Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz
Kommunikation in verteilten Systemen 2011
(WowKiVS 2011)

Modeling Process-Related Duties with Extended UML Activity and Interaction Diagrams

Sigrid Schefer, Mark Strembeck

12 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Modeling Process-Related Duties with Extended UML Activity and Interaction Diagrams

Sigrid Schefer, Mark Strembeck

Institute for Information Systems and New Media
Vienna University of Economics and Business (WU Vienna), 1090 Austria

Abstract:

Business processes are an important source for the engineering of customized software systems. In this context, the definition, monitoring, and enforcement of the *duties* associated with different tasks in a business process is one important factor to ensure compliance of an IT system with certain laws and regulations. In this paper, we present a UML2 extension for an integrated modeling of business processes and process-related duties. In particular, our extension allows for the modeling of duties and associated tasks in business process models.

Keywords: business processes, duties, UML, compliance management

1 Introduction

Business processes define an organization's operational procedures and are performed to reach the operational goals of the corresponding organization. Therefore, business processes are an important source for the engineering of customized software systems. In this context, the definition, monitoring, and enforcement of the *duties* associated with different tasks in a business process is one important factor to ensure compliance in an IT system. For example, adequate support for the definition and enforcement of process-related policies, including separation of duty constraints, is one important part of SOX compliance [CB06, Dam04]. *Separation of duty* (SOD) constraints enforce conflict of interest policies (see, e.g., [AS00, LTB07]). Conflict of interest arises as a result of the simultaneous assignment of two mutually exclusive tasks to the same subject. However, modeling support for process-related duties is largely missing today. Especially in distributed environments the joint documentation of business processes and duties would facilitate their proper implementation and enforcement.

In this paper, we therefore propose an approach for the *integrated modeling of duties and business processes*. In particular, we present a UML extension to model processes, process-related duties, and responsibilities. An integrated modeling approach yields a number of advantages, such as allowing for a proper mapping of models to software systems, facilitating communication between different stakeholders, and detecting separation of duty conflicts. Moreover, on the one hand an integrated modeling approach for duties and processes allows for tracing duties to the (regulatory) reasons they exist, and on the other hand it allows to trace duties to the software components that have to ensure their monitoring and enforcement. This multi-directional traceability is also known as forward and backward traceability [DP98, GF94]. As a result, it is more easy to control and report on a company's fulfillment of compliance requirements. Furthermore, a complete and correct mapping between models and the respective software system assures consistency between modeling-level specifications and the software system supporting respective duties and process instances.

The remainder of this paper is structured as follows. Section 2 gives a motivating example for the definition of duties in a business process context. In Section 3, we present our extension for UML Activity diagrams and provide bindings to integrate duties into UML Interaction diagrams. Subsequently, Section 4 discusses related work and Section 5 concludes the paper.

