

Checking Correctness and Compliance of Integrated 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

Models of different kinds are used in the area of business process management. Abstract process knowledge as well as executable process definitions can be visualized and edited in a graphical manner. The same holds true for models of process instances in some process-aware information systems (PAIS), which allow for dynamic modifications in a process instance during process runtime. An appropriate modeling tool must not only provide means to graphically edit different kinds of models on different abstraction layers but also detect or prevent violations of certain model constraints. These constraints comprise a model's internal correctness as well as a model's compliance with general process knowledge expressed in another more abstract model. In this paper we contribute an approach that allows for uniformly specifying correctness as well as compliance checks for diverse graphical process models. This is achieved by means of an integrated meta-model for models of different kinds. The integrated meta-model is referenced by a fixed set of Object Constraint Language (OCL) expressions, which specify correctness and compliance checks. We exemplify some OCL expressions and their application within a prototypical modeling tool.

1. Introduction

Models of manifold kinds are widely used in the area of business process management. Some kinds of models are just used for communication between humans whereas others are processed by process-aware information systems (PAIS) [1] in order to support real world processes.

In order to be useful, models have to follow certain *constraints*. Particularly models which are processed by some PAIS must strictly adhere to *correctness* constraints. For instance, executable process definition models have to be *syntactically* correct. Otherwise, they cannot be interpreted at all by a PAIS. Regarding executable process definition models, correctness also refers to the dynamic *semantics* of the respective model. This includes absence of possible deadlocks or reading invalid data. Furthermore, models of a certain kind have to follow *compliance* constraints, which stem from other models that capture legal or in-company

regulations. For instance, in a claim handling process verifying completeness of amounts is reasonable only if the coverage of a claim is checked, too.

In this paper we depict how checks for correctness and compliance can be formalized such that they can be directly applied in a modeling tool. The *structure* of this paper is as follows: In Section 2 we show how processes can be modeled on different layers of abstraction. At this, we exemplify how imprecise process knowledge can be formulated using graphical models. Different types of constraints imposed on models are discussed in Section 3. In Section 4 we formalize the abstract syntax of the models via an integrated meta-model, which takes relationships between different model kinds into account. Section 5 describes how correctness and compliance checks can be declaratively specified in OCL. We give some implementation details of our prototypical modeling tool in Section 6. Related works are discussed in Section 7 and a conclusion is given in Section 8.

2. Process Model Layers

In this section we discuss different kinds of process models, which are associated with different layers of abstraction. Two of these model kinds are exemplified in Figure 1, which depicts a screenshot of our modeling tool.

2.1. Business Process Compliance

As stated before, processes usually need to comply with legal and in-company regulations. These regulations and other types of process knowledge must be externalized and defined in an *abstract* way in order to be applicable for many processes. We advocate the usage of graphical *process compliance models* for modeling such information. For this purpose, we have developed a simple diagrammatic language named Business Process Compliance Language (BPCL). BPCL models are still *precise* yet deliberately too abstract for being executable. Within a BPCL model, a domain expert can graphically specify certain constraints which processes have to comply with.

The upper model of Figure 1 exemplifies BPCL. This model incorporates common compliance constraints for claim handling processes. The gray triangles denote *activity*

