

Enabling Resource Assignment Constraints in BPMN

Ahmed Awad, Alexander Grosskopf, Andreas Meyer, Mathias Weske

Hasso Plattner Institute, Potsdam, Germany

{ahmed.awad,alexander.grosskopf,andreas.meyer,mathias.weske}@hpi.uni-potsdam.de

Abstract. Due to the critical role humans play in automated business processes, a BPEL extension has been recently released BPEL4People and WS-HumanTask to express issues like authorization to execute a task. However, there is currently little support to human resource allocation to tasks in high level modeling languages such as BPMN. In this paper, we show how resource allocation can be captured in BPMN. We extend the meta model to hold the information needed and we use Object Constraint Language (OCL) to express resource allocation constraints. Using OCL enables us to directly reflect the assignments and constraints on the model level in order to automatically propagate them to the execution. Thus, we benefit from a model driven approach in the area of business process management.

1 Introduction

Software models represent the centerpiece in deriving executable software from early abstract models. In Model Driven Architecture (MDA), raising the amount of information put into the abstract models, so called Platform Independent Models (PIM), allows to automatically derive executable models, so called platform specific models (PSM), with less effort of manually editing them to fill missing requirements.

In the business process management field, Business Process Modeling Notation (BPMN) would be seen as the PIM modeling language, whereas languages like BPEL play the role of a PSM. To transform PIM to PSM, more than only the control flow needs to be automatically translated to executable models. Two other aspects of business process models are of equivalent importance, namely data flow and resource allocation to tasks in the process. We consider resources to be human beings at all times. Talking about reflecting human executable tasks in a workflow. There has been a recent extension to BPEL, the WS-HumanTask [4] and BPEL4People [5], to address this on the executable level, there is currently limited support to express such resource allocation issues on higher level modeling languages such as EPCs, ADs, or BPMN.

BPMN was widely adopted by business experts. There is a need for a genuine representation as well as a technically sound solution to express resource allocation expressions on a high level. By that, we can empower the business to govern their processes and we can build on a technical solution by design.

So far, BPMN [3] has little support for human resource modeling [17]. Lanes can be used to loosely express roles or responsibilities. The draft version of BPMN2.0 [6] indicates that organizational models and resources will even lose ground in the specification. We argue that this is an upcoming flaw for the specification and we present an approach to overcome this. Our contribution in this paper is an extended metamodel for BPMN that is oriented to human beings as resources to accomplish work in a process. We use OCL [2] to express resource allocation constraints because it natively integrates with UML class diagrams. Additionally, there exist a lot of experience as well as sophisticated tools [14, 1] to validate OCL constraints.

The remainder of this paper is structured as follows. Section 2 sets the requirements for expressing allocation constraints on a model level. An extended BPMN metamodel for resource and related organizational aspects is presented in Section 3. A case study with different allocation patterns expressed as OCL constraints is given in Section 4. Tool support to validate the constraints and to enable modeling them is discussed in Section 5. Discussion of related work takes place in Section 6. We conclude the paper with a discussion of future work in Section 7.

2 Resource Patterns As A Requirements Framework

To express a comprehensive set of resource allocation constraints, we consider the Workflow Resource Patterns (WRP) [15] as a framework. They were defined by the Workflow Patterns Initiative by evaluating existing BPM languages and workflow systems for their resource related capabilities. Not all patterns described in [15] are related to process model information. As an example, Selection Autonomy (WRP-26) deals with the possibility that resources might select an arbitrary work item from a set of offered work items to process next. This is an ability related to a workflow system. It is not reflected in the process model itself. We consider the subset of resource patterns that is captured in the process and related to the specification of allocation constraints upon creation of the work item. We stick to the original definitions given in [15] disregarding refinements and changes made on the website¹. There are deviating opinions about the interpretation of patterns and the requirements to support them. Therefore we use this section to state the patterns considered in this paper and to directly derive the requirements associated with them.

WRP-01 Direct Allocation is the ability to specify the identity of the resource that will execute a task at design time. Support for this pattern is given, if it is possible at design time to reference the identity of a resource to which the work item will be allocated at runtime.

WRP-02 Role-based Allocation is the ability to specify at design time that a task can only be executed by resources which correspond to a given role. Support for

¹ www.workflowpatterns.com

this pattern is given if a role concept exists and roles can be specified to work on tasks. The role is resolved to individuals at runtime and allocated to one of them.

WRP-04 Authorization is the ability to specify the range of resources that are authorized to execute a task. Support for this pattern is given, if a set of roles or individuals can be specified to be authorized for a task.

