinement layers is useful in many scenarios, for example when numerous departments in the management hierarchy of an enterprise have to cope with compliance in their business processes. These departments can then be mapped to corresponding refinement layers of the overall modelling process. Thus, every department has the ability to implement individual compliance requirements independently from the other departments, retaining the lower layer constraints. When changes on the process are performed on a lower layer, the persons responsible for the upper layers have to be informed to adjust their refinement layers accordingly. With the approach of refinement layers also external experts can be involved into the design phase of a business process. Enterprises typically do not want to disclose the internals of a business process. Thus, it is possible to give away parts of the business process and restrict the external expert. This is done by attaching constraints to the constrained regions that do not allow the expert to build a non-functional process.

4.1 Example

The multilayer process design approach described in this section can be applied recursively on each refinement layer. This facilitates the collaboration of independent process designers which are allowed to implement new compliance requirements on each layer of process refinement. The abstract process is refined until all constrained regions are filled and all constraints are successfully validated. Figure 5 shows an example spanning three layers.

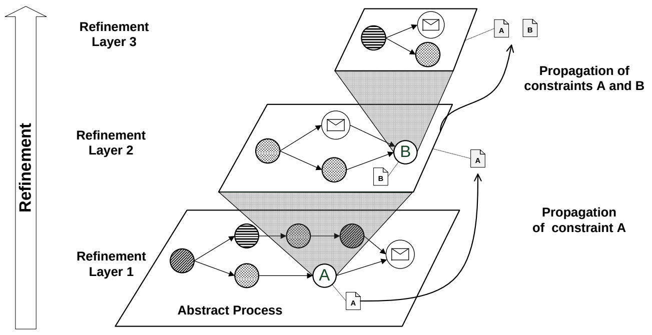


Fig. 5. Constraint-propagation over refinement layers

In the following we want to describe our intentions of compliant process modelling with a more detailed example than in Section 2. We look at a process designer working at the head office of an international bank, let us name it *InterBank*. This process designer defines a compliance template to be used to design credit-application processes. It is used on layer one of the refinement process like shown in Figure 5. The business processes that implements a credit application of the branch offices of InterBank may slightly vary in the way they handle their customer data. Thus, the process designer of layer 1 defined a constrained region where each branch office has to fill in its business logic. To ensure that the refinement of a branch office can not alter the process logic of the abstract process of layer 1, the constrained region contained in the abstract process of layer 1 is annotated with a constraint A. This constraint prevents the insertion of activities that terminate the process and modify existing variables of layer 1. After the process designer has completed the refinements, the compliance template is published to be refined in the next layer.

In layer 2 another process designer located in the head office of InterBank in Germany has to refine this abstract process to be compliant to the local customer data protection regulations. These regulations state that all data being sent away must be encrypted. Thus a process fragment is defined, adding the required business logic by using message-layer security and encryption techniques (denoted with an envelope symbol in the activity).

To allow further refinements by the local branch offices located e.g. in Hanover, the process designer integrates a constrained region within his compliance template at layer 2. To prevent further potential disclosure of information by the process designers working on the third layer, the constrained region of layer two is annotated with a constraint B which prevents the insertion of activities sending unencrypted messages. Constraints defined on lower layers must also hold on higher layers. To accomplish this we show how to propagate constraints between the layers of the refinement process in the next section.

5 Propagation of Constraints

In Figure 5 constraint A annotated to the constrained region A must hold for all activities inserted. Thus, constraint A must also hold for all activities being inserted in refinement layer 2. To meet this requirement we propose an approach called *constraint propagation*.

We assume, compliance templates always contain constrained regions. Otherwise they loose the property of being a template. Templates in a technical view are for example models for a piece of software to be built. Thus, a compliance template is a model for an eventual business process. In Figure 5, a new compliance template is inserted into constrained region A. All constraints of activity A are thus propagated to the next refinement layer. The two sets of constraints A and B are merged in refinement layer 2. Thus, constrained region B on refinement layer 2 now is annotated with the constraints A and B. This is done recursively for all further refinement layers.

