From Instance Spanning Models To Instance Spanning Rules

Manuel Gall, Stefanie Rinderle-Ma

University of Vienna, Faculty of Computer Science, Vienna, Austria
manuel.gall@univie.ac.at, stefanie.rinderle-ma@univie.ac.at

Abstract. Instance Spanning Constraints (ISCs) allow to control the behavior of multiple business processes and their instances which is crucial in many application domains (e.g., for process synchronization). The modeling and visualization of ISCs hence constitutes an essential brick in process compliance management. Currently, no approach exists for representing a set of Instance Spanning Rules (ISRs) based on an ISC. Existing work rather focuses on visualizing ISCs at the more abstract Instance Spanning Model (ISM) level. However, the gap between the ISM and the ISR level must be bridged in order to enable the enactment of an ISC at process level. Hence, this paper collects requirements for the implementation of ISRs through an ISM, e.g., by specification of data elements. Based on the requirements an existing visual modeling language is tailored towards the modeling of ISRs and the corresponding XML specifications are provided. Both, visual modeling and XML representation are prototypically implemented and illustrated by means of a set of use cases. Finally, an algorithm for deriving the common ISM of a set of ISRs is introduced and evaluated based on a given test set.

Keywords: Instance Spanning Constraints, Instance Spanning Rules, Compliance, Rule Visualization, Instantiable Constraint, Process-Aware Information Systems

1 Introduction

Business processes compliance deals with the enforcement of constraints such as new laws and regulations. Over time business process compliance became a billion dollar market [6]. Nowadays compliance rules tend to incorporate more and more business knowledge. Recent studies show that constraints spanning multiple instances and or processes are omnipresent [4]. Some approaches to deal with instance spanning constraints (ISCs) exist, e.g., [14,4,7]. However, the development life cycle from extracting, modeling, and implementing ISCs is not fully supported yet.

Contrary, for intra instance compliance the development life cycle has been established and starts with a constraint definition, e.g., a new law or regulation, followed by creating a non-executable model mainly for communication purposes [3]. This model is refined into instantiable rules which can be checked for process instances [11].

Establishing the development life cycle for ISCs requires a visual modeling notation in order to create instance spanning models (ISM). For this, for example, the visual notation ISC_Viz [5] can be used. However, ISMs are not executable. Hence, an ISM has to be instantiated and specified into executable instance spanning rules (ISRs), but an approach for this is missing.

This work aims at bridging the gap between ISM and ISRs and vice versa based on the following questions: "(1) How to go from an ISM towards a set of instance spanning rules (ISRs) – visually and at the implementation level?" and "(2) How to merge a set of ISRs into an ISM again?". The second question is important particularly in large, living systems where for a given set of ISRs the common schema would be important for validation. In order to answer these questions we use design science research [17]. A set of requirements is gathered from literature. Based on these requirements we build the following artifacts: a visual notation for ISRs, a corresponding XML representation, and a merge algorithm. We evaluate the applicability on a set of real world examples and compare the merge results against previous research.

The paper is structured as follows: Sect. 2 provides a motivating example. Sect. 3 discusses related approaches. Requirements are collected in Sect. 4. Our proposed solution for ISRs is discussed in Sect. 5. Sect. 6 introduces a merge algorithm for ISRs. The applicability of our approach is discussed in Sect. 7 and Sect. 8 concludes with a summary.

2 Motivating example

We lay out the basic concepts of the constraint development life cycle and management by means of an example. A vial has to be examined. Figure 1 Ⓐ shows the process in BPMN notation. The process consists of three activities *PreTest*, *Centrifugation* and *PostTest*. In the *PreTest* the vial is examined. In the *Centrifugation* activity the vial is put into the centrifuge and the centrifuge is started and stopped when centrifugation is finished. Afterwards the vial is taken out of the centrifuge. In the last activity a post examination is performed.

Figure 1 Ⓑ shows an example for an intra-instance instantiable rule modeled with BPMN-Q [2]. It states that the *PreTest* has to precede the *PostTest*. Furthermore the figure shows two instances 1 and 2. Instance 1 complies to the constraint while 2 violates it. The decision on compliance or violation can be made for each instance in a separate manner. What we cannot see in the visualization is that in order to perform such a compliance check each task has to be linked to a task within the process.

