



Declarative Models for Business Processes and UI-Generation using OCL

*Jens Brüning, Andreas Wolff
Rostock University (Germany)
Department of Computer Science*

Overview of this talk

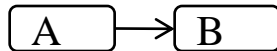
- Motivation for declarative process models using OCL
 - Brief introduction to process modeling
 - What further aspects can be described declaratively?
- Concept of declarative process models in UML and OCL
 - Meta model for business process models
 - Expressing temporal relations by OCL invariants
 - Animate and validate declarative models with *USE*
- UI aspect of the declarative process models

Motivation for declarative workflow-modeling

A brief introduction in process modeling

- 42 Process patterns of *van der Aalst* are very popular to define relationships between activities
- There are three standard relationships between activities

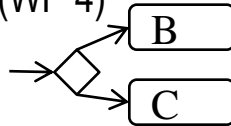
- Sequence (WP 1)



valid traces

start(A), finish(A), start(B), finish(B)

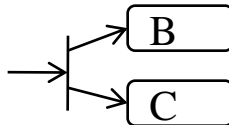
- Choice (WP 4)



start(B), finish(B)

start(C), finish(C)

- Concurrency (WP 2)



start(B), finish(B), start(C), finish(C)

start(B), start(C), finish(C), finish(B)

...

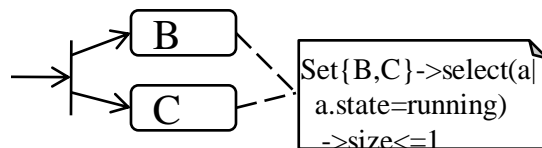
start(C), finish(C), start(B), finish(B)

Motivation for declarative workflow-modeling

More advanced relationships can be expressed by declarative annotations

- Workflow pattern 17: Interleaved Parallel Routing

- Execute a number of activities in any order, but do not execute any of these activities at the same time/simultaneously
- Pattern can be expressed by OCL expression

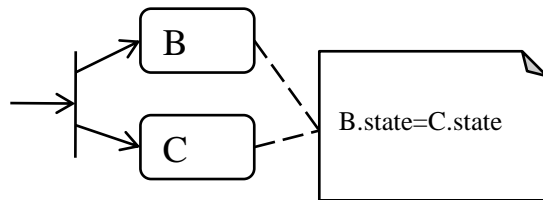


Valid traces

start(B), finish(B), start(C), finish(C)
start(C), finish(C), start(B), finish(B)

Motivation for declarative workflow-modeling

- Assurance of parallelism can be expressed declaratively
 - For example the operation of the surgeon must be assisted by the nurse