A problem for the process of constraint propagation is that blind merging of two sets of constraints can lead to a set of constraints not being satisfiable anymore in a logical sense. Mutual exclusive constraints can be the cause for this. An example for a set of constraints obviously not being satisfiable is the following:

– Activities of type invoke are allowed to be placed in this activity
– Activities of all other types but the invoke-type are allowed to be placed in this activity

In the following section we take a detailed look at the satisfiability of such constraints.

6 Satisfiability of Sets of Constraints

The satisfiability problem (SAT) deals with the question whether a boolean expression is satisfiable. A boolean expression is satisfiable if there is at least one assignment making the expression true. In order to use algorithms to verify if a set of constraints is satisfiable we have to map the constraints from the representation in natural language to a representation in classical logic. To automatically verify the satisfiability of a boolean expression a number of algorithms are presented in the literature. We want to keep the discussion of what algorithm fits best for the purpose of this paper out of scope.

In the following we show examples of how constraints written in natural language can be mapped to constraints written in classical logic to clarify how constraints in classical logic should be understood. We do not intent to provide a full formal mapping from natural language to classical logic. In the future we will extend our approach to more powerful languages like linear temporal logic (LTL) or deontic logic. These languages on the one hand enable constraint designers to design more complex constraints but on the other hand the more complex structure of these languages makes it harder to validate them automatically.

6.1 Mapping of Constraints in Natural Language to Formulas in Classical Logic

The following set contains two mutual exclusive constraints. With these constraints we want to show the mapping of constraints expressed in natural language to constraints expressed in classical logic and the need for the verification of satisfiability of a set of constraints. The mapping from natural language constraints to constraints in classical logic is presented to clarify how constraints in classical logic should be read. We do not intent to provide a full formal mapping from natural language to classical logic.

1. "This constrained region must not be replaced with an exit activity".
2. "This constrained region must be replaced with an exit activity".

The first constraint can be mapped to a term in conjunctive normal form (CNF) in the following way:

1. **Identify the subject the constraint has to be applied to.** In the example of the first constraint we have one subject, the constrained region. Thus, the constraint is added to this particular constrained region.
2. **Identify the objects of the constraint.** In the example of the first constraint the object is the exit activity. We map this object to a logical variable called e. If there is more than one object present in the constraint, we map each object to a separate logical variable and put it into a separate clause. The formula would now look like this:

$$(e) \tag{1}$$

For a number of objects (e, a, and b) in a constraint the formula would look like this:

$$(e) \wedge (a) \wedge (b) \tag{2}$$

3. **Identify if a negation is applied to certain objects.** In case of the first constraint the completed formula for this constraint would now look like this:

$$(\neg e) \tag{3}$$

The formula for the second constraint would look like this:

$$(e) \tag{4}$$

We assume that the first constraint is passed on to a higher layer of the refinement process where the second constraint applies to another constrained region. We want to merge formula 4 to formula 3. This is done by appending formula 4 with an $\wedge$ operator at the end of formula 3. The new formula would now look like this:

$$(\neg e) \wedge (e) \tag{5}$$

This formula is in CNF and it is not satisfiable. The outcome of the validation of formula 5 is always false for values of e of true or false.

Mutual exclusive constraints can also be expressed with classical logic. Let us assume the following constraint in natural language: "This constrained region can be replaced by either an invoke activity or an empty activity."

If we map the invoke activity to variable i and the empty activity to variable e the corresponding logical formula in CNF would look like this:

$$(i \vee e) \tag{6}$$

Another constraint can look like this: "This constrained region can be replaced by either an invoke activity and an assign activity or an invoke activity and an empty activity." We map this constraint to classical logic in the following way. We keep the mapping of the invoke activity and the empty activity to i and e respectively. And we add a mapping of the assign activity to a. The resulting logical formula in conjunctive normal form would look like this:

$$(a \vee e) \wedge (i) \tag{7}$$

6.2 Deletion of Constraints

It is worthwhile to keep the number of constraints that are annotated to constrained regions low. One reason for a low number of constraints is the polynomial complexity of the algorithms to compute the satisfiability of logical terms [6].