WRP-05 Separation of Duty is the ability to specify that two tasks must be allocated to different resources in a given workflow case. Support for this pattern requires the ability to express this interdependency between tasks in a workflow model.

WRP-06 Case Handling is the ability to allocate all work items within a given workflow case to the same resource. This pattern is supported if all tasks in a case are bound to the same resource that has accomplished the first work item at commencement of the process.

WRP-07 Retain Familiar is the ability to allocate work items to the same individual resource that has accomplished an earlier work item. Support for this pattern requires resource allocation descriptions to reference information from other work items.

WRP-08 Capability-based Allocation is the ability to offer or allocate instances of a task to resources based on specific capabilities that they possess. Support requires information about capabilities of resources and the ability to reference this information to specify resource allocation.

WRP-09 History-based Allocation is the ability to offer or allocate work items to resources on the basis of their previous execution history. Support for this pattern requires information about previously completed work items and the ability to reference this information to specify resource allocation.

WRP-10 Organizational Allocation is the ability to offer or allocate instances of a task to resources based on their position within the organization and their relationship with other resources. Support for this pattern requires an organizational model with positions and relationships. At design time, task allocation can be specified based on the absolute or relative position of a resource in the organizational model.

One might argue that further resource patterns should be listed here. We consider the core set of patterns required to describe task allocation to human resources that excludes systems and patterns such as Automatic Execution (WRP-11). We argue that this is trivial by using a certain task type or explicitly not specifying the allocation to a human resource. In this paper, we focus on the task allocation to humans. We assume that the constraints specified must

always hold and thereby are verified also for Delegation (WRP-27) or Escalation (WRP-28) of work items.

Fig. 1 provides a sample process comprising most of the patterns decided for above. It represents a typical process which is supported by software but the tasks are driven by humans. This scenario is about a leave request by an employee of a specific company which his supervisor needs to comment on and the manager, who works for the same company as well but shall not be the same resource as the supervisor, has to take the final decision before the supervisor, who did the commenting, can propagate the decision to his subordinate. For highlighting these three participating roles, we use three lanes in this scenario. The requirement that the supervisor is the superior of the employee cannot be considered for resource allocation yet, because hierarchical information of the appropriate organization is not available at design time. Furthermore, the separation and binding of duty constraints belonging to the last three tasks of the scenario in Fig. 1 cannot be represented with BPMN. Finally, authorization constraints are not representable within a standard BPMN diagram.

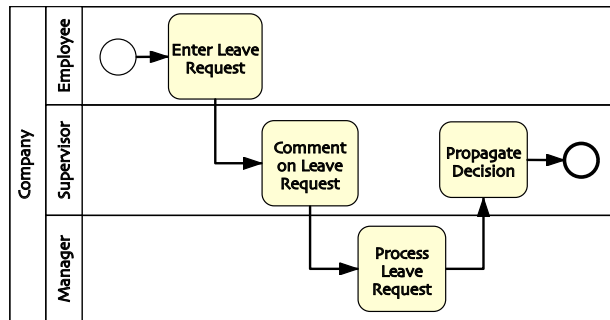


Fig. 1. A Leave Request Process

3 A Resource Aware BPMN Metamodel

In the previous section we determined the requirements. In this section, we present the BPMN core metamodel complemented by seven classes to hold resource relevant information. Fig. 2 represents the extended BPMN metamodel. Within this model, the classes `BPMNDiagram`, `Pool`, `Lane`, `Process`, `Activity`, and `Task` with stereotype `<<core>>` are already defined in BPMN and relate to design time artifacts. The other classes are the newly introduced ones.

A `Case` for example represents a process instance and thereby all information associated with the execution of it. This holds for still running cases as well as already completed executions, i.e. a `Case` holds the history information of a process model. Similarly, a `WorkItem` is the instance of a `Task`. Thus, `Cases` are composed of `WorkItems` just as `Processes` are composed of `Activities`. Each `WorkItem` is assigned to exactly one `Resource`, whereas a `Resource` may

