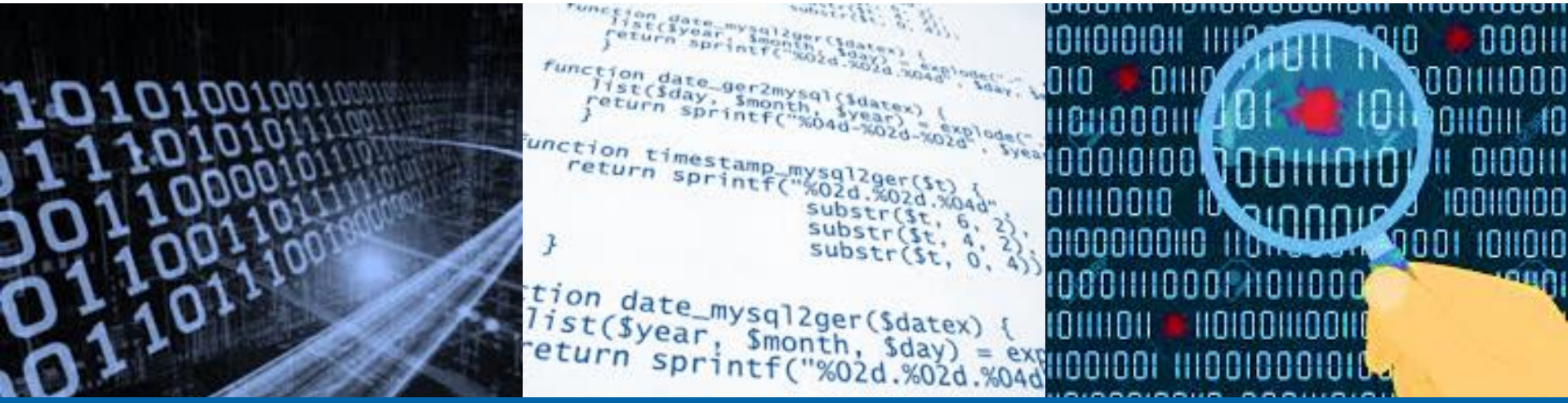
MODELLGETRIEBENE FUNKTIONALE SYSTEMTESTERSTELLUNG

mit dem MOdel Compiler for generating Complete Applications (MOCCA)



Jessica Schiffmann in Langen, 10.02.2017

Jessica Schiffmann

Referentenvorstellung

Ausbildung

Seit 2011	Berufsbegleitende Promotion im kooperativen Verfahren an der TU Freiberg und der FH Kaiserslautern
2005	Diplom Informatikerin (FH)
2003	Fachinformatikerin (Anwendungsentwicklung)

Beruflicher Werdegang

Seit 2015	IT-Expertin, FESTO AG & Co. KG
Seit 2014	Online-Tutorin für Softwarequalitätssicherung, IUBH
2011 - 2015	Senior Consultant für Softwaretest und Prozessgestaltung, PRISM Informatics
2008 - 2011	Inhouse Softwarequalitätssicherung, IDS Scheer AG
2003 - 2008	Softwareentwicklerin, T-Systems

AGENDA

1. Zielsetzung des Ansatzes

2. Einführung in MOCCA
   - Benötigte Artefakte
   - Generierungsprozess

3. Umsetzungsbeispiel
   - Kurzblick: Modellierung Systemmodelle
   - Modellierung Testfallstruktur und -ablauf

Zielsetzung

1. Abbildung der Systemtesteigenschaften
   - Eingaben über und Prüfen auf der Systemoberfläche
   - Testen von Prozessen / Prozessabläufen der Software