Figure 1 Ⓓ depicts all modeling elements from the ISM language used in this paper. *Dataelements* and *Resources* represent values specified during design or run-time and either within the rule or the process model. A *Conditional - Trigger* uses *Dataelements* and *Resources* to validate a business rule. The *Timer - Trigger* extends a *Conditional* with time related information. If a business rule within a trigger is true the action part is executed, i.e., *Start* the execution of a

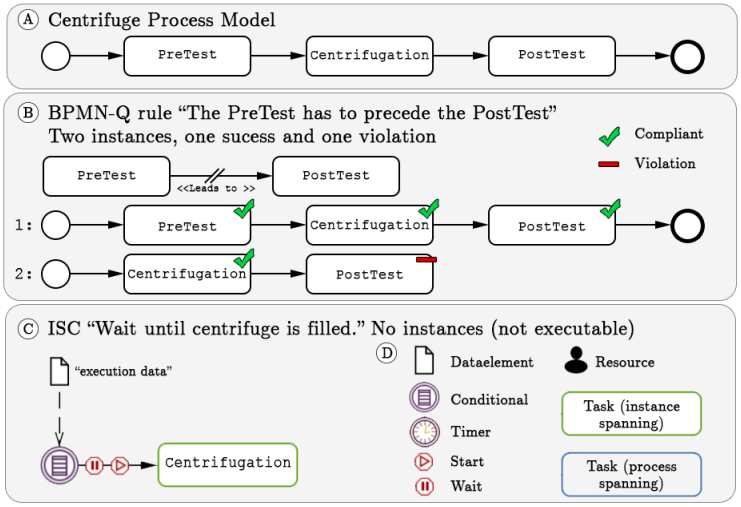

Fig. 1. Centrifugation process with intra-instance constraints and ISM.

task or *Wait* before a task is executed. Instance spanning tasks are represented with a green border while process spanning tasks with a blue one.

Figure 1 Ⓒ depicts an example ISC "Wait until centrifuge is filled." [15]. What differentiates an ISC from an intra-instance constraint is that the ISC involves multiple instances and/or processes. In this example the centrifuge offers a maximum load capacity of 4 vials and shall only be started when the maximum load capacity is reached. We used ISC_Viz to model the corresponding ISM. The visualization shows a *trigger* which performs a check resulting in a boolean value. A *data element* that stores a numeric value. Two *actions* wait and start and a *task*. What we cannot see is that the task is currently not linked to a process or instance. Based on the color *green* we know that the task has to be linked to one process in order to be executable. In its current state this model is not executable, the trigger does not state a condition, the data element and task are represented only by a label and the execution order of the actions is unclear.

3 Related Work

Most studies on constraints focus on modeling intra-instance constraints [3] or instantiating constraints [6,10,12] rather than widening the view towards ISCs. However, recent approaches such as [4,7,8] concentrate more and more on constraints spanning across instances and processes. Batching of multiple instances [14] is a common scenario when dealing with ISCs, however the work focuses on integrating batching regions into the process rather then separating the logic. At a formal level recent work even deals with going towards instantiation [16,9,4,8]. Also first steps are taken towards the discovery of ISCs from process logs [18]. Current approaches for visualizing constraints focus on intra-instance constraints [1,11]. These approaches can not be used when dealing with spanning constraints

due to more data and the involvement of multiple instances and or processes. ISC_Viz [5] is a visual notion for modeling ISMs. However, the language is limited to visually represent models for communication purpose only. The created models are not instantiable. With our research we want to extend ISC_Viz towards the modeling of ISRs.

4 Requirements for ISR Representation

This section states the requirements for the specification of ISRs based on an ISM. The requirements are derived from literature on instantiation of intra-instance compliance rules [11], instance-spanning constraints in general [4], and the visualization of ISMs [5]. This selection of approaches is feasible as the classification and visualization of instance-spanning constraints yields the necessary ingredients to go from their abstract and process-independent representation to an executable form. The related work on intra-instance constraints confirms these specification steps going from process-independent to process-specific rules. However, as ISCs contain additional, runtime related information such as trigger and actions when compared to intra-instance constraints, the approach in [11] cannot be directly applied to ISCs.

Nonetheless, when specifying executable rules the runtime-related parts of ISCs have to be specified, i.e., the data, the resources, and the linkage, leading to the following set of requirements:

Requirements derived from intra-instance constraints

1. *Data specification:* When an ISM is transformed into a rule all data elements have to be specified. For example, an ISM states that a loan amount is needed. In case of this example a rule either assigns a specific value to the variable or links to a source where to read the variables value.
2. *Resource specification:* While an ISM gives an overview of involved resources, a rule has to have a link to the resource and allow for e.g., checking if the resource is available.
3. *Linkage:* An ISM only states activity names. A rule has to specify a link between each activity of the rule and an activity within a process. An exception of this behavior are activities that link to no specific process, in this case the activity links to a repository of activities. An example rule that demonstrates this behaviour can be found in Section 7.