deal with several **WorkItems** at any time. A **Resource** itself is exactly identifiable by the attached id. Additionally, each **Resource** gets assigned to up to one **Profile** which represents the specific resource's capabilities, experiences, and advanced education courses using attribute value pairs. Lastly, a **Resource** is associated with at least one **Role**. Vice versa, each **Role** is linked to any number of **Resources** and is part of a hierarchy which depends on the type of **Role** used. Common for all types is the existence of none or one superior role and any number of subordinate roles for covering the root and the leaves within the hierarchy tree as well. There do exist two different types of roles: **FunctionalRoles** and **OrganizationalRoles**. **FunctionalRoles** correspond to roles performing a certain work and **OrganizationalRoles** represent the organizational structure of the organization. Using the hierarchy with **FunctionalRoles**, a superior role contains all the aspects of the appropriate subordinate roles, e.g. a senior developer may perform the same work as a normal developer in addition to some extra work reasoned by higher experience and trustworthiness. **OrganizationalRoles** do not inherit aspects of subordinate roles because in the sample in Fig. 1, the supervisor does not necessarily need to be an employee as well, although the supervisor is the superior of the employee. A **Role** is still assignable to any number of **Lanes** for highlighting which roles deal with tasks within a specific lane. Thereby, the relationship is on a m-to-n-basis.

With regards to the evaluation of Petia Wohed et al [17], the BPMN core supports direct allocation, role-based allocation, and organizational allocation. But the kind of support for the two latter mentioned patterns differs from the abilities expressed by the metamodel extension. Using BPMN, roles may be added as lane names and based on that allocation is possible. The extension allows a sophisticated view on the role concept as described above. Following, the allocation can be done in a more structured and economy closer way. Direct allocation is supported directly by naming the appropriate name after the specific resource. Organizational allocation is supported by the performers attribute of an activity which specifies the resources which may deal with a task. The resources entered there may be, amongst others, of the type of an organizational role. Since the entry is handled as simple string, the type is not recoverable from the process except by knowing which certain resource identifier belongs to which type. Based on these considerations, we identified direct allocation is supported sufficiently by the existing metamodel, whereas the other nine requirements needed additions to the metamodel.

Role-based allocation is clearer by using the **Role** entity because the distinction between **OrganizationalRole** and **FunctionalRole** is represented. Common role-based allocation bases on **FunctionalRoles**, whereas organizational allocation, i.e. aspects like superior of resource X, is comprised by the introduced **OrganizationalRole** and their hierarchy. Authorization is not covered directly by the metamodel, but each work item contains authorization constraints which resources must fulfill for being able to perform a specific task. Separation and binding of duties (patterns WRP-05 to WRP-07) are only considered by allocation expressions based on OCL which are attached to a **Task/WorkItem**. Capability-based allocation makes use of the profile introduced above.