2. Kompakte und wiederverwendbare Modellierung

3. Trennung von System- und Testverhalten bei gleichzeitigem Integrieren des Tests in das System

Grundlegende Modelltypen und Eingabefiles

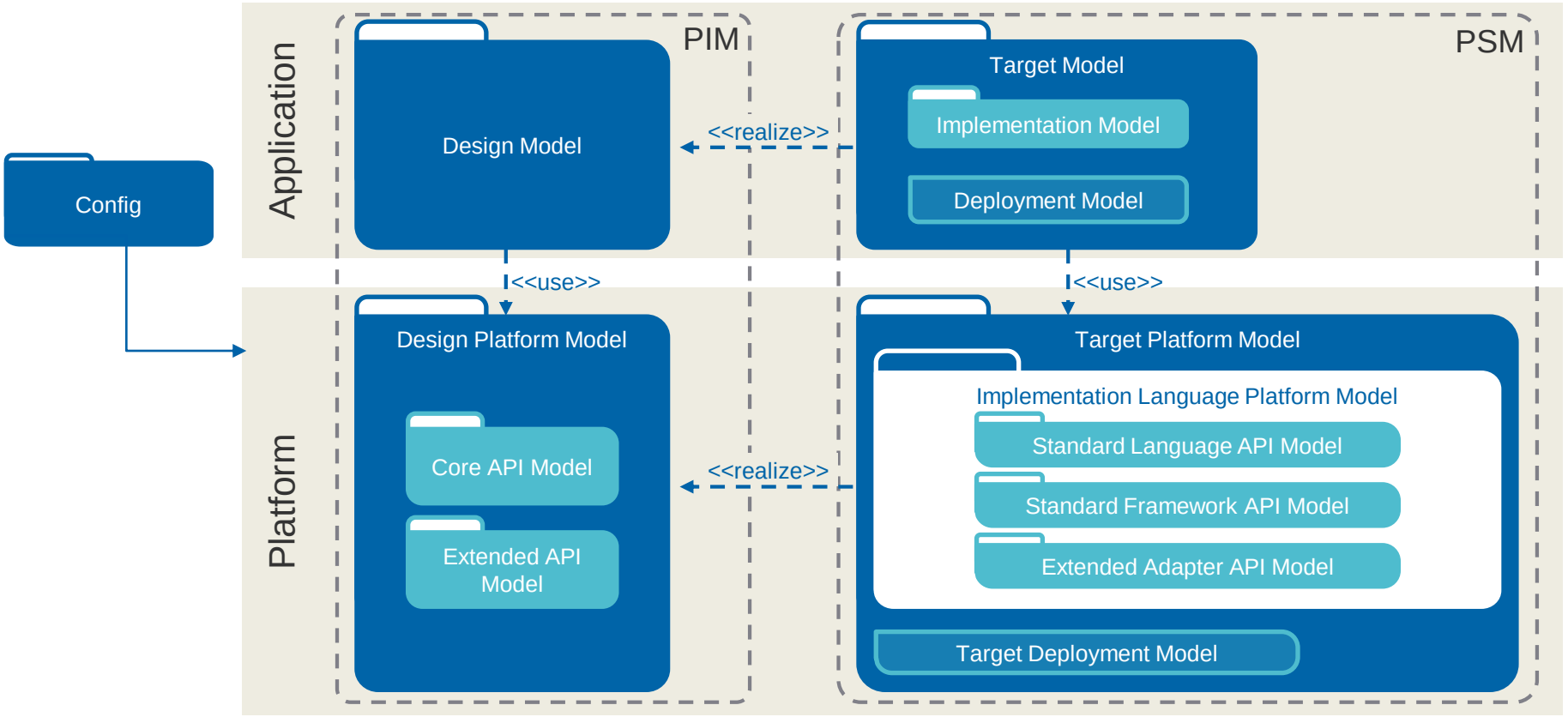


Abb. Quelle: Automatic Generation of Software Applications - A Platform-based MDA Approach, Phd Thesisis D.Liang

OCL

- Eränzung zu UML
- Textuelle Modellierung
- Aktuell in Version 2.4 (Februar 2014)
- Randbedingungen eines Modells
- Erzeugen widerspruchsfreie, maschinenlesbare Modelle
- 7 Arten von Zusicherung (Constrains)

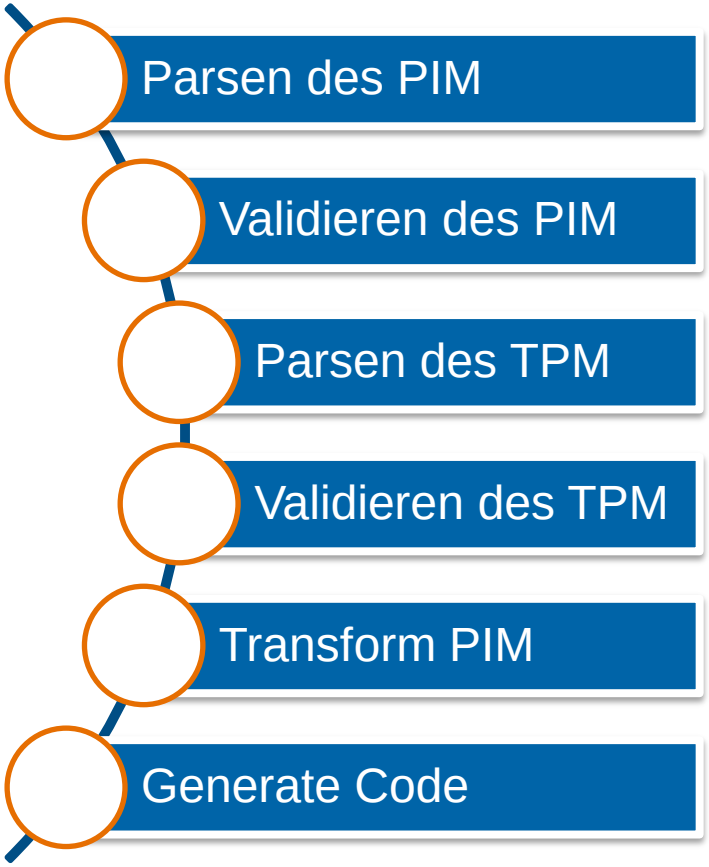
```
context Person inv: self.alter >=0
```

