

IUPC: Identification and Unification of Process Constraints

Juergen Mangler and Stefanie Rinderle-Ma

University of Vienna, Austria
Faculty of Computer Science
Workflow Systems and Technology Group
{stefanie.rinderle-ma, juergen.mangler}@univie.ac.at

Abstract. Business Process Compliance (BPC) has gained significant momentum in research and practice during the last years. Although many approaches address BPC, they mostly assume the existence of some kind of unified base of process constraints and focus on their verification over the business processes. However, it remains unclear how such an integrated process constraint base can be built up, even though this constitutes the essential prerequisite for all further compliance checks. In addition, the heterogeneity of process constraints has been neglected so far. Without identification and separation of process constraints from domain rules as well as unification of process constraints, the successful IT support of BPC will not be possible. In this technical report we introduce a unified representation framework that enables the identification of process constraints from domain rules and their later unification within a process constraint base. Separating process constraints from domain rules can lead to significant reduction of compliance checking effort. Unification enables consistency checks and optimizations as well as maintenance and evolution of the constraint base on the other side.

1 Introduction

Business Process Compliance (BPC) has gained significant momentum in research and practice during the last years. BPC requires that business processes comply with certain relevant rules, regulations, or norms. The rules can be derived from internal quality directives such as Six Sigma or ITIL. Examples for external rules comprise regulations by standards (e.g., ISO 001), regulations by authorizing bodies, or regulations based on contracts (business contracts) [14].

Although many approaches address BPC [12,9,14,10,3], they mostly assume the existence of some kind of unified base of process constraints and focus on the verification of these constraints over the business processes. By *process constraints* we refer to those rules in the domain of interest, that are associated with processes. As a general remark: the co-existence of processes and process constraints is common and desired (“Separate [Rules] From Processes, Not Contained In Them”, cf. [4]). However, it remains unclear how such an integrated process constraint base can be built up, even though this constitutes the essential prerequisite for all further compliance checks.