As mentioned before instance-spanning constraints contain additional information when compared to intra-instance constraints [4] which is necessary to define their scope (spanning instances or processes), their trigger (when do they become activated), and their actions, e.g., waiting.

Requirements derived from instance-spanning constraints

4. *Execution data*: Requirement 1 already handles that data elements are to be specified before a rule can be created. Execution data needs special treatment as this data is generated during process execution and thus is a specification of ISMs and not part of an intra-instance constraint.
5. *Trigger specification:* In order to allow validation of both trigger types *Conditional* and *Timer* a condition and behaviour have to be specified. A condition for the *Conditional* is specified by using data and resources in order to create a boolean condition, e.g., instance counter == load_capacity. A *Timer* condition allows for modeling the same condition as a *Conditional* with the addition of a time event, e.g., lastExecution >1 hour && instance_counter == load_capacity. The behaviour of a trigger is executed only if the condition is evaluated as true and enables modeling of data and resource manipulation.
6. *Action specification:* Some actions need to be specified in more detail when handling an ISR e.g. an *Alert* consists of a message, video, sound, or service that shall be executed.

An additional goal of this paper is to automatically create an ISM based on ISRs. This necessitates the following technical requirements.

Requirements for automatic ISR creation

7. *Unique ISM identifier:* Every set of ISRs has to store an ISM ID in order to enable grouping of ISRs e.g. a set of 3 ISRs stores the ISM ID of its corresponding ISM. If there is no corresponding ISM a unique value is created.
8. *Unique ISR identifier:* Every ISR is identified by an unique ID.

Current approaches for visualizing an ISM will be used to visualize ISRs. To allow the user to understand both, ISM and ISRs, we will follow a set of strict requirements on the visualization. Those requirements are based on Moodys principles [13] and ensure that users who are familiar with ISM modeling will have no problem in understanding and modeling ISRs.

Requirements on ISR visualization

9. *Principle of graphic economy:* The complexity of the constraint visualization shall not be changed. Therefore it is not allowed to add additional elements to the visualization, e.g., new symbols.
10. *Semiotic Clarity:* Avoid the definition of multiple semantics for a visual element.

5 ISR Representations

This section provides a visual as well as an XML-based representation for ISRs following the requirements as set out in Section 4. As ISRs are typically specified based on an ISM the visual ISR representation has to be designed in accordance with the visual ISM representation. ISC_Viz [5] is currently the only visual modeling language for ISMs. Thus ISC_Viz will serve as basis for the visual ISR modeling language. It will be discussed which of the ISC-related information is visualized, and which is only contained within the xml representation.

5.1 Visual ISR representation

Our focus for developing a visual ISR representation are Requirements 9 and 10 as stated in Section 4: 9 – no further visual elements when compared to ISC_Viz – and 10 – avoid multiple semantics per visual element. In order to meet these requirements we use the principle of complexity management and the principle of dual coding as proposed by Moody [13].

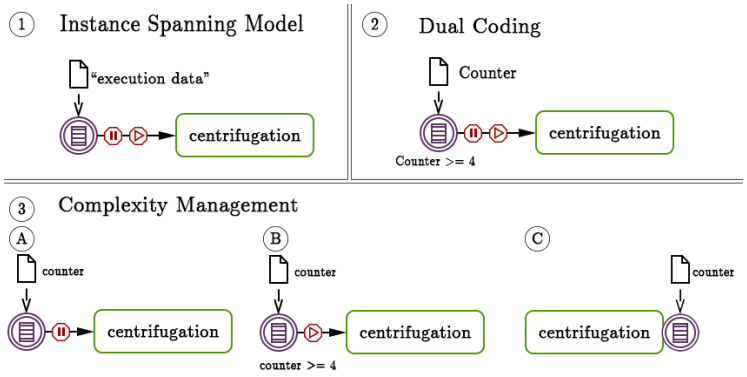

Fig. 2. Representing an ISM as a set of ISRs (example).

Dual coding Dual coding uses text in order to complement graphics. We will use this principle to add text to visual elements of an ISM, i.e., data, resource, task, and trigger, in order to meet Requirements 1 – 4 and partly 5. As text is already used within ISMs this is not considered as adding a new visual element and thus does not violate Requirement 9.

Applying dual coding on the ISM running example of Figure 2 ① results in the ISR ②. During the creation process, local data element *Counter*, a link to a task within a process as well as a condition and behaviour are specified.