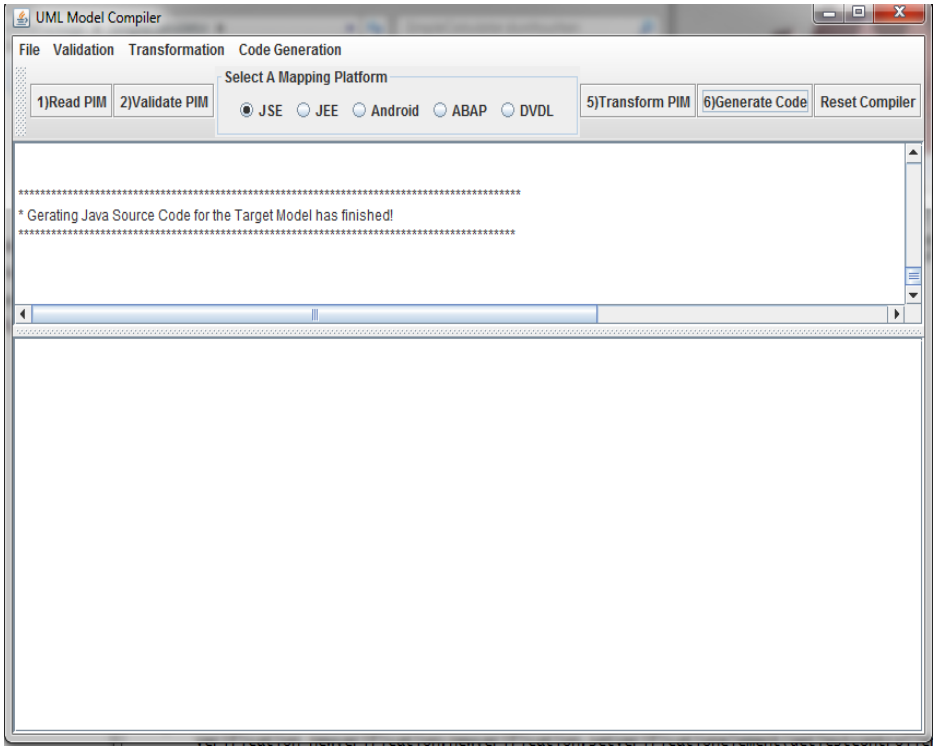
XOCL

- Ergänzt OCL zu einer vollwerten Action Language
- Ergänzungen
  - ⇒ Zustandsänderungen
  - ⇒ Kontrollfüsse
  - ⇒ Kompakte Eventmodellierung
- Kompaktere Modellierung

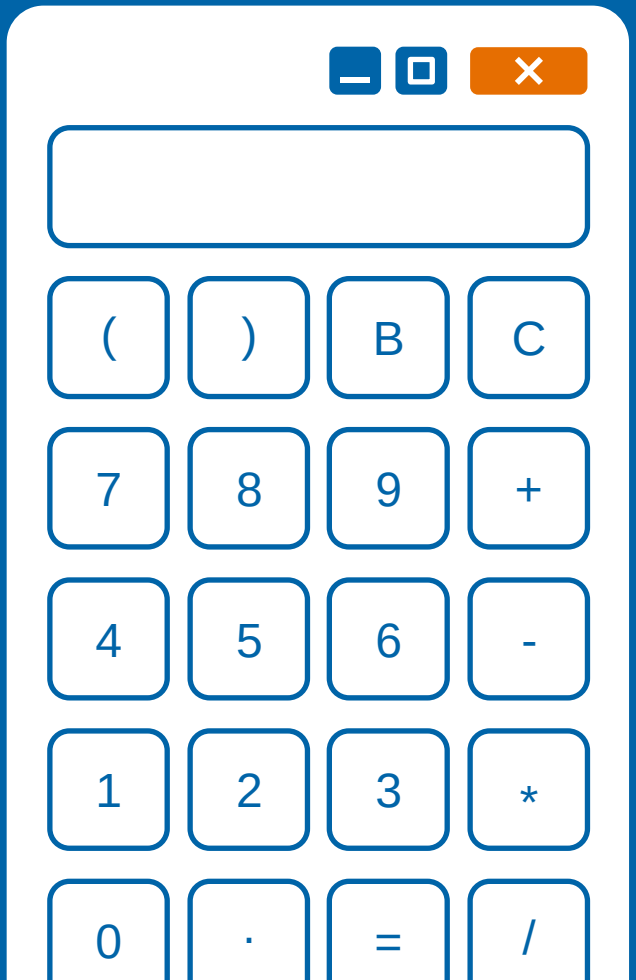
```
if t = 1 then begin
  update test.startTesting();
end endif
```