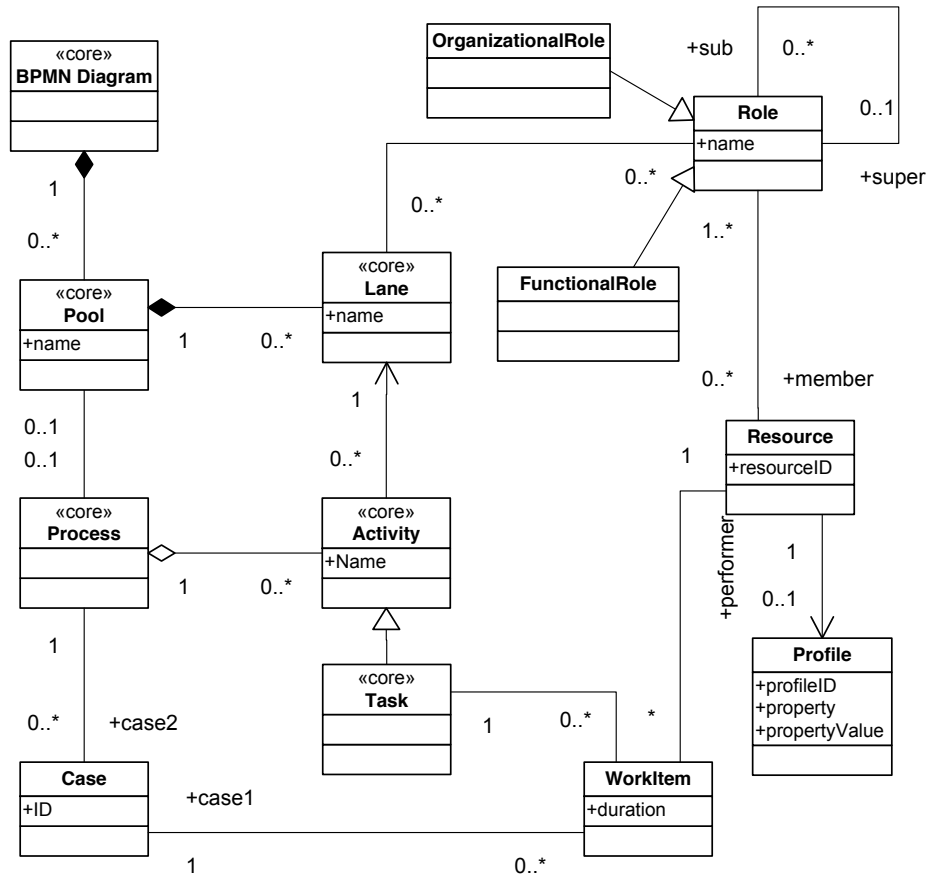


Fig. 2. An Extended BPMN Metamodel to Support Resource Allocation Constraints

The attribute value pairs are easy considerable by OCL-constraints within the allocation expressions. The *duration* attribute of class *WorkItem* can be exploited for cases of history based allocation. Organizational allocation, i.e. aspects like superior of resource X, is comprised by the introduced organizational role and their hierarchy.

4 Allocation Patterns as OCL Constraints and Their Relationship

Based on the extended metamodel introduced earlier, we examine how OCL constraints can be used to capture the semantics of the different allocation constraints discussed in Section 2. Moreover, we summarize our findings regarding the relationships we found between these allocation patterns.

4.1 Allocation Constraints

We exploit the sample business process in Fig. 1. When the allocation pattern is not suitable to the process, we offer example constraints with their specific context discussed briefly.

WRP-01 Direct Allocation In the context of process model in Fig. 1, it is required that only a single person is responsible for propagating the decision of the leave requests. The following OCL constraints ensures this by constraining the ID of the resource who executes this task to 101

```
context Task
inv directAllocation:
    self.Name = 'Propagate Decision' implies
self.workItem.performer->forAll(resourceID = 101)
```

WRP-02 Role-based Allocation As stated in the model, by placing the task 'Enter Leave Request' in 'Employee' Lane, it is required that only members of this role can execute this task. According to the OCL constraint,

```
context Task
inv roleBasedAllocation:
    self.Name = 'Enter Leave Request' implies
self.workItem.performer->forAll(
x | x.role->select(name ='Employee')->size(>0)
task 'Enter Leave Request' is allowed to be executed by performers (resources)
who are members of the role 'Employee'. So, performers who are not members
of this role i.e. role->select(name ='Employee')->size()=0, are not allowed
to perform this task.
```

WRP-04 Authorization According to the process model in Fig. 1, the task 'Process Leave Request' is meant to be executed by members of role 'Manager'. Imagine that in the organization, we have different disjoint management roles namely, 'First Line', 'Middle Management', and 'Top Management'. Only members of 'Middle Management' and 'Top Management' are allowed to execute this task.

```
context Task
inv roleLevelAuthorization:
    self.Name = 'Process Leave Request' implies
self.workItem.performer->forAll(x | x.role->
select(name ='Middle Management' or name='Top Management')->size(>0)
```

From the OCL constraint, we can see that it is an extended version of the constraint for the role based allocation, allowing performers of the task to be only members of the specified roles. We call this constraint a role level authorization

since at design-time, the designer can select from a set of roles those whose members are allowed to execute a task.

Similarly, a resource level authorization can take place when the designer can specify on a model a set of resources among which the runtime system can select one to execute the task. The OCL constraint to specify resource level authorization can be expressed as

```
context Task
inv resourceLevelAuthorization:
    self.Name = 'Process Leave Request' implies
self.workItem.performer->forAll(x|x.resourceID=1 or x.resourceID=50)
```

This type of resource level authorization can be handled as an extension of the direct allocation pattern.

WRP-05 Separation of Duty The separation of duty constraint is of great importance, especially in enforcing security issues in business policies. There are different levels of strictness of separation of duty constraints. Process level separation of duty, i.e. within each case every task must be executed by a unique resource, can be expressed as

```
context Case
inv processLevelDynamicSeparationOfDuty:
    self.workItem->forAll( wo1,wo2 | wo1.performer= wo2.performer
implies wo1 = wo2)
```

A less strict form of separation of duty is based on the task level, e.g. the case of tasks 'Comment on Leave Request' and 'Process Leave Request' in Fig. 1 are defined as mutual exclusive. This mutual exclusion or separation of duty can be either refined as either static, i.e. on the model level, or dynamic, i.e. on the instance level. Two tasks are statically exclusive if they must never be assigned to the same role or even if assigned to two different roles, these roles must be mutually exclusive. We can capture this formally by the following OCL constraint.

```
context Task
inv taskLevelStaticSeparationOfDuty:
    self.Name = 'Comment on Leave Request' implies
self.workItem.performer->forAll(x |
Task.allInstances()->select(Name = 'Process Leave Request')
.workItem.performer.excludes(x))
```

The above constraint states that whoever is executing the task 'Comment on leave Request' must not be any of those who can execute the task 'Process Leave Request'. Attaching this constraint to the `Task` class makes it globally applicable to all possible work items corresponding to different cases.