2 A Motivating Example

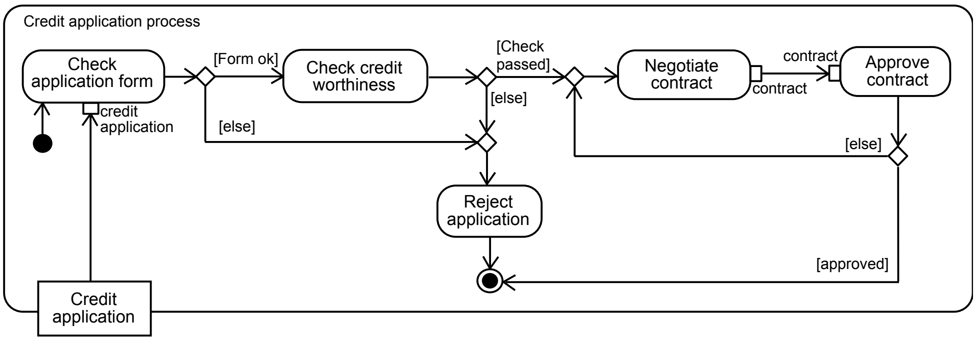


Figure 1: A simple credit application process

Figure 1 shows an example of a simple credit application process in a distributed environment modeled as standard UML Activity diagram. Below, we now provide a textual description of a duty D1 which affects the credit application process. In order to fulfill organisational compliance requirements, each *duty* defines an action which *must* be performed by a certain subject [Str05]. A *subject* may either be a human user or a software-based system. A *role* is a subject abstraction containing the tasks, associated permissions, and duties of a certain subject-type. In D1, a subject of the type/role "bank clerk" is required to discharge the duty "check applicant rating".

`D1: The bank clerk must check the credit applicant's rating within three days after receipt of the application form.`

A *process-related duty* is associated with a task in the corresponding business process. In our example, the duty "check applicant rating" needs to be performed when carrying out the "check credit worthiness" task (see Figure 1). Moreover, this duty can be refined via sub-duties, resulting in a duty-hierarchy. In a *duty-hierarchy*, higher level duties are more abstract duty descriptions which are refined by more concrete *subduties* [CDMR01, MS94]. For example, in D1.1-D1.3, the "check applicant rating" duty is refined via the three subduties listed below:

`D1.1: The bank clerk must check the validity of all data provided by the credit applicant.`

`D1.2: The bank clerk must check if the credit applicant is a black- listed customer.`

`D1.3: The bank clerk must check the credit applicant's current debt obligations.`

In addition, a duty may be associated with *constraints* [MS94, Str05]. In our example, D1 is associated with a time constraint which defines that D1 must be completed within three days after receipt of the application form. Furthermore, a duty may be associated with a *compensation action* which is carried out in case the time constraint is violated [ARD07]. In our example, D1 may be associated with the compensation action C_{D1} "forward duty to another bank clerk" if D1 is not completed in time:

`C`$_{D1}$`: If D1 is not discharged within three days after receipt of the application form, forward this duty to another bank clerk.`

In this context, it is also possible to define so called review duties. A *review duty* describes the goal of controlling the enforcement of an (ordinary) duty [SM02, SM04]. For example, the review duty R_{D1} "confirm applicant check" could be applied for the duty D1 if the special event "applicant has been rejected twice" occurs:

`R`$_{D1}$`: If an applicant has been rejected twice, a second bank clerk must confirm the credit applicant checks.`

This simple example already shows that it is difficult to describe all connections and implications of process-related duties in a textual manner. Therefore, a graphical representation for process-related duties is useful to facilitate their integration into process models and to support the elicitation and definition of *separation of duty* constraints (see, e.g., [BE01, SM10, WS07]).

3 Modeling Duties in a Business Process Context

Business process modeling and software system design is usually done via graphical modeling languages [RAHW06, WDHR06]. The Unified Modeling Language (UML) [OMG10b] offers a comprehensive and well-defined modeling framework and is the de facto standard for modeling and specifying information systems. Providing modeling support for duties in business process models using a standard notation like UML is intended to serve as a common language to bridge the communication gap between software engineers and non-technical stakeholders (see, e.g., [MJ10]). This means, integrated modeling of duties, responsibilities, and processes allows to document and communicate how duties are implemented in which parts of a business process and who is responsible for enforcing them.

To achieve the above, we model duties via *extended UML Activity diagrams* and propose a *refinement via UML Interaction models* to model bindings between duties and the classes implementing the duties' behavior. We use the Object Constraint Language (OCL) [OMG10a] to formally define the semantics of our newly introduced UML elements. Our UML extension can be applied to supplement other UML-based approaches and can be integrated in UML-based software tools. However, note that our general approach does not depend on the UML and may also be applied to extend other process modeling languages.

3.1 Extending UML2 Activity Diagrams

We introduce the *DutyNodes* package as a UML2 metamodel extension for modeling process-related duties (see Figure 2). This extension supplements the BusinessActivity extension introduced in [SM10]. Table 1 depicts corresponding notation elements. The associated OCL constraints defining the semantics for the new elements are found in Appendix A.