Codegenerierungsframework

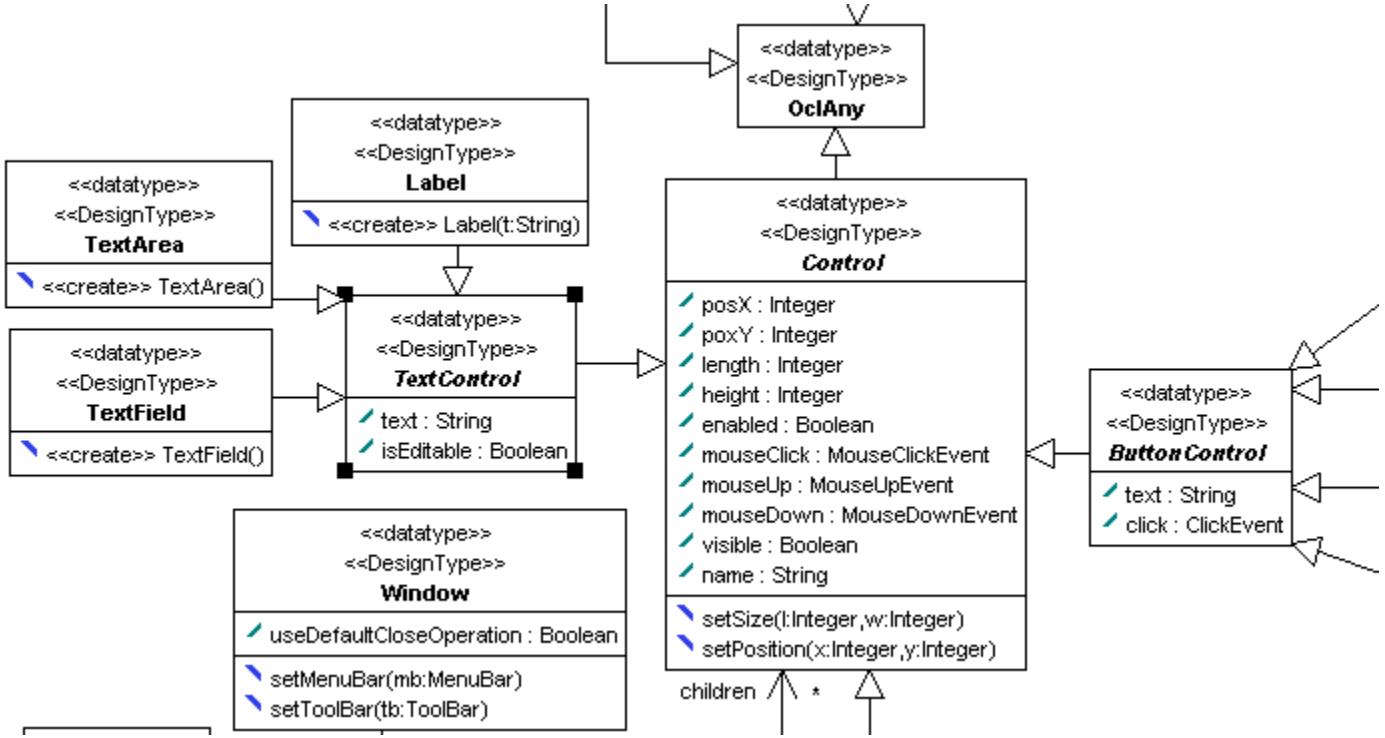




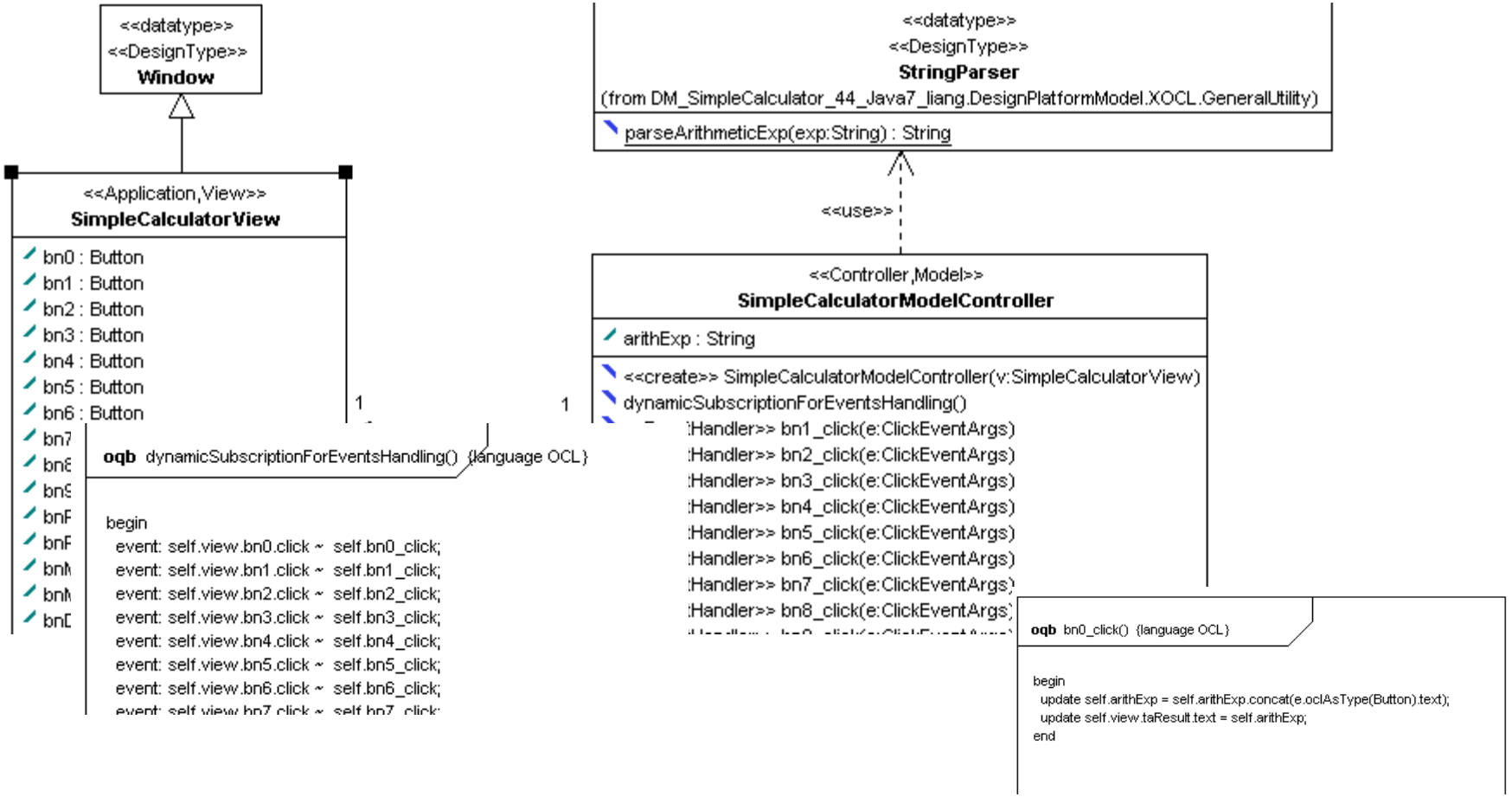
Umsetzungsbeispiel



Design Platform Model



