Declarative Workflow Modeling with UML class diagrams and OCL

Jens Brüning

University of Rostock, Department of Computer Science,
A.-Einstein-Str. 21, 18059 Rostock, Germany
jens.bruening@uni-rostock.de

Abstract: This paper describes an approach of modeling workflows with UML class diagrams and OCL constraints [OCL06] in a declarative way. These are modeled in the UML tool USE [USE08] that can generate object diagrams (snapshots) out of UML class diagrams. USE checks specified OCL constraints against the generated snapshots. With the declarative workflow modeling approach presented here, activity model states are integrated in the object model states. By analyzing these snapshots the model is validated against requirements.

In figure 1 a UML class diagram for modeling flat workflows is presented. The class *Process* has an attribute *name* and contains a set of activities which is described by the association *belongs* between *Process* and *Activity*. Atomic actions in the process that are executed in the workflow are expressed by the class *Activity*. Activities have also names that describe the actions and a state in which the activity is just in the time, the snapshot of the system is taken. Possible states for the activities are described in the enumeration *State*. A derived state of the process depends on the states of the included activities and is delivered by the operation *getState* in the class *Process*. Further on, the operation *getActivity* is needed by the subsequent OCL constraints and returns the requested activity instance. The described operations are specified in OCL expressions and will be interpreted by USE.

Figure 1: UML model for flat workflow specifications modeled in USE

Actual states of an activity instance can be changed by invoking the methods *skip()*, *start()* or *done()*.The functionality of the operations are expressed by OCL pre- and post-conditions. Operation *enabled()* proves on basis of the current object model state and the inner state of the activity itself, if it is enabled and thus can be started. This method is also coded in an OCL expression.

Without constraints, the model of figure 1 is insufficient to express concrete workflow definitions. OCL invariants are used to get process definitions with its containing activities. For example, the following invariant guarantees that the process “processing” consists of the activities “generate invoice”, “send invoice”, “debit” and “send goods”.

```
context Process inv OrderProcessing:
  self.name='processing' implies
    self.activity.name = Bag{'generate invoice', 'send invoice','debit','send goods'}
```

Declarative workflow models are flexible because all execution paths of the modelled activities are allowed if they are not forbidden explicitly [PA07]. All activities are in an interleaved relationship by default. Other classical temporal relations like sequences or alternatives can be modelled by constraints. In the example presented above there should be a sequence relation between “generate invoice” and “send invoice” and an alternative between “debit” and “generate invoice”. These relationships can be expressed by the following OCL invariants.

```
context Process inv Accounting_Generate_Send_Sequence:
  self.name='processing' implies
    (self.getActivity('send invoice').state=#running implies self.getActivity('generate invoice').state=#done)

context Process inv Accounting_Invoice_Debit_Alternative:
  self.name='processing' implies
    ((self.getActivity('generate invoice').state=#running implies  self.getActivity('debit').state=#skipped) and
     (self.getActivity('debit').state=#running implies self.getActivity('generate invoice').state=#skipped))
```

Further on, declarative workflow models can express additional relations between activities that are not possible to model in the traditional workflow modelling languages like BPMN or UML Activity Diagrams. For example, simply expressing that two activities must not occur at the same time [PA07]. This is modelled in the next OCL invariant where the activities “send goods” and “debit” are not allowed to be both in the state running at the same time.

```
context Process inv Debit_SendGoods_Entangled:
  self.name='processing' implies
    not (self.getActivity('generate invoice').state=#running and self.getActivity('send goods').state=#running)
```

The next working step for this modelling approach is, to express additional temporal relations in the declarative way on basis of the workflow patterns. Furthermore, it can be extended to hierarchical workflow models.

References

[OCL06] Object Constraint Language (OCL) Specification 2.0, OMG, http://www.omg.org/docs/formal/06-05-01.pdf, 2006.

[PA07] Pesic, M., Aalst, W., et.al.: Constraint-Based Workflow Models: Change Made Easy. In: LNCS 4103, pp. 77–94, Berlin, Springer, 2007.

[USE08] A UML-based Specification Environment, University of Bremen, http://www.db.informatik.uni-bremen.de/projects/use/, Bremen, 2008.