We can specify dynamic separation of duty between tasks by moving the constraint to the context of work item i.e. within the scope of a single case.

```

context WorkItem
inv taskLevelDynamicSeparationOfDuty:
    self.task.Name = 'Comment on Leave Request' implies
self.performer->forall(x |
self.case1.workItem->reject(wo | self = wo)->
select(task.Name='Process Leave Request').performer->excludes(x))

```

By assigning the scope to the `WorkItem` class, we can constrain performers of work items related to the task 'Comment on Leave Request' to be not the performer of a work item related to the task 'Process Leave Request' within the same case.

WRP-06 Case Handling Although case handling is a runtime allocation matter where you ensure that all tasks within the case are handled by the same resource, it is still possible to model this restriction on the model level by means of a constraint of the form

```

context Case
inv caseHandlingWithDirectAllocation:
    self.workItem->forall(performer.resourceID = 101)
The dynamic case handling situation would be expressed as
inv caseHandlingWithoutDirectAllocation:
    self.workItem-> forall(wo1 , wo2 | wo1.performer = wo2.performer)

```

In the first case, the designer can choose the specific resource to which the tasks in the model can be allocated. The latter case is a less restricted version where the engine can decide about the specific resource to be allocated but it has to be the same resource for all tasks.

WRP-07 Retain Familiar Retain familiar pattern can be seen as the opposite of the separation of duty pattern. Also, it could be named as binding of duty. Several levels of binding of duty can be identified in the same way as those for separation of duty. Considering the process level binding of duty, it appears that it is similar to case handling, so the constraint to express case handling as shown above fits. The task level binding of duty either statically or dynamically are expressed by the following two OCL constraints respectively.

```

context Task
inv tasklevelStaticOrBindOfDuty:
    self.Name = 'Propagate Decision' implies
self.workItem.performer->forall(x |
Task.allInstances->select(Name ='Comment on Leave Request')
.workItem.performer->includes(x))

```

```

context WorkItem
inv taskLevelDynamicBindOfDuty:
    self.task.Name = 'Propagate Decision' implies
self.performer->forall(x |
self.case1.workItem->reject(wo | self = wo)->
select(task.Name='Comment on Leave Request').performer->includes(x))

```

These statements represent the requirement that whoever has commented on the leave request is responsible to propagate the decision about this request.

WRP-08 Capability-based Allocation Capabilities of human resources can be thought of as the degree of education, the level of training, the number of years of experience, or anything else the organization would like to store about its workers in order to select them for task allocation based on their capabilities. In the metamodel of Fig. 2, the class `Profile` captures the capabilities of the different resources in form of two attributes, namely *'property'* and *'propertyValue'*. Imagine a task *'Develop Project Budget'* in a process model which requires to be assigned to a resource who has at least ten years of experience in this specific field of application. An OCL constraint catching this requirement can be stated as

```

context Task
inv capabilityAssignment:
    self.Name = 'Develop Project Budget' implies
self.performer->forall(x | x.profile->select(
property='YearsOfExperience' and propertyValue.oclAsType(Integer) >=
10))

```

Of course, there can be more than one way to capture capabilities of resources. The one, we introduced here, is simple but can still hold different capabilities for resources in an easy way.

WRP-09 History-based Allocation The specification of this pattern depends on how much information you can describe in the metamodel. In the following OCL constraint, we limit the set of available resources, from those the engine can select, to those, who have already executed this task in other cases with the least duration.

```

context WorkItem
inv historyBasedAllocation:
    self.Name = 'Process Leave Request' implies
self.workItem.performer->includesAll(self.process.case2.workItem->
select( wo1| WorkItem.allInstances->select(task.Name = 'Process Leave
Request')->
forall( wo2 | wo1.duration <= wo2.duration)).performer->asSet())

```

WRP-10 Organization-based Allocation The requirement, that only the supervisor of the employee, who entered the leave request, is allowed to comment on the leave request, can be expressed by the following constraint.

```
context WorkItem
inv organizationBasedAllocation:
    self.task.Name = 'Comment on Leave Request' implies
self.performer->forall(x | x.role.sub->includesAll(
Task.allInstances()-> select(Name='Enter Leave Request').workItem
->select( y| y.case1 = self.case1).performer.role->
select(name='Employee'))))
```

The above constraint restricts the performer of the task ‘Comment on Leave Request’ to be of a role that has the role ‘Employee’ as a subordinate to which the employee, who executed ‘Enter Leave Request’, belongs.