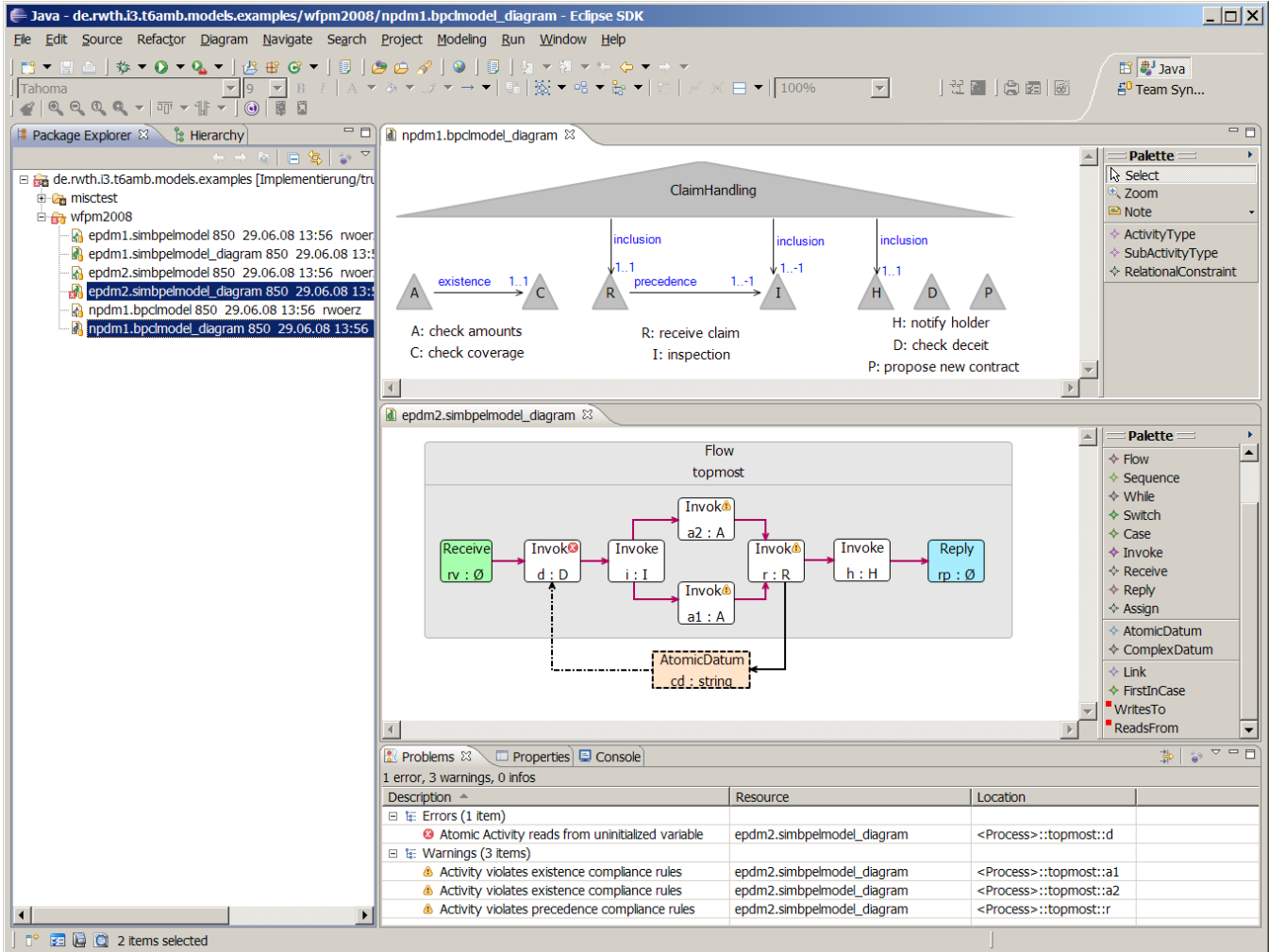


Figure 1. Modeling tool with opened BPCL (top) and SimBPEL model (middle)

types, the black arrows stand for certain constraints of activity types.

For better readability, we *abbreviate* the names of activities and activity types for the rest of the paper. The *meanings* of the abbreviations can be found below the BPCL model.

The BPCL model states five compliance constraints for claim handling processes, which are grouped into three *compliance types*:

- *inclusion*: A claim handling process requires the inclusion of exactly one (1..1) activity of type R as well as exactly one of type H. Furthermore, the inclusion of at least one activity of type I is demanded. This is modeled by the multiplicities 1..-1, wherein the upper bound -1 denotes that there is no upper bound.
- *existence*: The existence of an activity of type A implies the existence of an activity of type C.
- *precedence*: The existence of an activity of type R asks for at least one activity of type I. Furthermore, the

BPCL model requires that activities of type R directly or indirectly precede those activities of type I.

2.2. Simplified WS-BPEL

Actual process cases can be classified according to *process types*, e.g., “claim handling for damages to buildings” in an insurance company. In order to support actual processes a process modeler creates *executable process definition models* in a language like WS-BPEL [2]. In the following we use a simplified variant of WS-BPEL named *Simplified WS-BPEL (SimBPEL)*.

The middle of Figure 1 depicts an exemplary SimBPEL model reflecting a simplified claim handling process. The rounded boxes denote *activities* (of WS-BPEL type receive, invoke, and reply). They are nested in a flow and connected by links, which define the *control flow*. We also explicitly define *data flow* in SimBPEL models. In the example, activity r writes (drawn through arrow) to a string cd that

is read (dashed arrow) by activity d. Furthermore, activities can and should be typed in SimBPEL with activity types of a BPCL model. For instance, a1 is of type A (check amount).


2.3. Simplified WS-BPEL Instances

Besides abstract process compliance constraints and process types also actual *process cases* need to be modeled. To this end we use SimBPEL-Instance models. These models are derived from SimBPEL models inasmuch they also comprise the activity and data structure of SimBPEL models but also add *state information* to each activity. Due to space limits, we do not provide an example for SimBPEL-Instance.