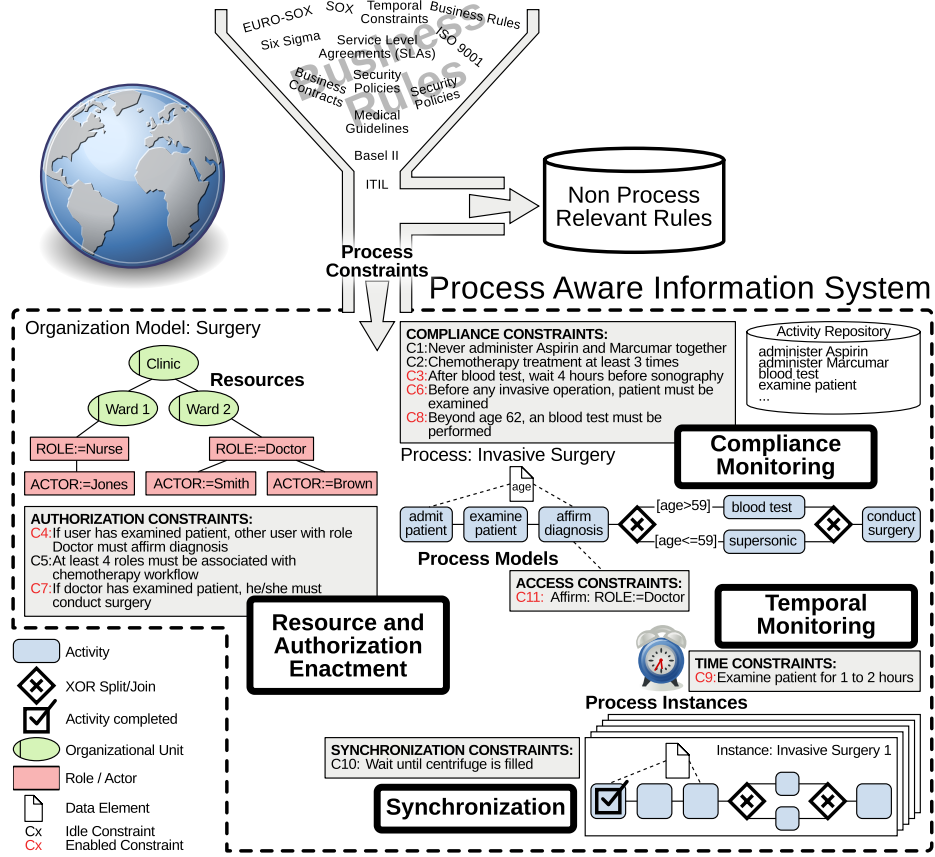


Fig. 1. Current Situation: “Zoo” of Process-Relevant Constraints

In addition, the heterogeneity of process constraints has been mostly neglected so far. For a specific application scenario consider Fig. 1 that displays example processes and rules from the medical domain: here, business processes might be subject to medical guidelines or clinical pathways on the one side [9] and also subject to authorization and privacy constraints on the other side. Fig. 1 also covers simple resource synchronization as well as temporal rules. Assuming that all of these rules are process constraints, how can they be integrated within a unified representation? In summary, building up a unified basis of process constraints out of all rules existing in a certain business context, basically poses the following requirements:

- REQUIREMENT 1: Process constraints should be identified and separated from the overall set of business rules.
- REQUIREMENT 2: Process constraints should be unified based on a common representation in order to facilitate consistency checking and optimization.

Requirement 1 is crucial in order to reduce BPC verification effort to those constraints that are associated with processes. This is particularly important since compliance verification for processes, which is mostly based on some kind of model checking technique, tends to be complex [10]. The second requirement demands for a unified representation that allows to fully match the heterogeneity of vastly different constraint types. It can then be used for filtering and optimization of process constraints. Furthermore, it also contributes to evolution and maintenance of the constraint base which is particularly important, since constraints are often subject to change and evolution [11].

In this technical report, we present a definition for process constraints that enables their distinction from business rules that are not associated with any process. This definition bases on results from general business rules frameworks as well as analysis of existing BPC approaches and case studies. Further, a unified representation for process constraints is provided. It covers process perspectives such as control flow, data flow, time, and resources which can be subject to process constraints. The unified representation also enables the specification of behavior which is relevant for process constraints that require some sort of action (e.g., synchronization between process instances).

In Sect. 2 we derive means to identify process constraints from domain rules. The IUPC unification representation is presented in Sect. 3. Sect. 4 discusses the IUPC representation along different constraint properties in PAIS followed by an evaluation in Sect. 5. Sect. 6 compares related approaches and Sect. 7 concludes with summary and outlook.

2 Identification of Process Constraints

This section addresses Requirement 1 as stated in the introduction: from all business rules relevant for “business practice and guidance” (cf. GUIDE [4]), those constraints that are relevant in the context of process execution should be filtered out. Reason is that the effort for checking constraints imposed over processes can be reduced to those which are actually associated with the process (structure). Note that verifying constraints over processes is often complex although different approaches offer optimization techniques [10].

In order to be able to identify and separate process constraints from business rules, a definition of process constraints is required. In turn, a definition requires to know the specifics of process constraints in contrast to business rules. Intuitively, process constraints should be somehow associated with a process. In literature on business rules, association of business rules and real-world objects is expressed by so called anchor objects [6]. In the example rule “A car must have a registration number”, real-world object “car” constitutes the anchor object [6]. More precisely, within the generally accepted “If ... Then” rule structure, the anchor object would be contained within the if part, triggering execution of the rule. The question is now which anchor objects are suitable to distinguish a process constraint from a general business rule.