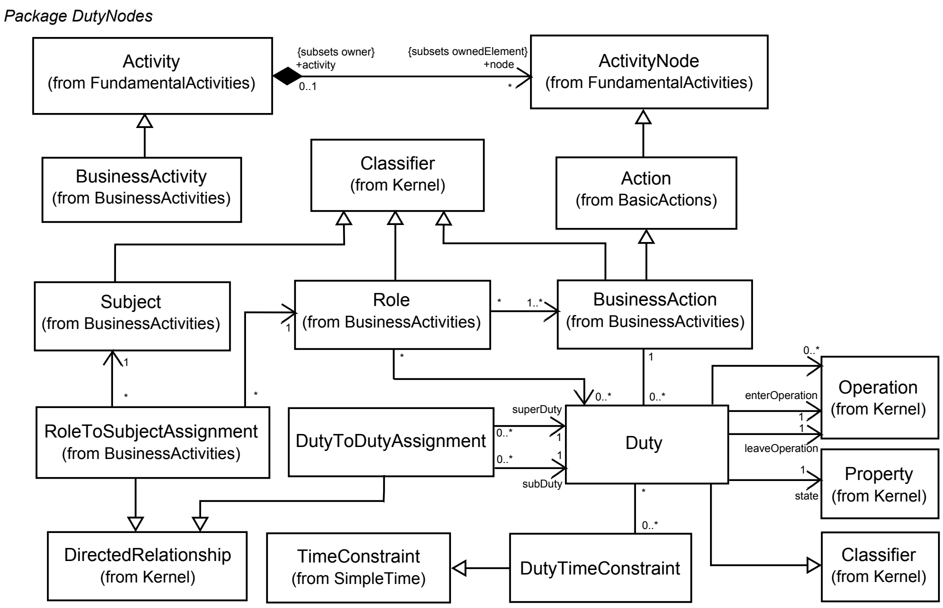


Figure 2: UML metamodel extension for DutyNodes

A *BusinessActivity* (from the BusinessActivity package specified in [SM10]) is a special UML Activity which can include all elements available for UML Activities in addition to our newly introduced elements (see OCL constraints 1, 2 and 3 in Appendix A). A *BusinessAction* corresponds to a task in a business process. Each instance of a BusinessAction may have an own state and a history, for example including attributes to capture how often the action instance has been executed, which subjects and roles executed the action instance, etc. In addition, SoD and BoD constraints can be defined for BusinessActions [SM10].

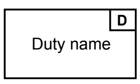
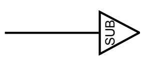
Node Type	Notation	Explanation
Duty		A *Duty* is shown as a rectangle. The compartment in the upper right corner includes a ”D”.
DutyToDuty Assignment		A *DutyToDutyAssignment* relation is shown as an arrow with a triangle arrowhead including the uppercase letters SUB indicating the end of the relation which points to the subduty.
DutyTimeConstraint	{t...t+n}—	A *DutyTimeConstraint* is shown as graphical association between a TimeInterval and a Duty.

Table 1: Graphical representation of DutyNodes in UML Activity diagrams

Our *Duty* element is defined as a special UML Classifier and is used in a UML Activity diagram to model that an action must be performed by a Subject which is assigned to this Duty (see Figure 2). Each Duty is linked to exactly one BusinessAction indicating that the Duty needs to be performed when carrying out the BusinessAction in order to fulfill compliance requirements. Moreover, Duties can be refined by subduties (see Section 2) and are linked to the Operation and Property metaclass (see Section 3.2). *Role* and *Subject* elements are linked to BusinessActions and Duties (see Figure 2 and OCL constraints 5 and 6). Furthermore, a Duty may be linked to a *DutyTimeConstraint* which is a specialised UML TimeConstraint (from the SimpleTime package, see [OMG10b]). If a DutyTimeConstraint has expired, a Compensation Action is triggered.