4.2 Relationship between Allocation Constraints/Patterns

We would like to summarize the relationships we could identify among the resource allocation patterns. Fig. 3 captures these relations. Within this figure, four types of relations are represented. Inheritance is represented by an arrow with a triangular head pointing to the parent pattern. The *extends* is represented with an arrow where the head points to the extended pattern. The extending pattern generalizes the extended pattern either by widening its selection criteria or by widening the context to which the constraint is allocated. Widening the selection criteria is for instance done in the case of ‘Role based Authorization’ *extends* ‘Role based Allocation’, whereas the case ‘Case handling with direct allocation’ *extends* ‘Direct Allocation’ widens the context by attaching “Direct Allocation” to the Case context. The relation *contradicts* is symmetric. Thus it is represented with arrow heads at both ends. Finally, relation *equals* is the opposite of contradicts.

5 Tool Support

In this section, we describe on the one hand the tools used to model and validate the various OCL constraints introduced earlier. On the other hand, we describe an extension of Oryx, a web-based BPMN modeling tool, to express resource allocation constraints.

5.1 OCL Validation

The OCL Environment [1] is a tool to develop UML Class diagrams, to express OCL constraints, to check their syntax, and to validate them. The tool was heavily used in the development and validation of the constraints discussed in Sect. 4. Via instantiating the metamodel in Fig. 2 by means of object diagrams, several use cases were designed to check the different allocation constraints. The tool was very helpful in enhancing the constraints, and testing the possibility

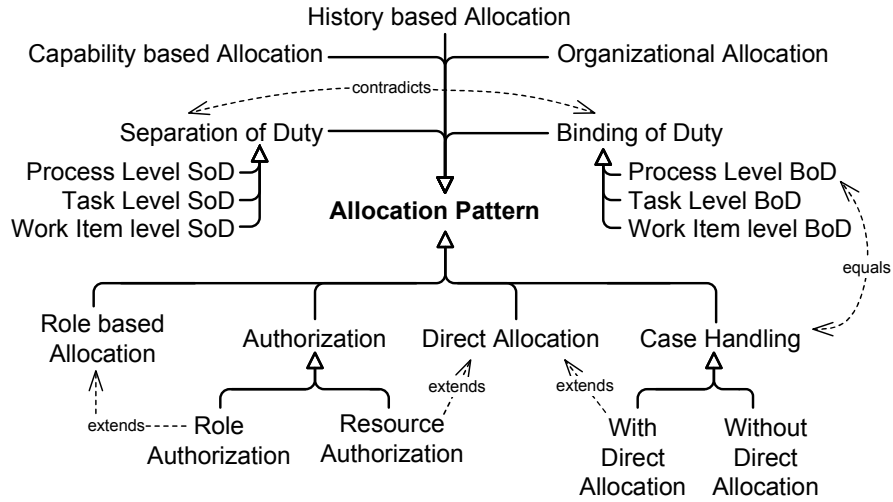


Fig. 3. Relationship Among Allocation Patterns

of merging more than one allocation constraint. Another tool, namely UML Specification Environment USE [14], was co-used with OCL to generate the test cases for the constraints.

5.2 BPMN Modeling

Oryx [10]² is an open source web based modeling tool. In addition to modeling BPMN in Oryx, users can check models for errors, transform the models to Petri nets, and simulate their models. We have developed a plug-in for Oryx through which we give the modeler the chance to define allocation constraints as discussed earlier in an easy way. Since OCL constraints need significant background to be expressed correctly, we have embedded the various constraints discussed above in a wizard driven user interface. First of all, the user selects the task to which he/ she wants to apply resource allocation patterns. Later, the user selects from a list the type of pattern to be applied. Let us imagine that he/ she selected the separation of duty pattern. In the second form, the user selects the set of exclusive tasks to the one selected before. After pressing the OK button, the respective allocation constraint is added to the **Performers** property of the task. To remind the modeler that this task has an associated allocation constraint, a text annotation is automatically associated to the task with information about the allocation constraint. We have chosen to store the allocation constraint in the **Performers** property of the task because it was dedicated by the BPMN specification to list those, who are eligible to execute the task (see [3] page 53).

