Auditing Business Process Compliance

Aditya Ghose and George Koliadis

Decision Systems Laboratory
School of Computer Science and Software Engineering
University of Wollongong, NSW 2522 Australia
{aditya, gk56}@uow.edu.au

Abstract. Compliance issues impose significant management and reporting requirements upon organizations. We present an approach to enhance business process modeling notations with the capability to detect and resolve many broad compliance related issues. We provide a semantic characterization of a minimal revision strategy that helps us obtain compliant process models from models that might be initially non-compliant, in a manner that accommodates the structural and semantic dimensions of parsimoniously annotated process models. We also provide a heuristic approach to compliance resolution using a notion of *compliance patterns*. This allows us to partially automate compliance resolution, leading to reduced levels of analyst involvement and improved decision support.

1 Introduction

Compliance management has become a significant concern for organizations given increasingly onerous legislative and regulatory environments. Legislation such as the Sarbanes-Oxley Act imposes stringent compliance requirements, and organizations are increasingly having to make heavy investments in meeting these requirements (arguably evaluated to approx. US$15 billion in year 2005 US corporate cost and $1.4 trillion in market costs [1]). Thus, we address the problem of auditing business processes for compliance with legislative/regulatory frameworks, as well as the problem of appropriately modifying processes if they are found to be non-compliant. We focus primarily on early-phase analysis and design (or model) time compliance analysis and resolution.

We will use Figure 1: a simple BPMN (see Section 1.2) "Screen Package" process owned by a $CourierOrganization$ as a motivating example. This process is concerned with scanning packages upon arrival to the organization to establish their $Status$ and ensure that they are screened by a $RegulatoryAuthority$ to determine if they should be $Held$. One simple (and critical) compliance rule imposed by a $RegulatoryAuthority$ may state that: (CR_1) *"Packages Known to be Held by a Regulatory Authority must not be Routed by a Sort Officer until the Package is Known to be Cleared by the Regulatory Authority"*.

Our challenge in this paper is to determine whether a process violates compliance requirements and to decide how to modify the process to ensure it complies. Several alternative approaches exist for the former - we therefore devote much of our attention to the latter. In particular we note that ad-hoc changes to processes in the face of non-compliance can lead to significant downsides, including additional inconsistencies,

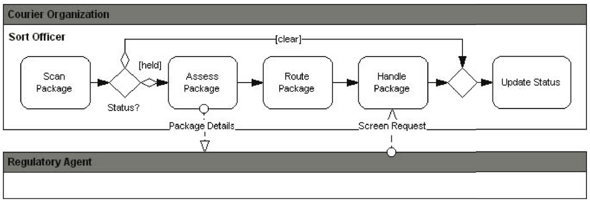

Fig. 1. Package Screening Process (*O*)

unwarranted side-effects as well as changes within the model and subsequent organization upon deployment.

We provide a conceptual framework that can be relatively easily implemented in decision-support tools that audit process models for compliance and suggest modifications when processes are found to be non-compliant. A key challenge with BPMN is that it provides relatively little by way semantics of the processes being modeled. Another challenge is that there is no consensus on how the semantics of BPMN might be defined, although several competing formalisms have been proposed. Since compliance checking clearly requires more information than is available in a pure BPMN process model, we propose a lightweight, analyst-mediated approach to semantic annotation of BPMN models, in particular, the annotation of activities with effects. Model checking is an alternative approach, but it requires mapping BPMN process models to state models, which is problematic and ill-defined.

We encode BPMN process models into semantically-annotated digraphs called Semantic Process Networks (or SPNets). We then define a class of *proximity relations* that permit us to compare alternative modifications of process models in terms of how much they deviate from the original process model. Armed with these tools we are able to resolve non-compliance by identifying minimally different process models (to the original) that satisfy the applicable compliance requirements. We are also able to focus analyst attention on the minimal sources of inconsistency (with the applicable rules) within a process model - appropriately modifying each of these is an alternative approach to restoring compliance. In addition to laying the semantic groundwork for reasoning about resolutions to process non-compliance, we also introduce the notion of process *compliance patterns*. These patterns provide heuristic guidance for detecting and resolving process non-compliance. This research lays the foundations for a new class of tools that would help analysts determine, using design-time artefacts, whether a process model satisfies the applicable compliance requirements and how best to modify these processes if they are found to be non-compliant.