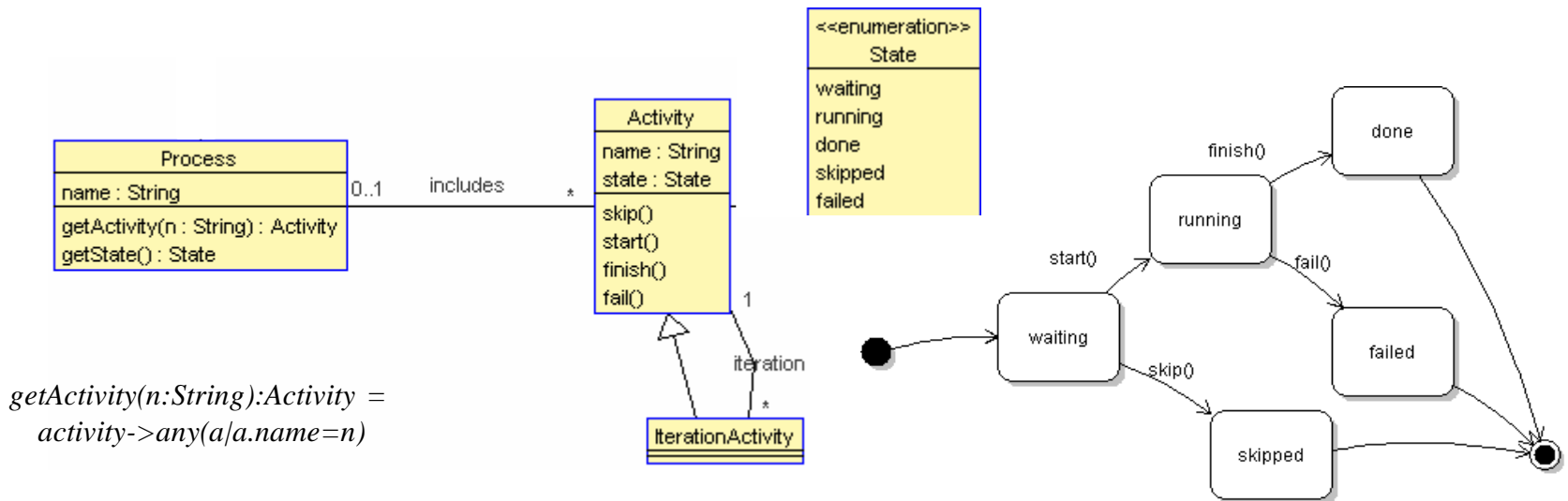
Valid traces

`start(B), start(C), finish(B), finish(C)`

...

`start(C), start(B), finish(C), finish(B)`

Concept of declarative process models in UML and OCL

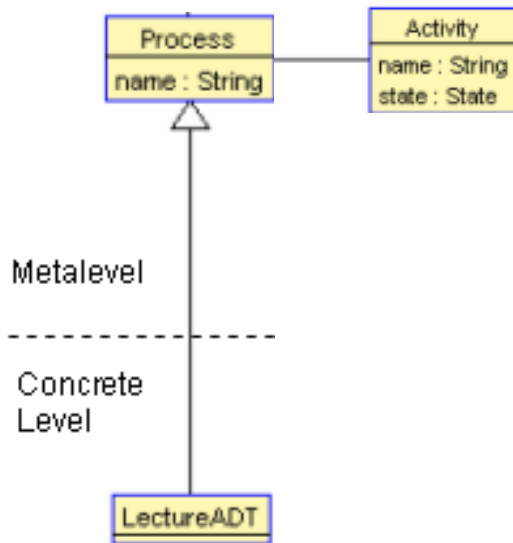


*getActivity(n:String):Activity =
activity->any(a/a.name=n)*

Meta model of declarative process models

Life cycle of activities

Concept of declarative process models in UML and OCL



- OCL invariants are used to define the process

context LectureADT inv LectureExamination:

self.activity.name = Bag{'TestPeriod','HomeworkPeriod','writeExam'}

Concept of declarative process models in UML and OCL

- Declarative process models are *flexible by design*
 - All execution orders of activities are allowed if they are not forbidden explicitly
 - OCL invariants are used to express temporal restrictions between activities

- Sequence of *HomeworkPeriod* and *writeExam*:

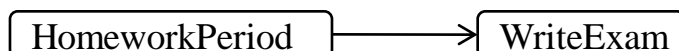
- declaratively described

context LectureADT inv Homework_Exam_Sequence:

self.getActivity('writeExam').state=#running implies

self.getActivity('HomeworkPeriod').state=#done

- graphically described





Concept of declarative process models in UML and OCL

- A further temporal restriction described by an OCL invariant:

TestPeriod must be in the *HomeworkPeriod*

context LectureADT inv TestPeriod_in_HomeworkPeriod:

self.getActivity('TestPeriod').state=#running implies

self.getActivity('HomeworkPeriod').state=#running

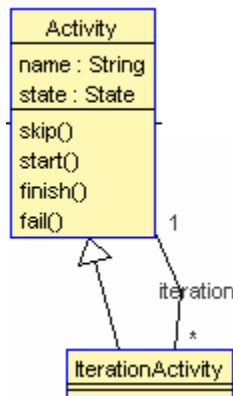
- This restriction cannot be described graphically

Concept of declarative process models in UML and OCL

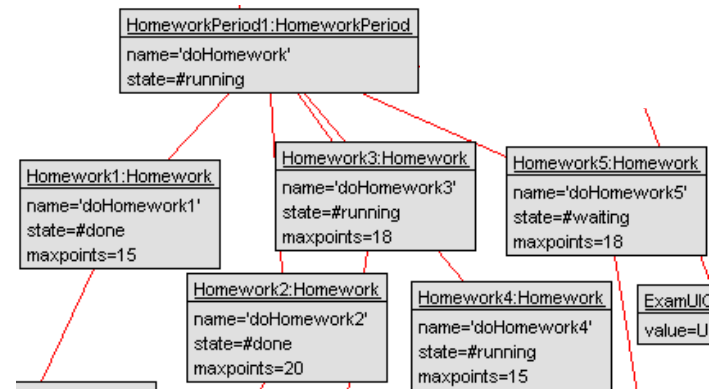
- *IterationActivity* describes activities that can be executed more than once
- OCL invariant for class *IterationActivity*

context Activity inv IterationActivities:

self.state=#done implies self.iterationActivity->forall(a|a.state=#done)