² www.oryx-editor.org

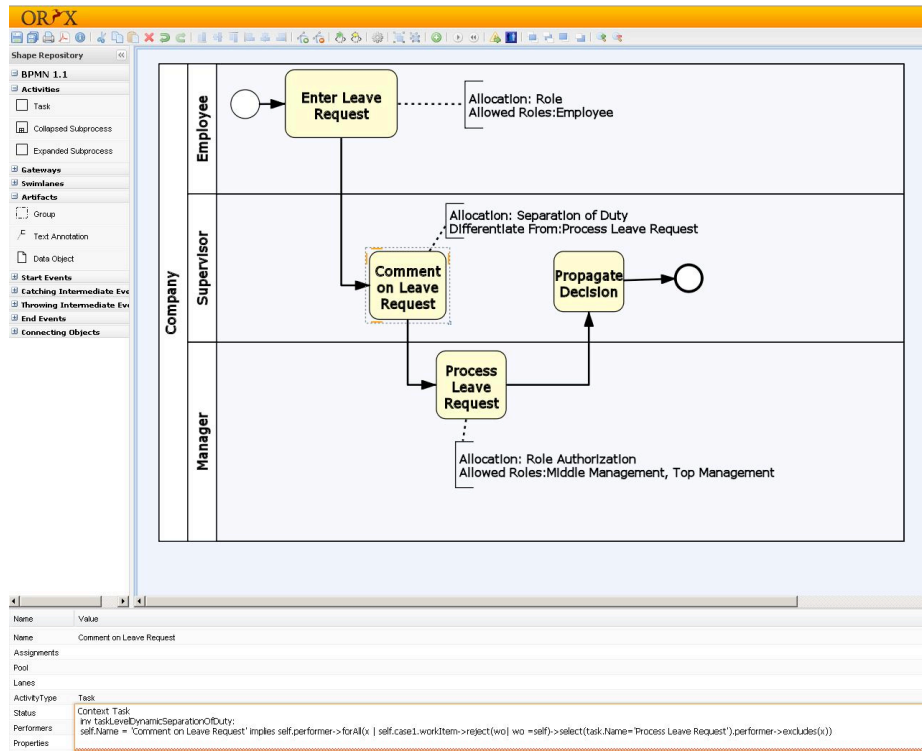


Fig. 4. A Snapshot of Oryx with Resource Allocation Constraints

6 Related Work

Enforcement of access control policies on workflow has been discussed a long time ago. In [8], the authors argue that specifying static separation of duties between roles in RBAC is insufficient in the area of workflow management systems. They provide a formal approach to model dynamic separation of duties as well as static ones. A closely related work on task level separation of duty specification is discussed in [9]. A reference model to describe how tasks in a workflow can be distributed among team members was introduced in [16]. In that approach, a mixed metamodel including workflow aspects and organizational aspects was introduced. In addition, OCL was used to express both correctness criteria for models and business constraints i.e. correctness criteria for instance level. As a reference model, it touched a set of general concepts in work distribution like the separation of duty constraints. In our work, we are oriented to extend BPMN to enable resource allocation requirements. We took the workflow resource patterns from [15] as our requirements framework. As shown, we were able to discuss different variants of these allocation patterns.

With the appearance of BPEL, several approaches studied possibilities of enforcing security issues in form of access control on BPEL processes. In [12], distributed and dynamic issues of access control are discussed with methods to

reason about dynamic role assignment to users. RBAC-WS-BPEL is an extension introduced in [7] to join WS-BPEL with an authorization model to enable specification of separation of duty constraints on the level of BPEL processes. With the introduction of BPEL4People [5], a recent approach [13] examined the different ways to model separation of duty constraints.

An approach to visually represent authorization constraints in BPMN was discussed in [18]. This approach discusses formalization and modeling of task-based authorization constraints in BPMN. Through identifying the set of conflicting tasks in a process model, the authors were able to express allocation constraints, like separation of duty between tasks based on their execution order. Comparing to our approach, we do not need the information of execution ordering between the two conflicting activities. Expressing a separation of duty constraint as an OCL constraint guarantees that it must hold during the lifetime of the process instance. Moreover, with our approach, stricter types of separation of duty could be expressed, for instance process level separation of duty.

An approach to avoid processes from deadlocking due to insufficient resources is introduced in [11]. This approach computes the minimal number of required resources in order to guarantee deadlock free execution of the process.