We address this requirement by deleting constraints that have been satisfied. An example is shown in Figure 6. Here the constrained region in refinement layer 1 is annotated with the constraints X and Y. In refinement layer 2 a compliance template is inserted. The activity labelled X satisfies constraint X. Thus, constraint X is deleted from the set of constraints of the constrained region in refinement layer 1. Constraint X is also not propagated to refinement layer 2 in contrast to constraint Y. Further, a new constraint Z is introduced in refinement layer 2. It is merged with constraint Y of refinement layer 1. The set of activities inserted in refinement layer 3 satisfies the constraints Y and Z. Thus, they are deleted from the set of constraints. The process is now executable because there is no constrained region left to be filled. In our example no constraints are left after all constrained regions are filled with activities. But there can also be the case that not all constraints are deleted. This is the case for constraints that never can be satisfied. Like the constraint: "This constrained region must not be filled with an activity of type invoke". This constraint will never be satisfied during the refinement process because the eventual business process would become non compliant.

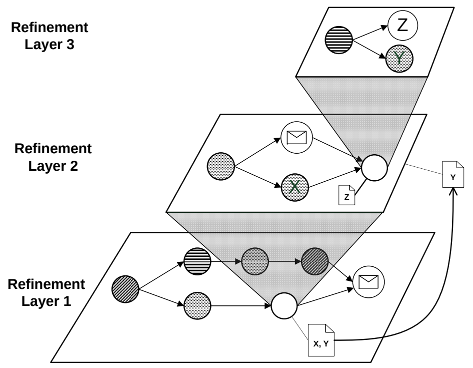


Fig. 6. Deletion of satisfied constraints

6.3 Direct and Indirect Conflicts

With the approach of deletion of constraints we solve another problem that we want to describe in the following. To do this we introduce the terms *direct* and *indirect* conflict in the refinement process.

Definition 1. *During the merge of two sets of constraints in the refinement process a direct conflict occurs when a negated literal of a lower layer clashes with a positive literal with the same name of a higher layer or vice versa.*

Figure 7 shows an example for a direct conflict. Let us assume that the constraint on refinement layer 1 states: "No activities of type A should be inserted into this constrained region". Whereas the constraint on refinement layer 2 states:"Only activities of type A can be inserted into this constrained region". In this situation the rule set of refinement layer 1 is in direct conflict with the rule set of refinement layer 2. If these sets of constraints are merged the resulting set of constraints is not satisfiable. Let us now take a look at the definition of an indirect conflict.

Definition 2. *During the merge of two sets of constraints in the refinement process an indirect conflict occurs when a positive literal of a lower layer clashes with a negative literal of a higher layer.*

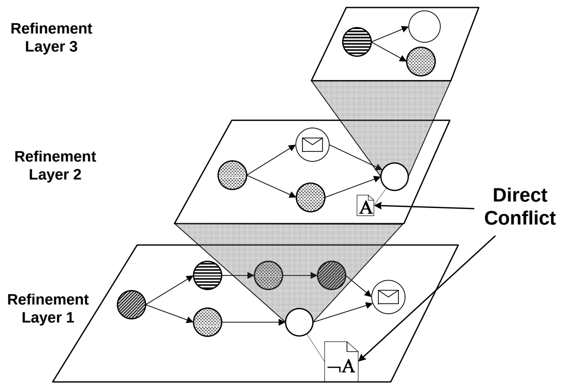


Fig. 7. Direct conflict of constraints in two refinement layers

Let us assume in Figure 7 the constraints of refinement layer 1 and refinement layer 2 have switched. When we merge these two sets of constraints, the resulting set of constraints obviously is not satisfiable. However, the constraint from refinement layer 1 could have been satisfied before the two sets of constraints are merged. This is the case when in refinement layer 2 an activity of type A has been inserted in the new constrained region. After the insertion of an activity of type A, the now satisfied constraint of refinement layer 1 is deleted. Thus, the indirect conflict is resolved.