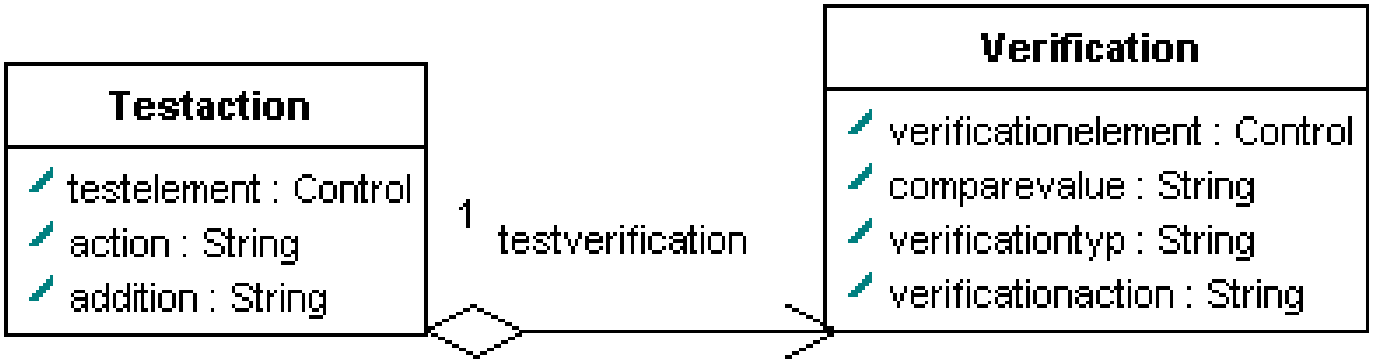
Design Model



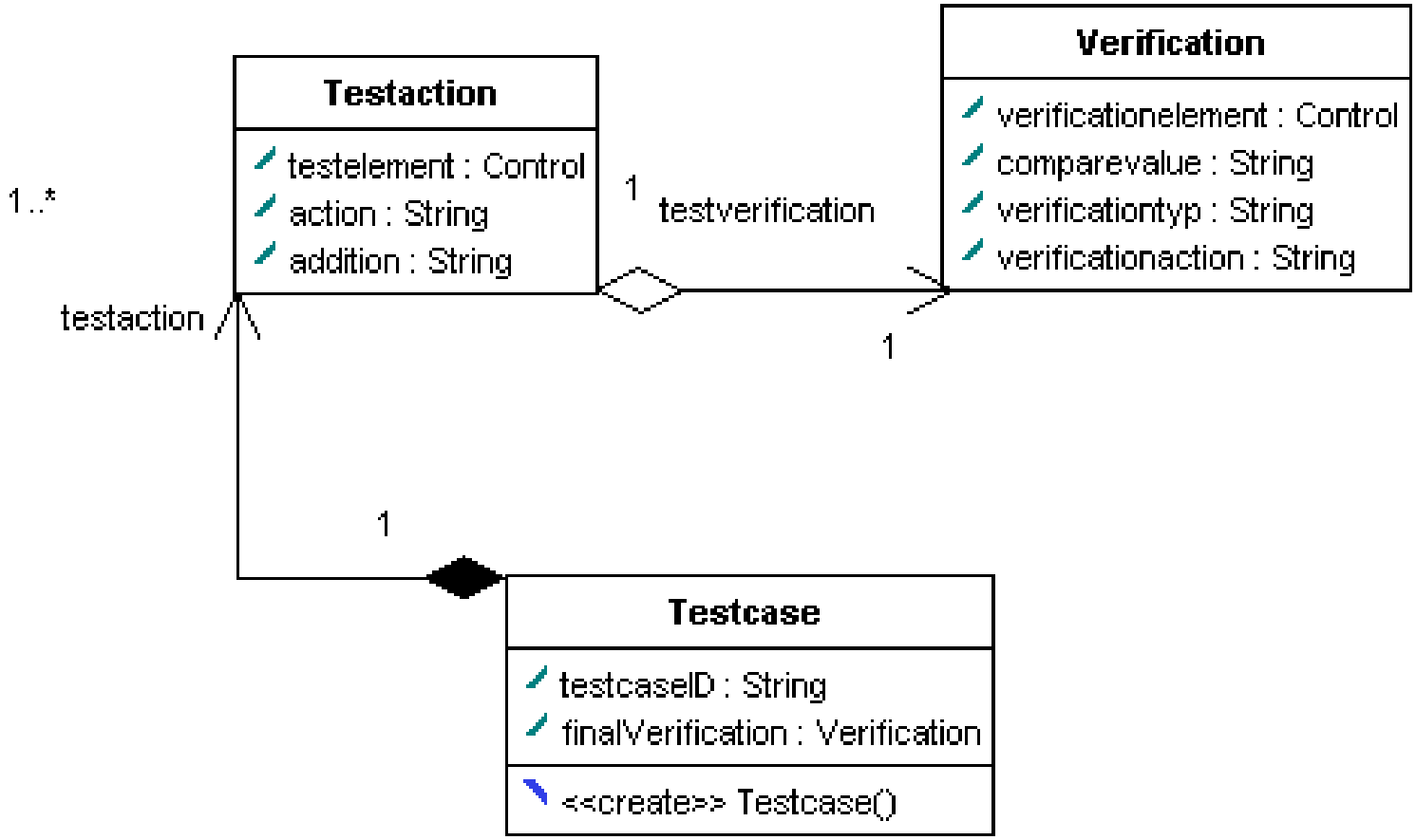
Gewählte Umsetzungsdetails für die Modellierung des Systemtests