As process literature study shows, all formalisms on process constraints (implicitly) contain a process-specific anchor object, i.e., a structural pattern contained within the process of interest which the process constraint refers to. A structural pattern contains at least one activity either executed within a process stored within the process activity repository. An example for the first case is constraint C3 depicted in Fig. 1, referring to process activity `blood test` executed within the treatment process. We denote such process constraints as *enabled*. By contrast process constraint C1 for example does not refer to any process activity currently executed within the treatment process. In turn, it refers to process activities `administer Aspirin` and `administer Marcumar`, stored within the associated process activity repository. Hence, any time one of these two activities will be added to some process, C1 will become enabled and thus is to be considered as process constraint. Until then, we denote C1 as *idle*.

Specifically, we use the general term structural pattern, since it might refer to a set of process activities as well as to control flow patterns [2], e.g., a sequence or a parallel branching.

Definition 1 (Process Constraint). *Let \mathcal{P} be a set of all process types of consideration. Let further $\mathcal{A} := \mathcal{A}_{\mathcal{P}} \dot{\cup} \mathcal{A}_{\mathcal{R}}$ be the set of all process activities within the domain where $\mathcal{A}_{\mathcal{P}}$ denotes the set of process activities executed within a process $P \in \mathcal{P}$ and $\mathcal{A}_{\mathcal{R}}$ denotes the set of process activities stored within the associated process repository.¹ Finally, let \mathcal{R} be the set of all domain rules. Then we denote a rule $r \in \mathcal{R}$ as process constraint if it contains a structural pattern SP_c as anchor object with SP_c is structural pattern over $\mathcal{A}_{\mathcal{P}}$ or $SP_c \subseteq \mathcal{A}_{\mathcal{R}}$.*

In the SeaFlows project [10] process constraints are tied to one or more process activities and consist of an *antecedent* and a *consequence* part. More precisely, based on a triggering antecedent pattern, a consequence pattern must hold to fulfill the constraint. This directly corresponds to the general notion of anchor object, since the structural pattern contained within the antecedent pattern triggers the constraint (if part). Process constraints as defined in DECLARE also refer to process activities and the semantics for relations between activities are based on LTL (Linear Temporal Logic) [13]. BPMN-Q offers compliance patterns containing structural patterns within the triggering part of the constraint [3]. Consequently, we can state that a business rule is a process constraint if its anchor object contains a structural pattern within a process.

3 Process Constraint Unification

As we can now identify process constraints, the second challenge is to unify the partly strongly varying process constraints. Hence, in this section we present the IUPC representation for unifying and structuring process constraints in to subsequently support process constraint optimization, maintenance, and evolution.

¹ For definition purposes we assume disjoint sets here. However, the definition can be easily adapted when used and idle process activities are contained in both sets.

3.1 Expressing Process Context by Linkage

The following definition of *Linkage* integrates the context Context_c of a process constraint c together with its structural pattern SP_c and a trigger position TP_c (cf. Fig. 2).

Definition 2 (Linkage). *Let \mathcal{P} be a set of all process types of consideration and \mathcal{I}_P be the set of process instances running according to a process type $P \in \mathcal{P}$. A process type $P \in \mathcal{P}$ is described by a process schema $S_P := (A_P, E_P, D_P)^2$ where A_P denotes the set of activities, E_P the set of control/data edges, and D_P the set of data elements S_P consists of. Then the linkage Linkage_c to a process type P is defined as follows:*

$$\begin{array}{ll} \text{Linkage}_c \subseteq \text{Context}_c \times \text{SP}_c \times \text{TP}_c & \text{where} \\ \text{Context}_c \subseteq \mathcal{P} \times \mathcal{I}_P & \wedge \\ \exists \text{SP}_c & \wedge \\ \text{TP}_c \in \emptyset, \text{before}(a_{n,P}), \text{after}(a_{n,P}) & \end{array}$$