1.1 Related Work

In [2], logic-based contractual formalisms are provided for specifying and evaluating the conformance of business process designs with business contracts. In comparison, we present a framework that permits the semi-automated alteration of non-compliant process models in a minimal, structure and semantics preserving manner. In [3], an approach for checking semantic integrity constraints within the narrative structure of web documents is proposed. Description logic extensions to Computational Tree Logic (CTL) are provided for specifying a formal model of a documents conventions, criteria, structure and content. In most cases, the ‘high-level’ nature of most business process models may not lead directly to detailed fine grained execution and interaction models used in model-checking based approaches to static analysis [4]. Furthermore, techniques employing model checking have limited support for localizing errors and inconsistencies to specific (or range of) elements on process models. In [5], an approach for integrating business rule definitions into process execution languages is presented. In addition, [6] have recently proposed a method for verifying semantic properties of a process w.r.t. execution traces once model change operations have been applied. Finally, heuristic change strategies have been used to provide additional guidance for scoping business process change requirements. For example, [7] present approx. thirty workflow redesign heuristics that encompass change in *task assignment*, *routing*, *allocation*, *communication* to guide performance improvement. [6] also define *insertion*, *deletion* and *movement* process change primitives to limit the scope of verifying semantic correctness of models.

1.2 Some Preliminaries

The *Business Process Modeling Notation (BPMN)* has received strong industry interest and support [8], and has been found to be of high maturity in representing the concepts required for modeling business process, apart from some limitations regarding the representation of process state and possible ambiguity of the swim-lane concept [9]. Processes are represented in BPMN using **flows**: *events*, *activities*, and *decisions*; **connectors**: *control flow links*, and *message flow links*; and **lanes**: *pools*, and *lanes* within pools. The process section in Figure 1 shows “Courier Organization” and “Regulatory Agent” participants collaborating to achieve the screening of a package.

Business (or Compliance) Rules (BR) declare constraints governing action, their coordination, structure, assignment and results, as well as the participants, their responsibilities, structure, interactions, rights and decisions. [10] provide a rich taxonomy of business rules that includes: *State Constraints*; *Process Constraints*; *Derivation Rules*; *Reaction Rules*; and, *Deontic Assignments*. In addition, the formal specification of business rules may include additional modal operators signifying the deontic (as in [2]) or temporal (as in [11]) characteristics of desirable properties of the model. For example, the following CTL expression refines the informal rule stated in Section 1:

$$\begin{aligned}&(CR_1)\ \mathbf{AG}[Knows(RegulatoryAgent, Package, Status, Held) \rightarrow\\&\mathbf{A}[\neg Performs(SortOfficer, Route, Package)\\&\mathbf{U}\ Knows(RegulatoryAgent, Package, Status, Clear)]]\end{aligned}$$

2 Modeling Business Processes for Compliance Auditing

Compliance of a business process is commonly concerned with the possible state of affairs a business process may bring to bear. *Activities* and *Sub-Processes* (i.e. represented in BPMN as rounded boxes) signify such transition of state, where the labeling of an activity (e.g. ‘Register New Customer’) abstracts one or more normal/abnormal outcomes. In order to improve the clarity and descriptive capability of process models for testing compliance, we augment state altering nodes (i.e. atomic activities and sub-processes) with parsimonious *effect annotations*. An *effect* is the result (i.e. product or outcome) of an activity being executed by some cause or agent. Table 1 below outlines the immediate effect of the tasks in Figure 1. Effects can be viewed as both: *normative* - as they state required outcomes (e.g. goals); and, *descriptive* Ð in that they describe the normal, and predicted, subset of all possible outcomes. Effect annotations can be *formal* (for instance, in first order logic, possibly augmented with temporal operators), or informal (such as simple English). Many of the examples we use in this paper rely on formal effect annotations, but most of our observations hold even if these annotations were in natural language (e.g. via Controlled Natural Languages - CNL). Formal annotations (i.e. provided, or derived from CNL), e.g. $Performs(Actor, Action, Object)$ / $Knows(Actor, Object, Property, Value)$, permit us to use automated reasoners, while informal annotations oblige analysts to check for consistency between effects.