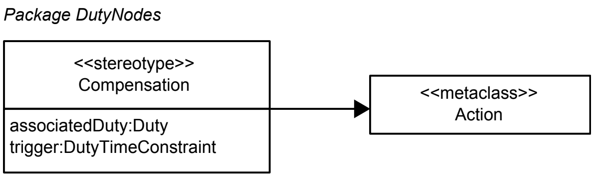


Figure 3: Stereotype for Compensation Action

In our extension, a *Compensation* Action is defined as a stereotype extending the semantics of the existing Action metaclass (see Figure 3). It is used in a UML Activity diagram to define actions which must be called if a Duty has not been performed in time (see OCL constraint 7 and 8). For this purpose, it specifies two properties. The *associatedDuty* property links a Compensation Action to its related Duty and the *trigger* defines a triggering time event for a Compensation Action. Examples for Compensation Actions are reassigning a Duty to another person, or sending a reminder email.

3.2 Defining Duty States

In our metamodel extension, a Duty is defined as special kind of Classifier (see Figure 2). According to [OMG10b], each Classifier may include an arbitrary number of Property attributes (from Kernel, AssociationClasses, Interfaces). In particular, each Duty contains a *state* Property. The *state* property refers to the actual state of a Duty and can take one of the following values: *passive*, *pending*, *discharged*, or *compensationActionCalled*. State descriptions are given in Table 2. In Appendix A, associated invariants are defined in OCL constraints 9 and 10.

State	Explanation
passive	The Duty is not activated.
pending	The Duty is activated but has not been discharged, yet.
discharged	The Duty has been successfully discharged in time.
compensationActionCalled	The Duty has not been successfully discharged in time. Thus, the corresponding Compensation Action was called.

Table 2: States of a Duty

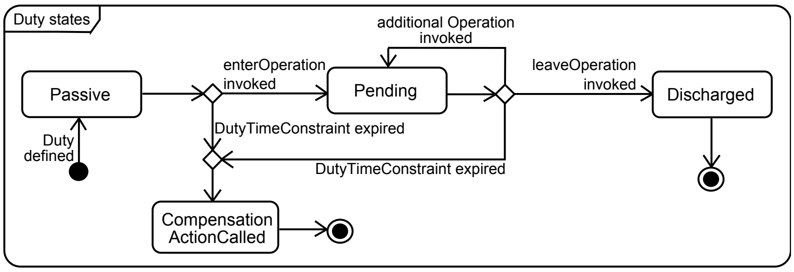


Figure 4: Duty state transitions

A Duty also defines at least two mandatory Operations: an *enterOperation* and a *leaveOperation* (see Figure 2). In addition, each Duty may include an arbitrary number of additional Operations. A Duty's *enterOperation* and *leaveOperation* define which Operation is invoked as soon as the corresponding Duty is entered or left. They can only be executed if the corresponding DutyTimeConstraint has not expired.

3.3 Modeling Duties from Different Perspectives

As mentioned above, integrating the specification of duties into business process models facilitates the communication and enforcement of compliance requirements. However, business process models need to express many different aspects. Capturing all of them in one model will presumably overload it. In our approach, the graphical representation of relations regarding our newly introduced elements can be limited to the most important modeling information. As each UML model needs to conform to its OCL constraints, the formally defined relations *exist independent of their actual graphical representation* (see [OMG10b, OMG10a]). Therefore, the graphical notation shown here primarily serves as a presentation option. For the graphical modeling of Duties, we propose four complementary perspectives (see Figure 5). Table 3 indicates which of the newly available modeling elements is used in which perspective.