While the specification of ISRs with respect to data, resources, and tasks at the visual level is sufficient using text, at the implementation level, the modeler has to specify either a link to a certain data element, resource and task within a process or repository and/or create a local variable within the rules. To avoid visual overload [13] we decided to not include URLs within the visual notion. Specifying URLs and values allows the rule execution engine access to values used within process instances and thus be used for evaluating conditions. Conditions and their behaviour have to be specified by the modeller. A condition is either true and the behaviour is executed or false and nothing else happens. As far as visualizing the condition and behaviour we opted to visualize the condition of a trigger and specify the details of the behaviour within the XML representation.

Dual coding enables the specification of several ISM elements such as data, but is not yet sufficient to realize executable ISRs. What is missing is the specification of the behavioural parts, i.e., the actions, that are triggered based on

the specified conditions. In Figure 2 ②, for example, we cannot decide what happens when the *Counter* reaches a value of four or greater. Shall the wait or start action be executed?

In addition, using dual coding only does not enable the specification of different conditions, i.e., the creation of ISRs that possess different conditions, but follow the same ISM. To tackle the specification of actions and multiple conditions, the visual representation of ISRs has to be further refined.

Complexity management The ISRs shall include a mechanism to allow for complexity management. Moody suggests two ways of dealing with complexity, modularization and hierarchy. Modularization divides a problem into multiple smaller problems. Hierarchy allows to represent a problem at different levels of detail. We will use this principle to refine the ISR visualization and tackle Requirement 5. We split one ISM into multiple rules. Each rule consists of one trigger and task, and zero to multiple resources, data elements, and actions. This allows the rule execution engine to know what behaviour and what action shall be executed in case a condition is true. A special type of trigger is a trigger without a condition such a trigger is always true.

Figure 2 ③ depicts the application of complexity management and dual coding on the ISM ①. ISR Ⓐ shows a rule without a condition and action wait. A rule engine will stop all instances before *centrifugation* is executed. The visualization does not show the default value of the *counter* (0) and the behaviour that every time this rule is true the *counter* is increased by 1. ISR Ⓑ depicts a condition *counter* >= *4* and a start action. This means that if an execution engine evaluates this rule as true the following task (i.e., *centrifugation* in the example) will be started. ISR Ⓒ does not state a condition but we can see that the *counter* is used again. This rule can be seen as a clean up. Within the behaviour we decrease the counter by one for each instance that finished the *centrifugation* task. If the process engine supports instance counting the *counter* of the engine can be used. In this case increasing and decreasing the counter within the rule engine is not needed.

In comparison to the ISM the ISRs show more information. When looking at the ISRs we know that we always stop the execution of the instances before the task *centrifugation*. We know that we start the instances when we reached the fourth instance. Specifying URLs and/or values for all elements creates a set of executable rules. The following XML representation details information that is not shown within the visual representation, e.g., URLs, behaviour and IDs.

5.2 XML representation

When transforming an ISM to ISRs there is more information needed than shown in the visualization, i.e., URLs, IDs, and behaviour. We use XML as file format to store all the necessary information for communication with our execution and rule engine as both support XML files as input and output. Furthermore

we opted for XML as it allows for validation against a schema. The schema describes all necessary information we know from the visualization as well as some "hidden" information. We will highlight this hidden information through the following discussion. The following code snippets are a short excerpt from the detailed Relax NG schema available here[1].

The visualization of data elements only visualizes their *label*, but to satisfy Requirements 1 and 4, we have to store additional information. Every *data* element consists of a *type*, e.g., *data* and *execution data*, a *label* and either *content* or an *url* and an *id*. Data elements that specify a value, e.g., MaximumNumber of concurrent instances in centrifuge, will most likely use the element *content* and specify the value. A more dynamic approach allows to link a value to the process itself by specifying the element *url* and *id*. Linking a value to a process allows access to process data and the possibility to use this data for *trigger conditions* and *behaviour*.

Listing 1.1. Relax NG data elements representation

```
<define name="dataelements">
  <element name="dataelements">
    <zeroOrMore>
      <element name="data">
        <element name="type"><choice>
          <value>data</value><value>execution</value>
        </choice></element>
        <element name="label"><text/></element>
        <choice>
          <element name="content"><text/></element>
          <group><element name="url"><text/></element><element name=
              "id"><text/></element></group>
<choice>  </element>  </zeroOrMore>  </element>  </define>
```

From a visual and XML perspective *resources* and *data* are nearly handled the same. As stated in Requirement 2 – a resource always links to a process or repository – we have to represent this behaviour within the XML representation. To do so we use an *URL* and *ID* pointing to a resource. The idea is that every resource is represented with a web interface i.e., all printer "/printer/id" are represented as rest services and to get data from this device one uses the URL and ID to for example determine the current ink level "/printer/id/inklevel".