In Def. 2, Context_c describes in which processes or process instances a constraint may occur. Possibilities include single instances (e.g. SLAs for services that have been dynamically selected), all instances of a process (attribution of resources to tasks) or even instances in multiple processes (when a resource is used in multiple processes, and synchronization has to take place). Structural patterns SP_c express the association between process constraint and process. In connection with *context*, it defines for which parts of process types and process instances the corresponding process constraint is to be enforced or verified. Structural patterns may not only spawn single activities but also several activities connected through complex control decisions. Finally, the trigger position TP_c is relevant for synchronization constraints, since synchronization constraints are constraints that not only set out certain conditions on process execution, but enforce an action. In this case the trigger position specifies whether the action should take place before the affected activity is started or after. The Trigger Position is solely present for run-time (behavioral) constraints. As a structural pattern may not only spawn a single activity but also several activities connected through complex control decisions, it is necessary to determine when exactly a constraint has to fire. This is the equivalent of describing the condition under which an event occurs, in an Event Condition Action (ECA) rule. The trigger position itself is simple: before or after an activity occurs. Of course multiple before / after positions can be specified.

Both, SP_c and TP_c are grouped as *Connection* in Fig. 2 to express their tight integration. A TP_c can not exist without a set of activities matched by a structural pattern SP_c . Apart from TP_c , linkage information can be statically matched against processes, thus it is possible to determine *enabled* and *idle* constraints (as described before).

² In order to stay meta-model independent, we assume a simple generic representation for process schemas that can be adopted by any (imperative) process meta model.

From now on we will denote a linkage Linkage_c in the compact form:

$$\text{Linkage}_c : ((\mathcal{P}, \mathcal{I}_P), \text{SP}_c, \text{TP}_c)$$

Consider compliance constraint C6 depicted in Fig. 1: “C6: Before any invasive operation, patient must be examined”.

The antecedent pattern “existence of activity invasive operation” within a process triggers the check whether this activity is preceded by an activity “patient examination”. The linkage for this compliance constraints turns out as

$$\text{Linkage}_{C6} : ((\text{Invasive Surgery}, \text{ALL}), \text{SP}_{C6}, \emptyset)$$

with

$$\begin{aligned} \text{SP}_{C6} : \exists a_1 \text{ Is}(a_1, \text{examine patient}) & \quad \wedge \\ \exists a_2 \text{ Is}(a_2, \text{conduct surgery}) & \quad \wedge \\ a_1 \mathcal{A}^* a_2 & \end{aligned}$$

where

$$\mathcal{A}^* : \text{arbitrary activities between } a_1 \text{ and } a_2$$

3.2 Integrating Data, Time, and Resources

Existing approaches mostly deal with control flow constraints, i.e., constraints that are only referring to structural patterns within a process [3]. The only approaches that have addressed data flow aspects within process constraints are SeaFlows [8] and BPMN-Q [3]. In accordance to these approaches, the data flow perspective within a process constraint can be represented as condition on the structural pattern it refers to. Consider constraint C8 from Fig. 1: “C8: Beyond age 62, a blood test must be performed”.

We can see that the data flow condition “Beyond age 62” imposes a restriction on the structural pattern of the constraint (“blood test”). However, control and data flow are only two perspectives of a process. As case studies show, process constraints might also refer to conditions on time and resources, e.g., constraints C3, C4, C5, C7, C9, and C11 within the example depicted in Fig. 1. Hence, constraint unification requires the specification of data, time, and resource conditions on top of the linkage part of the process constraint.

Definition 3 (Condition). Let c be a process constraint with linkage Linkage_c referring to a process type P described by process schema on linkage $S_P := (A_P, E_P, D_P)$. The condition of c imposed on Linkage_c is defined as

$$\text{Condition}_c \subseteq \text{expr}(D_P) \times \text{expr}(\text{time}_c) \times \text{expr}(\text{resource}_c)$$

where

- $\text{expr}(D_P)$ denotes a logical expression over the data elements of P
- $\text{expr}(\text{time}_c)$ denotes a temporal expression and

- $\text{expr}(\text{resource}_c)$ denotes a logical expression over the resources associated with P (typically modeled within a organizational or resource model)

It is important to mention that a condition cannot exist without referring to a linkage. Thus, a condition describes, under the premise of a given linkage, either (a) a combination of data, temporal or resource conditions that have to be present or (b) a combination of data, temporal or resource conditions that have to be present in order for a given behavior to be apply.

