This is an extended version of the paper published as: S. Schefer, M. Strembeck: Modeling Process-Related Duties with Extended UML Activity and Interaction Diagrams, In: Proc. of the International Workshop on Flexible Workflows in Distributed Systems, Workshops der wissenschaftlichen Konferenz Kommunikation in verteilten Systemen (WowKiVS), Electronic Communications of the EASST, Vol. 37, Kiel, Germany, March 2011

In the extended version, we reinserted the text that we had to cut from the paper due to the page restrictions for the conference version.

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

Sigrid Schefer, Mark Strembeck

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.

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 [3, 5, 17]. *Separation of duty* (SOD) constraints enforce conflict of interest policies [1, 7, 13]. 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 [6, 10]. 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 an example business process with duties. Section 5 discusses related work and Section 6 concludes the paper.

2 A Motivating Example

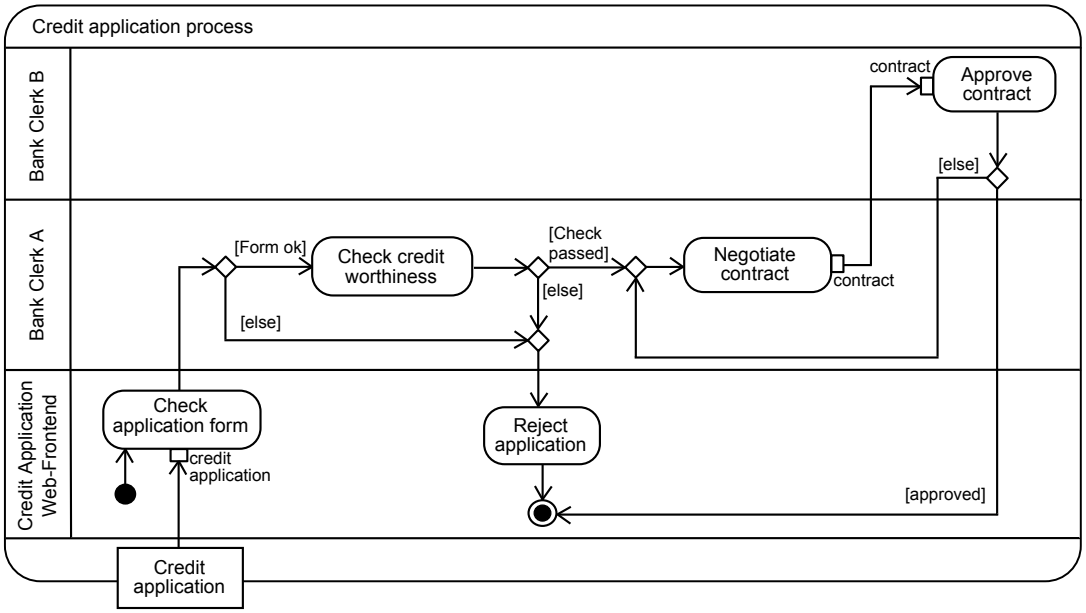


Figure 1: A simple credit application process

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 [27]. 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 on lower levels by more concrete *subduties* [4, 18, 26]. 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 blacklisted customer.`

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

In addition, a duty may be associated with *constraints* [18, 27]. 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 [29]. 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 [24, 25]. 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. Moreover, as already mentioned, the definition of duties in a business process context also supports the elicitation and definition of *separation of duty* constraints (see, e.g., [2, 28, 31]).

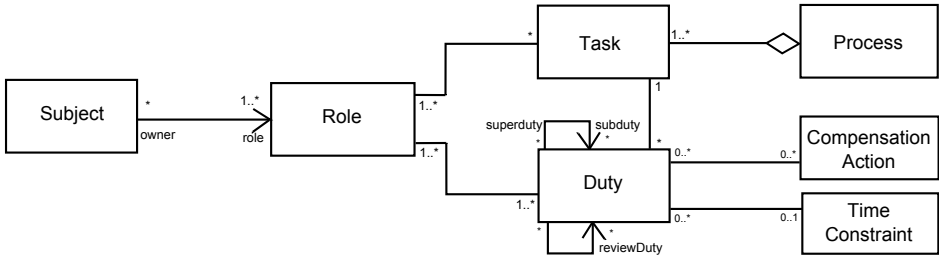


Figure 2: Relations between modeling elements

Figure 2 shows the essential relations of the elements used in our approach. Subjects may be assigned to roles. Roles are assigned to process-related tasks and duties. Each duty is related to a specific task. Compensation actions are associated with duties and define actions which must be taken if the time constraint associated with a duty is violated. A process instance defines one particular execution of a business process.