We assume that most of the instantiable rules consist of one *action*. But there might be cases where multiple actions are possible, e.g., execute an alert and then perform the *action* wait. To satisfy Requirement 6 an optional URL can be used for user notifications. An alert can show a message to a user, but one can think of further use cases like playing a sound or opening a service.

The *trigger* visualization is a bit different compared to other visualizations. A *trigger* can be represented in two ways by a *conditional* or a *timer*. This representation depends on what is stated within the XML element *type* shown in Listing 1.2. A *type* is specified either as *conditional* or *timer*. While the *condition* is visualized a trigger stores a behavioural part that is not visualized, but needed to meet Requirement 5. Additionally all previous defined elements *data*, *resources* and *actions* are included in the *trigger*.

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

Listing 1.2. Relax NG trigger representation

```
<define name="trigger">
  <optional>
    <element name="trigger">
      <element name="type"><choice>
        <value>conditional</value><value>timer</value>
      </choice></element>
      <ref name="dataelements"/>
      <ref name="resources"/>
      <element name="condition"><text/></element>
      <element name="behaviour"><text/></element>
      <ref name="actions"/>
    </element>  </optional> </define>
```

Listings 1.1 – 1.2 are combined within one rule definition shown in Listing 1.3. As specified in Requirement 7 and 8 every rule consists of an ID that groups all rules under a single ID. Additionally we allow for naming and describing a set of rules. Every rule set consists of one or more rules. To allow for identifying a single rule we again use a unique id for every rule within a set of rules. Furthermore rules consist of elements for name, description, and *priority*. The *priority* is needed within the rule engine to determine in which order rules shall be evaluated. A *trigger* before the task signals that the trigger is evaluated before the execution of a task while a *trigger* after a task is evaluated afterwards. As Requirement 3 states that a task within a rule has to be linked to a either a process or a repository. Within the XML file this linking is expressed by an element *spanning* with two values *single* for an URL to a process and *multi* for an URL to a repository. The *url* element specifies under which link the corresponding *process* or *repository* can be found. Within the *process* or *repository* an ID is used to narrow it down to a specific activity. One could argue that we do not store a label within the task, but the visualization of a rule shows a label within the task. To be consistent between the linked *process* or *repository* and the rule we use the label from the linkage. In some cases a task might not be needed, e.g., cleaning variables as shown in Section 7, Table 1 ①.

The key differences between the XML and visual representation are that *IDs*, *URLs* and *priority* and some element specifics, e.g., *types*, *behaviour* are not visualized, but stated within the XML file. To the contrary the *label* of a *task* is visualized but it is only indirectly stored within the XML file by specifying a *url* and an *id*.

Listing 1.3. Relax NG combination of all parts

```
<element name="rules">
  <element name="id"><text/></element>
  <element name="name"><text/></element>
  <element name="description"><text/></element>
  <oneOrMore>
  <element name="rule">
    <element name="id"><text/></element>
    <element name="name"><text/></element>
    <element name="description"><text/></element>
    <element name="priority"><data type="integer"/></element>
    <ref name="trigger"/>
    <element name="task">
      <optional>
        <element name="spanning"><choice>
```

```
15            <value>single</value><value>multi</value>
16          </choice></element>
17          <element name="url"><text/></element>
18          <element name="activityid"><text/></element>
19        </optional>  </element>
20      <ref name="trigger"/>
21    </element>  </oneOrMore>  </element>
```

6 Automatic creation of an ISM from a set of ISRs

The specification of ISRs based on an ISM is necessary in order to create executable rules. In addition, the other way round – deriving and ISM from a set of ISRs – is also of interest. Reasons include displaying common elements and structure as well as providing a schema for further ISRs. Creating the ISM from a set of ISRs should be done automatically in order to not burden the user with this task. A precondition for the automatic creation of an ISM from an ISR is that the ISR is valid with the schema set out in Section 5.2[2]. It should be possible to select ISRs from a given set for ISM creation. The reason is that the user might not be interested in a fully detailed ISM. This is achieved by selecting a priority value. Only rules with an higher priority will be merged to an ISM.

Algorithm 1 merges a given set of ISRs into an ISM by sorting all rules based on their priority. Then every rule that has a higher priority than a given value is considered for merging. All rule elements such as trigger, data elements, and tasks that are not represented within the generated ISM yet, will be added and simplified. Simplifying means to omit information that an ISM does not contain, i.e., links, conditions, and behavior. The implementation of Algorithm 1 as well as an illustrating example are provided as part of the evaluation in Section 7.2.