For constraint C6, for example, no specific condition is imposed on the linkage part. Hence:

$$\text{Condition}_{C6} : \emptyset$$

When considering constraint C3 in Fig. 1, things get more interesting, since “After a blood test wait 4 hours before sonography” obviously imposes a time condition on the linkage part. Less obviously, an additional condition is imposed on the data flow, since the 4 hours time frame between blood test and sonography are only required for the same patient, formally:

$$\text{Linkage}_{C3} : (\text{Invasive Surgery}, \text{ALL}), \text{SP}_{C3}, \emptyset)$$

with

$$\begin{aligned} \text{SP}_{C3} : \exists a_1 \text{ Is}(a_1, \text{blood test}) & \quad \wedge \\ \exists a_2 \text{ Is}(a_2, \text{sonography}) & \quad \wedge \\ a_1 \mathcal{A}^* a_2 & \end{aligned}$$

and

$$\begin{aligned} \text{Condition}_{C3} : \text{patient}(a_1) = \text{patient}(a_2) & \quad \wedge \\ \text{min_time_between}(a_1, a_2, 4h) & \end{aligned}$$

Similar considerations can be made for data (“C8: beyond age 62”) and resources (C7: examination by same user as surgery) which can be also represented by a condition part imposed on the linkage of a constraint.

3.3 Expressing Behavior within Process Constraints

The last missing piece is, given a certain linkage and condition, to allow for a certain assignment or behavior. Assignment becomes necessary for access constraints such as C11. Behavior specifies for example the action part of a synchronization constraint such as C10. Hence, we complete the unified constraint representation by a *Behavior* part.

Definition 4 (Behavior). *Let c again be a process constraint. For a given Linkage_c and Condition_c , a behavior can be either empty, the attribution of resource or time information or instructions specific for process execution (e.g. exceptions).*

3.4 Summary: Linking, Condition, Behavior

The overall representation for process constraints consisting of linkage, condition, and behavior is summarized in Fig. 2. In Sect. 5 we will evaluate the unified representation against constraints as shown Fig 1.

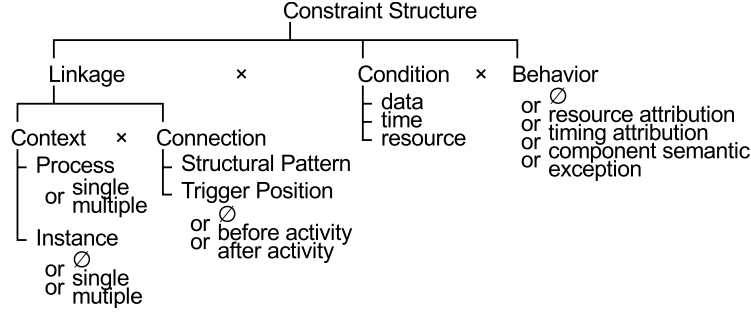


Fig. 2. Constraint Structure

4 Discussing Unified Representation along Constraint Properties in Process-Aware Information Systems

In the previous sections, a unified representation for process constraints has been presented. In this section, we discuss its usage within Process-Aware Information Systems (PAIS) along different constraint properties as set out in Fig. 3.

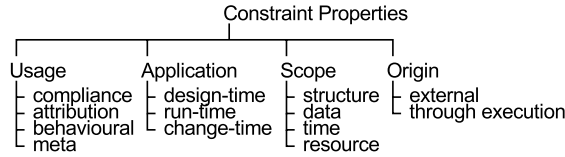


Fig. 3. Process Constraint Properties

Usage: Process constraints can be used to verify the compliance of business processes with relevant regulations and constraints. This is useful for process design and evolution. Constraints can also be used to alter or affect the behavior of processes (*behavioral*). Affecting the behavior of processes includes the attribution of process activities or, more generic, the attribution of structural patterns. Examples include resources allocation of process activities.

