On Designing a People-oriented Constraint-based Workflow Language

Frank Leymann, Tobias Unger, Sebastian Wagner

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{{INPROC-2010-13,
  author    = {Frank Leymann and Tobias Unger and Sebastian Wagner},
  title     = {On Designing a People-oriented Constraint-based Workflow
               Language},
  booktitle = {Proceedings of the 2nd Central-European Workshop on
               Services and their Composition, ZEUS 2010, Berlin,
               Germany, February 25--26, 2010},
  editors   = {Christian Gierds and Jan S{\"u}rmeli},
  year      = {2010},
  pages     = {25--31},
  publisher = {CEUR-WS.org}
}
```

On Designing a People-oriented Constraint-based Workflow Language*

Frank Leymann, Tobias Unger, and Sebastian Wagner

Institute of Architecture of Application Systems
University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{leymann, unger, wagner}@iaas.uni-stuttgart.de

Abstract The control-flow of business workflows is characterized by the strict execution order of the activities that is already defined at design time. This well-structured control-flow is for instance absolutely necessary if the workflows have to be performed fully automatically. However, this rigidity is not always appropriate for people-oriented workflows. Especially in scenarios where real world processes are only semi-structured humans should have more freedom to decide in which order they want to perform the activities. In this paper, we suggest an approach to design people-oriented workflows via constraints to make them more flexible.

1 Introduction

Many workflow languages address the importance of human interactions (e.g. BPEL [1] with the BPEL4People [2] extension) by providing means to model human activities (in the following called tasks). They also address issues like the assignment of people to tasks. Nevertheless, they do not take the control-flow of a business process with human interaction into consideration. Especially workflows that consist solely of activities that are performed by humans (called people-oriented workflows in this paper) should be more flexible than automated business workflows to give them more freedom to decide in which order they want to perform certain tasks. In the following, we propose a declarative approach of modeling people-oriented workflows. We suggest different classes of constraints to model them. This also encompasses constraints that cover scenarios where people have to collaborate. Additionally, techniques are sketched to validate the workflow models and to verify that the constraints are met during workflow execution.

2 Requirements

As mentioned before, people-oriented workflows should not be as strict as automated business workflows because the human factor makes them more difficult to

* This work is partially funded by the ALLOW project. ALLOW (http://www.allow-project.eu/) is part of the EU 7th Framework Programme (contract no. FP7-213339).

predict. Of course, also these workflows have to reflect real-world constraints (e.g. a customer can only be charged after she bought a certain product). However, each human has his own preferred way to work. They are also skilled in scheduling tasks [3], which allows us to relax the model of a workflow. Moreover, humans are aware of their environment. In a classical workflow the execution of the whole workflow is interrupted when resources are missing to execute a certain task. The person who has to perform the task is idle, even if she knows that she possesses all information to perform one or more of the succeeding tasks. This lack of flexibility can decrease the productivity of humans and lead to the fact that they reject a workflow model.

3 Declarative Workflows

Our approach to overcome the rigidity of imperative workflows are *declarative workflows* [4]. In this modeling paradigm a workflow is defined by using constraints that specify what must be done or which conditions must be satisfied during the execution of the workflow. These constraints approximate the desired behavior of the workflow. Any execution order of tasks is allowed as long as it does not violate one of the constraints. This results in the fact, that the user has much more freedom to decide in which order she wants to perform the tasks.

Before discussing the different constraints that we have identified, a simple example is presented that shows the benefits of modeling a workflow by using the declarative approach. A travel agency provides a workflow where customers can book a trip. The imperative version of the workflow consists of the three consecutive task models `Book Flight`, `Book Hotel` and `Pay`. This implies, that the customer has to perform the tasks in this predefined order. A disadvantage of the strict execution order is for instance that the customer can not book the hotel first if she only plans to fly if her favorite hotel has free rooms available. With the declarative approach this could be modeled much less restrictive. It would contain only one constraint that ensures that the task *Pay* is the last task that is executed. The execution order of the other two tasks is not modeled. Consequently, the customer can decide if she books the hotel or the flight first.