Table 1. Annotation of Package Screening Process (O) in Figure 1

Scan Package	$Performs(SortOfficer, Scan, Package)$
Assess Package	$Performs(SortOfficer, Assess, Package)$ $\wedge Knows(RegulatoryAgent, Package, Status, Held)$
Route Package	$Performs(SortOfficer, Route, Package)$ $\wedge Knows(SortOfficer, Package, Location, Forwarding)$
Handle Package	$Performs(SortOfficer, Handle, Package)$ $\wedge Knows(RegulatoryAgent, Package, Status, Clear)$
Update Status	$Performs(SortOfficer, Update, PackageStatus)$
General Rule (GR_1)	$\forall a : Actor\ Knows(a, PackageStatus, Held)$ $\Leftrightarrow \neg Knows(a, Package, Status, Cleared)$

An *annotated BPMN model*, for the purposes of this paper, is one in which every task (atomic, loop, compensatory or multi-instance) and every sub-process has been annotated with descriptions of its immediate effects. We verify process compliance by establishing that a business process model is consistent with a set of compliance rules. In general, inconsistencies exist when some domain / process specific rules contradict each other. We evaluate compliance locally at sections of the process where they apply. However, before doing this, we require that an analyst accumulates effects throughout the process to provide a local in-context description of the *cumulative effect* at task nodes in the process. We define a process for *pair-wise effect accumulation*, which, given an ordered pair of tasks with effect annotations, determines the cumulative effect after both tasks have been executed in contiguous sequence. The procedure serves as a methodology for analysts to follow if only informal annotations are available. We assume that the effect annotations have been represented in conjunctive normal form or CNF. Simple techniques exist for translating arbitrary sentences into the conjunctive normal form.

– **Contiguous Tasks:** Let $\langle t_i, t_j \rangle$ be the ordered pair of tasks, and let e_i and e_j be the corresponding pair of (immediate) effect annotations. Let $e_i = \{c_{i1}, c_{i2}, \ldots, c_{im}\}$

and $e_j = \{c_{j1}, c_{j2}, \ldots, c_{jn}\}$ (we can view CNF sentences as sets of clauses, without loss of generality). If $e_i \cup e_j$ is consistent, then the resulting cumulative effect is $e_i \cup e_j$. Else, we define $e'_i = \{c_k | c_k \in e_i$ and $\{c_k\} \cup e_j$ is consistent$\}$ and the resulting cumulative effect to be $e'_i \cup e_j$. In other words, the cumulative effect of the two tasks consists of the effects of the second task plus as many of the effects of the first task as can be consistently included. We remove those clauses in the effect annotation of the first task that contradict the effects of the second task. The remaining clauses are undone, i.e., these effects are overridden by the second task. In the following, we shall use $acc(e_1, e_2)$ to denote the result of pair-wise effect accumulation of two contiguous tasks t_1 and t_2 with (immediate) effects e_1 and e_2. For example: $acc(\{Knows(RegulatoryAgent, Package, Status, Held)\}, \{Knows(Reg$ $ulatoryAgent, Package, Status, Clear)\}) = \{Knows(RegulatoryAgent,$ $Package, Status, Clear)\}$ in the case that GR_1 (Table 1) is considered applicable and protected.

Effects are only accumulated within participant lanes. In addition to the effect annotation of each task, we annotate each task t with a cumulative effect E_t. E_t is defined as a set $\{es_1, es_2, \ldots, es_p\}$ of alternative *effect scenarios*. Alternative effect scenarios are introduced by OR-joins or XOR-joins, as we shall see below. Cumulative effect annotation involves a left-to-right pass through a participant lane. Tasks which are not connected to any preceding task via a control flow link are annotated with the cumulative effect $\{e\}$ where e is the immediate effect of the task in question. We accumulate effects through a left-to-right pass of a participant lane, applying the pair-wise effect accumulation procedure on contiguous pairs of tasks connected via control flow links. The process continues without modification over splits. Joins require special consideration. In the following, we describe the procedure to be followed in the case of 2-way joins only, for brevity. The procedure generalizes in a straightforward manner for *n*-way joins.