3. Process Model Constraints

Process models have to adhere to certain constraints to meet certain quality goals [3]. These constraints can be further subdivided into constraints for *correctness* of process models and *compliance* constraints between process models of different kind.


3.1. Correctness of Process Models

Only correct process models can be properly processed by a PAIS. Regarding WS-BPEL and SimBPEL, correctness includes simple *syntax* rules, e.g., links must have exactly one source and one target. There are also more complex syntax rules, e.g., links may not form a cycle. Furthermore, there are constraints pertaining the *static semantics* of the modeled process, e.g., a variable has to be written (initialized) at least once by some activity before it can be read by another one. This correctness constraint is particularly violated by the SimBPEL model in the middle of Figure 1. Here, d reads from cd before cd is written by r. The correctness violation is detected by the modeling tool. Consequently, it places an error marker  on the activity d and adds an error entry to the problems list.

Correctness constraints can be verified or falsified just by checking one single model. Hence, correctness constraints can be regarded as *intra-model constraints*.

3.2. Compliance of Process Models

As stated before there are different kinds of process models for different abstractions of processes. It is obvious that *process models* on different abstraction layers may *impose constraints* on each other. For example, a precedence relationship from one activity type R to another type I demands for all claim handling processes that activities of type R directly or indirectly precede activities of I. In Figure 1 the SimBPEL model violates several compliance constraints of the BPCL model. First, activity r of type R does not precede any activity of type I. Second, for activity

a1 of type A there is no corresponding activity of type C in the SimBPEL model. The same applies for a2. The modeling tool places a warning marker  onto each activity, which is reason of a compliance violation and also adds corresponding entries to the problems list to the warnings section.

Compliance can only be checked regarding two models, in contrast to correctness. Hence, we also call compliance of process models *inter-model constraints*.

4. Meta-Modeling Integrated Process Models

In order to automatically check constraints of process models, we need a *formal specification* of a process model's *abstract syntax*. For this purpose, we use the meta-modeling language *Ecore* [4] in order to define meta-models which strongly resemble class diagrams of the Unified Modeling Language (UML). These meta-models define the *constituents* (e.g., invoke or flow activities) as well as the *structure* (e.g., the possible nesting of invoke into flow activities) of process models.

Figure 2 depicts three meta-models defining the abstract syntax for each model kind described in Section 2. For brevity, we left out some meta-classes, which are not relevant in the context of this paper. Note that in the following the term “model (element) instance” refers to an instance of the respective meta-model (element) but not to the technical instantiation of a process instance according to a process definition in a PAIS.

4.1. Meta-Models for each Model Kind

The upper layer of Figure 2 depicts the meta-model for BPCL. Its essential *meta-classes* are *ActivityType* and *RelCon* (relationship constraint). The top of Figure 1 displays a model instance of this meta-model with a certain *concrete syntax*. Here, the gray triangles are model element instances of *ActivityType* and the arrows are model element instances of *RelCon*. The latter carries *meta-attributes* *conKind* of type *RelConKind*, whose values are reflected in a BPCL model instance as labels (inclusion, precedence etc.). The meta-attributes *lowerBound* and *upperBound* correspond to *multiplicities* of the shape *n..m* at the end of arrows in a BPCL model instance. Activity types that are source (target) of some *RelCon* reference this *RelCon* via *meta-reference* *fromSource* (*fromTarget*). Vice versa, an *RelCon* references its source (target) via meta-references *source* (*target*).

The middle layer of Figure 2 contains a meta-model for *SimBPEL* models. Basically, the meta-model defines possible *composition hierarchies* of WS-BPEL activities as well as the *relationship* of WS-BPEL links, activities and process data (WS-BPEL variables). For the time being, we

