Defining Instance Spanning Constraint Patterns for Business Processes Based on Proclets

Karolin Winter, Stefanie Rinderle-Ma

Faculty of Computer Science, University of Vienna, Vienna, Austria
{karolin.winter|stefanie.rinderle-ma}@univie.ac.at

Abstract. Instance Spanning Constraints (ISC) establish controls across multiple instances of one or several business process types. Consider, e.g., medical treatments during which drug-drug interactions might occur. Different treatments are likely to be modeled in separate processes, but yet have to be coordinated in order to avoid harm for patients. ISC typically stem from regulatory documents and must be integrated into business processes. In order to facilitate ISC integration, we provide six ISC patterns which are based on a real-world ISC collection as well as a categorization of ISC. The presented ISC patterns are formalized using Proclets based on timed colored workflow nets. This formalization choice results from an elaborated requirements analysis and enables the synchronization of instances of one or several process types while employing well-known process modeling approaches. The ISC patterns are evaluated through their application to i) selected business processes and ii) existing approaches for batching and security in business processes.

Keywords: Patterns and Reuse, Business Process Modeling, Business Process Compliance, Instance Spanning Constraints

1 Introduction

Today's highly flexible and interconnected business environments require the coordination of their business processes based on so called Instance Spanning Constraints (ISC) which span multiple instances of one or several process types. Figure 1 depicts a process model describing process type *laboratory process*. During runtime, for each lab sample a corresponding process instance is created and executed based on the process model. Assume that resource *centrifuge* employed is limited. An ISC *Wait until centrifuge is filled* can be employed to coordinate the efficient usage of the *centrifuge*. More precisely, if *centrifuge* offers n slots, the ISC realizes a synchronization of n process instances in order to execute the task `centrifugation` simultaneously.

ISC are present throughout many applications (e.g., security, batching, queueing) and domains (e.g., manufacturing, medicine). Various aspects connected with ISC, including modeling, enactment, and mining of ISC have been addressed by literature, e.g., [3,7,9,10,12,19]. However, a "common ground" for

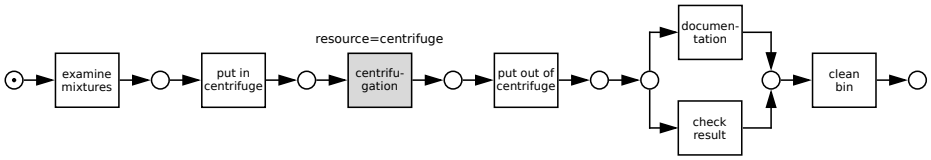

Fig. 1. Example of a laboratory process modeled as Petri net adapted from [18]

the different approaches, applications, and domains is still missing, though this would foster the understanding, transparency, reuse, and sharing of ISC.

In business process management, patterns have already proven useful for creating "common ground", including workflow patterns [1], change patterns [16], compliance monitoring functionalities [6], and compliance patterns [15]. In this spirit, we think that a set of ISC patterns will be useful and support the usage, transparency, and understandability of ISC across various approaches, applications, and domains. Hence, this work raises the following research questions:

1. Which ISC patterns are useful for establishing controls that span multiple process instances of one or several process types?
2. How to formalize ISC patterns?
3. How to realize ISC patterns?

When addressing RQ1, RQ2, and RQ3, the paper follows the methodology depicted in Fig. 2. The elicitation of ISC patterns is based on the ISC categorization elaborated in [19] which rests on the ISC collection presented in [3]. The ISC categorization [19] comprises *Category I: simultaneous execution of activities*, *Category II: constrained activity execution*, *Category III: order of activities*, and *Category IV: non-concurrent execution of activities* accounting for around 85.9% of the examples from the 114 ISC examples in [3]. The remaining 14.1% of the examples refer to ISC handling exceptions in process executions. This work will elaborate on Categories I – IV for ISC pattern formalization. In particular, the ISC patterns distinguish between ISC patterns for multiple instances of one process type and ISC patterns for multiple instances of multiple process types.

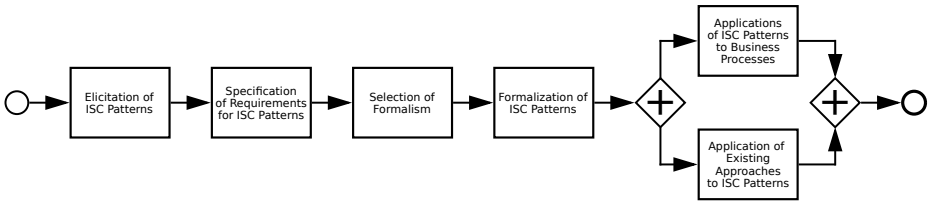

Fig. 2. Methodology adapted from [6]

The ISC pattern formalization poses several challenges, for example, the representation of an instance-spanning attribute such as the *centrifuge* in Fig. 1. Especially interesting is that ISC introduce virtual decision points between process instances and/or process types as shown in [18]. How can they be formalized,

but without allowing any undesired exchange of data between process instances or types? This is a differentiation to approaches for modeling interorganizational processes that are established based on the desired exchange of messages. Hence, Sect. 2 states requirements on ISC pattern formalization and discusses timed colored Workflow Nets and Proclets as formalization of choice. The formalized ISC patterns are presented in Sect. 3. The evaluation focuses on ISC realization based on the application to two business processes as well as to existing approaches followed by a discussion (cf. Sect. 4). The application to existing approaches also serves as related work discourse. The paper concludes in Sect. 5.

