"The final publication is available at springerlink.com"
published at: 13th International Conference on Web Information Systems Engineering - WISE 2012, Springer, LNCS 7651, pp. 652-658 (2012)

Definition and Enactment of Instance-Spanning Process Constraints

Maria Leitner, Juergen Mangler, and Stefanie Rinderle-Ma

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

Abstract. Currently, many approaches address the enforcement and monitoring of constraints over business processes. However, main focus has been put on constraint verification for intra-instance process constraints so far, i.e., constraints that affect single instances. Existing approaches addressing instance-spanning constraints only consider certain scenarios. In other words, a holistic approach considering intra-instance, inter-instance, and inter-process constraints is still missing. This paper aims at closing this gap. First of all, we show how the Identification and Unification of Process Constraints (IUPC) compliance framework enables the definition of instance-spanning process constraints in a flexible and generic way. Their enactment and enforcement is demonstrated within a prototypical implementation based on a service-oriented architecture.

Keywords: Instance-spanning Process Constraints, Process-based Compliance Management, Process Engine, Web-based Business Processes

1 Introduction

Process constraints have become an important instrument to define, enact, and enforce regulations, standards, or other requirements that are imposed on business processes and workflows. Powerful approaches have arisen that enable modeling, monitoring, and verifying process constraints throughout the entire process life cycle e.g., [8, 4].

Current approaches typically focus on a particular topic, like (1) authorization, (2) security in general, (3) checking of structural requirements at design-time or (4) result-checking at run-time. Additionally, they typically either deal with **rule enactment** (cmp. [9]), covering process specific topics such as providing and monitoring rules in conjunction with process models, or with **rule enforcement** which deals with certain (process) tasks e.g., separation/binding of duties of tasks (e.g. [2]) or synchronization [5, 12]. All the above mentioned approaches provide *intra-instance constraints*. They are typically defined in a process schema and enforced in single instances. As stated in [2], these constraints can be enforced statically in the process schema and dynamically during process execution.

However, this specialization on particular topics (and related components in Process-Aware Information Systems (PAIS)) for single process instances, poses a problem when considering inter-instance, inter-process or inter-organizational constraints (further denoted as *instance-spanning*). *Inter-instance constraints* apply to multiple instances of a single process schema. Typically, instances of a process are enacted within an organization. *Inter-process constraints* are defined over single or multiple instances of multiple process schemas. *Inter-organizational constraints* are a special case of inter-process constraints; the enforcement of these constraints is managed over multiple organizations.

As most of the above mentioned approaches cover different formalizations, implementations (for specific components), and/or topics; the instance-spanning aspect has to be separately handled for every approach. Examples for existing parallel evolution of instance-spanning approaches for different topics include e.g., inter-instance authorization constraints [13, 14] or inter-process task synchronization constraints e.g., [3]. An additional problem is that solutions for inter-process constraints often cannot directly be transferred to instance-spanning scenarios, as there is no standardized way for describing:

1. *Process Scope*: Which set of processes or instances does a constraint refer to?
2. *Constraint Scope:* Which set of tasks does a constraint refer to? I.e., there is no standardized way to describe that a constraint covers e.g., tasks from all instances of a certain process, or tasks from particular instances from different processes.
3. *Enactment & Enforcement Aspects:* How to define certain basics of integrating constraints with processes. This includes (1) referring to processes structure which is defined by e.g., Linear Temporal Logic (LTL) [11] and Compliance Rule Graphs (CRG) [4], or (2) dealing with process data, time and resource identification.