– **AND-joins:** Let t_1 and t_2 be the two tasks immediately preceding an AND-join. Let their cumulative effect annotations be $E_1 = \{es_{11}, es_{12}, \ldots, es_{1m}\}$ and $E_2 = \{es_{21}, es_{22}, \ldots, es_{2n}\}$ respectively (where es_{ts} denotes an effect scenario, subscript s within the cumulative effect of some task, subscript t). Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the AND-join. We define $E = \{acc(es_{1i}, e) \cup acc(es_{2j}, e) | es_{1i} \in E_1$ and $es_{2j} \in E_2\}$. Note that we do not consider the possibility of a pair of effect scenarios es_{1i} and es_{2j} being inconsistent, since this would only happen in the case of intrinsically and obviously erroneously constructed process models. The result of effect accumulation in the setting described here is denoted by $ANDacc(E_1, E_2, e)$.

– **XOR-joins:** Let t_1 and t_2 be the two tasks immediately preceding an XOR-join. Let their cumulative effect annotations be $E_1 = \{es_{11}, es_{12}, \ldots, es_{1m}\}$ and $E_2 = \{es_{21}, es_{22}, \ldots, es_{2n}\}$ respectively. Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the XOR-join. We define $E = \{acc(es_i, e) | es_i \in E_1$ or $es_i \in E_2\}$. The result of effect accumulation in the setting described here is denoted by $XORacc(E_1, E_2, e)$.

– **OR-joins:** Let t_1 and t_2 be the two tasks immediately preceding an OR-join. Let their cumulative effect annotations be $E_1 = \{es_{11}, es_{12}, \ldots, es_{1m}\}$ and $E_2 =$

$\{es_{21}, es_{22}, \ldots, es_{2n}\}$ respectively. Let e be the immediate effect annotation, and E the cumulative effect annotation of a task t immediately following the OR-join. The result of effect accumulation in the setting described here is denoted by $ORacc(E_1, E_2, e) = ANDacc(E_1, E_2, e) \cup XORacc(E_1, E_2, e)$.

We note that the procedure described above does not satisfactorily deal with loops, but we can perform approximate checking by partial loop unraveling. We also note that some of the effect scenarios generated might be infeasible. Our objective is to devise decision-support functionality in the compliance management space, with human analysts vetting key changes before they are deployed.

3 Detecting and Resolving Compliance Issues Within Business Process Models

Compliance detection involves some machinery takes semantically annotated process models and formal representations of compliance requirements, and generates a boolean flag indicating compliance or otherwise. A simple detection procedure in our context would involve exhaustive path exploration through effect-annotated BPMN models, checking for rule violations. Due to space limitations, we do not describe this any further. When a process model is found to violate a set of compliance requirements, it must be modified to ensure compliance. In our semantically annotated example (Figure 1 and Table 1) we can simply determine that the "Route Package" node will induce an effect scenario where both $Knows(RegulatoryAgent, Package, Status, Held) \wedge Performs(SortOfficer, Route, Package)$ is true upon accumulation. It is also easy to see that our aforementioned compliance rule CR_1 is violated. Figures 2 (R_1) and 3 (R_2) describe two simple resolutions of the inconsistent "Screen Package" process in Figure 1 (O). Both these examples illustrate slight consistency preserving alterations to the process models for illustrating how we may automate their selection.

Any approach to revising process models to deal with non-compliance must meet the following two requirements. First, the revised process must satisfy the intent or goals of the original process. Second, it must deviate as little as possible from the original process. The requirement for minimal deviation is driven by the need to avoid designing new processes from scratch (which can require significant additional investment) when an existing process is found to be non-compliant. While the analysis relies exclusively on design-time artefacts, the process in question might have already been implemented or resources might have been allocated/configured to meet the requirements of the original

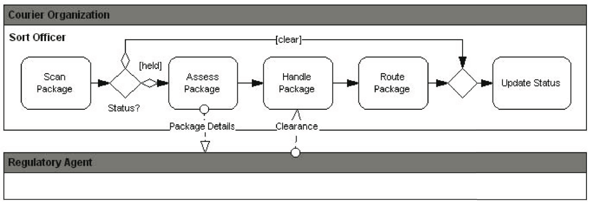

Fig. 2. Resolved Package Screening Process (R_1)

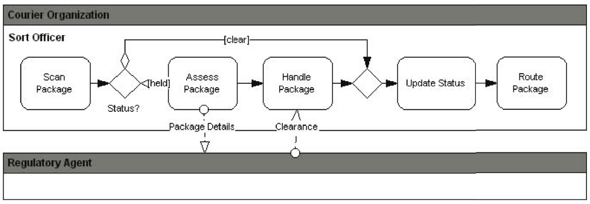

Fig. 3. Resolved Package Screening Process (R_2)

process. By seeking minimally different processes from the original one, we are able to avoid disruptive changes to the organizational context of the process.