3 Modeling Duties in a Business Process Context

Business process modeling and software system design is usually done via graphical modeling languages [22, 30]. The Unified Modeling Language (UML) [21] 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., [19]). 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) [20] 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 providing the following modeling elements (see Figure 3): Duty, DutyToDutyAssignment, and DutyTimeConstraint. This extension supplements the BusinessActivity extension introduced in [28]. Table 1 depicts

corresponding notation elements. The associated OCL constraints defining the semantics for the new elements are found in Appendix A.

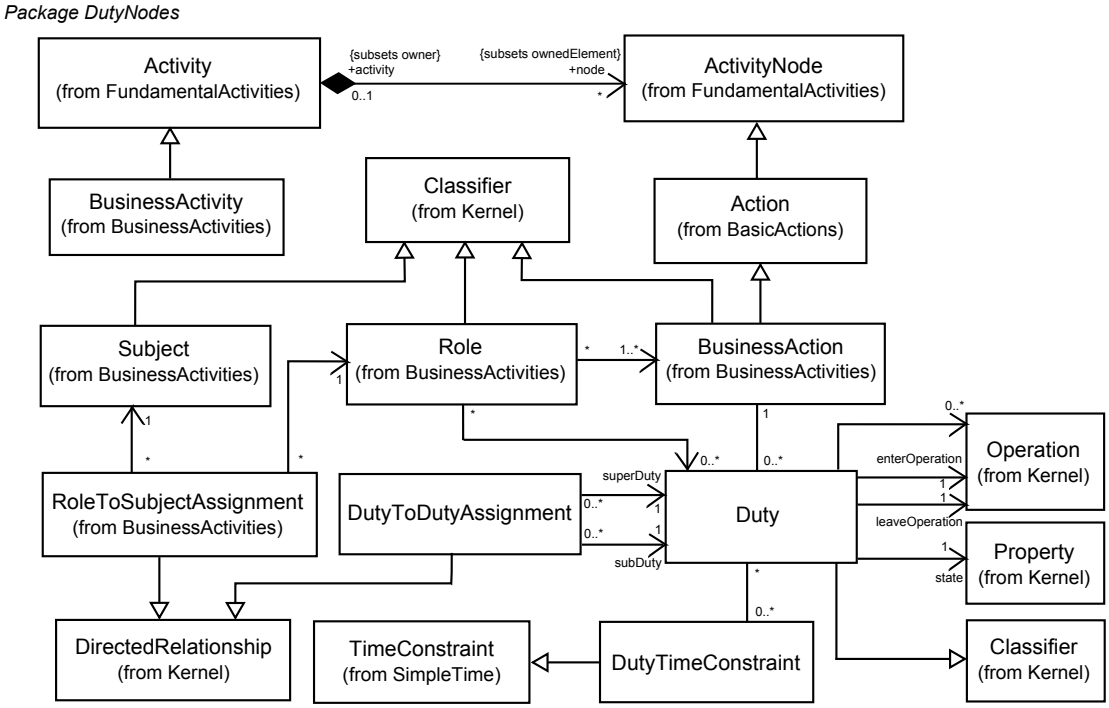


Figure 3: UML metamodel extension for DutyNodes

A *BusinessActivity* (from the BusinessActivity package specified in [28]) 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 [28].

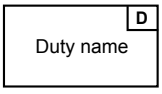
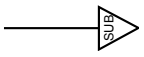
Node Type	Notation	Explanation
Duty		A *Duty* is shown as a rectangle. The compartment in the upper right corner includes a "D".
DutyToDuty Assignment		A *DutyToDutyAssignment* relation is shown as an arrow with a triangle arrowhead including the uppercase letters SUB indicating the end of the relation 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 3). 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 3 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 [21]). If a DutyTimeConstraint has expired, a Compensation Action is triggered.