The contribution of this paper is a comprehensive conceptual framework based on [7], for the specification of instance-spanning constraints. Instead of modifying specific approaches to make them instance-spanning aware, we want to introduce (1) a formalism, and (2) an architecture, how to enact and enforce instance-spanning constraints. We think that this can lead to unified enactment and enforcement of process constraints for PAIS, without the need of different infrastructures and/or components for different constraint topics. The semantic understanding of the enforcement has to be concentrated at enforcing components, while enactment can remain generic: checking conditions and forwarding the semantic part of constraints to enforcing components. Moreover, we illustrate our findings with examples for instance-spanning process constraints. In addition, we evaluate our findings with a proof-of-concept prototype for this architecture. In the following, Section 2 shows how instance-spanning constraints are specified in the IUPC framework. Furthermore, Section 3 displays the enactment of these constraints. Section 4 gives an overview on related work and Section 5 concludes the paper.

2 Design of Instance-spanning Process Constraints

As stated in the introduction, we will formalize instance-spanning constraints based on the Identification and Unification of Process Constraints (IUPC) frame-

work [7]. The purpose of the IUPC framework is to provide a means to integrate existing approaches that deal with various constraints topics (as explained in the introduction). Due to space limitations, we will only provide the extension of the main IUPC concepts (see [7] for a comprehensive definition), enriched with some examples how instance-spanning approaches fit in. We found instance-spanning constraints to typically have three dimensions - **Localization**, **Span** and **Dependency**:

Property 1 (Localization). If a constraint is connected to a task, this task can basically occur in a process, multiple processes, or processes in multiple organizations. Four different restriction scenarios are possible:

(1) A constraint should only be enacted for a certain instance (typically denoted as **intra-instance localization**). I.e.: $\forall a : a.\text{instance} = \text{CONST}$.
(2) A constraint is enacted for tasks in all instances of a process (typically denoted as **inter-instance localization**). I.e. $\forall a : a.\text{instance.process} = \text{CONST}$.
(3) A constraint is enacted for tasks in instances of many process (typically denoted as **inter-process localization**). I.e. $\forall a : a.\text{organization} = \text{CONST}$.
(4) A constraint is enacted for tasks in instances of processes that are present in more than one organization (typically denoted as **inter-organizational localization**). I.e. $\forall a : a.\text{organization} \neq \text{CONST}$.

Property 2 (Span). A constraint e.g., separation of duty, often affects multiple tasks at once. Five different scenarios are possible:

(5) All affected tasks are in the same instance (typically denoted as **intra-instance constraint**) e.g., $\forall a, b : a.\text{instance} = b.\text{instance} \wedge a.\text{organization} = b.\text{organization}$. As instances are unique, this definition should be sufficient.
(6) Affected tasks are spread over multiple instances of a process (typically denoted as **inter-instance constraint**). This is the typical case of inter-instance synchronization, as described in the related work e.g., $\forall a, b : a.\text{instance.process} = b.\text{instance.process} \wedge a.\text{organization} = b.\text{organization}$.
(7) Affected tasks are spread over multiple instances of multiple processes (typically denoted as **inter-process constraint**). E.g., $\forall a, b : a.\text{organization} = a.\text{organization}$.
(8) Affected tasks are spread over multiple instances of multiple processes of multiple organization (typically denoted as **inter-organization constraint**) e.g., $\forall a, b : a.\text{instance.process} \neq b.\text{instance.process} \wedge a.\text{organization} \neq b.\text{organization}$.
(9) Affected tasks are spread over multiple instances of single processes that exist in multiple organizations (typically denoted as **trans-organizational constraint**) e.g., $\forall a, b : a.\text{instance.process} = b.\text{instance.process} \wedge a.\text{organization} \neq b.\text{organization}$.

Property 3 (Dependency). This characteristic deals with the temporal aspect of constraint enactment. We define **Dependent** constraints to utilize behavior

tuples to realize e.g., Case Handling, Retain Familiar patterns (workflowpatterns.com) in conjunction with a worklist. For preceding enactments, a constraint may save a value e.g., $\forall a : a.\texttt{behavior\_data}[count\_invocations] += 1$; now it is possible to enact e.g., only every second time. We define **Independent** constraints as being independent of subsequent or preceding enactments.

