

Generation of test case templates using the Boolean representation of software model

Christina Dorotska

Freiberg University of Mining and
Technology
Institute of Computer Science,
Freiberg (Sachs.), Germany

Outline

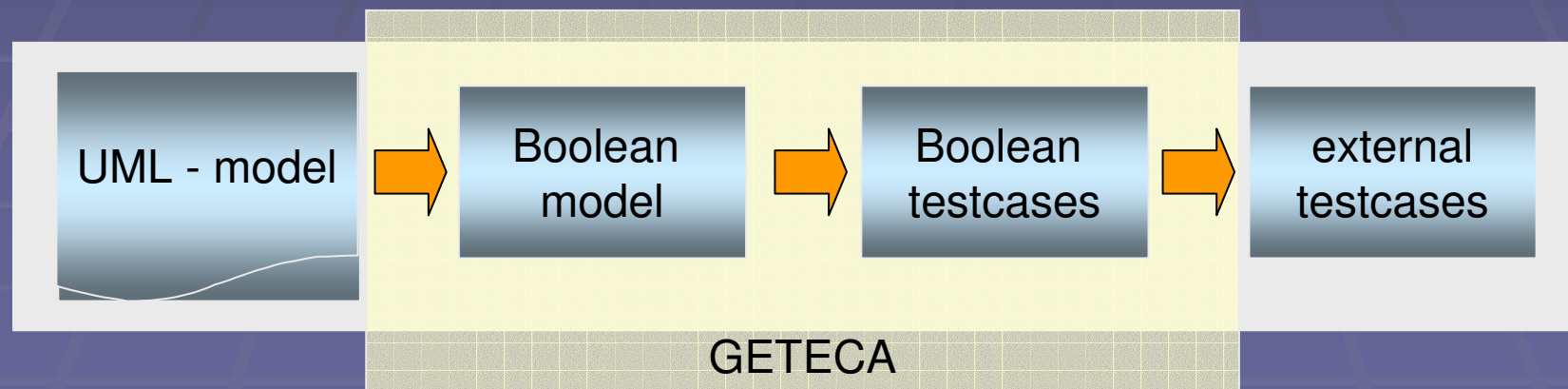
- Introduction
- UML
- Describing of a test template by a set of values
- Boolean representation of UML modell
 - Coding of model elements
 - Coding of test templates
- Example
- Experimental results
- Conclusion