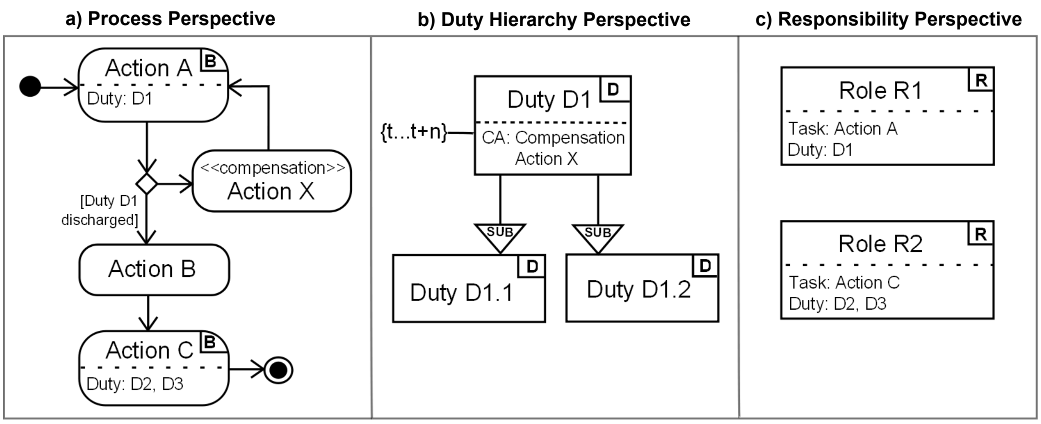


Figure 5: Modeling perspectives on a BusinessActivity with Duties

Perspective	Duty	Business Action	Compensation Action	DutyTime Constraint	Subduties	Roles
Process	√	√	√			
Duty Hierarchy	√		√	√	√	
Responsibility	√	√				√
Sequence	√			√		

Table 3: Modeling elements shown in different perspectives

3.4 Integrating Duties into UML2 Interaction Diagrams

We propose the refinement of extended Activity Diagrams via Interaction models providing a *Sequence Perspective* on Duties. Here, Interaction models define the detailed invocation sequence of messages for Duties which allows for the mapping of process definitions and related duties to the corresponding software system.

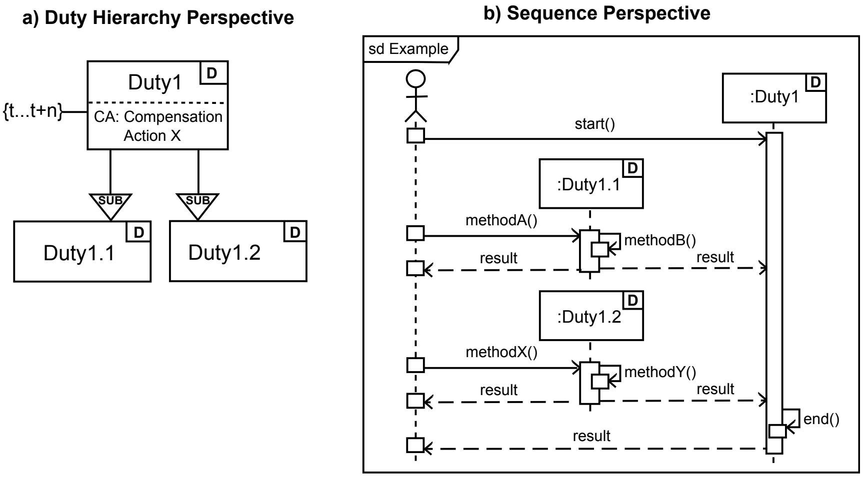


Figure 6: Modeling Duties in Interaction diagrams

Figure 6a) shows the *Duty Hierarchy Perspective* on a simple Duty hierarchy. The Interaction diagram in Figure 6b) presents the *Sequence Perspective* specifying the detailed invocation sequence of methods that occurs when these Duties are executed at runtime. In particular, it shows that an object of *Duty1* receives an invocation of an enterOperation called *start*. Subsequently, *methodA* and *methodX* trigger the execution of *methodB* and *methodY* in the subduty objects *Duty1.1* and *Duty1.2*. Finally, the leaveOperation *end* is executed. In UML2, each Lifeline head has the shape based on the Classifier that this lifeline represents (see [OMG10b]). Therefore, a Lifeline head node representing a Duty-Classifier is visualized by a rectangle including the name of the corresponding Duty and a small upper case letter D displayed in the upper right corner (see Table 1).