To **integrate** these characteristics **into the IUPC framework** (cf. [7]), we utilize the following rules: Often, simple intra-instance constraints are defined in the `Linkage`, `Condition` and/or `Behavior`. For example, separation of duty constraints are defined in the `Condition` as comparison if two tasks are executed in the same instance such as $b.instance = c.instance$. In case of inter-instance constraints, the `Context` ($\mathcal{P} \times \mathcal{I}_P$) defines which process ($\mathcal{P} \in \mathcal{P}_n$) and which instances (i.e. $\mathcal{I}_P \subseteq \mathcal{I}$) are affected by the constraint. On the other hand, the `Context(`($\mathcal{P} \times \mathcal{I}_P$)`)` specification of inter-process constraints defines a set of processes and instances i.e. $\mathcal{P}_n \subseteq \mathcal{P}$ and $\mathcal{I}_P \subseteq \mathcal{I}$. As a motivational example shown in Fig. 1, we specify intra-instance (*C1*), inter-instance (*C2*) and inter-process (*C3*) constraints based on the IUPC framework in [7].

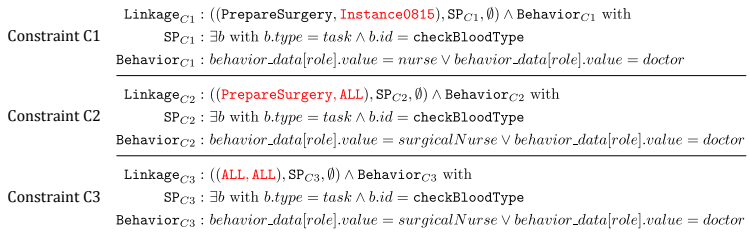


Fig. 1. Constraint Examples

3 Enactment of Instance-spanning Constraints

Architecture The architecture of our framework consists of a workflow execution engine (EE), worklist handler (WH), constraint engine (CE), users, external services, and data sources and is shown in Fig. 2. Depending on the type of task, manual or automated, the workflow execution engine delegates tasks either to a WH (manual task) or an external service (automated task). In case of manual tasks, the WH offers tasks to users which further can accept and execute them. On the other hand, automated tasks are delegated to an external service (e.g., a scheduler in scientific workflows) which further distributes tasks to data sources (e.g., nodes). Please note that the steps between a scheduler and a node are often not visible due to encapsulation. But to provide a comprehensive approach, these steps have to be considered for e.g., compliance checking. Throughout these two cases, a CE supports the enactment and monitoring of constraints during process execution.

Enforcement and Monitoring of Process Constraints For a better understanding, we will show how previously defined constraints (cf. constraint C1-3) are enforced

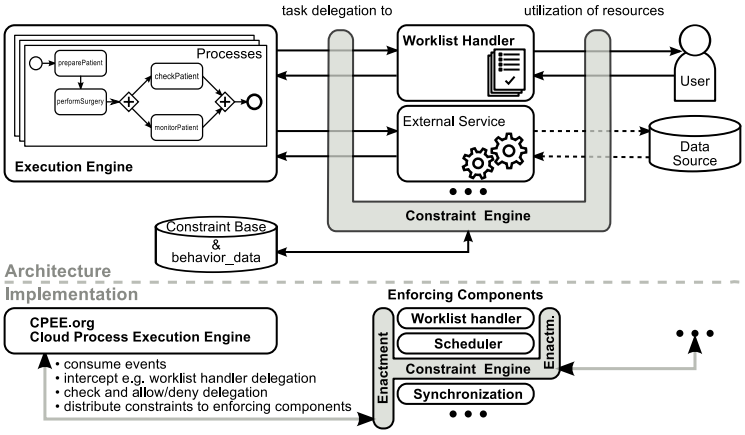
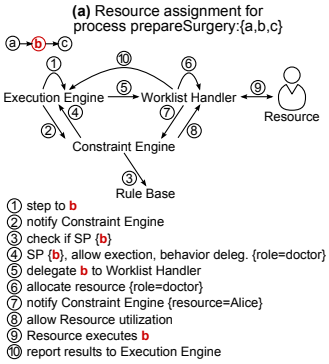


