Dynamic Aspects of People Allocation in BPEL4People

Andreas Meyer, Ahmed Awad, and Mathias Weske

Hasso Plattner Institute, Potsdam, Germany
{andreas.meyer,ahmed.awad,mathias.weske}@hpi.uni-potsdam.de

Abstract. BPEL4People and WS-HumanTask provide opportunities to specify which person shall perform the work determined by a certain task. These standards are based on the workflow resource patterns, against which they have been evaluated earlier. However, dynamic aspects especially like ownership changes or modified task execution with regard to work skipping during process execution has not been examined. We provide these considerations and appropriate implementation details for all allocation and constraint patterns focusing on delegation aspects, the most often used dynamic aspect. Moreover, we provide an implementation of organizational and capability-based allocation patterns.

1 Introduction

With the advent of service oriented architecture (SOA) many languages were proposed by different vendors and standardization committees to orchestrate the execution of disparate web services for the sake of realizing a business goal. Currently, BPEL [1] is the de facto standard language to describe orchestrations of web services. The language was designed to execute process models in which all steps are automated.

BPEL4People [2] and WS-HumanTask [3] extended the BPEL standard to support human tasks in a workflow. Therefore, the both extensions introduce so-called logical people groups as means to reflect roles and people assigned to roles in a BPEL process specification. Based on these logical people groups, it is possible to have different people assignments to a human task within the BPEL process. In [4, 5], an evaluation of the new extension was discussed based on the workflow resource patterns [6]. In [4], Russel and van der Aalst also provide concrete examples how to implement the supported patterns using these new extensions. This evaluation has also shown that BPEL4People and WS-HumanTask face limitations regarding the support of some allocation patterns, e.g., organizational and capability-based allocation. Additionally, dynamic aspects of such task allocations as well as task delegations during run-time were not considered.

In this paper, we focus on these dynamic aspects and revisit all creation patterns from this point of view. We distinguish the creation patterns from [6] into allocation and constraint patterns, where all patterns dealing with resource assignment as role-based and organizational allocation belong to the first group

of patterns and the others like separation of duties and authorization are considered as constraint patterns. We provide proposals to support allocation patterns having been stated as "not supported" or "partially supported" in [4, 5], to increase the flexibility in task allocation in BPEL. Besides the allocation patterns, we highly concentrate on the constraint patterns: separation and binding of duties as these are closely related. Thereby, binding of duties refers to retain familiar in [6]. The proposals are mainly based on already existing constructs of BPEL, BPEL4People, or WS-HumanTask.

In the upcoming section, we provide a business process motivating the usage of the different allocation as well as constraint patterns. In Section 3, we discuss our proposals regarding the patterns mentioned in Section 2. This discussion is meant to scope the patterns we are addressing. Afterwards, we present a short discussion of related work followed by the paper's conclusion in Section 5.

2 Motivating Example

The travel request scenario in Figure 1 comprises many important allocation and constraint patterns. First, an employee files a travel request, which will be processed by a manager, who must not be the same person as the one filing the request. The manager decides whether the request is granted or declined. Following, either the employee gets informed by the manager about the declination or the administrative assistant takes care of the travel reservations. Afterwards, this assistant provides all travel details to the employee. The actual allocation is based on roles for employee and manager and on organizational aspects for the manager and the assistant as these two persons rely on the employee who files the request directly (manager) or indirectly (assistant). Additionally, the travel request processing and travel arrangement tasks might be delegated to the defined persons. All other tasks are implicitly not delegateable.

Figure 1 shows that even in such small business processes a variety of different allocation approaches is in use. All but four approaches stated in [4] are comprised in the example. The missing allocation types are history- as well as capability-based allocation, direct allocation, and automatic execution. However, they are used in different scenarios. In industry, it is common to restrict the execution of specific tasks to persons having certain skills (capability awareness) - for instance only someone able to speak Spanish language is allowed to deal with the insurance claim of a Spanish customer. Additionally, history information provides insights about a person's experiences and therefore, critical tasks can be allocated to persons with much experience. The usage of direct allocation is obvious if there is exactly one person who shall be allowed to perform a certain task and automatic execution does not depend on any people support.