Process Meta model



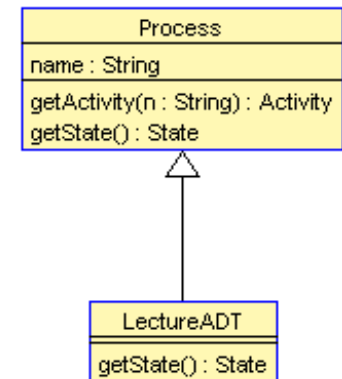
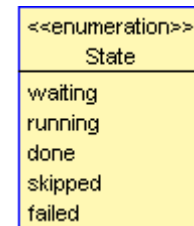
Process example

Concept of declarative process models in UML and OCL

- OCL function is used to calculate the state of the process

```
Process :: getState():State=
```

```
  if activity->forAll(a|a.state=#done) then #done
  else if activity->exists(a|a.state=#running) then #running
  else #waiting endif endif
```



- *getState()* can be overwritten by the concrete process *LectureADT*:

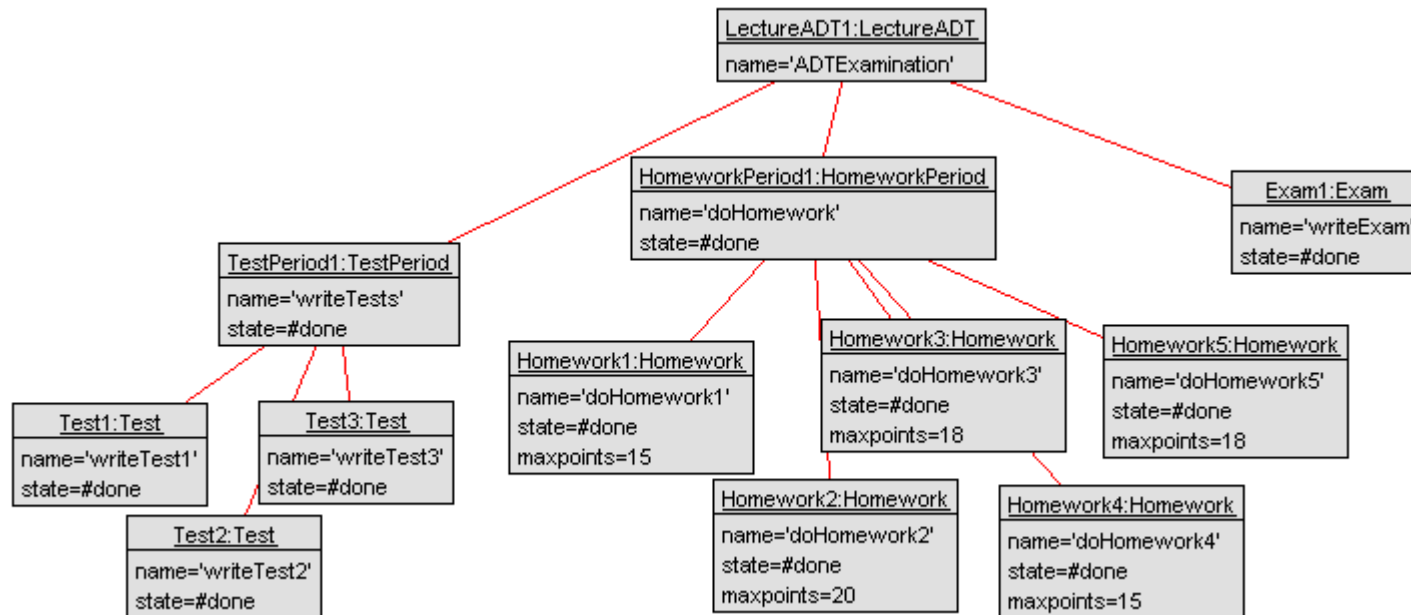
```
LectureADT :: getState():State=
```

```
  if getActivity('writeExam').state=#done then #done
  else if activity->exists(a|a.state=#running) then #running
  else #waiting endif endif
```