Fig. 2. Architecture of the Compliance Framework



(b) Common Attributes of Tasks and Data Elements

Attributes	Application design-time	Application run-time	Details
static task related			
id	X		e.g. prepareSurgery
type	X		type = task
input	X		list of inputs; e.g. patientID
output	X		list of outputs; e.g. duration, testResult
instance		X	e.g. InstanceChild
instance.process		X	e.g. TreatmentHeadTrauma
organization		X	e.g. Hospital, Laboratory
input[attribute].value		X	e.g. input[patientID].value=2645
output[attribute].value		X	e.g. output[testResult].value=false
static data-element related			
id	X		e.g. costs, age
type	X		type = data
instance		X	e.g. InstanceChild
instance.process		X	e.g. TreatmentHeadTrauma
organization		X	e.g. Hospital
value		X	e.g. age=5
(dynamic) behavior tuple			
behavior_data[id]		X	behavior directive; e.g. behavior_data[role] = doctor writable by constraint, readable by constraint and e.g. worklist
behavior_data[id].value		X	behavior value; e.g. behavior_data[role].value = Bob readable by constraint, writable by constraint and e.g. worklist

Fig. 3. Resource Assignment and Common Attributes

and monitored in the system. The function sequence to enact and monitor resource assignments (constraint **C1**) is shown in Fig. 3. Similar to constraint **C1**, this function sequence can be adapted to *all* instance-spanning constraints such as constraints C2 and C3.

Prototype In order to elaborate the prototype, we first define a set of common terms to be used. Processes contain tasks with a well defined input/ouput and data elements (i.e., variables modified at runtime) and are specified in our prototype. In Fig. 3, we give a set of common attributes that, as a precondition, have to be accessible in order to monitor or enact constraints. The static task and data element related attributes, are dealing with typical properties of tasks (`type`=*task*) and data elements (`type`=*data*) like `id`. We also assume that for each task it is possible to find out the `instance` it belongs to. Please note that `process` is a property of the `instance`, as every instance belongs to exactly one process. The attribute `organization` is independent of `instance` and `process` as one process is able to run in multiple organizations. This coordination of

processes in different organizations is a desirable side-effect of our approach. Furthermore, the dynamic behavior tuple realizes a shared space to coordinate the enactment of constraints. Constraints can (1) store values that are available for later enactments of constraints on the same task, and (2) share information with enforcing components such as worklists.

In Fig. 2 below the architecture, a short summary depicts the implementation of the prototype. For implementation, we rely on the service-oriented process testbed. As shown in the Fig., each component is a service e.g., the EE carries out tasks and the WH manages the assignment of tasks to users (i.e., manual tasks) or other services (i.e., automated tasks). As an EE we use CPEE (Cloud Process Execution Engine, http://cpee.org) as described in [10, 6]. This event based engine allows for a loosely coupled CE that only consumes events during the execution of instances. For the CE, we rely solely on the event type `running/syncing_before` which allows to delay the process execution.

By requiring only minimal event based interaction with enforcing components, we ensure that our approach can be easily integrated with other existing solutions. Typically, the EE and WH are tightly coupled in PAIS when specifying process constraints (e.g., authorization constraints). In our approach, EE and WH are independent from each other. Hence, a separated process model-related enactment and task-related enforcement of constraints is supported. execution engines in case of inter-organizational business processes.

4 Related Work

Mainly, research centers on intra-instance settings for constraints. First, most approaches for intra-instance constraints enable the definition and enforcement of constraints on structural patterns (e.g., [8]). Furthermore, how the scope can be extended towards data-aware process constraints is presented in e.g., [1]. Lastly, resource assignments such as roles (e.g., [2, 13]) are extensively investigated in literature. In case of inter-instance constraints, the workflow role-based access control model [13] defines *inter-case constraints* (e.g., the number of times activities are executed by single users) and *reciprocal separation of duties*. Moreover, the logic-based approach in [14] gives an overview on resource, data, and time inter-instance constraints. Since no generic solution is provided, inter-instance constraints are not comprehensively supported in [14] either.