2 Formalism Choice and Fundamentals

Overarchingly, the selection of a formalism for defining ISC patterns shall rest on well established concepts in business process management. This facilitates the understanding and sharing of ISC patterns for different business process scenarios. In literature, there is a debate on how to treat compliance constraints and business processes, i.e., keeping them separated as mostly the case for imperative approaches or "mixing" them as mostly the case for declarative approaches. With choosing an imperative formalism the definition of the ISC patterns becomes close to the business process definition (keeping the mental map) and ISC patterns can be directly used within the processes. This way it becomes transparent how the ISC are executed during runtime. Therefore, we opt for an imperative approach. Further on, the formalism should have a well-defined formal execution semantics in order to enable the smooth transformation into executable process code. Finally, we aim at a formalism that enables the formal analysis of ISC and related process types/instances.

Besides these general considerations, we postulate four requirements that result from an analysis of the ISC categories from [19] based on ISC examples from [3]. Let in the following, $\text{ISC}_S \subseteq$ ISC denote ISC that span one process type and $\text{ISC}_P \subseteq$ ISC denote ISC that span multiple process types.

Category I. ISC of this category refer to the simultaneous execution of events, e.g., that *Task* `centrifugation` *must be executed for five instances simultaneously.* An example of an ISC_P could be that *When the centrifugation is started a protocol must be created simultaneously.* In this case the two tasks, `centrifugation` and `write protocol` would be present in two separate processes but must be coordinated and started simultaneously. Simultaneous execution of tasks is crucial for an ISC pattern formalism.

Category II. ISC of this category refer to the constrained execution of events which can either refer to time or data constraints but also the absolute number of executions of a task or process. An example for the first case is *Loans may only be approved as long as the amount is below $1M per day.* This ISC depends on the data element *amount* and the timespan *day* for a task `approve loan`. An example for the latter is *Two tasks* `B` *and* `B'` *may only executed in sum 100 times a day.* Not only the absolute number but also the timespan *day* is of importance. If `B` and `B'` are located in different processes, this is an ISC_P.

Category III. ISC of this category refer to the order of event executions and only appear in the form of ISC_P since for ISC_S this would correspond to intra instance constraints. Consider for example *Before the centrifugation can be started, the blood sample must have been taken.* The corresponding tasks `centrifugation` and `take blood sample`, though located in separate processes, must be executed in a specific order.

Category IV. ISC of this category are also of type ISC_P and describe the non-concurrent execution of events, e.g., *Tasks* `take blood sample` *and* `administer inoculation` *may not be executed concurrently for one patient.*, whereas the two tasks are present in different processes.

In order to cover Categories I – IV for multiple instances of one, but also multiple process types, a formalism for defining ISC patterns should

- support instance correlation through an instance unique identifier (uid) in case of multiple process types ($\mapsto$ *Rq1*).
- represent attributes shared by multiple instances/processes (e.g., centrifuge) ($\mapsto$ *Rq2*).
- support the synchronization of instances at well-defined points such as tasks ($\mapsto$ *Rq3*).
- support the simultaneous execution of tasks across multiple instances for one or several process types ($\mapsto$ *Rq4*).

Rq1 refers to instance correlation. Consider, e.g., a patient being subject to multiple different treatment processes. If one examination has already been carried out within one process, the other processes should skip this task. In order to recognize for which instance, i.e., patient this task must be skipped, patient instances must be identifiable via, e.g., a unique patient id (cf. [19]). According to [3], an ISC is linked to event attributes, so-called *instance spanning attributes* which are time, resource and data. ISC can refer to multiple instance spanning attributes at once. Consider, e.g., an ISC stating *A user is not allowed to do event* `approve loan` *if the total loan amount per day and clerk exceeds $1M.* (cf. [18]) In this case, the ISC refers to three event attributes, i.e., the clerk (resource), the same day (time) and the current loan amount (data). Therefore, instance spanning attributes must be representable by an ISC pattern formalism (*Rq2*). Moreover, instances must synchronize like in the centrifuge example where $n-1$ instances have to wait before the `centrifugation` task until the $n-$th instance has arrived (*Rq3*). *Rq4* results from Category I. Note that *Rq1* – *Rq4* also respect the requirements for a visual ISC modeling notation as stated in [4].