Introduction

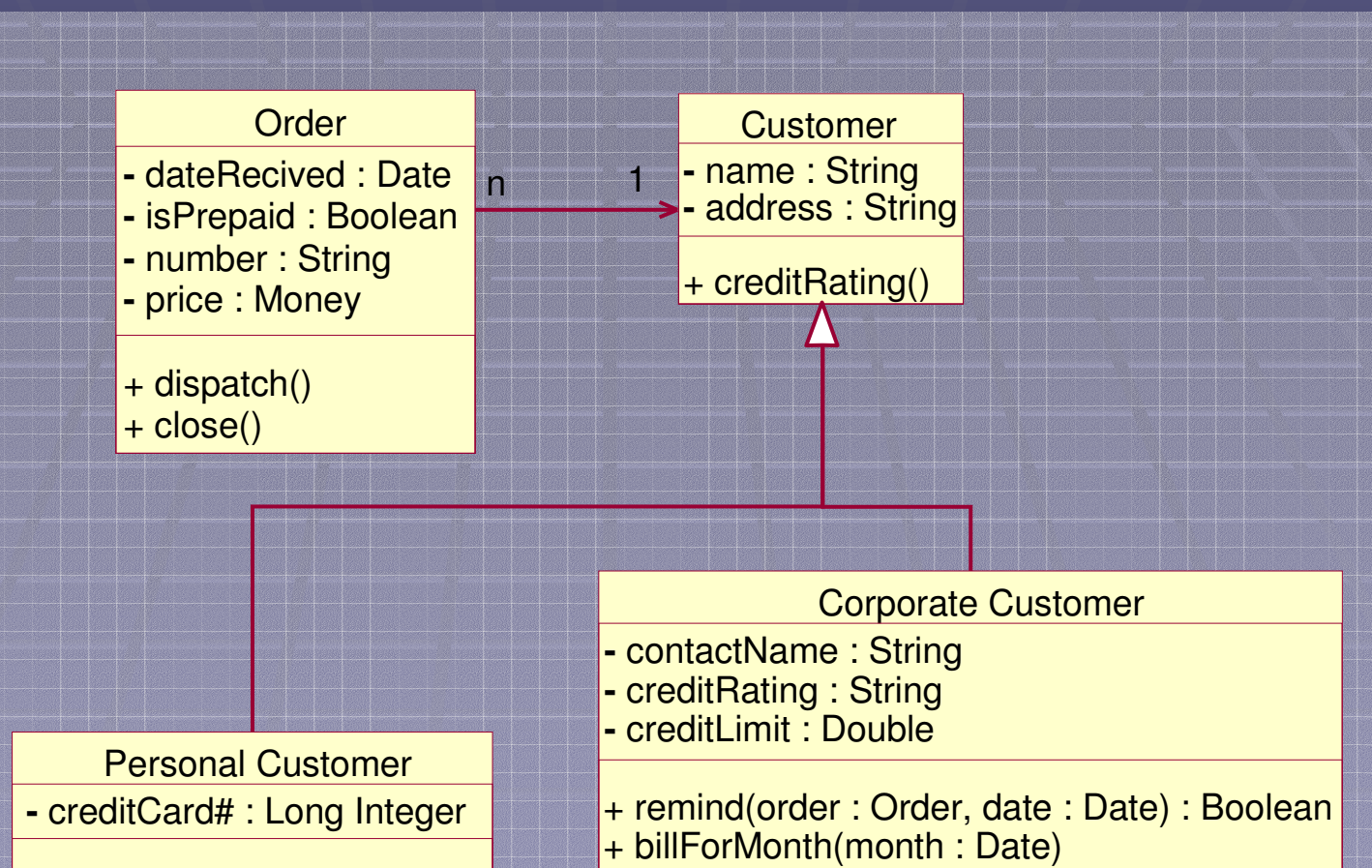
- Main goal

Automatically generation of test templates for the software

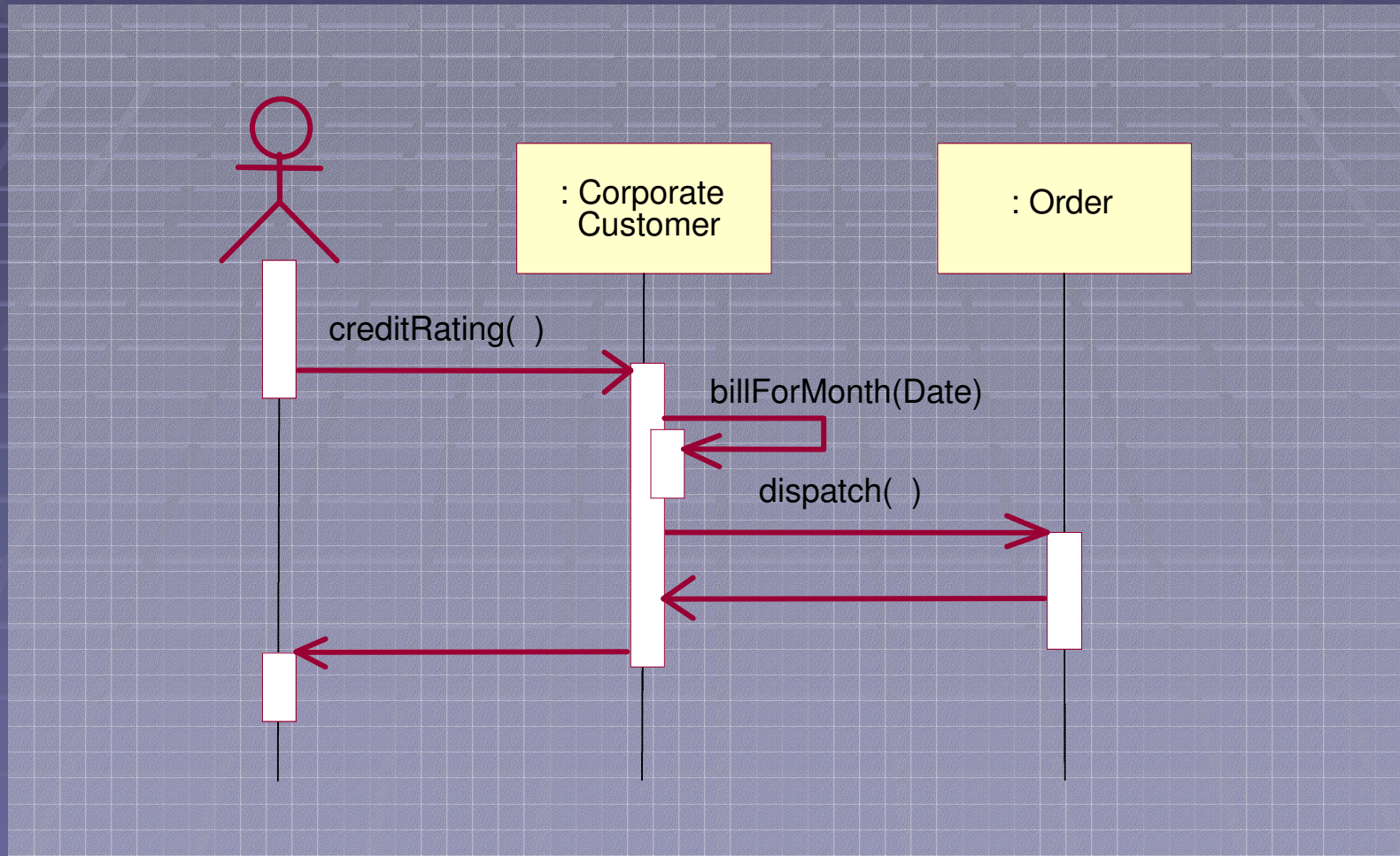
- Describing of the software with UML
- Creation of Boolean software model
- Calculation of Boolean testcases
- Transformation into the text form



Describing of software with UML diagrams



Describing of software with UML diagrams



Test template representation by a set of values

test template	value #1	value #2	...	value #i-1	value #i	value #i+1	...	value #n-1	value #n
#1	wrong	wrong	...	wrong	wrong	wrong	...	wrong	wrong
#2	wrong	wrong	...	wrong	correct	correct	...	wrong	correct
#3	wrong	wrong	...	correct	correct	wrong	...	correct	wrong
...
#j	wrong	correct	...	wrong	correct	wrong	...	correct	wrong
...
#2 ⁱ	correct	correct	...	correct	correct	correct	...	correct	correct

preconditions
postconditions

Test template representation by a set of values

test template	value #1	value #2	...	value #n	value #n+1	value #n+2	...	value #n+m	test result
#1	wrong	correct	...	correct	correct	wrong	...	wrong	✓
#2	correct	wrong	...	correct	correct	correct	...	correct	×
...
#n	correct	correct	...	wrong	wrong	wrong	...	wrong	✓
#n+1	correct	correct	...	correct	correct	correct	...	correct	✓