In summary, the enforcement of constraints has been merely addressed by single approaches for a certain process scope (mostly intra-instance settings). In this paper, we specify an extensive set of instance-spanning constraints which is the first comprehensive approach managing multiple process scopes.

5 Conclusion

In this paper, we provide a comprehensive approach for instance-spanning constraints based on the IUPC framework. Furthermore, we provide an novel SOA-based architecture where a constraint engine is tightly coupled with an execution

engine. Moreover, we demonstrate our findings with a proof-of-concept prototype. In future work, we aim at investigating instance-spanning constraints further within inter-organizational settings. Moreover, the IUPC approach and its implementation within the CPEE process engine will be evaluated by means of case studies in different domains such as care and virtual factories within the EU FP7 project ADVENTURE (http://www.fp7-adventure.eu/).

Acknowledgements This work was partially supported by the Commission of the European Union within the ADVENTURE FP7-ICT project (Grant agreement no. 285220).

References

1. Awad, A., Weidlich, M., Weske, M.: Specification, verification and explanation of violation for data aware compliance rules. In: Service-Oriented Computing. LNCS, vol. 5900, pp. 500–515. Springer (2009)
2. Bertino, E., Ferrari, E., Atluri, V.: The specification and enforcement of authorization constraints in workflow management systems. ACM Trans. Inf. Syst. Secur. 2(1), 65–104 (1999)
3. Heinlein, C.: Synchronization of concurrent workflows using interaction expressions and coordination protocols. In: CoopIS/DOA/ODBASE. pp. 54–71. LNCS (2002)
4. Ly, T., Rinderle-Ma, S., Knuplesch, D., Dadam, P.: Monitoring business process compliance using compliance rule graphs. In: OTM 2011. LNCS, vol. 7044, pp. 82–99. Springer (2011)
5. Mangler, J., Rinderle-Ma, S.: Rule-Based synchronization of process activities. In: 13th Conf. on Commerce and Enterprise Computing. pp. 121–128. IEEE (2011)
6. Mangler, J., Stuermer, G., Schikuta, E.: Cloud process execution Engine-Evaluation of the core concepts. Arxiv preprint arXiv:1003.3330 (2010)
7. Rinderle-Ma, S., Mangler, J.: Integration of process constraints from heterogeneous sources in Process-Aware information systems. In: Int'l Workshop Enterprise Modelling and Information Systems Architectures - EMISA 2011. LNI, GI (2011)
8. Sadiq, S., Orlowska, M., Sadiq, W.: Specification and validation of process constraints for flexible workflows. Inf. Syst. 30(5), 349–378 (2005)
9. Specification, W.M.C.: Workflow management coalition terminology & glossary (Document no. WFMC-TC-1011. document Status-Issue 3.0). Tech. rep., Workflow Management Coalition Specification (Feb 1999)
10. Stuermer, G., Mangler, J., Schikuta, E.: Building a modular service oriented workflow engine. In: IEEE Int'l Conf on Service-Oriented Computing and Applications. pp. 1–4. IEEE (Jan 2009)
11. van der Aalst, W.M.P., Beer, H.d., Dongen, B.v.: Process mining and verification of properties: An approach based on temporal logic. In: Int'l OnTheMove Conferences. LNCS, vol. 3761, pp. 130–147. Springer (2005)
12. 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)
13. Wainer, J., Barthelmess, P., Kumar, A.: W-RBAC - a workflow security model incorporating controlled overriding of constraints. International Journal of Collaborative Information Systems 12(4), 455–485 (2003)
14. Warner, J., Atluri, V.: Inter-instance authorization constraints for secure workflow management. In: Proc. of the 11th ACM symposium on Access control models and technologies. pp. 190–199. ACM (2006)