In summary, we are looking for a well-established, imperative design formalism with formal execution semantics and strong support of formal analysis. Workflow nets (WF-nets) and suitable extensions are good candidates for an initial selection. Table 1 evaluates this selection along requirements *Rq1* – *Rq4*.

Colored WF-nets support the representation of event attributes and hence meet *Rq1* and *Rq2*. In order to enable the synchronization of instances of different process types (*Rq3*) *Proclets* (cf. [14]), which allow for modeling interactions of processes, were identified as potential formalism candidate. An alternative

Table 1. Assessment of WF-nets and suitable extensions

Formalism	*Rq1*	*Rq2*	*Rq3*	*Rq4*
WF-net	-	-	+\-	-
Colored WF-net	+\-	+	+\-	-
Timed WF-net	-	-	+\-	+
Proclets Based on timed colored WF-nets	+	+	+	+

+ fulfilled, - not fulfilled, +\- partly fulfilled

would be *Workflow Modules* which are often employed in process choreography design (cf., e.g., [17]) and exchange information via incoming and outgoing places. In particular, exchange of instance specific information via colored tokens is enabled which is not a desirable behaviour for ISC patterns since execution of processes should just be coordinated without direct information exchange on running instances. For satisfying *Rq4* the formalism must be able to deal with time aspects which is only the case for timed WF-nets. Consequently, just the combination of timed colored WF-nets and Proclets fulfills all requirements. Proclet instances relying on timed colored WF-nets have a state, support the notion of a task and timed colored WF-nets are a graphical process notation providing soundness which is a prerequisite for a Proclet [8]. Definitions and concepts of the chosen formalism are described in the following.

Definition 1 (Petri/Workflow Net, [13]). *A* Petri net *is a triplet* (P, T, A) *where*

- P *is a finite set of places*
- T *is a finite set of transitions, such that* $P \cap T = \emptyset$
- $A \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs.*

A Petri net is called Workflow Net (WF-net) *if and only if*

- *there is a dedicated source place where the process starts and that has no incoming edge*
- *there is a dedicated sink place where the process ends and that has no outgoing edge*
- *all nodes are on a path from the source place to the sink place.*

The current state of a WF-net is determined by its markings.

Definition 2 (Marked Labeled WF-net, [13]). *A* marked WF-net *is a pair* $((P, T, A), M)$*, where* (P, T, A) *is a WF-net and* $M \in \mathbb{B}(P)$ *is a multiset over* P *denoting the marking of the net.*

Running example: Figure 3 depicts two processes in terms of marked labeled WF-nets. Based on Def. 2, process $\mathcal{P}1$ depicted in Fig. 3 is then given as $((P_{\mathcal{P}1}, T_{\mathcal{P}1}, A_{\mathcal{P}1}), [s])$

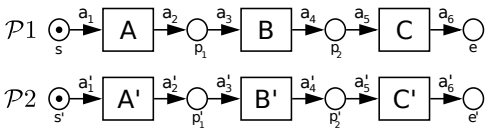

Fig. 3. Running Example, cf. [19]

with $P_{\mathcal{P}1} = \{s, p_1, p_2, e\}, T_{\mathcal{P}1} = \{A, B, C\}, A_{\mathcal{P}1} = \{a_1, a_2, ..., a_6\}$, whereas $a_1 = (s, A), a_2 = (A, p_1), a_3 = (p_1, B), a_4 = (B, p_2), a_5 = (p_2, C), a_6 = (C, e)$. The initial marking has one token in s. Process $\mathcal{P}2$ is given analogously.

Since the requirements for ISC patterns demand the display of event attributes as well as consumption times of tokens, we formalize the ISC patterns through so called *timed colored Workflow nets* which are capable of dealing with data- and time-related aspects [13].

Definition 3 ((Timed) Colored WF-net, [5,17]). *A* Colored WF-net *is a nine-tuple* $(P, T, A, \Sigma, V, C, G, E, I)$ *with*

- (P, T, A) *being a WF-net as described in Def. 1*
- Σ *is a finite set of nonempty types, called color sets*
- $V\colon A \to (P \times T) \cup (T \times P)$ *is a node function that maps each arc identifier to a pair (start node, end node) of the arc*
- $C\colon P \to \Sigma$ *is a color function that associates each place with a color set*
- $G\colon T \to BooleanExpr$ *is a guard function that maps each transition to a predicate*
- $E\colon A \to Expr$ *is an arc expression that evaluates to a multi-set over the color set of the place*
- I *is an initial marking of the colored WF-net.*

Timed colored WF-nets *carry in addition to the token colors a non-negative integer value called* timestamp *determining the time when the token can be consumed by a transition [5]. Markings of places having timestamps correspond to timed multisets and each colored WF-net also has a global clock which represents model time [5].*

For enabling the coordination across instances in the ISC pattern formalism, *Proclets* based on timed colored WF-nets are used which are defined as follows.

Definition 4 (Proclet, Proclet Instance, Proclet System, [2,8,14]).