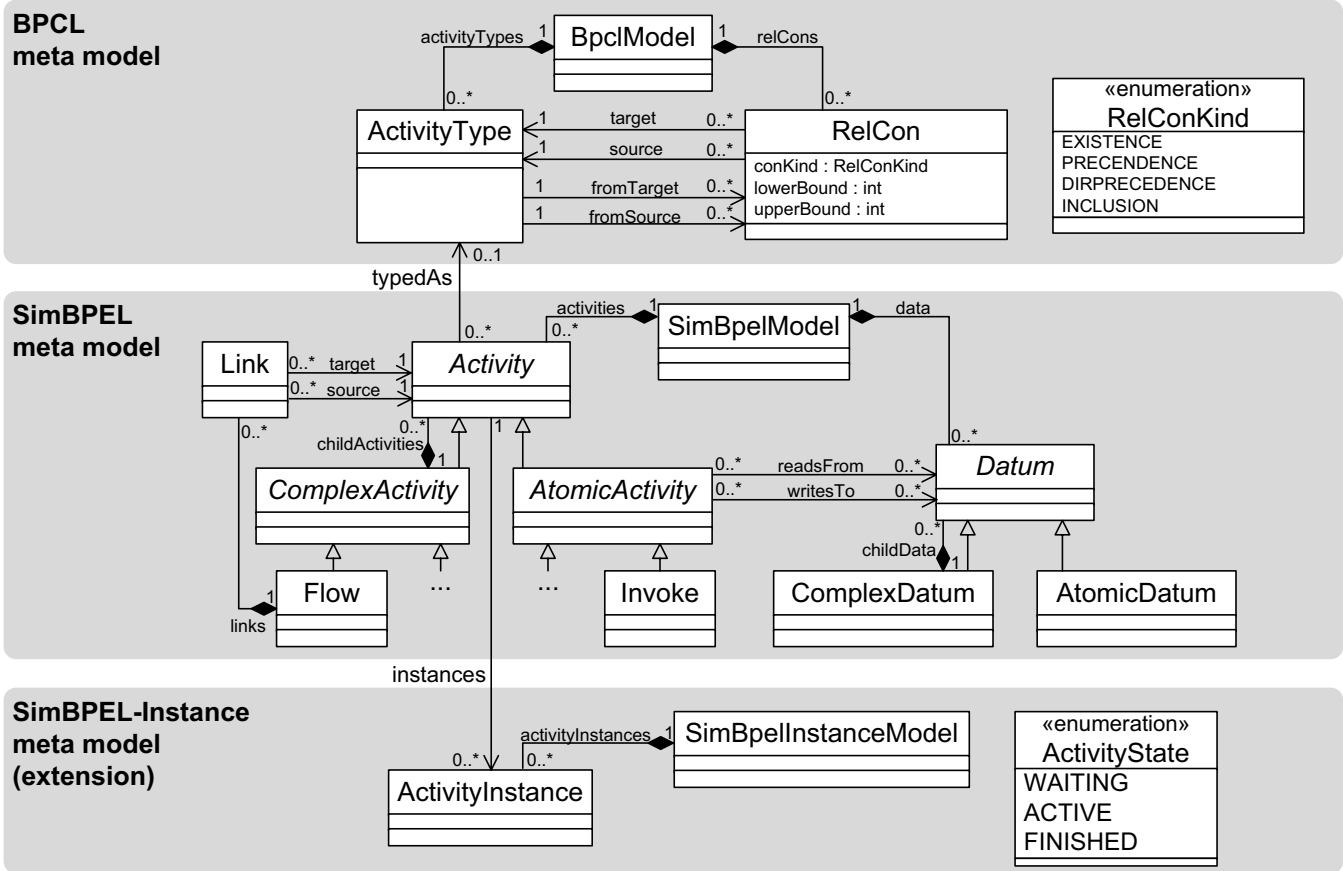


Figure 2. Integrated meta-models

deliberately leave out some advanced concepts of WS-BPEL like compensation handlers in SimBPEL.

SimBPEL-Instance models just extend SimBPEL models (cf. bottom of Figure 2). An **ActivityInstance** represents one *activation cycle* of an activity. Possible states are enumerated by **ActivityState**.

4.2. Integrated Meta-Model

As stated in Section 3.2, process models on different abstraction layers are *related* to each other. In our meta-models we account for this by meta-model *crossing meta-references*, which join the three meta-models to an *integrated meta-model*.

In a SimBPEL-Instance model an **Activity** associates a set of **ActivityInstances** via the meta-reference **instances**. Since an activity can be part of a case or while loop, it can be visited arbitrarily often. Thus, the multiplicity at the **ActivityInstance**-end is 0..*.

Activities in SimBPEL models can be *typed* with an **ActivityType** via meta-reference **typedAs**. This typing constitutes the connection between SimBPEL models and BPCL models.

5. Constraints in OCL

So far we have seen how to model the abstract syntax for different kinds of process models as well as how to integrate them with each other. Unfortunately, these meta-models are actually *underspecified*, i.e., there are model instances which are valid model instances of the respective meta-model but are nonetheless incorrect. The same applies for the integrated meta-model that does not itself ensure compliance of SimBPEL(-Instance) models with BPCL models.