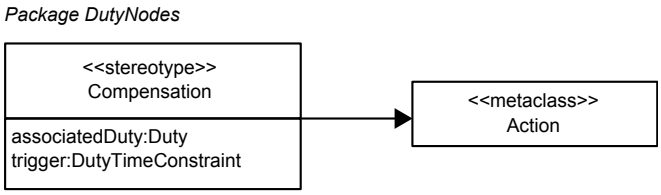


Figure 4: Stereotype for Compensation Action

In our extension, a *Compensation* Action is defined as a stereotype extending the semantics of the existing Action metaclass (see Figure 4). 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 3). According to [21], 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

A Duty also defines at least two mandatory Operations: an *enterOperation* and a *leaveOperation* (see Figure 3). 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. Thus, the following sequence of steps occurs when entering a Duty (see Figure 5):

Step 1: If a Duty's *enterOperation* is invoked before the respective DutyTimeConstraint has expired, its

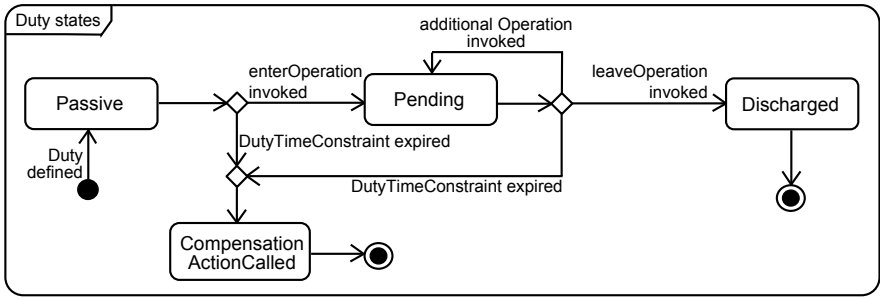


Figure 5: Duty state transitions

state changes from *passive* to *pending*. Otherwise, if a Duty's *enterOperation* is not executed in time, its state turns to *compensationActionCalled* and a Compensation Action is carried out.

Step 2: An arbitrary number of other Operations can be executed.

Step 3: If a Duty's *leaveOperation* is executed before the respective DutyTimeConstraint has expired, its state turns to *discharged*. Otherwise, its state turns to *compensationActionCalled*.

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 [21, 20]). Therefore, the graphical notation shown here primarily serves as a presentation option.

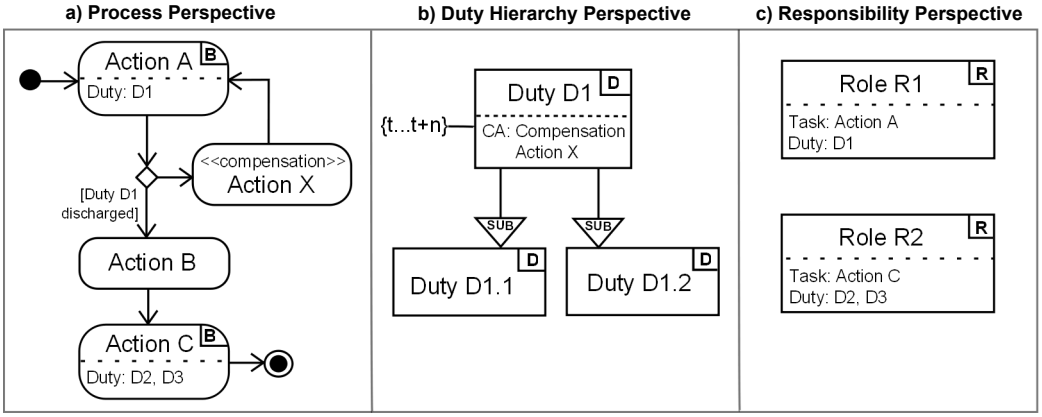


Figure 6: Modeling perspectives on a BusinessActivity with Duties

For the graphical modeling of Duties, we propose four complementary perspectives. Table 3 indicates which of the newly available modeling elements is used in which perspective.