Due to the page restrictions, we had to cut the detailed example from this paper. On our webpage we provide an extended version of this paper where we re-inserted the text we had to cut from the paper.

4 Related Work

Recent approaches emphasize the importance of ensuring compliance on a business process level. For example, in [LRGD10], Ly et al. present a framework allowing for integrated compliance support with regard to the process lifecycle. The framework also provides formal trace-based compliance criteria for static compliance validation and for dealing with process changes. Other approaches for modeling compliance aspects either use informal annotations or formal languages (see, e.g., [GK07]). In [SGN07], an approach for achieving compliance by design is introduced by providing a formal representation of control objectives. Approaches using visual patterns and languages are presented, e.g., in [LMX07] or [FESV07]. The importance for ensuring compliance not only at the design level, but also at runtime, is, e.g., emphasized in [LRD08]. The approach presented in this paper also provides support for ensuring compliance aspects on design and runtime level.

A number of different approaches exist that use methods and techniques from model-driven development to include business rules in the software development processes. However, these approaches concentrate on the modeling of authorization constraints or other security requirements (see, e.g., [JÖ5a, JÖ5b, WS07]). To the best of our knowledge, this work represents the first attempt to address process-related duties from a business process perspective. As each duty holder also needs sufficient authority to perform the assigned duties [SM02, Str05], our approach complements existing approaches in the compliance and security-related context (see also [SM10]).

5 Conclusion

The need for integrated modeling of business processes and process-related duties to fulfill compliance requirements has been repeatedly identified in research and practice. However, standard process modeling languages do not provide corresponding language elements. To overcome this limitation, we introduced an integrated approach for modeling business processes and process-related duties.

In particular, we presented a UML meta-model extension for Activity and Interaction diagrams. In our extension, duties are modeled in extended Activity diagrams, classes are used to define the behavior for each duty, and UML Interaction diagrams model the detailed invocation sequence of messages for executing duties. Moreover, we apply the Object Constraint Language to formally define the semantics of the newly introduced UML metaclasses and stereotypes. Therefore, our extension can be integrated with other UML-based approaches or tools.

In our future work, we will investigate how to model other aspects of process-related duties, such as delegation of duties. Providing suitable modeling primitives for delegating duties is especially important, as flexibility is demanded in many information systems in order to leave some room for individual decisions of users in the course of a business process.

Bibliography

[ARD07] W. M. P. van der Aalst, M. Rosemann, M. Dumas. Deadline-based escalation in process-aware information systems. *Decision Support Systems* 43(2), 2007.

[AS00] G. Ahn, R. Sandhu. Role-based Authorization Constraints Specification. *ACM Transactions on Information and System Security (TISSEC)* 3(4), November 2000.

[BE01] R. A. Botha, J. H. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal* 40(3), 2001.

[CB06] J. Cannon, M. Byers. Compliance Deconstructed. *ACM Queue* 4(7), September 2006.

[CDMR01] J. Cole, J. Derrick, Z. Milosevic, K. Raymond. Author Obliged to Submit Paper before 4 July: Policies in an Enterprise Specification. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*. 2001.

[Dam04] M. Damianides. How does SOX change IT? *Journal of Corporate Accounting & Finance* 15(6), September/October 2004.

[DP98] R. Dömges, K. Pohl. Adapting traceability environments to project-specific needs. *Communications of the ACM* 41(12), 1998.

[FESV07] A. Forster, G. Engels, T. Schattkowsky, R. Van Der Straeten. Verification of Business Process Quality Constraints Based on Visual Process Patterns. In *Proceedings of the First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering*. IEEE Computer Society, 2007.