In today's businesses, separation of duties and delegation are very common and frequently used constructs, e.g., to ensure compliance to regulations or to provide flexibility to keep processes running, if the chosen person is not able to finish an activity in time. Binding of duties and case handling are used to avoid context switches and therefore increase efficiency. Consequently, we decided to

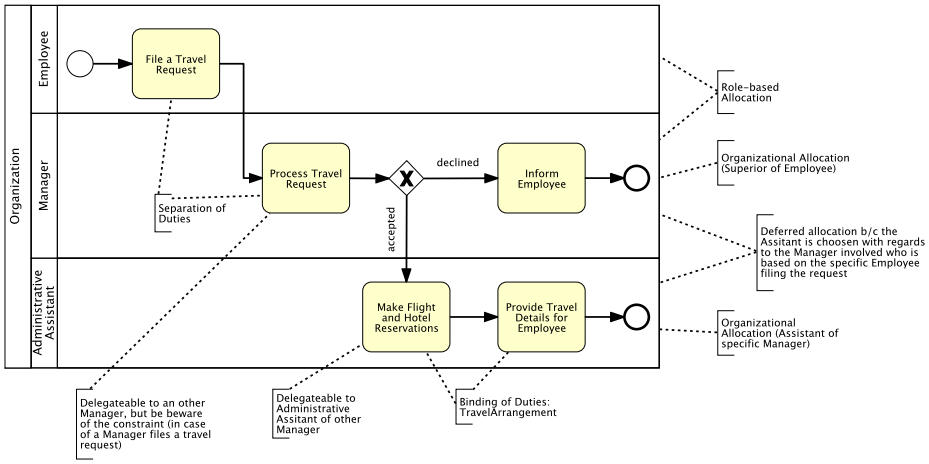

Fig. 1. Travel Request Scenario

consider all allocation patterns, the three constraint patterns (case handling, separation of duties, and retain familiar/ binding of duties) and delegation from [4].

3 Patterns Reflection

We revisit the patterns stated in the previous section. Following, on the one hand we discuss the patterns needing additional considerations regarding dynamic aspects and on the other hand we propose implementation details for currently not in BPEL supported patterns. We omit history-based allocation from our discussion as neither BPEL nor one of its stated extensions provide any opportunity to include execution data from other than the current process instance. However, the `search by` statement might be used to deal with intra-instance historical data. Therefore, providing access to the data of all instances would lead to a straightforward support of history-based allocation. However, in the WS-HumanTask specification, there is no sufficient description about the purpose of the `search by` statement. Additionally, we exclude automatic execution from our upcoming considerations because this allocation approach utilizes BPEL's core competencies by using web services or different machines for process execution instead of persons as performers.

During process execution, delegation is used very often to share workload, for instance to meet stated deadlines. In the scenario above, the assistant is able to delegate the travel preparation to another assistant if she is completely busy so that the employee gets the necessary travel information and bookings in time. During the reflection of the stated patterns, we identified many of them which are heavily affected by delegation. The main challenge is to make sure that delegation at run-time does not conflict with rules and compliance statements specified at design-time of the process. This includes, for instance, separation of

duties constraints. Therefore, the specification of delegation rules must be put into each people assignment statement, i.e., direct allocation, role-based allocation, deferred allocation, capability-based allocation, history-based allocation, and organizational allocation, and all constraint statements, i.e., authorization, separation of duties, retain familiar, case handling. Adding this information of to whom delegation is allowed at run-time ensures meeting the defined rules, if the potential delegatees are specified carefully during design-time. Listing 1 provides the general syntax to declare delegation. Besides "potentialOwners", the values for *potentialDelegatees* might be "none" or a specific logical people group as well as a specific person.

```
1 <htd:delegation potentialDelegatees="potentialOwners" />
```

Listing 1. Delegation

The authorization pattern can be seen either as another role-based allocation approach or as an access management system to deal with access rights to a certain task or to data objects processed by a task. The authorization pattern will remain untouched, because there are no possibilities within the given BPEL or BPEL extension constructs to integrate this pattern as introduced in [6]. Therefore, it remains partly supported regarding [4].