3.1 Constraints

This section gives a brief overview about control-flow constraints that can be used along with task models to design a people-oriented workflow. A more comprehensive description of the constraints can be found in [5].

On each constraint a so-called *activation condition* can be defined to support scenarios where constraints should be only enabled when a certain condition holds (e.g. when a process variable has a specific value). If the condition is met the constraint has to be satisfied during workflow execution. Otherwise, the constraint is not enabled and it is ignored. A similar approach is described in [6].

Unary constraints: In [6] several constraints were introduced that can be defined on a single task model, they are called *unary constraints* here. An example for these unary constraints is the *existence* constraint. It defines a lower and/or upper bound concerning the number of instances of a task model that must be executed. We extend this class by the *disable* constraint. If the *disable* constraint is defined on a task model no instances of this task model must be executed. The usage of this constraint makes only sense if an activation condition was defined on it. This prevents the execution of a task until the activation condition does not hold anymore.

Choice constraints: Another class of constraints suggested in [6] are the *choice constraints*. These constraints are used to specify that from a given set of task models a certain subset has to be chosen for instantiation and execution. If the constraint defines for instance that 2 instances from the set of task models $\{A, B, C\}$ have to be executed, one of the subsets $\{A, B\}$, $\{A, C\}$, $\{B, C\}$ of task models must be chosen for execution. However, the choice constraint defines only a lower bound concerning the size of the subsets, i.e. the execution of the task models $\{A, B, C\}$ is also valid. A more restrictive version is the *exclusive choice constraint* that was also introduced in [6]. When using this constraint the lower bound acts additionally as upper bound for the size of subsets of different task models that can be chosen for execution. If this constraint defines that 2 instances of the task models `A`, `B` and `C` must be performed, the instantiation and execution of the task model set $\{A, B, C\}$ is not permitted. A customer of an online book store has for instance the possibility to pay either by credit card, debit card or wireless transfer (each payment method is represented by a task model). To ensure that exactly one payment method is used an exclusive choice has to be defined that specifies that only one task model from the set of task models can be instantiated and performed.

Relation constraints: The class of *relation constraints* defines the execution order of the instances of two task models. In [6] several sequential relation constraints are described. They can be used to restrict the sequential execution order of two tasks. The *A precedence B* constraint is an example for these kind of constraints. It imposes the restriction that an instance of a task model `A` has always to be executed before an instance of task model `B`.

However, with sequential relation constraints it is not possible to model the requirement that two tasks must be performed exactly or at least partly at the same time. These kind of requirements usually emerge through collaborative work, i.e. for achieving a certain goal several tasks have to be executed simultaneously and each of them is performed by a different person. For instance when the pair programming technique is applied to develop software, one programmer writes the code (task A) and the other one has to review the typed code (task B) simultaneously. We introduce parallel relation constraints to model these kind of restrictions. Each of these constraints can be used to determine the degree of simultaneous execution (e.g. partly or completely). The constraints base on the interval relations of Allen's interval algebra [7] (briefly called IA). The IA was chosen because firstly, the interval relations proposed there cover all possible parallel relations between two intervals and secondly, algorithms exist for this algebra to verify that interval relations are consistent. An example for these kind of constraints is the *A during B* constraint which defines that task *A* has to be started and completed during the execution of task *B*. To realize the pair programming example an *equals* constraint has to be defined between the task models `Review` and `Write Code`.

In [5] the relation constraints are extended by time parameters. These time parameters can be used to reflect temporal restrictions between the tasks (e.g. that a task has to be executed within a certain time after its predecessor task was completed). Furthermore, for each relation constraint also a corresponding *negation constraint* exists. These constraints provide the capability to prohibit a certain execution order. The negation constraints are also described in [5].