7 Prototypical Implementation

We have extended the web based BPMN editor Oryx [1] with a number of concepts shown in this paper. Oryx does not require to install any third party software and runs in a web browser. With this editor it is possible to share process models over the Internet. Different stakeholders can use Oryx for modelling business processes in a collaborative way. It is platform independent and thus facilitates the integration of people with special skills into the process of business process modelling. Oryx has been adjusted to be run in google wave. This provides further possibilities to collaboratively design business processes. With the google wave integration two remote human process designers can see live changes of each other while modelling a business process [2].

Oryx consists of two architectural components, the front end and the back end. The front end component comprises the elements being visible in the browser window. The back end component comprises a number of Java servlets providing functionality like a process model database.

Figure 8 shows the front end of the Oryx process editor. It mainly consists of three parts. The modelling canvas in the middle, the shape repository (labelled 1),

[1] http://code.google.com/p/oryx-editor/
[2] http://www.processwave.org/

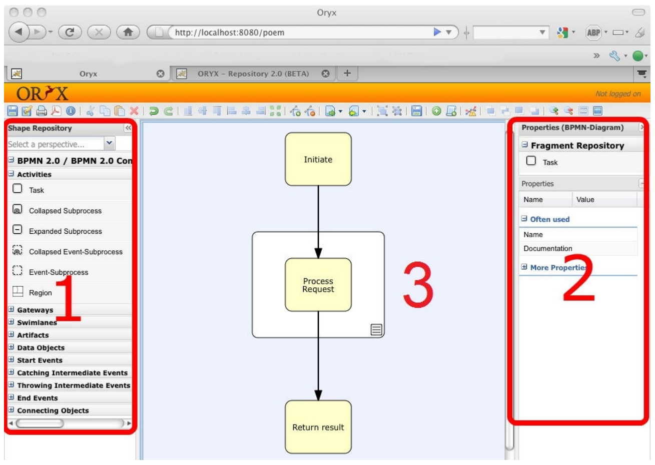


Fig. 8. Surface of Oryx BPMN modelling tool

and the properties pane (labelled 2). The shape repository contains all elements of the currently loaded process modelling language. This set of modelling elements is called a stencil set. Modelling elements can be used by dragging them from the shape repository to the canvas.

To provide the process designer with a means to develop process models containing constrained regions a new stencil has been added. We used the extension mechanism of Oryx to add the new stencil to the BPMN2.0 stencil set. This new stencil is called *Constrained Region*. The constrained region is labelled 3 in Figure 8. This stencil is capable of containing any kind of BPMN2.0 elements or group of BPMN2.0 elements up to a whole process model including other constrained regions. The stencil has a property called *compliance descriptor* used to store the constraints of the constrained region. This property is now shown on the properties pane on the right when a constrained region is selected on the canvas. The logical terms in a compliance descriptor have to be in conjunctive normal form.

Besides the new stencil we added three new functionalities to the Oryx editor, a fragment repository, a compliance checker, and a satisfiability checker for logical formulas.

The fragment repository is a frontend plugin. On the Oryx surface it is located on the right side above the properties pane in Figure 8. The fragment repository implements the concepts of the variability descriptor described in Section 3.2. It contains a list of process fragments that can be used to complete an abstract process to become an executable process.

The compliance checker is a backend plugin. It is responsible to check whether a set of activities being inserted into a constrained region does not violate any constraints. If an activity violates a constraint, red markers will be visible around the corners of the constrained region. Then it is not possible to insert the activity.

The satisfiability checker is a backend plugin. It is used to check the satisfiability of the merged sets of constraints of two constrained regions. These constraints are taken from the compliance descriptor property of a constrained region. We chose the SAT4J[3] framework to perform the check for satisfiability. If a set of constraints is not satisfiable the corresponding constrained region is marked with red corners.

With our additions the Oryx editor can now be used for two intended purposes. It can be used to design compliance templates and to complete a compliance template. Compliance templates can be created by modelling a process and inserting constrained regions into this process model. By adding compliance constraints to the compliance descriptor property, the user can restrict what kinds of activities can be inserted into specific constrained regions.

8 Related Work

An approach to annotate business processes with compliance rules is shown by Sadiq, et al. [13]. In this paper compliance objectives are modelled independently from the process models in the Formal Contract Language FCL [4]. These objectives are then used to visually annotate process models with compliance requirements. With this information process designers and compliance officers have a discussion basis to collaborate and eventually achieve the goal of a compliant process model.