3.1 Separation of Duties

We are concerned with dynamic task-based separation of duties. That is, two tasks must not be executed by the same person in the same process instance. Referring to the separation of duties constraint in Figure 1, it is not allowed that the person who files a travel request processes it. One might argue that the task "File Travel Request" being in the lane of an "Employee" role and the task "Process Travel Request" being assigned to the role "Manager" guarantees a static separation of duties. However, it is possible that a middle manager is a member in both roles. Thus, there has to be an explicit enforcement of separation of duties for the two tasks.

In Listing 2, the separation of duties constraint is enforced by means of evaluating the `htd:getActualOwner` method for the instance of the "File Travel Request" task and populating the list of *excluded owners* of the "Process Travel Request" task with that value. However, due to the dynamic nature of human resources and to the delegation possibility, the person who claims the "Process Travel Request" task might delegate it to the person who already executed the "File Travel Request" task, thus, violating the constraint. At that time, a BPEL4People compliant engine will not prevent this delegation since the WS-HumanTask specification states that the lack of an explicit delegation element allows the delegation to anyone in the organization's people directory. To prevent any chance of a violation to the separation of duties, we explicitly add a delegation element to the declaration of the "Process Travel Request" task, see line 30.

```
<htd:task name="File Travel Request">
 <htd:peopleAssignments>
  <htd:potentialOwners>
   <htd:from logicalPeopleGroup="Employee"/>
  </htd:potentialOwners>
  </htd:excludedOwners>
   <from>
    htd:getActualOwner("Process Travel Request")
   </from>
  </htd:excludedOwners>
 </htd:peopleAssignments>
 <htd:delegation potentialDelegatees="potentialOwners">
  <from>
   htd:except(htd:getLogicalPeopleGroup("Employee"),
   htd:getActualOwner("Process Travel Request"))
  </from>
 </htd:delegation>
</htd:task>
<htd:task name="Process Travel Request">
 <htd:peopleAssignments>
  <htd:potentialOwners>
   <htd:from logicalPeopleGroup="Manager"/>
  </htd:potentialOwners>
  </htd:excludedOwners>
   <from>
    htd:getActualOwner("File Travel Request")
   </from>
  </htd:excludedOwners>
 </htd:peopleAssignments>
 <htd:delegation potentialDelegatees="potentialOwners">
  <from>
   htd:except(htd:getLogicalPeopleGroup("Manager"),
   htd:getActualOwner("Process Travel Request"))
  </from>
 </htd:delegation>
</htd:task>
```

Listing 2. More Cautious Enforcement of Separation of Duties

Additionally, it is not guaranteed that the tasks, belonging to one separation of duties dependency group, are executed in sequence - sequential, parallel, as well as exclusive executions are possible. Thus, in case where tasks "A" and "B" are executed in parallel, it is easy to break the constraint, if the restriction to exclude performers of "A" is checked at allocation of "B" but not also during allocation of "A" while initiating task "B" and following, allocation for "B" takes place first. In this case, the function `htd:getActualOwner("A")` will result in an empty `htd:user` value and leads to allocation of "B". Moreover, the allocation procedure for task "A" would have no restrictions to follow. Therefore, it is possible to allocate it to the same person who was allocated to task "B". The mapping shown in Listing 2 resolves this issue generally by replicating the allocation restrictions on each task declaration.

3.2 Binding of Duties

Binding of duties is conceptually the opposite of separation of duties, i.e., the same person needs to perform all tasks associated to this one dependency group for saving context switching times to raise efficiency or with regards

to restrictions like determining one person as single contact person to a customer. This requirement is realized in BPEL via the X-Path extension function `getActualOwner` to identify the actual person dealing or rather have dealt with a specific task. This approach is proposed in [4]. However, delegation opportunities make this pattern much more complex and lead to three different delegation approaches. First, delegation might be forbidden to ensure that the person, the tasks were allocated to, really does perform these tasks (Listing 1 with the value "none" for potential delegatees).