This underspecification stems from the *insufficient expressiveness* of the meta-modeling language Ecore. Therefore, we utilize the standardized *Object Constraint Language (OCL)* [5] to formulate a *fixed set of OCL expressions*, which remedy this shortcoming. Briefly, OCL is a textual language that can be used to *refine meta-models*, which are expressed in, e.g., Ecore or UML. Using OCL one can particularly rule out unwanted model instances. The *semantics* of OCL is based on *First Order Logic (FOL)*, e.g., OCL provides quantifiers. However the *syntax* is borrowed from *programming languages*.

One *subset* of the *OCL expressions* refines the specification of the meta-models by complex *correctness criteria*.

```

context AtomicActivity
inv variablesInitialized:
  Datum.allInstances () ->forAll (d|self.readsFrom->includes (d)
  implies AtomicActivity.allInstances ()->exists (a|
  a.writesTo->includes (d) and a.allSuccs->includes (self) ) )

```

Listing 1. OCL-invariant for checking a correctness constraint

Another subset specifies *compliance relationships* between different models. These expressions particularly make use of the meta-model crossing meta-references. Thus, they formally relate the process knowledge in BPCL models to SimBPEL(-Instance) models.

5.1. Correctness Checks in OCL

In the following we *exemplify* our usage of OCL for correctness checks with a simple correctness constraint for SimBPEL models. Informally, we can delineate our example correctness constraint as follows: “Process data has to be written (initialized) before being read”. This invariant is well known from compiler theory but also applies to executable process definition models. It cannot be directly expressed in the SimBPEL meta-model but in the OCL invariant of Listing 1.

In OCL an *invariant* is introduced by the keyword `inv` and always evaluates to a boolean value. Furthermore, an invariant is declared within the `context` of an element of the meta-model, which is `AtomicActivity` in our case. The keyword `self` refers to the atomic activity to which the invariant is applied.

In our example `variablesInitialized` is true regarding a particular atomic activity (`self`) if and only if for all datum instances `d` of the respective model the following holds true: If `self` reads from datum `d`, then there is an atomic activity `a` that writes to `d` and precedes `self` in the control flow definition. The precedence is inversely checked via the OCL operation `allSuccs`. This operation returns all direct and indirect control flow successors of an activity. We omitted the definition of this lengthy operation due to space limits.

5.2. Compliance Checks in OCL

In the previous subsection we have exemplified how to define intra-model correctness constraints for SimBPEL models using OCL. Since we have an integrated meta-model, *model instances* of different meta-models can *refer to each other*, e.g., activities in SimBPEL models reference activity types in BPCL models. Thus, model instances are integrated as well so that we can use OCL also for *checking compliance* of SimBPEL models against compliance constraints defined in BPCL models.

Listing 2 demonstrates how the *precedence compliance* constraint is expressed in OCL. In the context of `ActivityType` we define an operation `allAdjPrec`, which returns all adjacent model element instances of `RelCon` that have the activity type as source and are of kind `PRECEDENCE`. If we would apply this operation to activity type `R` in the BPCL model of Figure 1, it would yield a set just containing the element, which corresponds to the outgoing *precedence*-arrow. Furthermore, we defined the OCL operation `matchBounds` that receives an integer parameter `c` and returns true if and only if `c` is within the bounds specified by `self`, which is a `RelCon`.

Both operations are applied in the definition of the invariant `invCompliesPrecedence` for activities. In this invariant we navigate from an activity `self` (e.g. `r`) to its activity type via `typedAs`. Here we apply `allAdjPrec` and demand for each resulting element `rcl` that `self` has (indirect) control flow successor activities (`allSuccs`) of the right type (e.g. `l`) and right amount (e.g. `1..-1`). This invariant is false for activity `r` in the SimBPEL model of Figure 1.

6. Implementation of the Modeling Tool

Our prototypical modeling tool has been rapidly developed using technologies of the Eclipse project, which we detail in [6]. In particular, we modeled the *abstract syntax* of our different model kinds with Ecore meta-models using the *Eclipse Modeling Framework (EMF)* [4]. These meta-models were merged with a graphical *concrete syntax* specification using the *Graphical Modeling Framework (GMF)*. From that, we generate a set of Eclipse Plugins, which implement our modeling tool. At runtime of the modeling tool, the *OCL constraints* are evaluated using the *Eclipse Modeling Tools (MDT)*. By means of these frameworks, we could *minimize* the amount of hand written *Java code* to some initializing statements.

Since the modeling tool is itself hosted within an Eclipse environment at runtime, we can also use standard Eclipse Plugins like *Subclipse* in order to provide basic *versioning* support for our process models.