Additionally, constraints can be used to specify *meta* constraints. Meta constraints are intended to check the consistency of other constraints, for example, constraint C5 in Fig. 1. Another example is a meta constraint specifying that

for each process activity referring to a resource `centrifuge` synchronization constraint C10 in Fig. 1 is assigned to.

Application: Application deals with the question at which phase of a process life cycle constraints may be enforced. Basically, at *design-time* constraints are checked to verify that a process complies to certain criteria (compliance) constraints. All constraints that are checked at design-time are *compliance* constraints or *meta* constraints.

All *behavioral* constraints, and some *compliance* constraints are checked at *run-time*. Examples include the checking for data value constraints, or the attribution of structural patterns (the attributes are used at run-time, though can be checked by meta constraints at design-time). Compliance constraints may affect run-time under certain circumstances. E.g. although a process may not conform to a specific constraint at design-time, it may do so at run-time, because only a certain execution path violates the constraint. Hence corresponding process instances are to be monitored at run-time [10].

Scope: The four perspectives of the constraint scope have been discussed in Sect. 2, capturing which kind of information constraints a PAIS can contain. The most important scope perspective is *structure* as in all constraints structural information has to be present (either explicit or implicit through connections of information to structure), as otherwise they would not be connected to processes and could thus not be enforced in PAIS. Note that if a constraint holds on only structural information, it is always a *compliance* constraint.

Data is also an integral part of a process, that is tightly connected to structure. At design-time, it is possible to check if data types and data flow conform to a certain schema. Further it can be checked whether certain data values will lead to compliance violations at run-time. If, for example, a treatment process states that a lab test is to be performed for all patients beyond 65 years and the corresponding medical guideline states that the lab test is mandatory for patients beyond 62 years, it can be already checked at design-time, that for patients between 62 and 64 there will be a violation of the corresponding constraint at run-time.

Resource and *time* are attributes that are typically connected to structure (or data). Their purpose is to describe information that aids the execution of a process. *Resource-aware* and *time-aware* constraints in PAIS are typically checked or enforced at run-time including:

- Resource assignment to process activities, typically specified based on access constraints (e.g., constraint C11 in Fig. 1). Based on role assignment, process activities are offered to authorized actors in their work lists at run-time (e.g. by a RBAC component).
- On top of access constraints, authorization constraints can be specified such as dynamic separation of duties. Dynamic authorization constraints are verified during run-time.
- Assign temporal information to activities (e.g. the normal duration of a certain activity is 2 hours with a standard deviation of 10 minutes).

Origin: When do constraints become available for application to a process? All constraints become available through not specified **external** resources, either at design time, run-time or change time. They are identified and structured by constraint designers and then made available through a constraint-base, and have to be enacted from this point on. One exception is constituted by SLAs for dynamic service selection. These constraints become available *through execution* of a process instance, and vanish after the instance finishes.

5 Evaluation

The feasibility of the unified representation is evaluated by means of the overall 16 process constraints depicted in Fig. 1 and along the constraint properties described in Section 4. For this we classify the 16 process constraint into different constraint types that embody a combination of properties as represented by the unified representation.