4 Workflow Model Validation

When creating a declarative workflow model contradictions between the constraints can emerge that are hard to discover at the first glance. This is especially true if many relation constraints were defined. To validate the consistency of the relation constraints we follow an approach similar to the one suggested in [8]. The relation constraints are transformed to interval relations of the IA and each task model is represented by an interval. The different relations between the intervals form a constraint network like the one that is illustrated in Figure 1. On the constraint network reasoning techniques that were introduced in [7] and in [9] are performed. These reasoning techniques infer new relations between all intervals and discover contradictions between them. For instance in Figure 1, it can be inferred from the relations "`C` has to be executed during the execution of `B`" and "`B` has to be executed before `D`" that `C` must be executed before `D`. Moreover, it can be also inferred that a contradiction exists in the network since `A` can not be performed after `D` but before `B`.

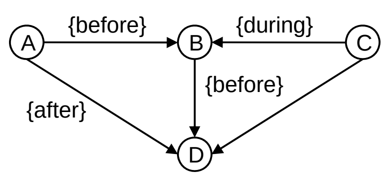

Figure 1. Example Constraint Network

5 Workflow Execution

During the execution of a workflow instance the workflow engine has to verify that all constraints are met. For the unary and choice constraints this can be done very easily. To check if the relation constraints were met we utilize the constraint networks introduced in section 4. Since a network contains the interval relations that must hold between all pairs of tasks the engine can determine if a violation has occurred. If for instance task C in Figure 1 is not performed during the execution of task B the workflow is obviously violated. To reduce these violations the engine should actively decide which tasks can be started by the user. Based on the example before, the engine would not schedule task C for execution when task B is not executed. In [5] more detailed information about the workflow instance verification is provided.

The possible constraint violations have to be also reflected by the the lifecycle model of a declarative workflow. In Figure 2 an extended lifecycle model for declarative workflows is proposed. It bases on the lifecycle for imperative workflows that is suggested in [10]. For the sake of shortness, only the states are discussed that are different from those described in [10]. Here, the *running* state consists of the two substates *satisfied* and *temporarily violated* (these states are inspired by the constraint states proposed in [4]). A workflow instance is in the state *running satisfied* if the workflow is running and no constraints are violated. If a constraint is violated but the violation is still resolvable (e.g. by executing another task) the workflow is put into the *temporarily violated* state. An instance transitions into the *violated* state if it is permanently violated, i.e. if a constraint violation can not be resolved. In this state the workflow execution is interrupted and a process stakeholder can either stop the workflow or, if possible, perform corrective actions (e.g undo a task execution). As declarative workflows are not modeled by a directed acyclic graph there are no end-tasks that cause the workflow to be completed. Hence, declarative workflows have to be completed explicitly by an authorized user. As soon as the request to complete the workflow has been received it is put into the *completing* state. In this state no new tasks can be started anymore and all running tasks have to be completed. Since there is still the chance that constraints are violated during the completion of the workflow it can also transition into the *violated* state instead of the *completed* state.

6 Related Work

In [4], [6] and [11] a declarative approach for modeling and executing workflows is discussed. There is also a declarative workflow management system presented which is called Declare. As mentioned in 3.1, we extend the constraints proposed in [6] by time parameters to add support for temporal restrictions.

To provide a more flexible way for executing workflows in [12] a paradigm called *case handling* is proposed. The central concept behind this paradigm is the *case* which denotes a product that is created during the process execution (e.g. a document). A case consists of different data objects. The data objects are linked to one or more activities and an activity can be only started when its data objects are present. This means, that not control-flow related information drive the process but the state of the case, i.e. the existence of data objects. In this approach the user is focused on the whole case and not only on one activity like in traditional workflows.