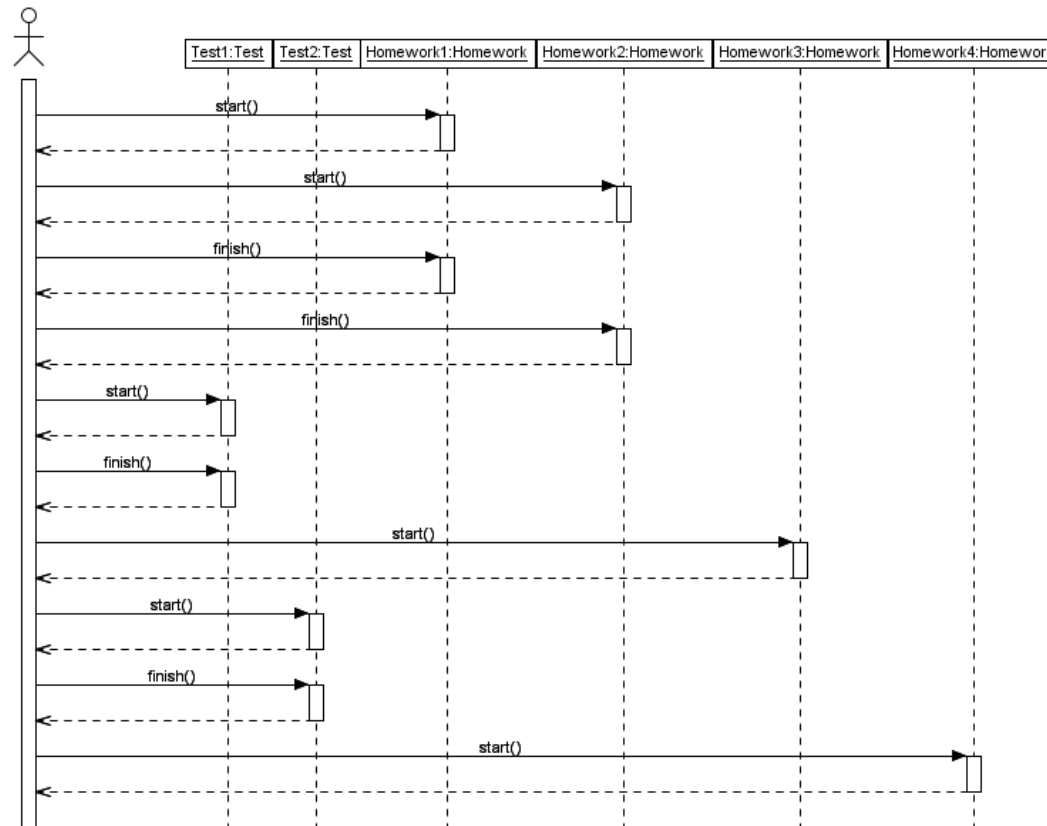
Animate and validate declarative models with *USE*

- *USE* can instantiate the declarative process model
- Processes are displayed as UML object diagrams
- Process scenarios can be tested by invoking the following operations in activities:
 - start(), finish(), skip()
 - These operations have side effects on the activity objects
 - OCL cannot be used
 - ASSL (A snapshot sequence language) provides this functionality
- OCL-Process-Invariants are observed by *USE*
 - ASSL procedures are only executed in *USE* if no invariant is violated