An orthogonal approach is shown in [9]. In this paper compliance of a process model is checked by transforming the process model into a Pi-Calculus model and then subsequently into an finite state machine. Compliance rules are modelled in the Business Property Specification Language BPSL. The BPSL model is then translated into linear temporal logic LTL. The finite state machine and the LTL model are used to check, if the original process model is compliant to the rules defined in LTL. This is done with model checking techniques as shown by Clarke, et al. [1].

Our approach differs from the approaches above by the fact that in our approach the original process model implicitly contains the rules it should be compliant with. During the design process the software used to design the process model automatically checks the compliance of the process model.

In [16] Yi et al. show how constraints in real time systems can be verified. Here constraints like *process P will never reach state S* are verified by a backward analysis of the process graph. The analysis starts at the final state of the process and searches back to the start. Either the algorithm reaches the start of the process or not. Depending on that a constraint can be verified. Our approach differs drastically. We do not consider the whole process model to verify constraints. Instead, we consider important parts of the process when a new activity is inserted at design time. This can reduce verification time.

Eberle et al. [3] understand process fragments as building blocks for the creation of process-based applications. They discuss an algebraic foundation

[3] http://www.sat4j.org

in order to enable the representation of small pieces of process knowledge and to allow the composition of fragments into more complex structures. Ma [10] proposes process fragments as a concept for flexible and modularised reuse. In this approach a formal definition of a process fragment for the business process execution language BPEL [12] is presented. Khalaf [8] presents a static approach for distributed process execution, including the identification, creation and execution of BPEL process fragments while keeping the original execution semantics of the original process. Those approaches are relevant for our overall approach, however those approaches do not address compliance aspects.

9 Conclusion and Future Work

Building on the work presented in [14] we introduced the concept of a refinement process. The refinement process starts with an abstract business process that is refined in a layered way. The refinement can be made at different levels of granularity: either single activities can be included, higher level structures (i.e. process fragments) can be reused, or new compliance templates can be integrated in a layer. This refinement approach can be executed in enterprises facilitating the involvement of different departments in the development process. To be able to develop compliance constraints coming with constrained regions, we showed how to map constraints in natural language to logical formulas in conjunctive normal form. Furthermore we brought up concepts on how constraints on different layers of the refinement process are handled and propagated. The propagation of constraints lead us to the problem of satisfiability of boolean expressions. We addressed this problem by discussing approaches to delete satisfied constraints from these boolean expressions. Furthermore we discussed the notion of direct versus indirect conflicts of constraints and showed that only direct conflicts always lead to not satisfiable boolean expressions.

With an eye to process fragments, we will investigate how process fragments can be split up and integrated into more than one constrained region of an abstract process. In Figure 9 two process fragments are merged. We consider the process fragment containing the activity A as the abstract process from the compliance template. Now the process fragment containing the activities E, F, and G is inserted into this process. To do this, a suitable place has to be found. Requirements for this place are defined by the structure of the process fragment to be inserted. In our case the abstract process has to have two constrained regions with at least one activity in between them. Note, there could be an arbitrary set of activities between these two constrained regions. One possibility to insert the new process fragment is to place activity E in constrained region B, and activity C in constrained region F.

There is a need to support a variety of constraint languages in the refinement process. This requirement arises from the nature of the refinement process presented in this paper. As there could be separate parties developing parts of the eventual business process it is necessary to support the introduction of separate constraint languages for each refinement layer. E.g. to check properties

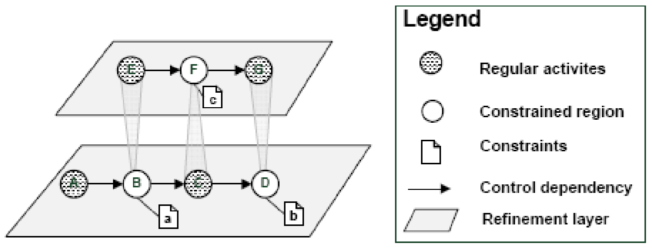


Fig. 9. Composition of process fragments

of activities used within a XML-based workflow language such as BPEL [12], XPath might be used. Also, graphical notations, like UML or BPMN, will be examined to determine if they are suitable to express compliance requirements.