- *A* Proclet *is a tuple* $Pr = (N, ports)$ *consisting of a timed colored WF-net* N*, a set of ports,* $ports \subseteq 2^T \times \{in, out\} \times \{?, 1, *, +\} \times \{?, 1, *, +\}$[1] *where each port* $pr = (T_{pr}, dir_{pr}, card_{pr}, mult_{pr})$
  - *is associated to a set* $T_{pr} \subseteq T$ *of transitions, s.t.* $\forall t_1, t_2 \in T_{pr}$ *holds:* $l(t_1) = l(t_2)$
  - *has a direction of communication* $dir_{pr} \in \{in, out\}$
  - *has a cardinality* $card_{pr} \in \{?, 1, *, +\}$ *specifying how many performatives may or have to be sent or received upon an occurence of one* $t \in T_{pr}$
  - *has a multiplicity* $mult_{pr} \in \{?, 1, *, +\}$ *specifying how often all transitions* T_{pr} *may occur together during the lifetime of an instance of* Pr
  - *no two ports share a transition,* $T_{pr} \cap T_{pr'} = \emptyset, \forall pr, pr' \in ports, pr \neq pr'$.

  and has a unique transition with no incoming arcs, and a unique transition with no outgoing arcs. These transitions denote actions to create and finish an instance of Pr *respectively.*

[1] ? means 0 or 1, 1 exactly one, $*$ arbitrary number, $+$ at least one

– *A* Proclet instance *corresponds to an instance of the process definition. Ports exchange* performatives*, which have at least six attributes (time, channel, sender, set of recipients, action and content) and are stored in the* knowledge base *of a Proclet instance. The knowledge base can be queried by tasks and contains public as well as private parts. The public part is identical for all instances of the class, i.e. this part resides at the class level even though it holds information about instances. The private part resides exclusively at the instance level. A task may have a* precondition *based on the knowledge base. A task is enabled if i) the corresponding transition in the WF-net is enabled, ii) the precondition evaluates to true, and iii) each input port contains a performative.*
– *A* Proclet System *consists of a finite set of Proclets together with a set of channels via which a Proclet can interact with other Proclets. A* naming service *keeps track of all Proclet instances.*

3 Formalization of ISC Patterns

In the following, 6 ISC patterns are formalized, two for Cat. I and II, whereas the distinction is made between one or multiple process types, and one for Cat. III and IV. The ISC patterns are exemplary illustrated for two Proclets.

Figure 4 depicts the basic building blocks for ISC patterns which are similar, in the case of Category I and II for ISC_S as well as ISC_P. These ISC patterns are just distinguishable based on the underlying timed colored WF-net. In case of Category III, the ISC pattern requires only a onedirectional information exchange. Consider, e.g., `B` must be executed before `C'` (cf. Fig. 3). The further execution of process $\mathcal{P}_1$ containing `B` is independent of whether `C'` has been executed or not. Categories I and II in contrast require a bidirectional information exchange. For Category IV two output ports and one input port are necessary to formalize the ISC pattern.

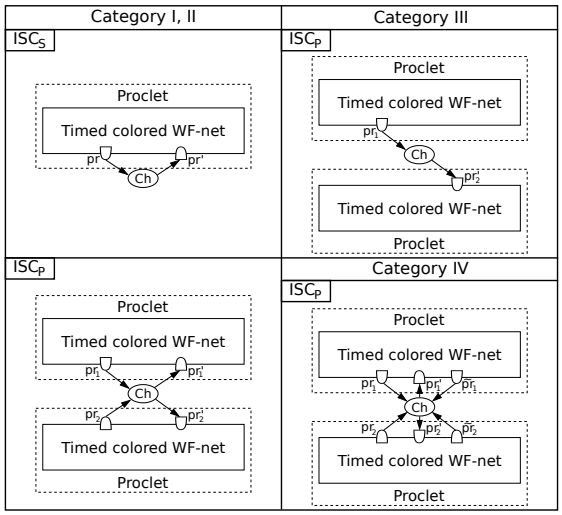

Fig. 4. Building blocks for ISC patterns

Let $\mathcal{N} = \{N_i\}_{i=1,\ldots,n}$ be a set of timed colored WF-nets with $N_i = (P_i, T_i, A_i, \Sigma_i, V_i, C_i, G_i, E_i, I_i)$ as defined in Def. 3 with $\bigcap_i P_i = \emptyset, \bigcap_i T_i = \emptyset, \bigcap_i A_i = \emptyset$.

- The set of ISC transitions is given as $T_{ISC} \subseteq \bigcup_i T_i$. ISC transitions are those tasks affected by the ISC.
- The set of transitions preceding the ISC transition is given as $T_{pre} \subseteq \bigcup_i T_i$. For Category IV, the set of transitions succeeding the ISC transition is defined analogously $T_{post} \subseteq \bigcup_i T_i$.
- The set containing all colors representing the instance spanning attributes is denoted as $\Sigma_{ISC} \subseteq \bigcup_i \Sigma_i$.