Disallowing delegation has drawbacks. In industry, it is hard to ensure that nothing will change during process execution. Assume, a certain amount of tasks of one process are allocated to one specific person and after executing the first task, this person is temporary unavailable. Nevertheless, the process is time-critical. Therefore, execution cannot be suspended until the person gets back to work. Somebody else has to take over, but with regards to the process definition, delegation is not allowed as stated earlier. One might argue that escalation handling should take place to overcome this issue, but the question regarding follow-up tasks remain. Escalation of each of these tasks is a possible behavior but abuses the sense behind escalation. The handling of situations of unavailability should be defined beforehand. Besides being absent, a person might get into timing problems for unforeseen reasons and need to ask for help. Giving the person the chance to delegate the task herself allows her to hand it over to an in-mind appropriate person [7] instead of calling the escalation handler.

Allowing delegation for binding of duties leads to two different behaviors from which one is chosen at design-time to be applied. Delegating the current task either results in delegating all upcoming tasks as well or delegating just this one task and performing all upcoming tasks herself. Assuming binding of duties is mainly used for avoiding context switches, choosing the first option leads to a better result as after delegating the current task, the new "actual owner" will get into the area of application and the former owner does not need to take over again, i.e., a second context switch is avoided. Therefore, this option might be the one to go for. However, from the designer's point of view or organization specific regulations and the following task allocation, there might be some reason that this person needs to deal with as many of the tasks as possible but may be allowed to delegate single tasks for some reasons. Following, both options should be available besides the "delegation forbidden" option. However, implementation of both approaches deals with certain issues. First, we will provide details for delegating the current and all upcoming tasks followed by a solution for delegating only one specific task and keep the actual owner for all others.

Referring to the travel request in Figure 1, the two tasks in the third lane shall be executed by the same person. Therefore, the appropriate dependency is determined and both tasks belong to the same dependency group (set) named `TravelArrangement`. In more complex scenarios, there might exist overlapping dependency groups. If so, these will be merged into one dependency group set. Altogether, we assume consistent dependency group definitions during process design-time, i.e., dependencies that do not lead to execution deadlocks [8].

Our approach to realize the delegation of all tasks beginning from the current one onward is the introduction of a new X-Path function named `getLastActualOwnerFromSet`. Thus, we will allow to query for the actual owner of the last task being enabled from the binding of duties set, the current task is part of: in our example the set `TravelArrangement`. Using the function's result, the appropriate person can be allocated to the current task.

The X-Path function takes a set of tasks as input and parses through the complete set reading the timestamp of task allocation to a person of each task. If a task does not contain such a timestamp, it is ignored in further calculations. If there is not any valid timestamp found in the complete set, the function returns an "empty user" - comparable to the existing X-Path function `getActualOwner` - and one of the potential owners is chosen for allocation. After the timestamp identification, the newest timestamp is selected and for the appropriate task, the X-Path function `getActualOwner` is applied. Subsequently, the actual owner of the task being enabled latest before the current one will be returned and allocated accordingly. The timestamp can be obtained by calling the `htd:getTaskHistory` operation which returns a list of task events associated with their timestamps.

Using this function, the facts of BPEL's "allocation on enablement" and strict sequential execution ensure that each newly enabled task gets allocated the actual owner of the task being enabled before, if any, considering probable delegations. An example allocation presenting the aforementioned approach is given in Listing 3. If there exists an earlier allocation for a task in the same binding of duties set (`TravelArrangement`) as the current task, the actual owner of this last task is assigned to the current task. Otherwise, a person from the set of potential owners is chosen for allocation. Delegation is allowed to all persons being an administrative assistant.

```
1  <htd:task name="Provide Travel Details for Employee">
2   <htd:peopleAssignments>
3    <bpel:if>
4     <bpel:condition>
5      htd:getLastActualOwnerFromSet("TravelArrangement") != emptyUser
6     </bpel:condition>
7     <bpel:then>
8      <htd:potentialOwners>
9       <htd:from>
10       htd:getLastActualOwnerFromSet("TravelArrangement")
11      </htd:from>
12     </htd:potentialOwners>
13    </bpel:then>
14    <bpel:else>
15     <htd:potentialOwners>
16      <htd:from logicalPeopleGroup="Administrative Assistant"/>
17     </htd:potentialOwners>
18    </bpel:else>
19   </bpel:if>
20  </htd:peopleAssignments>
21  <htd:delegation potentialDelegatees="Administrative Assistant" />
22 </htd:task>
```