1. Benutzersicht auf das Gesamtsoftwaresystem, d.h. Nutzen der Systemoberfläche durch Nutzen der Ein- und Ausgabeelemente
2. Bindung der Identifikationsmöglichkeit der Elemente an den Programmcode/das Systemmodell für frühe Integration und Wartbarkeit
3. Testablauf als Verknüpfung elementarer Funktionen der Oberflächenelemente (Testaction)
4. Über Daten gesteuerten prozessorientierter Testablauf (Keyword-driven) zur Erhöhung der Wiederverwendbarkeit
5. Übertragbares Testverhalten abgestimmt auf die als Basis genutzte Teststruktur (Modellierung der Teststruktur als Teil der Lösung)

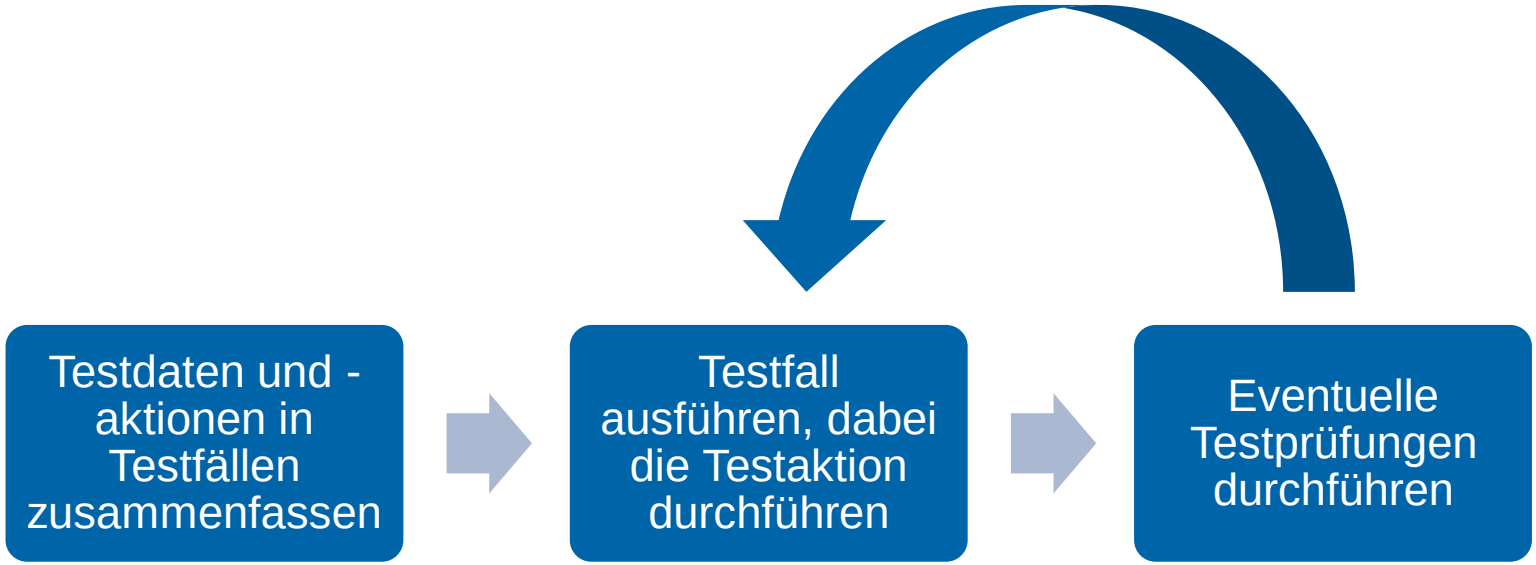
Testaction als kleinste wiederverwendbare Struktur