We begin by describing what it means for a process model to minimally deviate from another. This task is complicated by the fact that there is no consensus on the semantics for BPMN (our chosen process modeling notation, selected for its widespread use in industry). Little exists in the literature on measures of deviation for process models ([12] provides some similarity measures, but these rely on petri net models of processes). We address this problem by exploiting both the structure of BPMN process models and the lightweight semantic annotations described earlier in the paper. To provide a uniform basis for conjoint structural and semantic comparisons, we encode effect-annotated BPMN models into *semantic process networks* (or SPNets).

Definition 1. *A Semantic Process Network (SPNet) is a digraph (V, E), where:*

- *each node is of the form $\langle ID, nodetype, owner, effect_I, effect_C\rangle$, and*
- *each edge is of the form $\langle\langle u, v\rangle, edgetype\rangle$.*

Each event, activity or gateway in a BPMN model maps to a node, with the $nodetype$ indicating whether the node was obtained from an event, activity or gateway respectively in the BPMN model. The ID of nodes of type $event$ or $activity$ refers to the ID of the corresponding event or activity in the BPMN model. The ID of a $gateway$ type node refers to the condition associated with the corresponding gateway in the BPMN model. The $owner$ attribute of a node refers to the role associated with the pool from which the node was obtained. The $effect_I$ of a node corresponds to the set of sentences describing the immediate effects of that node, and $effect_C$ the cumulative effect at the node within the network - these are only defined for nodes obtained from activities, and are empty in other cases. Note that $effect_I$ is a set of sentences, while $effect_C$ is a set of sets of sentences, with each element of $effect_C$ representing a distinct effect scenario. The $edgetype$ of an edge can be either control *or* message *depending on whether the edge represents a control flow or message flow in the BPMN model.*

We note that a unique SPNet exists for each process model in BPMN. This can be determined objectively by transforming BPMN models into a predetermined normal form. The BPMN notation illustrates how certain modeling patterns can be transformed into equivalent and far less ambiguous format.

Definition 2. *Associated with each SPNet spn is a* proximity relation $\leq_{spn}$ *such that* $spn_i \leq_{spn} spn_j$ *denotes that* spn_i *is closer to spn than* spn_j. $\leq_{spn}$, *in turn, is defined by a triple* $\langle \leq_{spn}^{V}, \leq_{spn}^{E}, \leq_{spn}^{EFF} \rangle$ *where:*

- $\leq_{spn}^{V}$ *is a proximity relation associated with the set of nodes* V *of spn,*
- $\leq_{spn}^{E}$ *is a proximity relation associated with the set of edges* E *of spn and*
- $\leq_{spn}^{EFF}$ *is a proximity relation associated with the set of cumulative effect annotations associated with nodes in spn.* $spn_i \leq_{spn} spn_j$ *iff each of* $spn_i \leq_{spn}^{V} spn_j$, $spn_i \leq_{spn}^{E} spn_j$ *and* $spn_i \leq_{spn}^{EFF} spn_j$ *holds. We write* $spn_i <_{spn} spn_j$ *iff* $spn_i \leq_{spn} spn_j$ *and at least one of* $spn_i <_{spn}^{V} spn_j$, $spn_i <_{spn}^{E} spn_j$ *or* $spn_i <_{spn}^{EFF} spn_j$ *holds.*

The proximity relations $\leq_{spn}^{V}, \leq_{spn}^{E}$ and $\leq_{spn}^{EFF}$ can be defined in different ways to reflect alternative intuitions. For instance, the following, set inclusion-oriented definition might be of interest: $spn_i \leq_{spn}^{V} spn_j$ iff $(V_{spn} \Delta V_{spn_i}) \subseteq (V_{spn} \Delta V_{spn_j})$, where $A \Delta B$ denotes the symmetric difference of sets A and B. An alternative, set cardinality-oriented definition is as follows: $spn_i \leq_{spn}^{V} spn_j$ iff $|V_{spn} \Delta V_{spn_i}| \leq \left|V_{spn} \Delta V_{spn_j}\right|$ (here $|A|$ denotes the cardinality of set A). Similar alternatives exist for the $\leq_{spn}^{E}$ relation. Both $\leq_{spn}^{V}$ and $\leq_{spn}^{E}$ define the structural proximity of one SPNet to another.