Coding of model elements

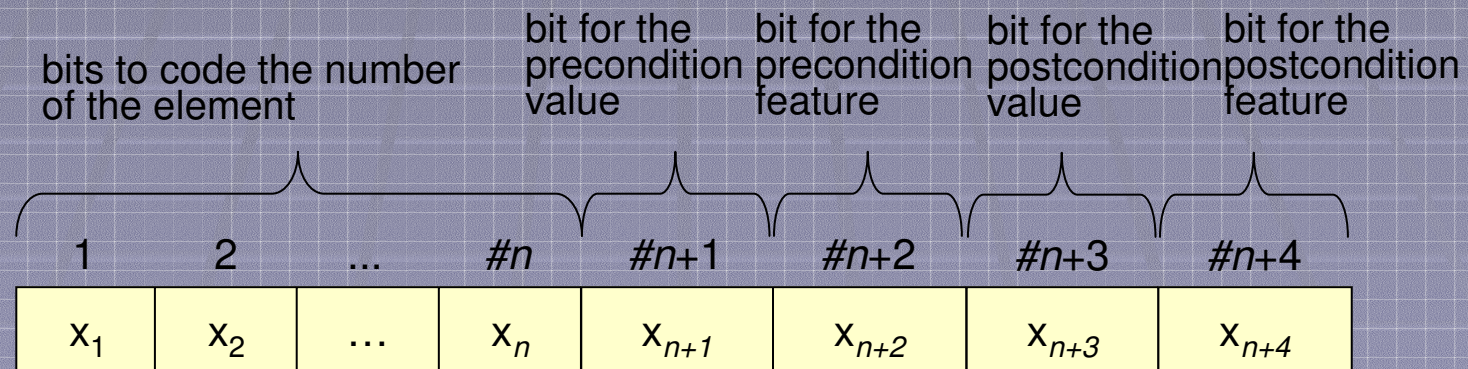
Class 1 #c₁
 Class 2 #c₂
 ...
 Class n #c_n

Attribut 1 #c_{p1}#a₁
 Attribut 2 #c_{p2}#a₂
 ...
 Attribut s #c_{ps}#a_s

Operation 1 #c_{k1}#o₁
 Operation 2 #c_{k2}#o₂
 ...
 Operation m #c_{km}#o_m

Parameter 1 #c_{i1}#o_{j1}#p₁
 Parameter 2 #c_{i2}#o_{j2}#p₂
 ...
 Parameter l #c_{il}#o_{js}#p_l