One common remark to approaches related to controlling resource assignments is that they all stem from a security background. For instance, an approach to select a resource based on his/ her experience or based on history of executions were not addressed at all in these approaches. In our approach we have addressed such patterns and presented a way to capture them.

7 Conclusion

An approach to enable the expression of resource allocation constraints in BPMN process models was introduced. The resource allocation patterns were used as the requirements framework to extend the BPMN metamodel. To formally express these allocation patterns, OCL was used as an expression language for these patterns. With OCL expressiveness and the developed metamodel, we were able to cover different variants of resource allocation patterns. We could also identify three types of relationships between the variants of resource allocation patterns. A BPMN modeling tool, Oryx, was extended with a user friendly approach to express resource allocation constraints. As the user selects the required pattern to apply, the tool automatically appends the appropriate OCL expression to the required task(s). With this, we have enabled an easy to use approach to express complex constraints.

Mapping from BPMN resources enabled models into executable processes such as BEPL4People and WS-HumanTask specifications is considered as future work step. Moreover, with more than one allocation constraint attached to a single task, it becomes necessary to analyze the possibility of conflicts between them before generating executable processes.

References

1. OCLE: Object Constraint Language Environment, 2003.
2. Object Constraint Language OCL 2.0 Specification. Technical report, OMG, 2005.
3. Business Process Modeling Notation (BPMN) 1.1 Specification. Technical report, OMG, 2008.
4. A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. Knig, F. Leymann, R. Mller, G. Pfau, K. Plsler, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller. *Web Services Human Task (WS-HumanTask), Version 1.0*. June 2007.
5. A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. Knig, F. Leymann, R. Mller, G. Pfau, K. Plsler, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller. *WS-BPEL Extension for People (BPEL4People), Version 1.0*. 2007.
6. M. Amend, A. Barros, M. Benitez, M. Chapman, M. Das, R. Day, D. Frankel, D. Ings, P. Irassar, O. Kieselbach, M. Kloppmann, J. Koehler, F. M. Kraft, F. Leymann, J. Mischkinsky, R. Mueller, K. Ploesser, M. Rowley, S. Samoojh, V. Saxena, M. Srinivasan, I. Trickovic, H. Voelzer, F. Weber, , and S. A. White. Draft proposal for: Business process model and notation (bpnm) specification 2.0 v0.5.2, 2008.
7. E. Bertino, J. Crampton, and F. Paci. Access control and authorization constraints for ws-bpel. *Web Services, IEEE International Conference on*, 0:275–284, 2006.
8. E. Bertino, E. Ferrari, and V. Atluri. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. *ACM Trans. Inf. Syst. Secur.*, 2(1):65–104, 1999.
9. R. A. Botha and J. H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3), 2001.
10. G. Decker, H. Overdick, and M. Weske. Oryx - an open modeling platform for the bpm community. In M. Dumas, M. Reichert, and M.-C. Shan, editors, *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pages 382–385. Springer, 2008.
11. M. Kohler and A. Schaad. Avoiding Policy-based Deadlocks in Business Processes. In *ARES*, pages 709–716. IEEE Computer Society, 2008.
12. H. Koshutanski and F. Massacci. Interactive Access Control for Web Services. In *In Proceedings of the 19th IFIP International Information Security Conference (SEC 2004)*, pages 151–166. Kluwer Press, 2004.
13. J. Mendling, K. Ploesser, and M. Strembeck. Specifying Separation of Duty Constraints in BPEL4People Processes. In W. Abramowicz and D. Fensel, editors, *BIS*, volume 7 of *Lecture Notes in Business Information Processing*, pages 273–284. Springer, 2008.
14. M. Richters. The USE Tool: A UML-based Specification Environment, 2001.
15. N. Russell, A. H. M. T. Hofstede, and D. Edmond. Workflow resource patterns: Identification, representation and tool support. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232. Springer, 2005.
16. W. M. P. van der Aalst and A. Kumar. A Reference Model for Team-enabled Workflow Management Systems. *Data Knowl. Eng.*, 38(3):335–363, 2001.
17. P. Wohed, W. M. P. van der Aalst, M. Dumas, A. T. Hofstede, and N. Russell. On the Suitability of BPMN for Business Process Modelling. In *Proceedings 4th International Conference on Business Process Management (BPM 2006)*, LNCS, Vienna, Austria, 2006. Springer Verlag.
18. C. Wolter and A. Schaad. Modeling of Task-Based Authorization Constraints in BPMN. In G. Alonso, P. Dadam, and M. Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2007.