Take R_1 (Figure 2) and R_2 (Figure 3) as examples to illustrate our structural proximity relations. Trivially, R_1 and R_2 share all their nodes with O, and therefore, no comparison can be made across this structural dimension. Next, we determine a significant edge difference between R_1 and O, including the "Handle Package' $\rightarrow$ 'Route Package' edge. R_2 also differs with O across some edges including "Update Status" $\rightarrow$ "Route Package". If an inclusion-oriented definition for proximity (i.e. $\leq \,_{spn}^{E}$ in Definition 2) were applied, we would not be able to differentiate R_1 and R_2 w.r.t. structural proximity to O. On the other hand, if we choose to apply the cardinality-oriented definition, we'd determine $R_2 \leq \,_{spn}^{E} R_1$ as $|R_1 \Delta O| = 6$ and $|R_2 \Delta O| = 4$ (see Table 2). We can comprehend that an inclusion-oriented definition would ensure less commitment and greater control for analysts.

Defining the proximity relation $\leq_{spn}^{EFF}$ is somewhat more complicated, since it explores semantic proximity. One approach is to look at the terminating or leaf nodes in an SPNet, i.e., nodes with no outgoing edges. Each such node might be associated with multiple effect scenarios. The set of all effect scenarios associated with every terminating node in an SPNet thus represents a (coarse-grained) description of all possible end-states that might be reached via the execution of some instance of the corresponding process

Table 2. Edge Difference of R_1 and R_2 w.r.t. O

$R_1 \Delta O$	$AssessPackage \rightarrow HandlePackage\ (R_1)$, $XORjoin \rightarrow RoutePackage\ (R_1)$ $RoutePackage \rightarrow UpdateStatus\ (R_1)$, $XORjoin \rightarrow UpdateStatus\ (O)$ $RoutePackage \rightarrow HandlePackage\ (O)$, $AssessPackage \rightarrow RoutePackage\ (O)$
$R_2 \Delta O$	$AssessPackage \rightarrow HandlePackage\ (R_2)$, $UpdateStatus \rightarrow RoutePackage\ (R_2)$ $AssessPackage \rightarrow RoutePackage\ (O)$, $RoutePackage \rightarrow HandlePackage\ (O)$

model. For an SPNet spn, let this set be represented by $T_{spn} = \{es_1, \ldots, es_n\}$ where each es_i represents an effect scenario. Let $Diff(spn, spn_i) = \{d_1, \ldots, d_m\}$ where d_i is the smallest cardinality element of the set of symmetric differences between $es_i \in T_{spn_i}$ and each $es \in T_{spn}$. In other words, let $S(es_i, T_{spn}) = \{es_i \Delta e \mid e \in T_{spn}\}$. Then d_i is any (non-deterministically chosen) cardinality-minimal element of $S(es_i, T_{spn})$. Then we write $spn_i \leq^{EFF}_{spn} spn_j$ iff for each $e \in Diff(spn, spn_i)$, there exists an $e' \in Diff(spn, spn_j)$ such that $e \subseteq e'$.

The definition of $\leq^{EFF}_{spn}$ above exploited set inclusion. An alternative, cardinality-oriented definition is as follows: $spn_i \leq^{EFF}_{spn} spn_j$ iff

$$\sum_{d \in Diff(spn, spn_i)} d \leq \sum_{d \in Diff(spn, spn_i)} d$$

The two approaches to defining $\leq^{EFF}_{spn}$ presented above focus on the cumulative end-effects of processes, thus ensuring that modifications to processes deviate minimally in their final effects. In some situations, it is also interesting to consider minimal deviations of the internal workflows that achieve the end-effects. In part this is evaluated by the $\leq^{V}_{spn}$ and $\leq^{E}_{spn}$ proximity relations, but not entirely. Analysis similar to what we have described above with end-effect scenarios, but extended to include intermediate effect scenarios, can be used to achieve this. We do not include details here for brevity.