- The *Process Perspective* is exemplified in Figure 6a) illustrating a BusinessActivity including three Duties. This perspective presents a general view on how Duties – and optionally their Compensation Actions – are integrated into a specific process.
- The *Duty Hierarchy Perspective* provides a detailed view on Duties as visualized in Figure 6b) showing relations between Duties and subduties, DutyTimeConstraints as well as Compensation Actions.

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

Table 3: Modeling elements shown in different perspectives

- The *Responsibility Perspective* depicts Role-Duty and Role-BusinessAction relations indicating the responsibility for discharging a Duty and for performing a Business Action as modeled in Figure 6c).
- The fourth perspective, called *Sequence Perspective* is described in the next subsection.

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 7a) shows the *Duty Hierarchy Perspective* on a simple Duty hierarchy. The Interaction diagram in Figure 7b) 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 [21]). 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).

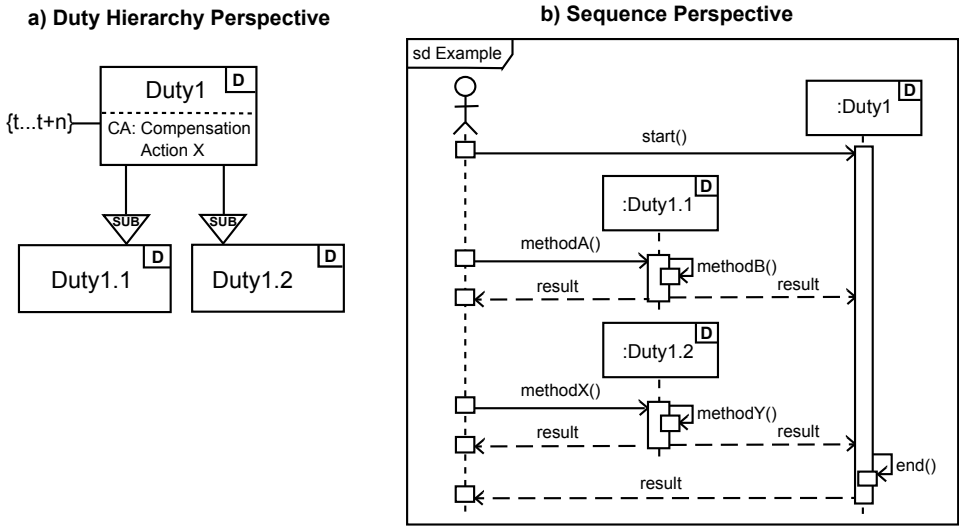


Figure 7: Modeling Duties in Interaction diagrams

4 Example Process with Duties

This section presents an example modeling the four perspectives on a process including Duties (see Table 3). For this purpose, we extend the credit application process from Figure 1 by including the new modeling constructs proposed in Section 3. As each UML model needs to conform to its OCL invariants (see Appendix A), the graphical representation can be limited to the most important modeling information.

Figure 8a) shows the *Process Perspective* on a BusinessActivity that remodels the credit application process from Figure 1 by integrating process-related duties. The process includes five actions, three of which are defined as BusinessActions. Two BusinessActions are associated with Duties: the BusinessAction "Check credit worthiness" is associated with the Duty "Check applicant rating" and the BusinessAction "Negotiate contract" with the Duty "Fulfill precontractual duties". In addition, the Compensation Action "Reassign Duty" is triggered if the Duty "Check applicant rating" is not discharged in time.

The *Responsibility Perspective* in Figure 8b) shows the Role BankClerk which is assigned to the two BusinessActions and the associated Duties defined in the credit application process. Thus, a Subject assigned to the BankClerk role will be responsible to perform these Duties and all of their subduties. When integrating Duties and responsibilities into business process models, the detection of conflicts – such as separation of duty conflicts – is simplified compared to if Duties are defined in separate textual documents.

Figure 8c) presents the *Duty Hierarchy Perspective* on the Duty "Check applicant rating" which is connected to the BusinessAction "Check credit worthiness". It is refined via three subduties and associated with a DutyTimeConstraint and a Compensation Action. The DutyTimeConstraint expresses that the Duty "Check applicant rating" – and all of its subduties – needs to be completed within three time units (e.g. days) after the corresponding BusinessAction has been started. Otherwise, the Compensation Action "Reassign Duty to other bank clerk" is called.