[GF94] O. C. Z. Gotel, A. C. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of 1st International Conference on Requirements Engineering., IEEE Computer Society Press*. 1994.

[GK07] A. Ghose, G. Koliadis. Auditing Business Process Compliance. In *Service-Oriented Computing – ICSOC 2007*. Lecture Notes in Computer Science 4749. Springer Berlin / Heidelberg, 2007.

[JÖ5a] J. Jürjens. *Secure Systems Development with UML*. Springer Verlag, 2005.

[JÖ5b] J. Jürjens. Sound Methods and Effective Tools for Model-based Security Engineering with UML. In *Proc. of the 27th International Conference on Software Engineering (ICSE)*. 2005.

[LMX07] Y. Liu, S. Müller, K. Xu. A static compliance-checking framework for business process models. *IBM Syst. J.* 46, April 2007.

[LRD08] L. T. Ly, S. Rinderle, P. Dadam. Integration and verification of semantic constraints in adaptive process management systems. *Data Knowl. Eng.* 64, January 2008.

[LRGD10] L. Ly, S. Rinderle-Ma, K. Göser, P. Dadam. On enabling integrated process compliance with semantic constraints in process management systems. *Information Systems Frontiers*, 2010.

[LTB07] N. Li, M. Tripunitara, Z. Bizri. On Mutually Exclusive Roles and Separation-of-Duty. *ACM Transactions on Information and System Security (TISSEC)* 10(2), May 2007.

[MJ10] H. Mouratidis, J. Jürjens. From Goal-Driven Security Requirements Engineering to Secure Design. *International Journal of Intelligent Systems* 25(8), 2010.

[MS94] J. D. Moffett, M. S. Sloman. Policy Conflict Analysis in Distributed System Management. *Journal of Organizational Computing* 4(1), 1994.

[OMG10a] OMG. Object Constraint Language Specification. available at: http://www.omg.org/technology/documents/formal/ocl.htm, February 2010. Version 2.2, formal/2010-02-01, The Object Management Group.

[OMG10b] OMG. Unified Modeling Language (OMG UML): Superstructure. available at: http://www.omg.org/technology/documents/formal/uml.htm, May 2010. Version 2.3, formal/2010-05-03, The Object Management Group.

[RAHW06] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, P. Wohed. On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling. In *Proc. of the Third Asia-Pacific Conference on Conceptual Modelling (APCCM)*. 2006.

[SGN07] S. Sadiq, G. Governatori, K. Naimiri. Modeling Control Objectives for Business Process Compliance. In *5th International Conference on Business Process Management (BPM07)*. 2007.

[SM02] A. Schaad, J. D. Moffett. Delegation of Obligations. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*. 2002.

[SM04] A. Schaad, J. Moffett. Separation, review and supervision controls in the context of a credit application process: a case study of organisational control principles. In *SAC '04: Proceedings of the 2004 ACM Symposium on Applied Computing*. ACM, New York, NY, USA, 2004.

[SM10] M. Strembeck, J. Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 2010. doi:10.1016/j.infsof.2010.11.015

[Str05] M. Strembeck. Embedding Policy Rules for Software-Based Systems in a Requirements Context. In *Proc. of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*. June 2005.

[WDHR06] P. Wohed, M. Dumas, A. H. M. T. Hofstede, N. Russell. On the Suitability of BPMN for Business Process Modelling. In *In Proceedings 4th International Conference on Business Process Management (BPM 2006), LNCS*. 2006.

[WS07] C. Wolter, A. Schaad. Modeling of Task-Based Authorization Constraints in BPMN. In Alonso et al. (eds.), *Business Process Management*. Lecture Notes in Computer Science 4714. Springer Berlin / Heidelberg, 2007.