Algorithm 1 Merging ISRs to create an ISM.

```
rules = all ISRs from the XML input
priority = value from input
model = null
rules.sort()
while rules do
    if rules[i].priority > priority then
        while each element from rules[i] do
            if !model.contains(rules[i].element[k]) then
                model.add(rules[i].element[k])
            end if
        end while
    end if
end while
model = model.simplify()
```

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

7 Evaluation

Roughly, the evaluation is conducted in a circle where we start with an ISC and an ISM. We visually specify ISRs based on the ISM. This visual specification of ISRs has been prototypically realized in the modeling tool *ISR modeler*[3] based on the principles introduced in Sect. 5.1. We export and validate these rules with the schema described in Sect. 5.2. Finally we apply Alg. 1 from Sect. 6 and create an ISM again. We visualize the created ISM with the ISC_Viz[4] modeling tool and compare the visual and XML representation with the original model.

7.1 Creating ISR visual models and XML representations

To show the applicability of the approach, four ISMs are visually specified as ISRs, one for each of the categories of the classification introduced by [4]. In a nutshell, the classification has two dimensions, i.e., requirement and context. Requirement comprises categories single (only one modeling perspective) and multi (multiple perspectives, e.g., data and time) the ISC can be built of. Dimension context refers to single if the ISC is defined for multiple instances of one processes and multi for ISCs spanning multiple processes. The examples depicted in Table 1 are picked from a set of ISC examples [15]. Our visual language is complete in a sense that we can model all 114 of these examples with our language. The corresponding ISMs have been created in our previous work [5]. These ISMs will help to verify the results of merging the created ISRs with Alg. 1. One has to bear in mind that multiple interpretations of the text are possible as the description is not precise. However Table 1 also comprises specification details of the examples to give more details on how we interpret the rule and what is special about this rule.

Figure 3 shows the visualization of the example ISRs. In the following we discuss how the visualization realizes the requirements set out in Section 4.

- Requirement 1 is shown in rule ② and specifies a data element as *kwp_connections*.
- Requirement 2 is visualized in rule ① and sets a link to a repository where the role *user* is selected.
- Requirement 3 is the linkage of a task which is represented in all rules by either selecting a repository ① ② or a link to a specific task of one process ③ ④. We want to point out that this represents the classification of context multi ① ② and single ③ ④.
- Requirement 4, execution data, is represented in rule ④ as *counter* that counts all instances.
- Requirement 5, specification of a trigger, is shown in every rule. Some rules do not have a condition e.g. ② Ⓒ, ④ Ⓐ.
- Requirement 6, actions, are visualized in nearly all examples.

[3] ISR visualization: `http://gruppe.wst.univie.ac.at/~gallm6/ISC_Viz/ISR/`
[4] ISM visualization: `http://gruppe.wst.univie.ac.at/~gallm6/ISC_Viz/ISM/`

Rule	Context	Requ.	ISC	Specification details
①	Multi	Multi	A user is not allowed to execute more than 100 tasks (of any workflow) in a day	For the cleanup ①Ⓒ we specify a rule without a task. This rule triggers every day even when no task is executed.
②	Multi	Single	Maximal KWP-2000 Connections The number of connections to KWP2000 should not exceed 10.	In this case the KWP (communication protocol) allows for a maximum of 10 concurrent connections. The example shows that a set of rules can consist of different trigger positions before and after a task.
③	Single	Multi	There should not exist more than one instance of W such that the input parameters (say loan customer) is the same and the loan amount sums up to $100K during a period of one month.	Each customer is allowed to have multiple loans but in total their loan amount shall not exceed $100K per month. This rule set shows a timer and that multiple data elements can be used within a trigger.
④	Single	Single	Wait until centrifuge is filled.	We set a limit that the centrifuge is filled when 4 instances arrive. In this case we perform the action *wait* for all instances before the task *centrifugation*. After finishing the task we use a trigger without condition to perform a cleanup i.e. change variable values.