Listing 3. Binding of Duties Using an X-Path Extension

Turning the focus to the second approach to delegate the current task only, we propose a straightforward solution. We use the firstly enabled task of the binding of duties set as reference and allocate the actual owner of this task to all other tasks belonging to the same set. Admittedly, this approach has two drawbacks. First, the initial task needs to be known. Secondly, some limitations appear for this first task compared to all other tasks of the set, which can be affected freely by all control flow or resource changes like skipping and delegation, whereas the first task must be executed completely and must not be delegated to any other person. Listing 4 provides the BPEL code for any but the first task of the `TravelArrangement` set. For the first task, the `potentialOwners` tag needs to refer to a specific logical people group and the value of `potentialDelegatees` needs to be changed to "none" in the `delegation` tag. Additionally, in the `peopleActivity` tag surrounding the `task` tag, `isSkipable` must be set to false.

```
<htd:task name="Provide Travel Details for Employee">
 <htd:peopleAssignments>
  <htd:potentialOwners>
   <htd:from>
    htd:getActualOwner("Make Flight and Hotel Reservations")
   </htd:from>
  </htd:potentialOwners>
 </htd:peopleAssignments>
 <htd:delegation potentialDelegatees="Administrative Assistant" />
</htd:task>
```

Listing 4. Binding of Duties - Delegation of Current Task Only

3.3 Case Handling

Case handling is not supported by BPEL4People and WS-HumanTask as already stated in [4]. This issue relates to the fact that BPEL only allows "allocation on enablement", i.e., allocation of all tasks of a specific case at creation of this case as specified by the case handling pattern introduced in [6] is out of scope for BPEL. The resulting effect of case handling can be achieved by relaxing the constraint "allocation on case creation" to "all tasks associated to the case have been executed by the same resource (or resource pool) after case completion". Based thereon, case handling is a special binding of duties implementation. Furthermore, there exists another approach to introduce this part of case handling into BPEL. WS-Human Tasks provides "Composite Tasks" which allow the definition of any number of contained subtasks. The case itself can be mapped to the composite task and all tasks associated to the case are added as subtasks. By setting the value for instantiation to automatic, it is achieved that all defined allocations for the subtasks are performed at composite task's creation, i.e., case's creation. This approach has one main limitation: All subtasks are either strictly sequential or strictly parallel. Subsequently, control flow would not be definable within cases. However, considering the case handling notion the Workflow Patterns initiative used in [6], exactly one person will execute the complete case. Therefore, it is logical to either give this person free choice of task execution

order as he should know best what to do when or force him to follow a defined sequence.

Van der Aalst et al. [9] introduced a new case handling approach in 2005 which also allows different persons to execute tasks within one case. Additionally, they added a distinction between three types of roles which can be assigned to a task simultaneously: the execute role, the skip role and the redo role. The skip role was empowered to decide whether the specific task is skipped. Accordingly, the redo role has the power to decide about repeating tasks within the case. Regarding the execute role and the possibility of more than one executor, the proposed composite task approach works fine as each subtask can get allocated a different person as task owner. Evaluating the other two roles leads to some issues. Generally in BPEL, only the executor role is assigned to a task (or subtask) and this one has all rights to deal with task management and flow issues, i.e., the executor role can decide about skipping a role - if skipping is generally allowed through the process description. Redo is completely out of scope in the BPEL standard [4].

Putting these facts together, BPEL4People and WS-HumanTask do not support case handling in the form introduced in [9] but as specialization of binding of duties with forbidden delegation or by utilizing composite tasks in conjunction with dropping the control flow information. The latter two approaches comply to the definition given by the Workflow Patterns initiative in [6].