A Invariants for the DutyNodes package

OCL Constraint 1 *If a BusinessAction node is used in an Activity diagram, this Activity is always of the type BusinessActivity:*

```
context Activity inv:
   if self.node->exists(n |
      n.oclIsKindOf(BusinessAction) then
      self.oclIsKindOf(BusinessActivity))
   endif
```

OCL Constraint 2 *In order to unambiguously identify different instances of the same BusinessActivity, we require that each BusinessActivity defines an attribute called processID:*

```
context BusinessActivity inv:
self.instanceSpecification->forAll(i |
   i.slot->exists(s |
      s.definingFeature.name = processID))
```

OCL Constraint 3 *Each Duty instance defines an attribute called associatedProcessInstance and needs to be discharged in the context of an instance of the corresponding BusinessActivity:*

```
context Duty
inv: self.instanceSpecification->forAll(i |
        i.slot->exists(s |
           s.definingFeature.name = associatedProcessInstance ))
inv: self.instanceSpecification->forAll(i |
        self.businessAction.activity.instanceSpecification->exists(a |
           i.slot->select(si |
              si.definingFeature.name = associatedProcessInstance
              a.slot->select(sa |
                 sa.definingFeature.name = processID and
                 si.value = sa.value))))
```

OCL Constraint 4 *Each Duty defines an attribute called isReviewDuty stating if a special Duty is a review-duty or not (see Section 2):*

```
context Duty inv:
self.instanceSpecification->forAll(i |
   i.slot->exists(s |
      s.definingFeature.name = isReviewDuty))
```

OCL Constraint 5 *Each Duty defines an attribute called responsibleSubject to assign an instance of a Duty to a Subject that is responsible for executing this particular Duty. Therefore, the responsibleSubject must refer to a Subject that is allowed to execute this Duty (due to its role membership):*

```
context Duty inv:
self.instanceSpecification->forAll(i |
   i.slot->exists(s |
      s.definingFeature.name = responsibleSubject and
```

```
      (self.role->exists(r |
         r.roleToSubjectAssignment->exists(rsa |
            rsa.subject.name = s.value)))))
```

OCL Constraint 6 *Each Duty defines an attribute called responsibleRole to assign an instance of a Duty to the executing Role of this particular Duty. Therefore, the responsibleRole must refer to a Role that is directly associated with the corresponding Duty:*

```
context Duty inv:
self.instanceSpecification->forAll(i |
   i.slot->exists(s |
      s.definingFeature.name = responsibleRole and
      (self.role->exists(r |
         r.name = s.value))))
```

OCL Constraint 7 *If a Duty is associated with a DutyTimeConstraint, a Compensation Action needs to be defined. Thus, a Compensation Action only exists if an associated Duty and a triggering DutyTimeConstraint are defined (see Section 2):*

```
context Compensation inv:
if self.trigger->exists() then
   self.associatedDuty.oclIsKindOf(Duty) and
   self.trigger.oclIsKindOf(DutyTimeConstraint)
endif
```

OCL Constraint 8 *Each DutyTimeConstraint defines an attribute called currentDate which defines the current date when a process instance is executed:*

```
context DutyTimeConstraint inv:
self.instanceSpecification->forAll(i |
   i.slot->select(s |
      s.definingFeature.name = currentDate))
```

OCL Constraint 9 *Each Duty is associated with the state property taking one of the predefined values:*

```
context Duty inv:
self.instanceSpecification->forAll(i |
   i.slot->exists(s |
      s.state.value = "passive"|"pending"|"discharged"| "compensationActionCalled" ))
```

OCL Constraint 10 *The enterOperation and the leaveOperation of a Duty can only be executed if the corresponding DutyTimeConstraint is not expired:*

```
context Duty
inv: self.enterOperation->forAll(e |
       if e.duty.dutyTimeConstraint.notEmpty then
          e.duty.dutyTimeConstraint.currentDate <
             e.duty.dutyTimeConstraint.specification.max)
inv: self.leaveOperation->forAll(e |
       if e.duty.dutyTimeConstraint.notEmpty then
          e.duty.dutyTimeConstraint.currentDate <
             e.duty.dutyTimeConstraint.specification.max)
```