Further, let PS be a Proclet system as defined in Def. 4. For modeling ISC the following must be at least fulfilled

- Each Proclet in PS consists of one element in $\mathcal{N}$ and up to three ports. The port connected to the ISC transition is always an input port, the others are output ports. The output ports send performatives, holding information on ISC colors in their content attributes. This enables to check ISC.
- If PS consists of more than one Proclet, Σ_{ISC} must, based on *Rq1*, contain at least one element. This element represents the corresponding instance uid and is defined as type *string*, i.e., $colset\ ID = string; \Sigma_{ISC} = \{ID\}, var\ iscid : ID$.
- Each Proclet instance, corresponding to a process instance, has a $proc_{id}$ which serves for identifying instances within the WF-net, i.e., there exists a corresponding variable $var\ proc_{id} : ID$. Each Σ_i is therefore not empty.
- The channel linking the Proclets based on the ISC is denoted as Ch. If the ISC conditions are fulfilled, the corresponding $proc_{id}$(s) are handed to the content attribute of a performative.
- Since the knowledge base keeps track of all performatives we know which instances have already executed their ISC task(s). Instances that have done so may not further be considered for checking the ISC condition.

Whether ISC transitions can fire depends on the tokens in the preceding place and whether the input port contains a performative.

The ISC patterns are formalized and illustrated based on the timed colored WF-nets depicted in the running example (cf. Fig. 3, Sect. 2). Note that the ISC patterns can be extended to more than two processes as well.

3.1 Category I – Simultaneous Event Execution

For ISC of this category, no matter if one or several process types are considered, it holds that, as long as the condition for the ISC is not fulfilled, all instances have to wait before the ISC transition(s). As soon as the condition is fulfilled, all tokens have to be consumed simultaneously by the ISC transition(s). Therefore, all tokens have a suitable timestamp indicating that they need to be consumed without delay. This timestamp corresponds to the timestamp of the global glock when the transition preceding the ISC transition fires. By that, we ensure that the timestamp is "old enough" such that the ISC transition can always fire. Moreover, we demand that transitions fire as soon as the enablement conditions according to Def. 4 are fulfilled. In the case of multiple process types, all global

clocks start with 0. We assign the timestamp to the color set representing the $proc_{id}$, i.e., this becomes a timed color set with initial timestamp 0.

Simultaneous ISC Pattern (one process type)
Description: As long as the ISC is not fulfilled, all instances have to wait before the ISC transition. As soon as the ISC condition is fulfilled, all affected instances must execute the ISC transition simultaneously.
Formalization: Let $A \in T_{pre}, B \in T_{ISC}, pr = (\{A\}, out, +, 1)$ and $pr' = (\{B\}, in, 1, +)$. The guard function G for B must identify the correct tokens based on the $proc_{id}$, i.e., transition B may only consume the tokens having the correct $proc_{id}$. This information is encoded in the content attribute of the performative received via pr'. All tokens must be consumed by B simultaneously.

Simultaneous ISC Pattern (multiple process types)
Description: Until the ISC condition is not fulfilled, all instances have to wait before the ISC transitions. As soon as the ISC condition is fulfilled, all affected instances must execute the ISC transitions simultaneously.
Formalization: Let $A, A' \in T_{pre}, B, B,' \in T_{ISC}, pr_1 = (\{A\}, out, +, 1), pr_2 = (\{A'\}, out, +, 1), pr'_1 = (\{B\}, in, 1, +)$ and $pr'_2 = (\{B'\}, in, 1, +)$. The guard functions G_1, G_2 for B, B' must identify the correct tokens based on the $proc_{id}$, i.e., transitions B, B' may only consume the tokens having the correct $proc_{id}$. This information is encoded in the content attribute of the performative received via pr'_1, pr'_2. All tokens must be consumed by B and B' simultaneously.

3.2 Category II – Constrained Event Execution

The ISC patterns for this category are based on the same building blocks as those for Category I. However, there are two differences. First, simultaneous execution is not of importance and second, according to the real-world ISC examples, mostly ISC conditions correspond to thresholds, like, e.g., the amount in the before mentioned example for Category II. This leads to the situation that, as long as this threshold is not met, instances may be executed.

Constrained ISC Pattern (one process type)
Description: As long as the ISC threshold is not reached, all instances may execute the ISC transition. As soon as the ISC threshold has been reached, all affected instances must wait until the ISC threshold can be reset.
Formalization: Let $A \in T_{pre}, B \in T_{ISC}, pr = (\{A\}, out, +, 1)$ and $pr' = (\{B\}, in, 1, +)$. The guard function G for B must identify the correct tokens based on the $proc_{id}$, i.e., transition B may only consume the tokens having the correct $proc_{id}$. This information is encoded in the content attribute of the performative received via pr'.

