

On Correctness, Compliance, and Consistency of Process Models

René Würzberger and Thomas Kurpick and Thomas Heer
 RWTH Aachen University
 Department of Computer Science 3 (Software Engineering)
 D-52074 Aachen, Germany
 {woerzberger, kurpick, heer}@i3.informatik.rwth-aachen.de

Abstract

Process management incorporates a plethora of models, which are expressed in different languages for different layers of abstraction. A holistic process modeling environment must provide means for dealing with three types of modeling constraints. First, it must ensure a model's correctness. Second, a modeling environment must account for compliance of models with respect to models on higher layers of abstraction. Third, consistency of different models of the same language layer should be supported. In this paper, we exemplify these three constraint types and discuss how they can be enforced in a holistic modeling environment.

1 Constraint types for process models

Modeling support is crucial for the acceptance of a process-aware information system (PAIS). We distinguish between *instance models* for running processes, *executable models* like WS-BPEL and *non-executable models* for general process knowledge. Fig. 1 depicts which orthogonal types of explicit and implicit *model constraints* have to be considered, namely correctness, compliance, and consistency.

1.1 Correctness

The most rigid constraint for a single process model is its *correctness*. Process models that are incorrect cause problems, in particular if they are processed in a PAIS runtime environment. Correctness comprises several nuances known from programming language theory like syntactical and semantical correctness, which cannot be discussed here due to space limitations.

In Fig. 1, the executable model *ep3* is incorrect since the data flow from H to P points to the opposite direction of the corresponding control flow. The same sort of incorrectness

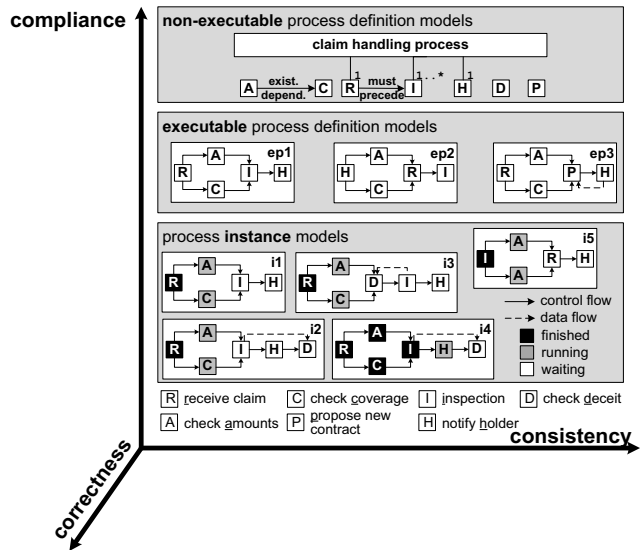


Figure 1. Constraint types

can be found in instance model *i3*. Instance model *i2* is correct whereas *i4* becomes incorrect, if activity D is added to the instance model after activity I has already been finished.

1.2 Compliance

Models on higher layers of abstraction often constitute a *compliance* authority to lower models. Generic process constraints, which cannot be appropriately expressed in executable models can be *explicitly* specified in non-executable models. The non-executable model of Fig. 1, which is expressed in a simple hypothetic language, states five *constraints*: (1) Every claim handling process must have exactly one activity R, (2) exactly one activity H and (3) at least one activity I. (4) If activity A is part of a claim handling process, also C has to be carried out and vice versa. (5) R must (indirectly) precede I in the control flow.

Compliance to non-executable models concerns instance

	<i>correctness</i>	<i>compliance</i>	<i>consistency</i>
<i>constraint violation</i>	unacceptable	acceptable	acceptable
<i>check invocation</i>	automatically or manual	manual	manual
<i>tool behavior</i>	exclude operations or display errors	display warnings	display notifications

Table 1. Properties of and requirements for a holistic modeling tool

and executable models. Changes in *instance* models might contradict constraints specified in *non-executable* models. For example, instance model i5 violates constraints (4) and (5). There is also a compliance relationship between *executable* models and *non-executable* models. The executable model ep3 has no activity l, which contradicts constraint 3.

Compliance to executable models affects instance models. While most PAIS technically disallow any structural deviations of an *instance* model from the *executable* model, from its corresponding executable model, flexible PAIS explicitly allow for or even call for such deviation [3, 4]. In Fig. 1, instance models do not comply to any executable model but i1, which complies to ep1.

1.3 Consistency

Consistency refers to constraint relationships between models *within* a single language layer. Usually, most companies maintain an abundance of executable models that embody *implicit* knowledge, e.g., implicit dependencies between activities. Thus, a user surely likes not only to be informed about incorrectnesses and incompliance but also about possible inconsistencies of his model with respect to other models on the same layer.

Executable models ep1 and ep3 have in common that activity H succeeds activity R in the control flow. This is reasonable with regard to the meanings of H and R (cf. Fig.1). However, an user might model the contrary in ep2. Then, this would be correct and compliant to the non-executable model but inconsistent with the other two executable models.

2 Towards a holistic modeling tool

Currently, we are implementing a holistic modeling tool for models on different abstraction layers. Table 1 summarizes some of the key properties and requirements for this tool with regard to the different types of constraints. Violation of correctness generally leads to technical problems and is therefore *unacceptable* whereas compliance and consistency violations may be intended. Thus, the tool should prevent those operation, which violate correctness if time complexity of the corresponding check is sufficiently low to be *automatically* invoked during an editing operation. Other violations are temporarily allowed but detected dur-

ing a complete check run, which the user has to invoke manually. The results of that check are classified according to the severity of the violation, e.g., correctness violation is an error whereas inconsistencies just yield notifications.

Presently, our modeling tool supports the editing of graphical models. Some correctness and compliance checks have been implemented using diverse frameworks of the Eclipse project, e.g., for meta modeling and for interpreting constraints formulated in the standardized Object Constraint Language (OCL). Consistency is not yet treated.

3 Related work and conclusion

There are several approaches that deal with certain types of constraints on process models. *Consistency*: Reichert uses a graph based calculus with certain optimizations in order to detect errors in editable instance models [3]. *Compliance*: Schleicher proposes the usage of UML class diagrams as a language for non-executable process definition models within the AHEAD process management system [4]. Ly et. al. suggest a first order logic formalism with a textual language as foundation for (non-executable) “semantic constraints” [2]. *Consistency*: Studying relationships between process models of the same language is the objective of several research activities, e.g., the formalization of behavioral *equivalence* of processes models [1].

Detecting or preventing constraint violations is a crucial function in a modeling environment. To our best knowledge there is no approach that aims at consistency between process models in the same layer in our sense. Furthermore, there is no holistic approach for a modeling environment that accounts for all three types of constraints.

References

- [1] Hidders, Dumas, v. d. Aalst, ter Hofstede, and Verelst. When are two workflows the same? In *CATS '05: Proc. of the 2005 Australasian symposium on Theory of computing*, 2005.
- [2] L. T. Ly, S. Rinderle, and P. Dadam. Integration and verification of semantic constraints in adaptive process management systems. *Data Knowl. Eng.*, 64:3–23, 2008.
- [3] M. Reichert. *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. PhD thesis, Univ. Ulm, 2000.
- [4] A. Schleicher. *Management of Development Processes: An Evolutionary Approach*. PhD thesis, RWTH Aachen University, 2002.