Now, we establish their semantic proximity of R_1 and R_2 w.r.t. O based on the final cumulative effect scenarios at terminating nodes. In the case of the simple annotations defined in Table 1, we can determine that the final cumulative effect of both R_1 and R_2 result in two effect scenarios such that R_1 actually remains identical to O in terms of final state approximation. R_2 on the other hand receives the additional effects of $Performs(SortOfficer, Route, Package) \wedge Knows(SortOfficer, Package, Location, Forwarding)$ on the effect scenario now generated by placing the "Route Package" activity in line with both process trajectories. Therefore, $Diff(O, R_1) = \emptyset$ and $Diff(O, R_2) = \{\{Perf \ldots\}\}$, and R_1 would be nominally chosen over R_2.

Finally, consider a more detailed analysis where, say for instance, we also evaluate non-terminating nodes using the aforementioned cardinality-oriented definition. In this situation, only the cumulative scenario in R_1 at "Handle Package" minimally differs from the scenario in R_2 at the corresponding node by $\{Performs(\ldots, Route, \ldots) \wedge Knows(\ldots, Forwarding)\}$. R_2 on the other hand differs w.r.t. a scenario at "Handle Package" by (2), at a scenario in "Update Status" by (2), and at a scenario in "Route Package" by (2). This in-turn reinforces the selection of R_1.

Definition 3. *A process model m' is R-minimal with respect to another process model m and a set of rules R iff each of the following hold:*

- *m violates R.*
- *m' satisfies R.*
- *There exists no process model m'' such that $spn'' <_{spn} spn'$ and m'' satisfies R, where spn, spn' and spn'' are SPNets corres. to m, m' and m'' respectively.*

The definition of R-minimality above provides a "semantic" yardstick for evaluating whether a process is being minimally modified to restore compliance with a set of rules. It also provides an outline of a procedure for dealing with compliance violations: generate the set of R-minimal process models and select one. The selection process could be analyst-mediated, or might involve the application of extraneously encoded preference criteria. An alternative approach is to identify the minimal sources of inconsistency with a given set of rules, thus focussing analyst attention to the portions of a model that require editing to restore compliance.

Definition 4. *Given a process model m that violates a set of rules R, a* minimal source of inconsistency *with respect to m and R is a process model m' such that each of the following hold:*

- *spn', (SPNet of m'), is a sub-graph of spn, (SPNet of m).*
- *m' violates R.*
- *m'' does not violate R for any process model m'' whose corresponding SPNet spn'' is a sub-graph of spn', the SPNet corresponding to m'.*

An obvious modification to restore compliance is to replace the terminal activities of the process models that are minimal sources of inconsistency. Note that in general, every minimal source of inconsistency must be appropriately modified to restore compliance. Useful guidance can be provided to the analyst on what the best modifications might be (given, for instance, a repertoire of possible alternative activities to replace a given activity with), using analysis that involves measuring deviations of effect annotations of activities. We ommit details for brevity. Much of our discussion above assumes only the existence of process models and their analyst-mediated effect annotations. Sometimes, goal-based annotations are also available, which describe the objectives that processes, sub-processes or individual activities are designed to achieve. While effect annotations are descriptive, goal annotations are normative. Goal annotations can impose “hard" constraints on how process models might be modified, given that modifications must still achieve the original goals of a process or its constituent elements, wherever possible. Analysis of the kind that we have discussed above could be performed to support such reasoning, but we ommit details again due to space restrictions.

4 Heuristics for Asserting and Resolving Compliance Issues

In the previous section we have provided a semantic basis for reasoning about alternative resolutions to non-compliant processes. In this section we introduce the notion of a *compliance pattern* as a heuristic basis for supporting (even in partially automated ways) the resolution of non-compliance in process moels. Informally, a compliance pattern captures a commonly occurring mode of compliance violation, including both the compliance requirement that is violated and the actions required to restore compliance. In the following, we summarize the main types of process compliance patterns. Further details have been omitted due to space restrictions.

Structural Patterns

- **Activity/Event/Decision Inclusion.** Activity, event and decision inclusion may be defined against deontic modalities (permitted, mandatory, prohibited) and/or based on path quantifiers (as in CTL). For example: *The action of receiving package details must always occur during the screening of a package*. **Resolution:** Add or remove an activity. This may require co-ordination and assignment change to occur in structured processes.