7. Related Work

There are plenty of works dealing with constraints for process models. Below, we distinguish these works according to

```

context ActivityType
def: allAdjPrec: Set (RelCon) =
    self.fromSource->select (rcl|rcl.conKind = RelConKind::PRECEDENCE)->asSet ()

context RelCon
def: matchBounds (c: Integer): Boolean =
    self.lowerBound <= c and self.upperBound >= 0 implies self.upperBound >= c

context Activity
inv invCompliesPrecedence:
    self.typedAs.allAdjPrec->forall (rcl|rcl.matchBounds (
    self.allSuccs->select (act|act.typedAs = rcl.target)->size () ))

```

Listing 2. OCL-invariants for checking a compliance constraint

our classification of constraints, i.e., intra-model correctness and inter-model compliance. In some works, correctness of process models is called *soundness*. Compliance is also termed *conformance* by some authors.

Correctness. In [7] Verbeeck et al. describe a workflow analyzer named *Woflan* for checking *correctness* of process definitions at process *design time*. This approach covers sophisticated correctness constraints like the absence of potential deadlocks in a process definition. For the time being, correctness constraints of such complexity are not supported by our approach. *Woflan*'s formal foundations are based on the Petri-net class *WF Nets*. Thus, for every process definition language *Woflan* requires a *transformer* that translates to *WF Nets*. This contrasts our approach, where *adapting* to different process definition languages can be realized by adapting the OCL constraints to the respective meta-model instead of translating the process definition models.

Mendling et al. also deal with *correctness* of process models at design time [8]. They provide formal syntax and semantics definitions for unrestricted *Event-driven Process Chains (EPCs)*, which resemble those of Petri-nets. Moreover, they use *graph rewritings* to verify the correctness of EPCs, which particularly might have more than just one start element. Though we use OCL for implementing correctness checks, graph rewriting is surely also a powerful tool in this context. This is particularly true since there are many sophisticated languages and development environments for graph rewriting systems, like *PROGRES*, *Fujaba* or *AGG*, for which [9] provides a comparison.

In [10] Reichert describes a *calculus* that is used to preserve *correctness* in modifiable process instances modeled in a non-standard language (Kontrollflussgraphen) at *process runtime*. Reichert particularly focuses on computational *efficiency*, which we have neglected so far. Apparently, there are no further research efforts transferring his results to other process languages.

Compliance. We follow up *preceding works* [11] of our

group, which have been done in the context of the collaborative research center 476 *IMPROVE* [12]. Krapp et al. developed a *layered system* of graphical process modeling languages. The language *DYNAMITE* [13] particularly suits modeling process instances of *highly dynamic development processes* and the language *MADAM* [14] can be used to define *compliance* constraints for *DYNAMITE* models. Schleicher et al. *adapted* *MADAM* to the standardized *Meta Object Facility (MOF)* [15] using stereotyped class and collaboration diagrams for modeling compliance constraints [16], [17]. *Our work* continues the basic ideas of the preceding work but *differs* in several regards: First, we *abandoned* the hitherto utilized graph oriented programming language and environment *PROGRES* [18]. This particularly eliminates the necessity for recompilation of the modeling environment after changes in the compliance constraints. Changes of compliance constraints in *BPCL* models become immediately effective. Second, we focus on *business processes* of insurance companies instead of development processes in chemical engineering like our predecessors. This particularly requires new types of constraints like indirect precedence or existence dependency. Third, we take *standards* like *WS-BPEL* into account. The languages of preceding work were fully designed from scratch.

The *DECLARE* project [19] provides a *purely declarative*, extensible and graphical language similar to our *BPCL*. However, the usage of *DECLARE*-models is different. *DECLARE*-models are *interpreted* in a dedicated process engine in order to compute the set of currently executable activities. *DECLARE*-processes can invoke or be invoked from imperatively defined processes, e.g. *YAWL*-processes [20]. In contrast, we use *BPCL* models for *checks* of imperative models. The semantics of *DECLARE* elements is explicitly defined in *Linear Temporal Logic (LTL)* whereas we use OCL.

In [21] Andersson et al. discuss a *declarative foundation* of process models. They also provide a *layered system* of process model kinds ranging from abstract but comprehensi-

ble “business models” to detailed but complicated “process models”. The model kind in the *middle layer* (“activity dependency models”) resembles our BPCL. However, the authors are rather concerned with *top-down mappings* towards detailed model kinds instead of *bottom-up compliance checks*.

Ly et al. also consider “compliance and validation support in each phase of the process lifecycle” an important issue [22]. They provide interesting *requirements* for ensuring process compliance, which are not yet covered by our models, e.g., “context information” like drug incompatibilities. However, prototypical implementation of these requirements in their framework seems to be ongoing work.