Constraint Structure (Fig. 2)					Constraint Properties (Fig. 3)												Constraint Type	Example	
Linkage		Context		Condition	Behaviour	Application				Scope			Origin		Usage				
Process Instance	Struct. Pat.	Trigger Pos.	Connection			design-time	run-time	change-time	structure	data	time	resource	external	through exec.	compliance	attribution	behaviour		meta
X	X	X	○	~	e.g. ROLE:=Doctor	X	X	-	-	X	X	-	X	X				Resource Attribution	C11
X	X	X	○	~	e.g. DURATION:=12 minutes	X	X	-	X	X	X	-	X	X				Timing Information Attribution	
X	X	X	○	X	○	X	-	X	-	-	-	-	X	X				Business Compliance Constraints	
X	X	X	○	X	○	X	X	-	X	X	X	-	-	X	X			Separation/Binding of Duty	
X	X	X	○	X	○	X	X	X	X	-	X	-	X	X		-		Data Constraints	C4,R4
X	X	X	○	X	~ exception, e.g. send warning	X	X	X	-	X	-	X	X	X		-		Temporal Constraints	C1*,C8,R1
X	X	X	○	○	○	X	-	X	-	-	-	-	X	X				Structural Constraints	C3,C9,R2
X	X	X	○	○	○	X	X	X	-	-	-	-	X	X		-		Structural Constraints (run-time)	C1*,C6,C7
X	X	X	-	~	~ exception, e.g. send warning	-	X	-	X	-	-	-	X	-	-	-		Security	C2,R3
X	X	X	X	X	comp. semantic, e.g. delay spec.	X	X	-	-	-	X	-	-	X		X		Synchronisation	C10
X	X	X	X	X	comp. semantic, e.g. algorithm	X	X	-	-	-	X	-	-	X		X		Active	
X	X	X	-	~	~	-	X	-	X	-	-	-	X	X		-		SLA (dyn. service sel.)	
X	X	X	-	~	~	-	X	-	X	-	-	-	X	X		-		SLA (service sel. at design-time)	R5
X	X	X	○	X*	component semantic	X	X	X	-	-	-	-	X	-		X		Meta (check other constraints)	C5

X ... necessary

X* ... check if e.g. behavior parts are correct

○ ... not allowed / empty

- ... optional

X ... default

- ... possible

... not possible

Table 1. Constraint Types

As depicted in Tab. 1 we identified 14 different process constraint types. The first two constraint types deal with *Resource Attribution* (e.g., roles, actors, nodes) and *Timing Information Attribution* (e.g., minimal, maximal, average duration) to process structure. All this information is necessary at run-time, for a temporal monitor as a basis to verify if a process behaves correctly, or an RBAC system to select actors in a worklist. The only difference between these two is, that during change-time timing information may have to be adapted to account for the duration of changes. E.g. constraint C11 in Fig. 1 can be represented

as follows:

$$\begin{aligned}
&\text{Linkage}_{C11} : ((\text{Invasive Surgery}, \text{ALL}), \text{SP}_{C11}, \emptyset) \quad \text{with} \\
&\quad \text{SP}_{C11} : \exists a_1 \text{ Is}(a_1, \text{affirm diagnosis}) \\
&\text{Condition}_{C11} : \emptyset \\
&\text{Behavior}_{C11} : \{ \text{ROLE} := \text{Doctor} \}
\end{aligned}$$

For attribution TP_c is always empty, as the matched structural pattern implies, that the attribution is available all the time.

The third constraint type describes *Business Compliance Constraints* in general. We decided to include this very generic constraint type (which should not be confused with several of the other constraint types) to show some specific characteristics we derived by studying constraint sources as mentioned in Section 2. Business compliance constraints, are exclusively compliance constraints, they never carry a behavioral part, they most of the time verify a certain structure of the process, with a possibility of checking also data, resource and temporal aspects. Therefore the design-time *Structural constraints* also in this list are *Business Compliance Constraints*, as well as some *Temporal Constraints* or *Data constraints* (whenever only design-time aspects are covered). *Separation / Binding of Duty*, *Resource Attribution*, and *Timing Information Attribution* are the exception to the rule. They may be very well applied at design time, just only the impact can only be seen at run-time.

6 Related Work

Many approaches for checking BPC either at design or run time exist, e.g., [9,14,5]). As argued before, identification and unification of process constraints has been outside the scope of these approaches so far. Validation of the IUPC representation compared to selected existing approaches is presented in Table 2.