Constrained ISC Pattern (multiple process types)
Description: As long as the ISC threshold is not reached, all instances may execute the ISC transitions. As soon as the ISC threshold is has been reached, all affected instances must wait until the ISC threshold can be reset.

Formalization: Let $A, A' \in T_{pre}, B, B,' \in T_{ISC}, pr_1 = (\{A\}, out, +, 1), pr_2 = (\{A'\}, out, +, 1), pr'_1 = (\{B\}, in, 1, +)$ and $pr'_2 = (\{B'\}, in, 1, +)$. The guard functions G_1, G_2 for B, B' must identify the correct tokens based on the $proc_{id}$, i.e., transitions B, B' may only consume the tokens having the correct $proc_{id}$. This information is encoded in the content attribute of the performative received via pr'_1, pr'_2.

3.3 Category III – Ordered Event Executions

ISC of this category span multiple process types. It holds that ISC transitions may only consume tokens if the preceding place contains the associated token and the preceding transition, that needs to be executed first and is present in a different process, has already been executed.

Ordering ISC Pattern
Description: As long as the instance containing the transition which precedes the ISC transition has not arrived at the synchronization point, the corresponding instance, i.e., the one with the same instance uid, located in a different process may not execute the ISC transition. As soon as the transition preceding the ISC transition has been executed the associated instance may execute the ISC transition.
Formalization: Let $B \in T_{pre}, C' \in T_{ISC}, pr_1 = (\{B\}, out, +, 1)$ and $pr'_2 = (\{C'\}, in, 1, +)$. The guard function G for C' must identify the correct tokens based on the $proc_{id}$. Transition C' can only consume the token having the correct $proc_{id}$. This information is encoded in the content attribute of the performative received via pr'_2.

3.4 Category IV – Non-concurrent Event Execution

For ISC of this category only one transition involved in the ISC is allowed to consume a token, all other transitions have to wait until it is finished. Consider, e.g., that two tasks `B` and `B'` may not be executed concurrently. In order to formalize this ISC pattern, an additional set $T_{post} \subseteq \bigcup_i T_i$ containing the transitions succeeding an ISC transitions must be introduced. This allows for checking whether one task has already finished since its succeeding transition was executed.

Non-concurrent ISC Pattern
Description: If an ISC transition has been started all other associated instances, i.e., those having the same instance uid, have to wait until it is completed. Afterwards, only one other associated instance may start its ISC transition and all remaining instances have to wait until it is finished. This continues until all instances have executed their ISC transitions.
Formalization: Let $A, A' \in T_{pre}, B, B' \in T_{ISC}, C, C' \in T_{post}$ and ports $pr_1 = (\{A\}, out, +, 1), pr_2 = (\{A'\}, out, +, 1), pr'_1 = (\{B\}, in, 1, +), pr'_2 = (\{B'\}, in, 1, +)$, $\tilde{pr}_1 = (\{C\}, out, +, 1)$ and $\tilde{pr}_2 = (\{C'\}, out, +, 1)$. The guard functions G_1, G_2 for B, B' must identify the correct tokens based on the $proc_{id}$. Transitions B, B' can only consume the token having the correct $proc_{id}$. This information is encoded in the content attribute of the performative received via pr'_1, pr'_2.

4 Evaluation of ISC Patterns

ISC patterns are evaluated through their application in processes in Sect. 4.1 and in existing approaches in Sect. 4.2, followed by a discussion in Sect. 4.3.

4.1 Application to Business Processes

Centrifuge Example. Figure 5 picks up the centrifuge example outlined in the introduction with corresponding ISC *Wait until centrifuge is filled.* In order to model the Simultaneous ISC pattern, a Proclet system is given by a timed colored WF-net and $ports = \{pr, pr'\}$, $pr = ((put\ in\ centrifuge), out, +, 1)$, $pr' = ((centrifugation), in, 1, +)$ as well as one channel Ch.
Moreover, $(put\ in\ centrifuge) \in T_{pre}, (centrifugation) \in T_{ISC}, \Sigma = \{NUM, ID\}$, $var\ procid : ID, var\ slots : NUM$ and $\Sigma_{ISC} = \{NUM\} \subseteq \Sigma$ whereas ID is a timed color set.

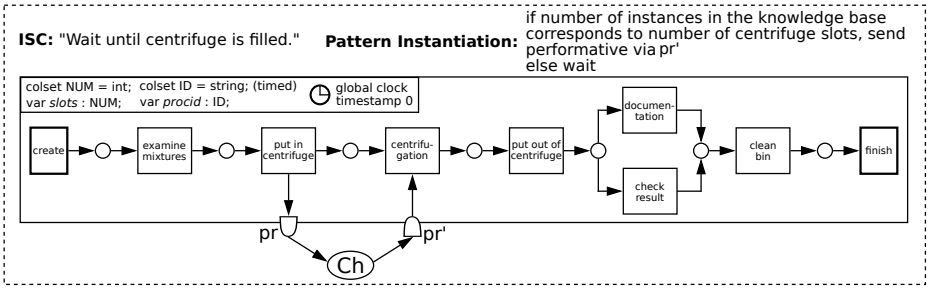

Fig. 5. Recap of laboratory process including the Simultaneous ISC Pattern (one process type)