Code patterns for model elements



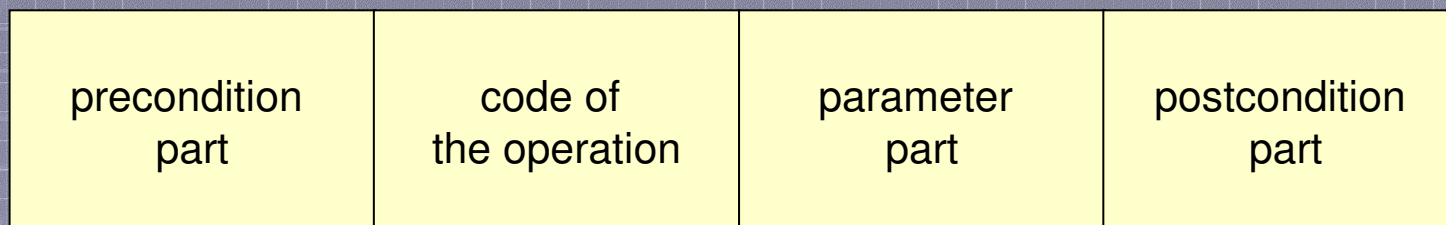
Code pattern for parameter

Possible kinds of parameter and its coding

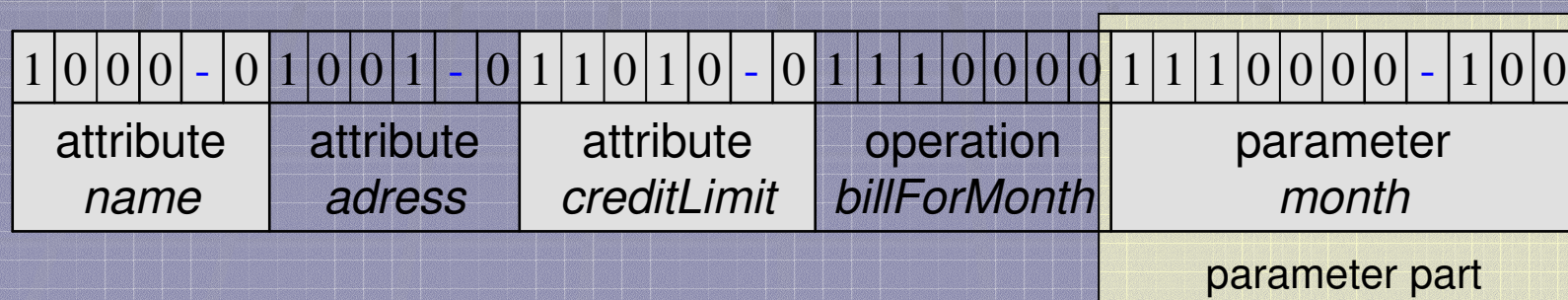
Bit for the precondition value	Bit for the precondition feature	Bit for the postcondition value	Bit for the postcondition feature	Note
-	1	0	0	<i>in</i> -parameter
0	0	-	1	<i>out</i> -parameter
-	1	-	1	<i>inout</i> -parameter

Coding of a test template

Dependency vector



Dependency vector for the function *billForMonth*



Test templates for the function *billForMonth*

1 0 0 0 - 0 1 0 0 1 - 0 1 1 0 1 0 - 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 - 1 0 0

1	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0
1	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0
1	0	0	0	1	0	1	0	0	1	1	0	1	0	1	1	1	0	0	0	1	1	1	0	0	0	0	1	1	0	0
1	0	0	0	1	0	1	0	0	1	1	0	1	0	1	1	1	0	0	0	1	1	1	0	0	0	0	1	1	0	0
1	0	0	0	1	0	1	0	0	1	1	0	1	0	1	1	1	0	0	0	1	1	1	0	0	0	0	1	1	0	0
1	0	0	0	1	0	1	0	0	1	1	0	1	0	1	1	1	0	0	0	1	1	1	0	0	0	0	1	1	0	0
attribute <i>name</i>	attribute <i>adress</i>				attribute <i>creditLimit</i>				operation <i>billForMonth</i>				parameter <i>month</i>																	



Interpreted list of test templates

```

Testfälle.txt - Editor
Datei Bearbeiten Format Ansicht ?

Interaktion: billForMonth(OPERATION)
Testtemplate #. 1
Customer          name(ATTRIBUT)          precondition  wrong
                  address(ATTRIBUT)       precondition  correct
Corporate Customer creditLimit(ATTRIBUT)  precondition  correct
Corporate Customer month(PARAMETER)  precondition  correct

Interaktion: billForMonth(OPERATION)
Testtemplate #. 2
Customer          name(ATTRIBUT)          precondition  correct
                  address(ATTRIBUT)       precondition  wrong
Corporate Customer creditLimit(ATTRIBUT)  precondition  correct
Corporate Customer month(PARAMETER)  precondition  correct

Interaktion: billForMonth(OPERATION)
Testtemplate #. 3
Customer          name(ATTRIBUT)          precondition  correct
                  address(ATTRIBUT)       precondition  correct

```

Experimental results

#	# model elements	# test templates	generation time (ms)
1	148	57	156
2	138	60	250
3	60	23	46
4	65	23	46
5	78	13	31
6	75	10	31
7	78	11	31

Conclusion

- Automatically generation of test cases using a UML model and its representation in Boolean form is possible
- To generate the test templates was used the dependency vector of the function
- All values, which influence the test result were covered in a dependency vector with a dash
- To create a test case all dashes should be replaced with one or zero