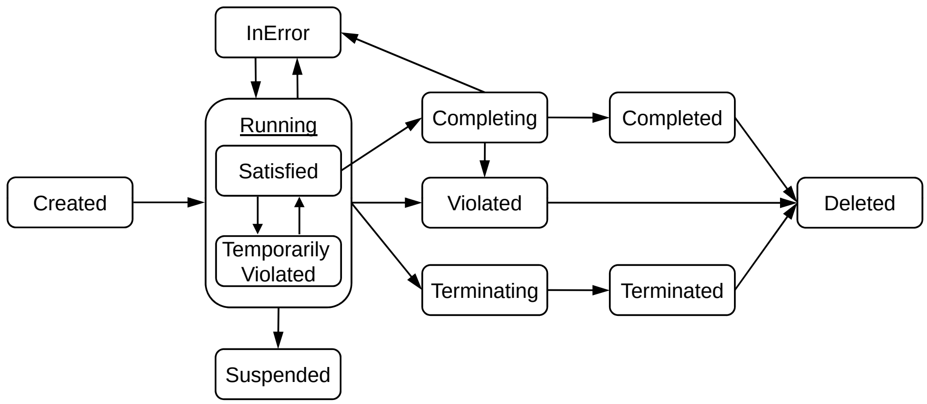

Figure 2. Declarative Workflow Lifecycle

BPEL4People [2] adds support for human interactions to BPEL. It extends BPEL by introducing *people activities* to enable human interactions with the BPEL process. It focuses on human interaction patterns, like the 4-eyes principle, escalation handling and nominations. The people activities are implemented by human tasks that are defined in the WS-HumanTask specification [13]. The concept of human tasks that is described there covers the most important aspects of human activities and could be used in conjunction with the constraints to model people-oriented workflows. In [5] an approach is presented that shows how this can be done.

7 Conclusion & Future Work

In this paper, we proposed a constraint-based language for modeling people-oriented workflows because the imperative paradigm tends to overspecify workflows and to be to restrictive. We introduced constraint networks to validate the consistency of these workflow models and to verify that the constraints were met during workflow execution.

A downside of the declarative approach is that these workflows are more difficult to understand by people because unlike in imperative workflows there is no directed graph that connects the different tasks. This is especially true when the workflow contains a lot of tasks and constraints. On the one hand, it is more sophisticated to observe the overall progress of a running workflow instance and on the other hand, it can be challenging to comprehend the transitive effects of the constraints. In future work, we plan to develop a proper visualization for the declarative workflows to make them more understandable for the end-users.

References

1. OASIS: Web Services Business Process Execution Language Version 2.0. Committee specification, OASIS WSBPEL TC (January 2007)
2. OASIS: WS-BPEL Extension for People Specification Version 1.1, Committee Draft 06. (2009)
3. Card, S.K., Newell, A., Moran, T.P.: The Psychology of Human-Computer Interaction. L. Erlbaum Associates Inc., Hillsdale, NJ, USA (1983)
4. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D **23**(2) (2009) 99–113
5. Wagner, S.: A Concept of Human-oriented Workflows. Diploma thesis, University of Stuttgart, Germany (January 2010)
6. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Control to Users. PhD thesis, Eindhoven University of Technology (2008)
7. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11) (1983) 832–843
8. Lu, R., Sadiq, S.W., Padmanabhan, V., Governatori, G.: Using a temporal constraint network for business process execution. In Dobbie, G., Bailey, J., eds.: ADC. Volume 49 of CRPIT., Australian Computer Society (2006) 157–166
9. Vilain, M.B., Kautz, H.A.: Constraint propagation algorithms for temporal reasoning. In: AAAI. (1986) 377–382
10. Leymann, F., Roller, D.: Production Workflow - Concepts and Techniques. PTR Prentice Hall (Januar 2000)
11. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: OTM Conferences (1). (2007)
12. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data Knowl. Eng. **53**(2) (2005) 129–162
13. OASIS: Web Services Human Task Specification Version 1.1, Committee Draft 06. (2009)