Printer Example. Figure 6 depicts three processes adapted from [19] whereas $\mathcal{P}1$ outlines the handling of flyer orders, $\mathcal{P}3$ the handling of poster orders and $\mathcal{P}2$ the corresponding billing process for both flyer and poster orders. In [19], six different ISC are outlined, but for illustration purposes, only one example is depicted in this paper while the remaining ISC examples are provided as supplementary material.[2]. The ISC states *Flyers and posters as well as bills and posters cannot be printed concurrently on one printer since they require a different paper format* which can be modeled using the Non-concurrent ISC pattern. Each process is represented by a Proclet system $PS_i = (N_i, ports_i), i = 1, 2, 3$ based on a timed colored WF-net N_i, $ports_i = \{pr_i, pr'_i, \tilde{pr}_i\}$, with $pr_1 = ((send\ draft\ to\ customer), out, +, 1), pr'_1 = ((print\ flyer), in, 1, +)$, $\tilde{pr}_1 = ((deliver\ flyer), out, +, 1)$ and the remaining ports defined analogously, as well as one channel Ch. Moreover, $(send\ draft\ to\ customer), (write\ bill)$,

[2] `http://gruppe.wst.univie.ac.at/projects/crisp/index.php?t=iscpatterns`

$(design\ poster) \in T_{pre}, (print\ flyer), (print\ bill), (print\ poster) \in T_{ISC}$, $(deliver\ flyer), (deliver\ bill), (deliver\ poster) \in T_{post}, \Sigma_i = \{ID^2, RES\}$ and $\Sigma_{ISC} = \{ID^2, RES\}$ with corresponding variables as depicted in Fig. 6.

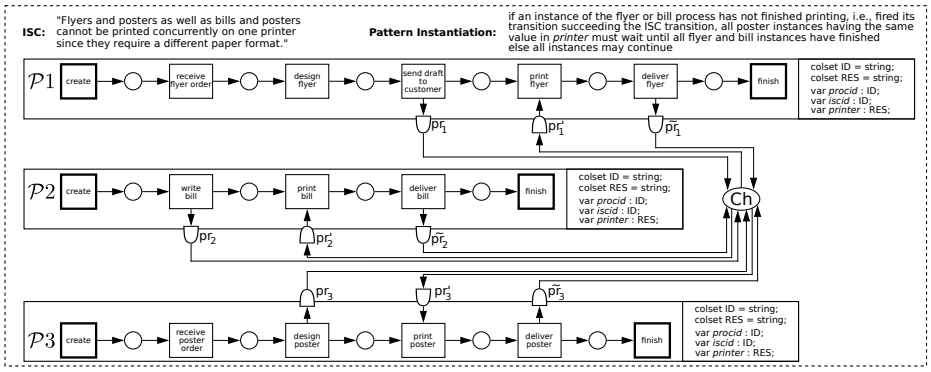

Fig. 6. Printer example adapted from [19] including the Non-concurrent ISC Pattern

4.2 Application to Existing Approaches and Related Work

Existing approaches in the context of ISC address the modeling, implementation, and execution of ISC in business processes [3,7] and the usage of ISC for applications such as batching and security. How ISC patterns can be applied for batching and security is discussed in the following.

Batching [10]. Batching can be either parallel, meaning that "instances for one activity are executed simultaneously and get terminated before the next activity is executed the same way" or sequential, i.e., "activities within a batch region are executed for one process instance (case) before the next one can be started." A batching region is determined by four parameters, the *i) groupingCharacteristic, ii) activationRule, iii) maxBatchSize* and *iv) executionOrder*. The identification of instances based on instance spanning attributes and related instances via the instance uid corresponds to i). The ISC, e.g., task `centrifugation` may only be started if the centrifuge is filled corresponds to ii) activationRule whereas iii) the maxBatchSize is determined by, e.g., the number of centrifuge slots. The execution order in this case is parallel and the Simultaneous ISC pattern can be applied. For sequential processing, the Non-concurrent ISC Pattern can be applied. Since batching typically spans just one process type, the process must be duplicated in order to be able to apply the Non-concurrent ISC Pattern.

Security [11]. Security constraints such as access control and separation of duties within processes can be modeled using the Constrained ISC pattern, in one or multiple process type form. Consider, e.g., two tasks `grant loan` present in process $P1$ and `check loan accommodations` present in process $P2$. A user

u having granted a loan with $uid = 123$ may not check the accommodation of the loan case with $uid = 123$. Based on token colors representing the user and the uid the execution of task `check loan accommodations` by user u can be prohibited for the case with $uid = 123$. Only a second user is allowed to execute task `check loan accommodations` for the instance with $uid = 123$.