Process Snapshot taken in USE

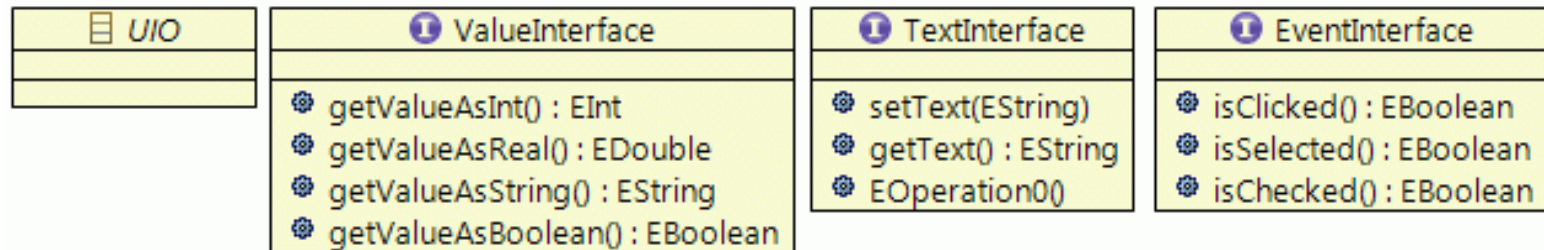


Process scenario logged by UML sequence charts

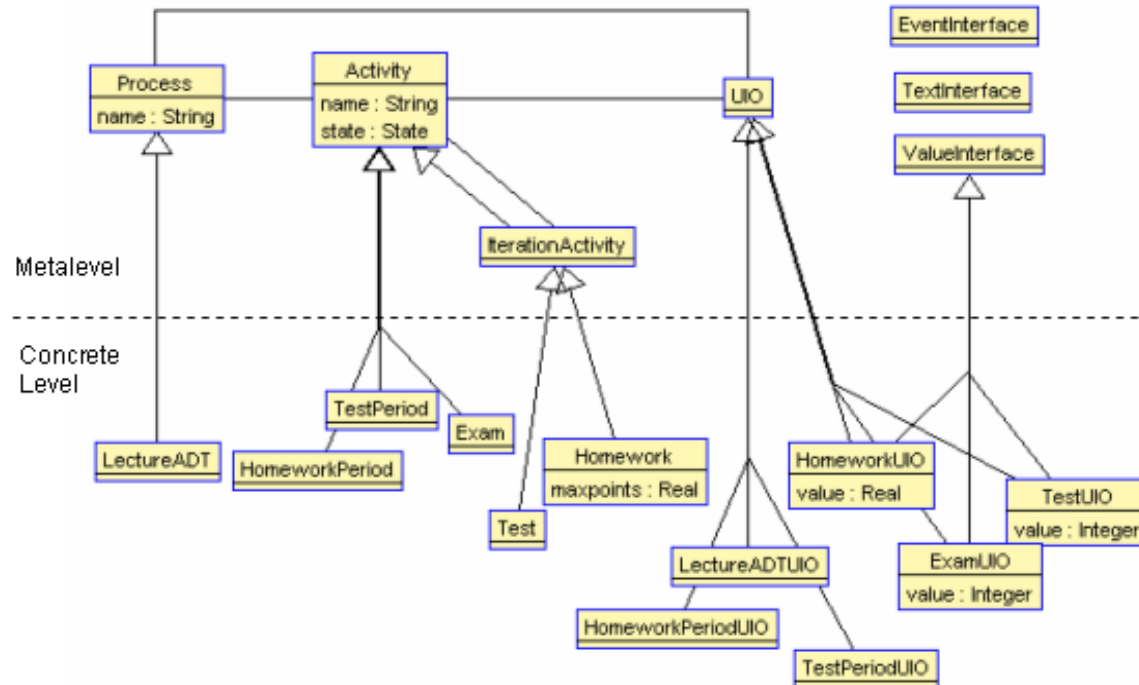


Standardizing In- and Output

- Abstract UIO Class and Interfaces



Resulting concrete process model (omitting CUI)



Constraints

- OCL invariants make statements of UI objects to process execution states
- If the Exam process is running the tests have to be passed with more then 50 percent

context Exam inv ExamOnlyIf_TestsPassed:

let percents:Bag(Integer) =

self.process.getActivity('HomeworkPeriod').iterationActivity.uIO.oclAsType(HomeworkUIO).getValueAsInt()

in

self.state=#running implies

percents->sum() / percents->size() > 50

Conclusion & future work

- Declarative process models can describe further aspects than imperative process models
- A Meta model for declarative process modeling using UML and OCL was presented
- Animation of these models in *USE* has been demonstrated
- User interface aspects were discussed

Future work:

- Process view for declarative process models should be implemented in *USE*
- The process Meta model should be extended to express more workflow patterns



Thank you for your attention!