- **Activity/Event/Decision Coordination.** Activity coordination may be serial, conditional, parallel, and/or repetitive. In the case of branching constructs, CTL [**?**] assertions provide a natural means to refer to the required temporal relations between activities. Interval algebra [13] is also applicable. For example: *If a package is cleared, then it must have been screened some time in the past*; *If a package is held then it should not be delivered until it is cleared, along all possible paths globally*. **Resolution:** Add or remove an activity. Re-order existing activities.
- **Activity/Event/Decision Assignment.** The assignment of an activity to a role is defined using deontic operators. A statement making an action mandatory for some role may or may not preclude its assignment to other roles, and vice-versa. For example: *Clearing a package must only be assigned to a Regulatory Authority role*; *The customer must provide the details of the package to a Courier role (however the courier is still able to provide package details to another role)*. **Resolution:** Add or remove an activity. Re-assign an activity.
- **Actor/Resource Inclusion.** The participation of an actor or availability of a resource within a process may also be defined using deontic operators. An actor's existence in a process model (e.g. using BPMN lanes) will also indicate their participation. For example: *A regulatory authority must be included in the process for screening a package*. **Resolution:** Add or remove a participant/resource. This may require the addition and removal of activities and/or interactions.
- **Actor/Resource Interaction.** The interaction between actors and/or the transfer of resources may be governed by security, privacy or other concerns. For example: *A Customer must never interact with a Regulatory Authority during the screening of a package*. **Resolution:** Add or remove a participant/resource. Add or remove an interaction/transfer.

Semantic Patterns: To resolve the following compliance issues, almost any (or combination of) changes may be required - e.g. Add or remove an action. Add or remove an effect. Re-assign an action. Add or remove an actor. Add or remove an interaction.

- **Effect Inclusion.** An effect may be permitted, mandatory or prohibited to hold at a set of final or intermediate states of the process. For example: *Delivered packages must not be held*; *Delivered packages must be cleared.*
- **Effect Coordination.** The temporal relationship among the effects of a process (i.e. declared discretely in process models such as BPMN) may also be constrained. For example, *In all possible cases, a cleared package must be delivered unless it is held some time after clearance*.
- **Effect Modification.** Temporal rules may also refer to allowable changes upon intermediate effects within a process. For example: *If a package is held, then it cannot be cleared by a delivery process*.

5 Conclusion

We define a novel framework for auditing BPMN process models for compliance with legislative/regulatory requirements, and for exploring alternative modifications to restore compliance in the event that the processes are found to be non-compliant. This lays the foundations for tool support in the area, which we are in the process of implementing, but whose details we have had to omit due to space constraints. Parts of this framework have been empirically validated, but a complete industry-scale validation remains future work.

References

1. Zhang, I.X.: Economic consequences of the sarbanes-oxley act of 2002. AEI-Brookings Joint Center 5 (2005)
2. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: Proc. 10th Int. Enterprise Dist. Object Computing Conf. (2006)
3. Weitl, F., Freitag, B.: Checking semantic integrity constraints on integrated web documents. In: Workshop Proc. of ER., pp. 198–209 (2004)
4. Janssen, W., Mateescu, R., Mauw, S., Springintveld, J.: Verifying business processes using spin. In: Holzman, G., Serhrouchni, E.N. (eds.) Proceedings of the 4th International SPIN Workshop, Paris, France, pp. 21–36 (1998)
5. Rosenberg, F., Dustdar, S.: Business rule integration in bpel - a service-oriented approach. In: Proc. of the 7th Int. IEEE Conf. on E-Commerce Technology, IEEE Computer Society Press, Los Alamitos (2005)
6. Ly, L.T., Rinderle, S., Dadam, P.: Semantic correctness in adaptive process management systems. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, Springer, Heidelberg (2006)
7. Reijers, H.A.: Design and Control of Workflow Processes: Business Process Management for the Service Industry. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, Springer, Heidelberg (2003)
8. White, S.: Business process modeling notation (bpmn), Technical report, OMG Final Adopted Specification 1.0 (2006), http://www.bpmn.org
9. Becker, J., Indulska, M., Rosemann, M., Green, P.: Do process modelling techniques get better? In: Proc. 16th Australasian Conf. on I.S. (2005)
10. Wagner, G.: How to design a general rule markup language. In: Proc. of the Workshop XML Technologies for the Semantic Web (2002)
11. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Proc. of the Int. Joint Conference on R.E., Toronto, pp. 249–263. IEEE Press, Los Alamitos (2001)
12. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: Proc. of the Fourth Asia-Pacific Conf. on Conceptual Modelling (2007)
13. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, Cambridge (2004)
14. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)