Table 1. ISC examples with specification details

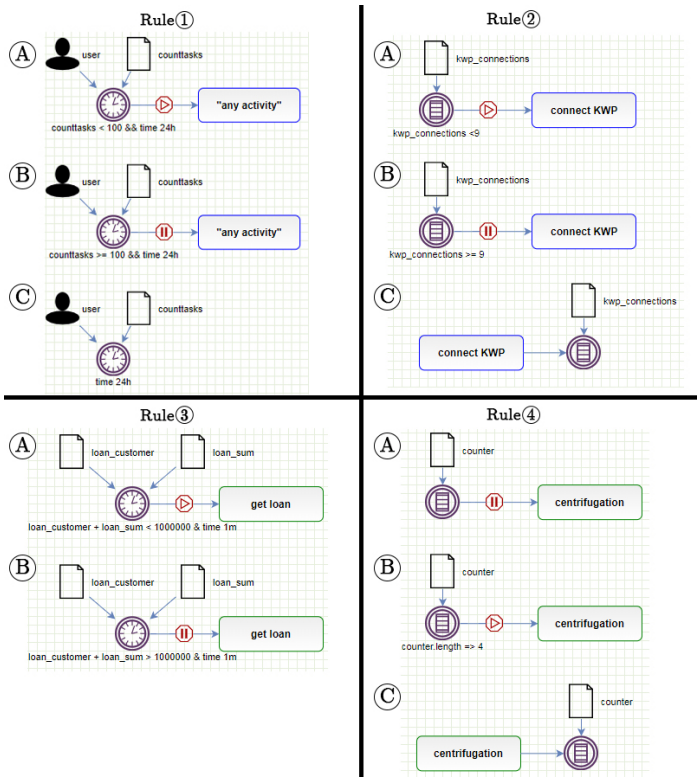

Fig. 3. Rules modeled with our extension to ISC_Viz.

– Requirement 7 and Requirement 8 cannot be seen in the visual representation, but rule numbers ①, ②, ③ and ④ could represent unique ISM IDs and the alpha numeric values Ⓐ, Ⓑ and Ⓒ represent rule IDs

Listing 1.4 shows an excerpt of the XML file exported from our tool ISC_Viz. The exported XML file is valid against the schema introduced in Section 5.2. The Listing specifies the centrifuge process from Table 1④. Every time three dots ... are shown within the listing there is some information skipped due to space limitations, this information is available on our website [5]. ISM ID (1) is followed by an ID (1) of the first rule ④Ⓐ. The rule is further specified with a priority of 11, no condition, a behaviour where the counter gets increased, and a single spanning task linked with an url and an id. Rule ④Ⓑ starts with an other ID (2), a different priority value of 7, a condition that checks if the counter is =>4, and behaviour that starts the first 4 instances if the condition is true. In the third rule ④Ⓑ the trigger is specified after the task without a condition and a behaviour that removes the instance from the counter. Our evaluation shows that we can visually model examples from each category of the classification and express them as XML.

Listing 1.4. Relax NG combination of all parts

```
...<id>1</id>              <!-- ISM ID -->
<rule><id>1</id> ...       <!-- Rule ID -->
<priority>11</priority> ...
  <condition></condition>
  <behaviour>counter &lt;&lt; event</behaviour> ...
<task>
  <spanning>single</spanning>
  <url>.../CentProcess.xml</url>
  <activityid>centrifugation</activityid>
</task>...
<id>2</id> ...              <!-- Rule ID -->
<priority>7</priority> ...
  <condition>counter.length =&gt; 4</condition>
  <behaviour>counter[0..3].each do |event      event.continue end</
      behaviour> ...
<id>3</id>                  <!-- Rule ID -->
<priority>3</priority> ...
  <task> ... </task> ...
  <condition></condition>
  <behaviour>counter.shift</behaviour> ...
```

7.2 Testing the algorithm

Algorithm 1 is applied to the example from Sect. 7.1 ④. The algorithm requires a priority value as input. The values in the XML file are *11*, *7*, and *3*. In the following, we decided to use the values *10*, *5* and *2* as input for the merge algorithm. The values are chosen to reflect all possible combinations. A value above 11 would result in an empty model as no rule in Listing 1.4 has a priority value of 11 or above. A value of *10* merges all rules with a value above 10, in

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

this case the first rule. A value of *5* merges rule 1 and 2, a value of *2* merges all rules.

Figure 4 shows the visualization of the ISM from example ④. Ⓐ shows the ISM taken from our previous work [5]. Based on the selected priority Algorithm 1 creates different ISMs. Figure 4 Ⓑ uses priority 10 and includes only Rule ④ Ⓐ the visual differences between Rule ④ Ⓐ and ISM Ⓑ are that the counter is now represented as *execution data*. When the priority is reduced to 5 an additional rule is included in the ISM. ISM Ⓒ, is identical when compared to the given ISM Ⓐ. If the counter is reduced to value below 3 all three rules will be incorporated into one ISM. Ⓓ depicts an ISM where two triggers are visualized. Due to space limitations we showcase all four XML files and merged models on our website[6].