4.3 Discussion

The presented ISC patterns can be considered *comprehensive* w.r.t. the existing ISC categorization [19] and collection of real-world ISC [3]. Still, new ISC examples might lead to additional ISC patterns. Besides, the current collection of ISC patterns does not cover ISC for handling exceptions which might be quite diverse since exceptions can be manifold. Existing workflow patterns for exception handling[3] seem to be a useful starting point for modeling ISC exception handling patterns. Moreover, processes and process instances might be subject to multiple ISC at once, i.e., ISC patterns must be *combinable* without interrupting and influencing each other. Regarding the *simplicity* of the formalization choice: Consider as an ISC example *five tokens need to be consumed simultaneously by transition B*. This could simply be modeled by having capacity 5 for the arc before task `B` and a timed WF-net without colors. So, for this simple ISC, Proclets and colors are actually not required. However, as mentioned before, ISC can have further conditions. Consider, e.g., *task `B` must be executed for five instances simultaneously by the same resource `R1`*. In this case, we need to check whether five instances have arrived before task `B` for `R1`, i.e., token colors are required in order to model and consequently check the resource. Additionally, we need a knowledge base that has stored all instances and corresponding data values and can be queried whether enough instances have arrived for resource `R1`. Hence, the expressiveness of Proclets based on timed colored WF-nets is mandatory.

5 Conclusion

This work presents 6 patterns for constraints spanning multiple process instances of one or multiple process types. These ISC patterns are based on a collection of real-world ISC examples and an existing ISC categorization. Several requirements imposed by ISC, like support of instance correlation or the need for representing instance spanning attributes, led to selecting Proclets based on timed colored Workflow nets for ISC pattern formalization. ISC patterns are evaluated through an application to business processes as well as existing approaches such as batching and security. The discussion outlines several links to future work including the elicitation and formalization of ISC patterns for exception handling as well as investigating how ISC patterns can be combined without introducing, e.g., a blocking of the process execution. When implementing and executing ISC at runtime, aspects such as scalability become subject for investigation well.

[3] `http://www.workflowpatterns.com/patterns/exception/`

References

1. van Der Aalst, W.M., Ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and parallel databases 14(1), 5–51 (2003)
2. Fahland, D., de Leoni, M., van Dongen, B., van der Aalst, W.: Checking behavioral conformance of artifacts. BPM Center Report BPM-11-08, BPMcenter. org (2011)
3. Fdhila, W., Gall, M., Rinderle-Ma, S., Mangler, J., Indiono, C.: Classification and formalization of instance-spanning constraints in process-driven applications. In: Business Process Management. pp. 348–364 (2016)
4. Gall, M., Rinderle-Ma, S.: Visual modeling of instance-spanning constraints in process-aware information systems. In: Advanced Information Systems Engineering. pp. 597–611 (2017)
5. Jensen, K., Kristensen, L.M.: Coloured Petri nets: modelling and validation of concurrent systems. Springer Science & Business Media (2009)
6. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Inf. Syst. 54, 209–234 (2015)
7. Mangler, J., Rinderle-Ma, S.: Rule-based synchronization of process activities. In: Commerce and Enterprise Computing. pp. 121–128 (2011)
8. Mans, R., Russell, N.C., van der Aalst, W.M., Bakker, P.J., Moleman, A.J., Jaspers, M.W.: Proclets in healthcare. Journal of Biomedical Informatics 43(4), 632–649 (2010)
9. Martin, N., Swennen, M., Depaire, B., Jans, M., Caris, A., Vanhoof, K.: Retrieving batch organisation of work insights from event logs. Decis. Support Syst. 100, 119–128 (2017)
10. Pufahl, L., Meyer, A., Weske, M.: Batch regions: Process instance synchronization based on data. In: Enterprise Distrib. Object Comp. pp. 150–159 (2014)
11. dos Santos, D.R., Ranise, S.: On run-time enforcement of authorization constraints in security-sensitive workflows. In: Software Engineering and Formal Methods. pp. 203–218 (2017)
12. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Inf. Syst. 53, 278–295 (2015)
13. Van Der Aalst, W.: Process mining: discovery, conformance and enhancement of business processes, vol. 2. Springer (2011)
14. Van Der Aalst, W.M., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. International Journal of Cooperative Information Systems 10(04), 443–481 (2001)
15. Voglhofer, T., Rinderle-Ma, S.: Collection and elicitation of business process compliance patterns with focus on data aspects. Bus Inf Syst Eng (2019)
16. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. 66(3), 438–466 (2008)
17. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007), https://doi.org/10.1007/978-3-540-73522-9
18. Winter, K., Rinderle-Ma, S.: Discovering instance-spanning constraints from process execution logs based on classification techniques. In: Enterprise Distributed Object Computing Conference. pp. 79–88 (2017)
19. Winter, K., Stertz, F., Rinderle-Ma, S.: Discovering instance and process spanning constraints from process execution logs. Information Systems 89, 101484 (2020)