Testfall als Strukturelement



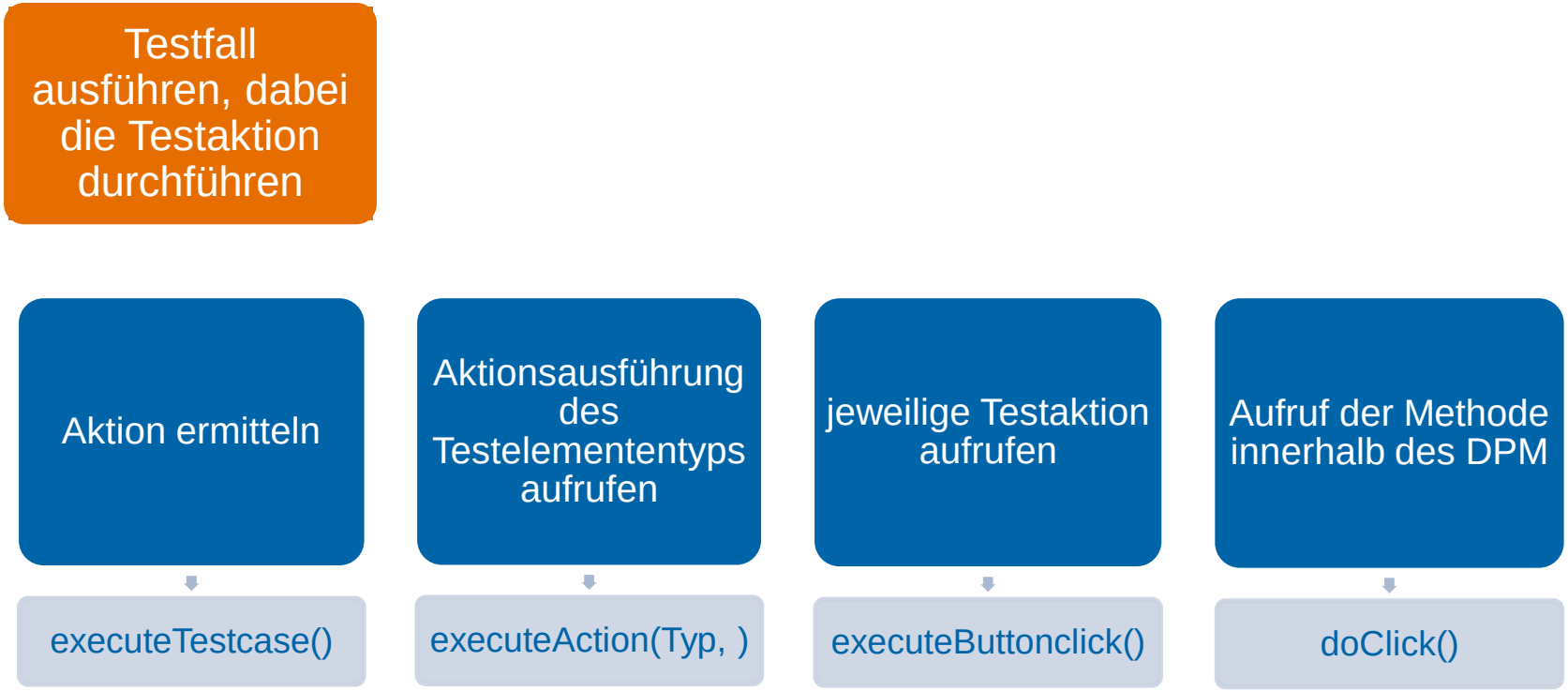
Testablaufsteuerung



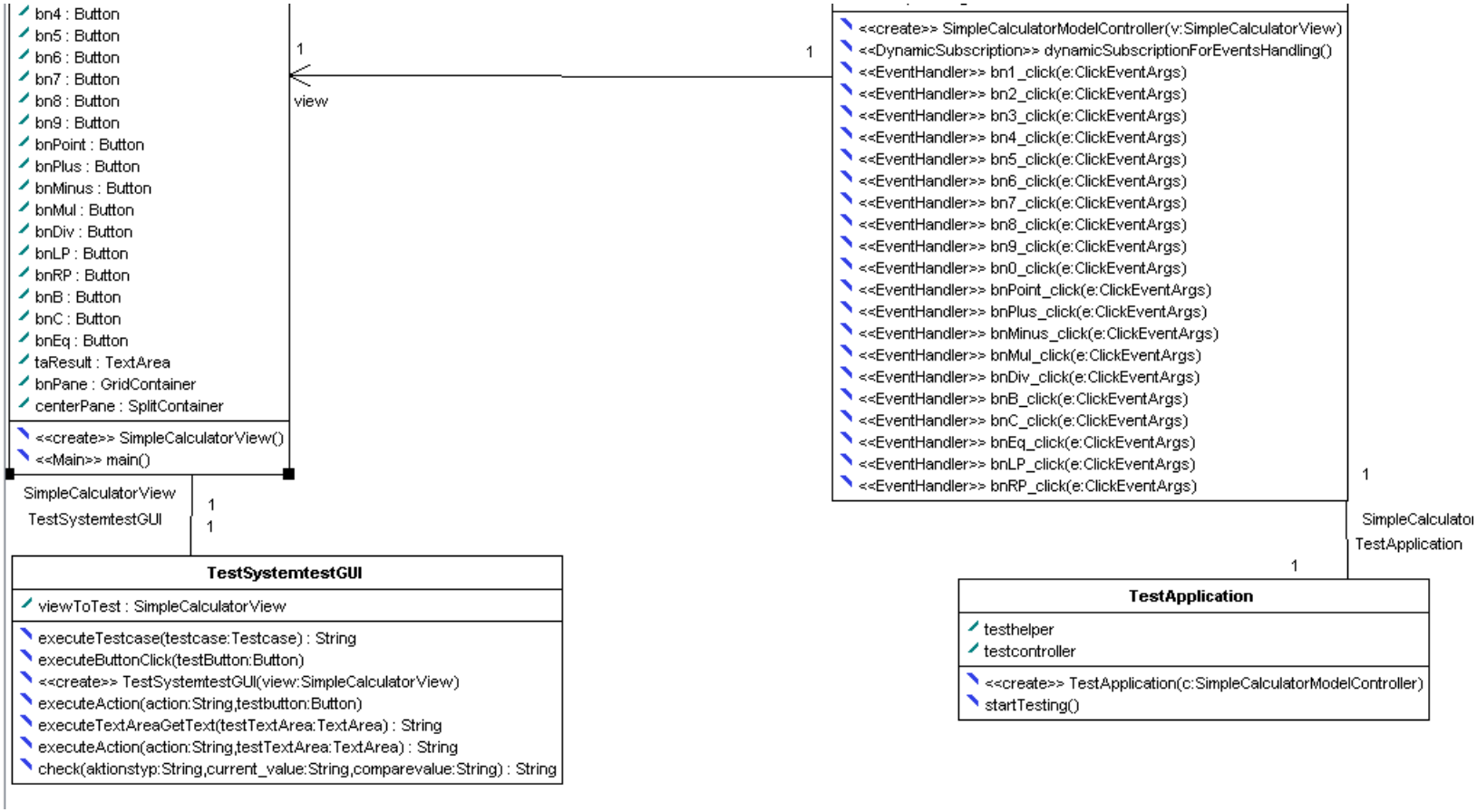

```
test1 : Testcase;
update test1.testcaseID  = "01";
testaction1 : Testaction;
update testaction1.action = "click";
update testaction1.testelement = testcontroller.view.bn1;
update testaction1.addition = "";
update testaction1.testverification = newverification;
update test1.testaction->including(testaction1);
update test1.testcaseresult =testhelper.executeTestcase(test1);
end
```

```
begin
 current: Testaction;
 update current = currentTestcase.testaction->at(i);
 testElement: Control;
 update testElement = current.testelement;
update className = "JButton";
if className = "JButton"
 then
  begin
   testbutton: Button;
   update testbutton = testElement.oclAsType(Button);
   update executeAction(current.action, testbutton);
 end
```

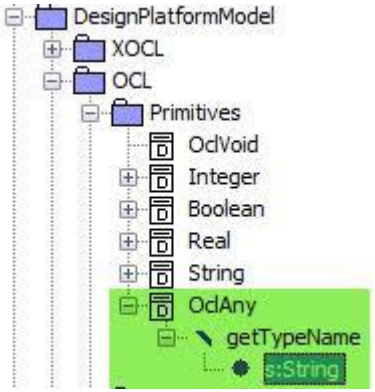
Testaktion durchführen



Integrations in Anwendungsmodell



Funktionen aus dem Platformodel modellieren




```
514   <dpmOperationMappingRule>
515     <sourceOperation>OCL.Primitives.OclAny.getTypeName()</sourceOperation>
516     <operationSubstitution>