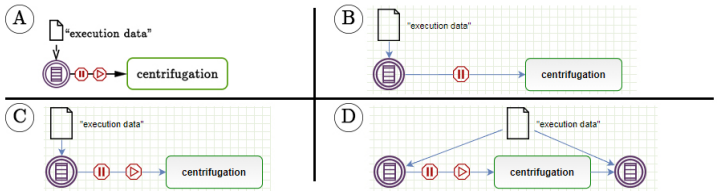

Fig. 4. ISM visualization from the files created by Algorithm 1.

ISMs created manually and ISMs created from a set of corresponding rules can differ in the number of elements depending on the priority set for Algorithm 1. Currently, Algorithm 1 creates ISM models that are complete in the sense that all visual elements will be used, meaning that all data elements, resources, trigger, and actions allowed within an ISM will be visualized.

8 Conclusion

Compliance management demands to bridge the gap between models of instance-spanning constraints (ISM) and their executable rules (ISRs). Visual representations of constraints are used for reflecting and discussing regulations, but can be specified in different ways for implementing constraints across multiple instances. This work provides a visual representation including a modeling tool for the specification of ISRs as well as the export and further specification of XML representations. Moreover, merging ISRs into an ISM is enabled. In future work, we will examine the automatic creation of ISMs based on ISRs, i.e., for creating ISMs without the need of unique IDs. Furthermore we want to evaluate how to visually model the behaviour part of a trigger.

Acknowledgment This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-072.

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

References

1. Awad, A., Weidlich, M., Weske, M.: Visually specifying compliance rules and explaining their violations for business processes. J. Vis. Lang. Comput. 22(1), 30–55 (2011)
2. Awad, A.: Bpmn-q: A language to query business processes. In: In Proceedings of EMISA07. pp. 115–128 (2007)
3. Becker, J., Ahrendt, C., Coners, A., Weiß, B., Winkelmann, A.: Business rule based extension of a semantic process modeling language for managing business process compliance in the financial sector. In: GI Jahrestagung (1). pp. 201–206 (2010)
4. Fdhila, W., Gall, M., Rinderle-Ma, S., Mangler, J., Indiono, C.: Classification and formalization of instance-spanning constraints in process-driven applications. In: International Conference on Business Process Management 2016 (June 2016)
5. Gall, M., Rinderle-Ma, S.: Visual modeling of instance-spanning constraints in process-aware information systems. pp. 597–595. Advanced Information Systems Engineering (May 2017)
6. Ghose, A., Koliadis, G.: Auditing business process compliance. In: Int'l Conf. on Service-Oriented Computing. pp. 169–180 (2007)
7. Heinlein, C.: Workflow and process synchronization with interaction expressions and graphs. In: Int'l Conference on Data Engineering. pp. 243–252 (2001)
8. Indiono, C., Mangler, J., Fdhila, W., Rinderle-Ma, S.: Rule-based runtime monitoring of instance-spanning constraints in process-aware information systems. In: On the Move to Meaningful Internet Systems. pp. 381–399 (2016)
9. Leitner, M., Mangler, J., Rinderle-Ma, S.: Definition and enactment of instance-spanning process constraints. In: Int'l Conf. on Web Information Systems Engineering. pp. 652–658 (2012)
10. 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. Information Systems 54, 209–234 (2015)
11. Ly, L., Rinderle-Ma, S., Dadam, P.: Design and verification of instantiable compliance rule graphs in process-aware information systems. In: Conference on Advanced Information Systems Engineering. pp. 9–23 (2010)
12. Mangler, J., Rinderle-Ma, S.: IUPC: identification and unification of process constraints. CoRR abs/1104.3609 (2011), `http://arxiv.org/abs/1104.3609`
13. Moody, D.: The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on Software Engineering 35(6), 756–779 (Nov 2009)
14. Pufahl, L., Herzberg, N., Meyer, A., Weske, M.: Flexible batch configuration in business processes based on events. In: Int'l Conference on Service-Oriented Computing. pp. 63–78 (2014)
15. Rinderle-Ma, S., Gall, M., Fdhila, W., Mangler, J., Indiono, C.: Collecting examples for instance-spanning constraints. Tech. Rep. abs/1603.01523, CoRR (2016)
16. Warner, J., Atluri, V.: Inter-instance authorization constraints for secure workflow management. In: Symp. on Access Control Models and Techn. pp. 190–199 (2006)
17. Wieringa, R.: Design Science Methodology for Information Systems and Software Engineering. Springer (2015)
18. Winter, K., Rinderle-Ma, S.: Discovering instance-spanning constraints from process execution logs based on classification techniques. In: 21st IEEE International Enterprise Distributed Object Computing Conference. pp. 79–88 (2017)