We will further examine how semantics of activities can be integrated into constraints. There is a need to be able to express e.g. if a BPEL invoke activity invokes Web service A or Web service B. There is also a need to create a universal mapping from BPMN constructs to literals of logical formulas. One problem here is that a task in BPMN is a generic construct to execute something. The naming of the task tells the process developer what the task is really doing. Thus, if we want to provide a mapping from tasks to literals we need to introduce semantics in the naming of tasks.

The current prototype based on the Oryx BPMN editor should be further extended. We want to integrate two design modes. The first mode is the compliance template design mode. In this mode compliance templates can be created. This means in this mode abstract business processes can be built, compliance descriptors defined, and variability descriptors can be added to the compliance template. In the second mode only the variability descriptor is shown to the process designer. With the activities provided in the variability descriptor the process designer can complete the process.

To use the tools in enterprises we want to come up with a methodology describing what stakeholders are needed to be part of a refinement process and how tools can support this process.

10 Acknowledgements

The work published in this article was partially funded by the MASTER project[4] (contract no. FP7-216917) and the COMPAS project[5] (contract no. FP7-215175) of the EU 7th Research Framework Programme Information and Communication Technologies Objective.

[4] http://www.master-fp7.eu
[5] http://www.compas-ict.eu

References

1. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 2000.
2. U. S. Code. Sarbanes-Oxley Act of 2002, PL 107-204, 116 Stat 745, 2002.
3. H. Eberle, T. Unger, and F. Leymann. Process fragments. In R. Meersman, T. Dillon, and P. Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2009*, volume 5870 of *Lecture Notes in Computer Science*, pages 398–405. Springer, 2009.
4. G. Governatori and Z. Milosevic. A formal analysis of a business contract language. *Int. J. Cooperative Inf. Syst.*, 15(4):659–685, 2006.
5. A. Halfmann. *Siemens versiebenfacht Zahl der Compliance-Mitarbeiter*, January 2009.
6. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Pearson Addison-Wesley, Upper Saddle River, NJ, 3. edition, 2007.
7. I. Jureta, A. Siena, J. Mylopoulos, A. Perini, and A. Susi. Theory of regulatory compliance for requirements engineering. *CoRR*, abs/1002.3711, 2010. informal publication.
8. R. Khalaf. *Supporting business process fragmentation while maintaining operational semantics : a BPEL perspective.* Doctoral thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, March 2008.
9. Y. Liu, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM Syst. J.*, 46(2):335–361, 2007.
10. Z. Ma and F. Leymann. Bpel fragments for modularized reuse in modeling bpel processes. In *ICNS '09: Proceedings of the 2009 Fifth International Conference on Networking and Services*, pages 63–68, Washington, DC, USA, 2009. IEEE Computer Society.
11. R. Mietzner and F. Leymann. Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors. In *Proceedings of the International Conference on Services Computing, Industry Track, SCC 2008.* IEEE, Juli 2008.
12. OASIS. *Web Services Business Process Execution Language Version 2.0 – OASIS Standard*, 2007.
13. S. W. Sadiq, G. Governatori, and K. Namiri. Modeling control objectives for business process compliance. In G. Alonso, P. Dadam, and M. Rosemann, editors, *BPM 2007*, volume 4714 of *Lecture Notes in Computer Science*, pages 149–164, Berlin, 2007. Springer.
14. D. Schleicher, T. Anstett, F. Leymann, and R. Mietzner. Maintaining Compliance in Customizable Process Models. In *Proceedings of the 17th International Conference on Cooperative Information Systems (CoopIS 2009)*, pages 1–16. Springer Verlag, November 2009.
15. D. Schumm, F. Leymann, Z. Ma, T. Scheibler, and S. Strauch. Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. In *Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI'10).* Universitaetsverlag Goettingen, 2010.
16. W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *In Proc. of the 7th International Conference on Formal Description Techniques*, pages 223–238, 1994.