3.4 Capability-Based Allocation

The capability-based allocation pattern distributes work to a user based on a skill that she possesses. In [4], the authors indicated that BPEL4People does not support that pattern. However, we argue that capabilities can be supported by using the `parameter` element of a logical people group. For instance, in a logical people group specifying employees, we can define the argument "Language" to list the languages an employee can speak. It is generally possible to define parameters with arbitrary data types, e.g., integers, which might be utilized to differentiate the level an employee fulfills regarding a specific skill. Consequently, it is possible to perform filtering with complex conditions using appropriate X-Path operators or functions.

3.5 Organizational Allocation

Similarly to capability-based allocation, organizational allocation can be addressed by adding `parameter` elements in the definition of logical people groups corresponding to the appropriate organizational roles. For instance, we can add a "Department" parameter to reflect the organizational department to which an employee belongs.

To model hierarchical dependencies among roles, we can *flatten* the hierarchy within logical people group definitions. For instance, consider the "Employee" and the "Manager" roles shown in Figure 1. Assume that there is a hierarchical relationship between the managers and the employees. We can define the

three logical people groups “Employee”, “Manager” and “EmployeesManager” as follows:

```
<htd:logicalPeopleGroup name="Employee">
 <htd:parameter name="Manager" type="htt:user" />
</htd:logicalPeopleGroup>

<htd:logicalPeopleGroup name="Manager">
</htd:logicalPeopleGroup>

<htd:logicalPeopleGroup name="EmployeesManager">
 <htd:parameter name="Employee" type="htt:user" />
</htd:logicalPeopleGroup>
```

Listing 5. Flattening Organizational Hierarchy

For the “Employee” people group, we defined the parameter “Manager” so that we can find employees working under control of a specific manager. The people group “EmployeesManager” is the means to flatten the relationship between a manager and his employees. The group has the parameter “Employee” so that we can filter for the employees a specific manager is responsible for. Another possibility to provide this behavior would be to drop the “EmployeesManager” group and add a parameter “Employee” in the “Manager” group. In this case, the parameter would be of a complex user-defined type, e.g., “ManagedEmployees”.

Based on the first design choice to flatten the hierarchy, Listing 6 shows how the “Process Travel Request” task is allocated to the manager of the employee who executed “File Travel Request”.

```
<htd:task name="Process Travel Request">
 <htd:peopleAssignments>
  <htd:potentialOwners>
   <htd:from logicalPeopleGroup="EmployeesManager">
    <htd:argument name="Employee">
     htd:getActualOwner("File Travel Request")
    </htd:argument>
   </htd:from>
  </htd:potentialOwners>
 </htd:peopleAssignments>
</htd:task>
```

Listing 6. Organizational Allocation

A limitation of the hierarchy flattening, the way shown above, is that it supports traversal of one level only. To support multiple level traversal, it is necessary to build complex types for the different depth of the hierarchy and a logical people group with a parameter of that complex type.

4 Related Work

All considerations in this paper are based on the workflow resource patterns derived in [6] and the opportunities arising from the BPEL4People [2] and WS-HumanTask [3] extensions to BPEL [1]. Russel and van der Aalst evaluated these new extensions against the workflow resource patterns in [4, 5] and provided BPEL expressions for all supported patterns. They focused on a pattern

by pattern analysis. We expanded this view by considering dynamic changes during process execution and extending their expressions accordingly. We also introduced new proposals to raise BPEL's coverage of patterns where appropriate.

Mendling et al. provided an almost exhaustive view on separation of duties constraints and determined many different types of separation of duties [10]. Only considerations regarding the handling of delegation or task reorder during process execution have not been addressed.

In [9], the authors introduced a new approach of case handling connecting three different types of roles to each task of the case. These roles need not to remain consistent through the whole case. Therefore, similarities to the case handling described in [6] are very low.

In [11], the authors propose a model-driven approach to separate the modeling of human resource allocation on the conceptual model from the technical implementation in BPEL4People and WS-HumanTask. Following a view-based approach, the authors introduce a human view on business processes where business experts can express role-based assignments of tasks. Using a model-to-code transformer, the authors generate BPEL4People code out of the human view. Unfortunately, the authors neglect defining task delegation constraints at the human view. Thus, our contribution in this paper could be integrated with the work in [11] to provide more control on resource behavior at the process design phase.