The Interaction diagram in Figure 8d) depicts the *Sequence Perspective* visualizing the detailed invocation sequence of methods defined for the Duties. This mapping supports consistency between process definitions, related duties and the corresponding software system. In particular, an instance of the Duty "Check applicant rating" receives an invocation of an enterOperation called checkData() which changes the Duty's state from passive to pending. Next, the subduties' execution is triggered. Each subduty also defines at least an enter- and a leaveOperation. After successfully discharging the subduties, the Duty's leaveOperation writeCustomerAccountEntry() is completed and its state finally turns to discharged.

5 Related Work

Recent approaches emphasize the importance of ensuring compliance on a business process level. For example, in [15], 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., [9]). In [23], 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 [14] or [8]. The importance for ensuring compliance not only at the design level, but also at runtime, is, e.g., emphasized in [16]. 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., [11, 12, 31]). 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 [25, 27], our approach complements existing approaches in the compliance and security-related context (see also [28]).

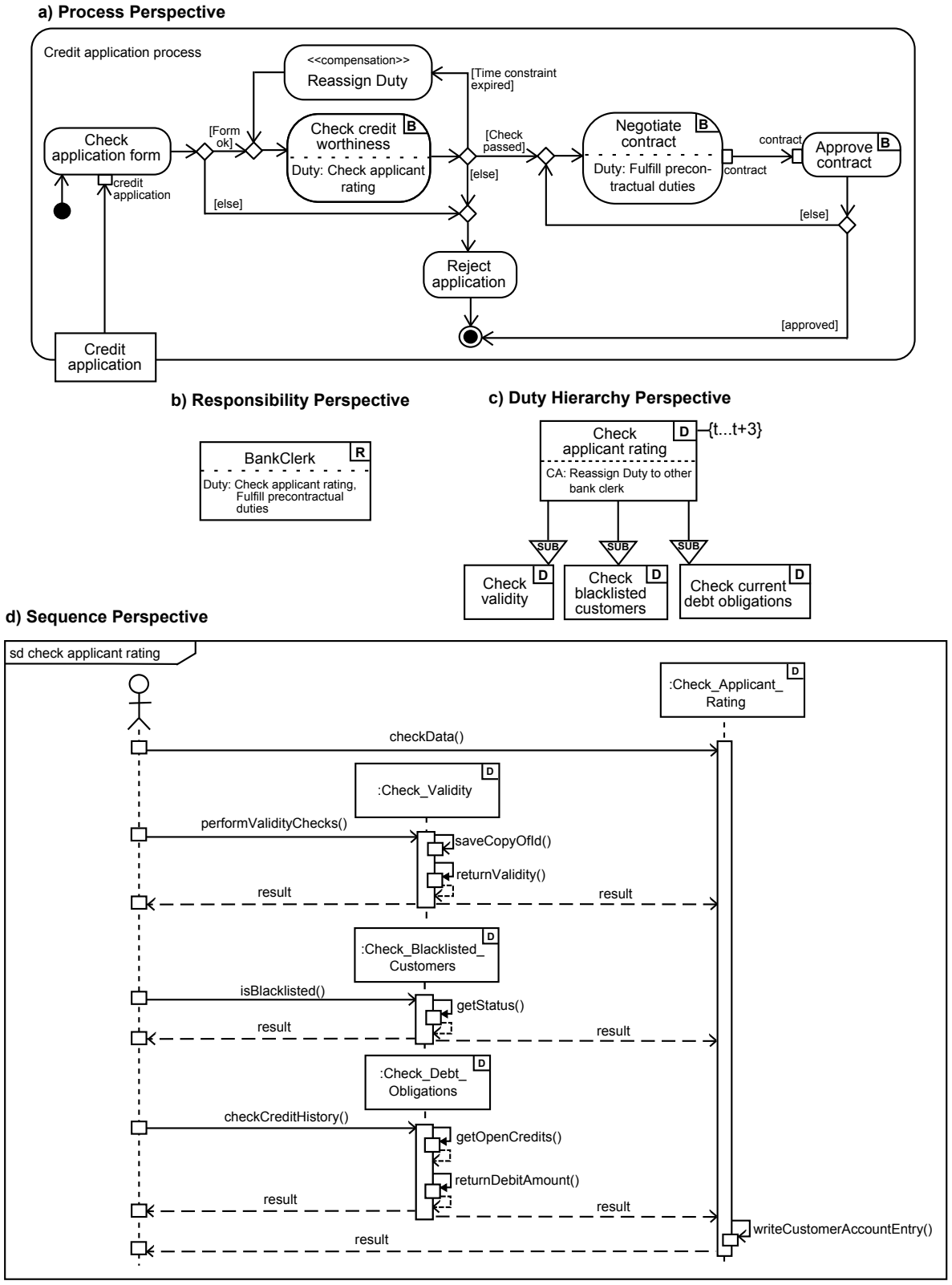


Figure 8: Four perspectives on Duties in the credit application process

6 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.

References

[1] AHN, G., AND SANDHU, R. Role-based Authorization Constraints Specification. *ACM Transactions on Information and System Security (TISSEC) 3*, 4 (November 2000).

[2] BOTHA, R. A., AND ELOFF, J. H. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal 40*, 3 (2001).

[3] CANNON, J., AND BYERS, M. Compliance Deconstructed. *ACM Queue 4*, 7 (September 2006).

[4] COLE, J., DERRICK, J., MILOSEVIC, Z., AND RAYMOND, K. 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).

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

[6] DÖMGES, R., AND POHL, K. Adapting traceability environments to project-specific needs. *Communications of the ACM 41*, 12 (1998).

[7] FERRAIOLO, D., BARKLEY, J., AND KUHN, D. A Role-Based Access Control Model and Reference Implementation within a Corporate Intranet. *ACM Transactions on Information and System Security (TISSEC) 2*, 1 (February 1999).

[8] FORSTER, A., ENGELS, G., SCHATTKOWSKY, T., AND VAN DER STRAETEN, R. 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* (2007), IEEE Computer Society.

[9] GHOSE, A., AND KOLIADIS, G. Auditing business process compliance. In *Service-Oriented Computing – ICSOC 2007*, vol. 4749 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007.