Ghose et al. provide a *conceptual framework* and heuristics for minimizing and automating *revisions* of noncompliant BPMN processes models [23]. Since their approach targets BPMN, which is rather a rich and high level notation without strict execution semantics, they have to *annotate BPMN models* to deliver semantics in addition.

Goedertier et al. point out that “*sequence and timing constraints* [...] are an important aspect of compliance” [24]. They developed a language called PENELOPE based on *temporal deontic assignments* in order to formalize compliance rules. In contrast to our approach these rules are not used to check existing process models but to *generate* compliant process models.

Rozinat et al. address the *compliance relationship* between a *process log* of a completed process and the corresponding *process definition* model [25]. In terms of our work this would amount to check process instance models against executable process definition models, which we have not dealt with so far.

Lu et. al propose a methodology for *measuring* design time *compliance* of process definition models to compliance rules, which are expressed in a formal logic language [26]. This language is *textual* in contrast to our BPCL.

After all and to our best knowledge there is no other approach that *uniformly* addressed both *correctness and compliance*. We are aware of the fact that some related works address even more sophisticated constraints on process models compared to those presented in this paper. However, we consider our work an approach to *rapidly realize tool support* for editing correctness and compliance constrained process models in a unified way based on formal meta-models and OCL expressions.

8. Conclusion and Future Work

In this paper we have presented an approach that allows for specifying *different checks* of process model constraints in a *unified manner* using *integrated meta-models*, which are supplemented with *fixed set* of additional *OCL constraints*. This approach is aligned with existing *standards* and *programming frameworks* of the Eclipse project. Thus, we were

able to *rapidly develop* a prototypical modeling tool, which particularly provides a dedicated notation for each process model kind and detects constraint violations. Nonetheless, there is still *work to do*:

SimBPEL is just a *simplified* version of WS-BPEL. We have developed a *transformer* that translates WS-BPEL models to SimBPEL models. Eventually, we want to *redundantize* this transformer by providing support for *real* WS-BPEL models instead of SimBPEL models.

In a past work, we have extended a commercial process management system such that it provides modification operations on running process instances [27]. Thus, we want to *concentrate on* editing support for process *instance* models, i.e., SimBPEL-Instance models. According to our integrated meta-model, a SimBPEL-Instance model is just a SimBPEL model augmented by activity instances, which are associated to their respective activity. Checking compliance of SimBPEL-Instances models requires a *revision* of the OCL constraints. The revision has to consider numbers of present and possible future activity instances. This is not trivial if the SimBPEL-Instance model contains decisions and loops.

For the time being, a weak point in our approach is clearly the *need to type* activities before they can be considered in compliance checks. Eventually, we want to replace this manual typing by an *automated* mechanism that infers the type of an activity from available information, e.g., the activity’s interface from the WS-BPEL definition.

References

- [1] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, Eds., *Process-Aware Information Systems*. John Wiley & Sons, 2005.
- [2] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C. K. Liu, R. Khalaf, D. Knig, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu, “Web services business process execution language v2.0,” Organization for the Advancement of Structured Information Standards (OASIS), Tech. Rep., 2007.
- [3] J. Krogstie, G. Sindre, and H. Jorgensen, “Process models representing knowledge for action: a revised quality framework,” *Eur. J. Inf. Syst.*, vol. 15, no. 1, pp. 91–102, 2006.
- [4] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. Grose, *Eclipse Modeling Framework*, 1st ed., ser. The Eclipse Series, E. Gamma, L. Nackman, and J. Wiegand, Eds. Addison-Wesley Professional, 2004.
- [5] *Object Constraint Language (OMG) Specification - Version 2.0*, Object Management Group (OMG), May 2006. [Online]. Available: <http://www.omg.org/docs/formal/06-05-01.pdf>