517       <tpmOperationName>getClass().getSimpleName</tpmOperationName>
518     </operationSubstitution>
519   </dpmOperationMappingRule>
```

```
oqb drawFunction() {language OCL}

begin

it_var:Integer = 1;

while it_var < self.pointsX->size()
 begin
  update g.drawLine( self.lineColor,self.pointsX->at(it_var-1), self.pointsY->at(it_var-1), self.pointsX->at(it_var), self.pointsY->at(it_var));
  update it_var = it_var + 1;
 end
endwhile
update OutputStream::writeToConsole( self.getTypeName());
end
```

```
443   <dpmOperationMappingRule>
444     <sourceOperation>XOCL.(
445     <operationSubstitution>
446       <useTPMcodeTemplate>t
447       <tpmCodeTemplate>
448           java.io.ByteArray
449           extraLib.parser.a
450           extraLib.parser.a
451           parser.Parse();
452           if(parser.errors.
453               %Param1% = St
454           }
455       </tpmCodeTemplate>
456
457       <tpmCodeTemplateParameter>%Param0%</tpmCodeTemplateParameter>
458       <tpmCodeTemplateParameter>%Param1%</tpmCodeTemplateParameter>
459     </operationSubstitution>
460   </dpmOperationMappingRule>
```

Offene Punkte

- Architektur innerhalb des Systemmodells (zwei Klassen)
- Kompaktere Weise der Testdatenmodellierung