[10] GOTEL, O. C. Z., AND FINKELSTEIN, A. C. An analysis of the requirements traceability problem. In *Proceedings of 1st International Conference on Requirements Engineering., IEEE Computer Society Press* (1994).

[11] JÜRJENS, J. *Secure Systems Development with UML.* Springer Verlag, 2005.

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

[13] LI, N., TRIPUNITARA, M., AND BIZRI, Z. On Mutually Exclusive Roles and Separation-of-Duty. *ACM Transactions on Information and System Security (TISSEC) 10*, 2 (May 2007).

[14] LIU, Y., MÜLLER, S., AND XU, K. A static compliance-checking framework for business process models. *IBM Syst. J. 46* (April 2007).

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

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

[17] Mishra, S., and Weistroffer, H. A Framework for Integrating Sarbanes-Oxley Compliance into the Systems Development Process. *Communications of the Association for Information Systems (CAIS) 20*, 1 (2007).

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

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

[20] 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.

[21] 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.

[22] Russell, N., van der Aalst, W. M. P., ter Hofstede, A. H. M., and Wohed, P. 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).

[23] Sadiq, S., Governatori, G., and Naimiri, K. Modeling control objectives for business process compliance. In *5th International Conference on Business Process Management (BPM07)* (2007).

[24] Schaad, A., and Moffett, J. 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* (New York, NY, USA, 2004), ACM.

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

[26] Sloman, M. S. Policy Driven Management for Distributed Systems. *Journal of Network and Systems Management 2*, 4 (1994).

[27] Strembeck, M. 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).

[28] Strembeck, M., and Mendling, J. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology* (2010).

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

[30] Wohed, P., Dumas, M., Hofstede, A. H. M. T., and Russell, N. On the suitability of bpmn for business process modelling. In *In Proceedings 4th International Conference on Business Process Management (BPM 2006), LNCS* (2006).

[31] Wolter, C., and Schaad, A. Modeling of Task-Based Authorization Constraints in BPMN. In *Business Process Management*, G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714 of *Lecture Notes in Computer Science*. 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)
```