- [6] R. Wörzberger and T. Heer, "Process model editing support using eclipse modeling project tools," in *Proceedings of the Second Workshop on MDS Today*, ser. Lecture Notes in Informatics, P. Friese, S. Zambrovski, and F. Zimmermann, Eds. Shaker Verlag, 2008.
- [7] H. M. W. Verbeek, T. Basten, and W. M. P. van der Aalst, "Diagnosing Workflow Processes using Woflan," *The Computer Journal*, vol. 44, no. 4, pp. 246–279, 2001. [Online]. Available: <http://comjnl.oxfordjournals.org/cgi/content/abstract/44/4/246>
- [8] J. Mendling and W. M. P. van der Aalst, "Formalization and verification of epscs with or-joins based on state and context," in *CAiSE*, ser. Lecture Notes in Computer Science, J. Krogstie, A. L. Opdahl, and G. Sindre, Eds., vol. 4495. Springer, 2007, pp. 439–453.
- [9] C. Fuss, C. Mosler, U. Ranger, and E. Schultchen, "The jury is still out: A comparison of AGG, Fujaba, and PROGRES," in *Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'07)*, 2007, 14 pp.
- [10] M. Reichert, "Dynamische Ablaufänderungen in Workflow-Management-Systemen," Ph.D. dissertation, Universität Ulm, May 2000.
- [11] M. Heller, D. Jäger, C.-A. Krapp, M. Nagl, A. Schleicher, B. Westfechtel, and R. Wörzberger, *An Adaptive and Reactive Management System for Project Coordination*. Springer, 2008, ch. 3, pp. 300–366.
- [12] M. Nagl and W. Marquardt, Eds., *Collaborative and Distributed Chemical Engineering: From Understanding to Substantial Design Process Support*, ser. Lecture Notes in Computer Science. Springer, 2008, vol. 4970.
- [13] P. Heimann, G. Joeris, C.-A. Krapp, and B. Westfechtel, "DYNAMITE: Dynamic task nets for software process management," in *Proceedings of the 18th International Conference on Software Engineering*. IEEE Computer Society Press, 1996, pp. 331–341. [Online]. Available: citeseer.ist.psu.edu/heimann96dynamite.html
- [14] C.-A. Krapp, "An adaptable environment for the management of development processes," Ph.D. dissertation, RWTH Aachen University, 1998.
- [15] *Meta-Object Facility (MOF) Specification - Version 1.4*, Object Management Group (OMG), April 2002.
- [16] A. Schleicher, "Management of Development Processes: An Evolutionary Approach," Ph.D. dissertation, RWTH Aachen University, 2002.
- [17] A. Schleicher and B. Westfechtel, "Beyond stereotyping: Metamodeling for the UML," in *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS), Minitrack: Unified Modeling – A Critical Review and Suggested Future*, 10 pages. IEEE Computer Society Press, 2001.
- [18] A. Schürr, A. Winter, and A. Zündorf, "The PROGRES approach: Language and environment," in *Handbook on Graph Grammars and Computing by Graph Transformation – Volume 2: Applications, Languages, and Tools*, H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, Eds. World Scientific, 1999, pp. 478–550.
- [19] M. Pesic, M. Schonenberg, and W. Aalst, "Declare: Full support for loosely-structured processes," in *EDOC '07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*. Washington, DC, USA: IEEE Computer Society, 2007, p. 287.
- [20] W. van der Aalst and A. Hofstede, "YAWL: Yet Another Workflow Language," *Information System*, vol. 30, no. 4, pp. 245–275, 2005.
- [21] B. Andersson, M. Bergholtz, A. Edirisuriya, T. Ilayperuma, and P. Johannesson, "A declarative foundation of process models," in *CAiSE*, 2005, pp. 233–247.
- [22] L. T. Ly, K. Göser, S. Rinderle-Ma, and P. Dadam, "Compliance of semantic constraints - A requirements analysis for process management systems," in *Proc. 1st Int'l Workshop on Governance, Risk and Compliance - Applications in Information Systems (GRCIS'08)*, 2008.
- [23] A. Ghose and G. Koliadis, "Auditing business process compliance," in *ICSOC*, ser. Lecture Notes in Computer Science, B. J. Krämer, K.-J. Lin, and P. Narasimhan, Eds., vol. 4749. Springer, 2007, pp. 169–180. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74974-5_14
- [24] S. Goedertier and J. Vanthienen, "Designing compliant business processes with obligations and permissions," in *Business Process Management Workshops*, ser. Lecture Notes in Computer Science, J. Eder and S. Dustdar, Eds., vol. 4103. Springer, 2006, pp. 5–14. [Online]. Available: http://dx.doi.org/10.1007/11837862_2
- [25] A. Rozinat and W. M. P. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Inf. Syst.*, vol. 33, no. 1, pp. 64–95, 2008.
- [26] R. Lu, S. W. Sadiq, and G. Governatori, "Compliance aware business process design," in *Business Process Management Workshops*, ser. Lecture Notes in Computer Science, A. H. M. ter Hofstede, B. Benatallah, and H.-Y. Paik, Eds., vol. 4928. Springer, 2007, pp. 120–131. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78238-4_14
- [27] R. Wörzberger, N. Ehses, and T. Heer, "Adding Support for Dynamics Patterns to Static Business Process Management System," in *Software Composition*, ser. Lecture Notes in Computer Science, C. Pautasso and É. Tanter, Eds., vol. 4954. Springer, 2008, pp. 84–91.