5 Conclusion

In this paper, we revisited most of the allocation and constraint patterns as well as the delegation pattern from [6] regarding dynamic aspects as people allocation and task delegation during process execution using BPEL including the both extensions BPEL4People and WS-HumanTask as de facto business standard. This includes especially an explicit statement of delegation possibilities and potential delegatees.

With regards to [4], three allocation patterns cannot be expressed in BPEL: history-, capability-based and organizational allocation. Therefore, we proposed approaches based on existing constructs from BPEL and its extensions. History-based allocation may utilize the `searchBy` statement if access to other instances of the same or even other processes is provided and the other two mainly rely on `parameters` to filter existing information.

All constraint patterns need additional information in the expression to make sure that the constraints hold after changes in the process execution , e.g., task order and actual owner of a task. In separation of duties constraints, all dependent tasks need to be stated explicitly for all tasks part of this separation of duties dependency group. For binding of duties (retain familiar), the analogous information needs to be stated for each affected task as well. Additionally, the introduction of the new X-Path function `getLastActualOwnerFromSet()` eases the usage of this pattern. The function bases on already existing constructs and

information available. Case handling in the sense of van der Aalst et al. [9] is not manageable, but a relaxation of the requirements made in [6] leads to an implementation approach using composite tasks. Subsequently, case handling as a specialization of binding of duties can be expressed in BPEL4People and WS-HumanTask.

References

1. Alves, A., Arkin, A., Askary, S., Barreto, C., Block, B., Curbera, F., Ford, M., Goland, Y., Guizar, A., Kartha, N., Liu, C., Khalaf, R., Knig, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0. Technical report, OASIS (April 2007)
2. Clément, L., König, D., Mehta, V., Mueller, R., Rangaswamy, R., Rowley, M., Trickovic, I.: WS-BPEL Extension for People (BPEL4People) Specification Version 1.1. (2010) Committee Draft 07 / Public Review Draft 02.
3. Clément, L., König, D., Mehta, V., Mueller, R., Rangaswamy, R., Rowley, M., Trickovic, I.: Web Services ? Human Task (WS-HumanTask) Specification Version 1.1. (2010) Committee Draft 07 / Public Review Draft 02.
4. Russell, N., van der Aalst, W.: Evaluation of the BPEL4People and WS-HumanTask Extensions to WS-BPEL 2.0 using the Workflow Resource Patterns. BPM Center Report BPM-07-10 (2007)
5. Russell, N., van der Aalst, W.M.P.: Work distribution and resource management in bpel4people: Capabilities and opportunities. In Bellahsene, Z., Léonard, M., eds.: CAiSE. Volume 5074 of Lecture Notes in Computer Science., Springer (2008) 94–108
6. Russell, N., Ter Hofstede, A.H., Edmond, D., van der Aalst, W.: Workflow Resource Patterns. Technical report (2004)
7. van der Aalst, W.M.P., Song, M.: Mining social networks: Uncovering interaction patterns in business processes. In Desel, J., Pernici, B., Weske, M., eds.: Business Process Management. Volume 3080 of Lecture Notes in Computer Science., Springer (2004) 244–260
8. Kohler, M., Schaad, A.: Avoiding Policy-based Deadlocks in Business Processes. In: ARES, IEEE Computer Society (2008) 709–716
9. van der Aalst, W.M., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data & Knowledge Engineering **53**(2) (2005) 129 – 162
10. Mendling, J., Ploesser, K., Strembeck, M.: Specifying Separation of Duty Constraints in BPEL4People Processes. In Abramowicz, W., Fensel, D., eds.: BIS. Volume 7 of Lecture Notes in Business Information Processing., Springer (2008) 273–284
11. Holmes, T., Tran, H., Zdun, U., Dustdar, S.: Modeling human aspects of business processes — a view-based, model-driven approach. In: ECMDA-FA '08: Proceedings of the 4th European conference on Model Driven Architecture, Berlin, Heidelberg, Springer-Verlag (2008) 246–261