Representation	SeaFlows [10]	DECLARE [13]	BPMN-Q [3]	IUPC
Linkage	activity set	control flow pat.	control flow pat. global scope	control flow pat. process context trigger position
Condition	data	–	data	data, time, resource
Behavior	–	–	–	✓

Table 2. Comparison of Existing Approaches

The above mentioned approaches try to ensure BPC either a-priori at design time or by detecting inconsistencies at run-time. In addition, there are also a-posteriori approaches that offer techniques to analyze process logs (i.e., data of already executed and finished processes) with respect to certain properties such as adhering to compliance constraints [1]. In these approaches, different aspects logged during process execution can be checked, e.g., separation of duties. Doing so the a-posteriori approach reflects the need for unified compliance checking, not

only a-posteriori, but also a-priori. Other approaches use constraints to synchronize between business processes of different type (e.g., between a chemotherapy and a radiation process) [7]. Based on our unification approach, synchronization constraints can be unified and managed in combination with the other constraints in PAIS.

7 Summary and Outlook

This technical introduced the IUPC representation for identification and unification of process constraints. By separating process constraints from domain rules, effort for compliance verification can be significantly reduced. Offering a unified representation enables consistency checks and optimizations of the constraint base. Furthermore, unification and structuring of constraints in PAIS enables the development of an integrated checking component instead of isolated and distributed checking components as present nowadays.

In upcoming publications we provide (on the basis of this classification) extensive evaluation based on case studies from different domains, e.g., health care and financial sector. We furthermore work towards an implementation of the unified and structured IUPC representation on top of our adaptive cloud process execution engine CPEE [15]. Particular focus will be put on maintenance of the constraint base and an integrated verification component for the engine.

References

1. van der Aalst, W., de Beer, H., van Dongen, B.: Process mining and verification of properties: An approach based on temporal logic. In: Meersman, R., et al Tari, Z. (eds.) *Int'l OnTheMove Conferences*. LNCS, vol. 3761, p. 130–147 (2005)
2. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. *Distributed and Parallel Databases* 14(1), 5–51 (2003)
3. Awad, A., Weidlich, M., Weske, M.: Visually specifying compliance rules and explaining their violations for business processes. *Journal of Visual Languages & Computing* 22(1), 30–55 (Feb 2011)
4. Business Rules Group: The business rules manifesto, <http://www.businessrulesgroup.org/brmanifesto.htm>
5. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: *Int'l Enterprise Distributed Object Computing Conference*. p. 221–232 (2006)
6. GUIDE Business Rules Project: Defining business rules ~ what are they really? Tech. Rep. Final Report 1.3 (2000)
7. Heinlein, C.: Workflow and process synchronization with interaction expressions and graphs. In: *Proc. Int'l Conf. on Data Engineering*. p. 243–252 (2001)
8. Knuplesch, D., Ly, L., Rinderle-Ma, S., Pfeifer, H., Dadam, P.: On enabling Data-Aware compliance checking of business process models. In: *Conceptual Modeling – ER 2010*. vol. 6412, pp. 332–346 (2010)
9. Lu, R., Sadiq, S., Governatori, G.: Compliance aware process design. In: *Proc. Int'l Business Process Management Workshops '07* (2007)

10. Ly, L., Rinderle-Ma, S., Dadam, P.: Design and verification of instantiable compliance rule graphs in Process-Aware information systems. In: Proc. Int'l Conf. on Advanced Systems Engineering. pp. 9–23 (2010)
11. Ly, T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems - requirements, challenges, solutions. Information Systems Frontiers (Special Issue on Governance, Risk and Compliance) (2009)
12. Namiri, K., Stojanovic, N.: Pattern-Based design and validation of business process compliance. In: Int' OTM Conferences 2007, Part I. LNCS, vol. 4803, p. 59–76. Springer (2007)
13. Pesic, M., Schonenberg, H., Sidorova, N., van der Aalst, W.: Constraint-Based workflow models: Change made easy. In: OTM Conferences (1). p. 77–94 (2007)
14. Sadiq, S., Governatori, G., Naimiri, K.: Modeling control objectives for business process compliance. In: Business Process Management (2007)
15. Sturmer, G., Mangler, J., Schikuta, E.: Building a modular service oriented workflow engine. In: Service-Oriented Computing and Applications (SOCA), 2009 IEEE International Conference on. p. 1–4 (2009)