anderScore
trust in competence

Codegeneratoren mit Xtend2

11.04.2012, A. Arnold

© Copyright 2012 anderScore GmbH

Inhalt

anderScore
trust in competence

1. Was ist Xtend2? 3
2. Xtend2 Konzepte 4
3. Hands On! 8
20

Was ist Xtend2?

anderScore trust in competence

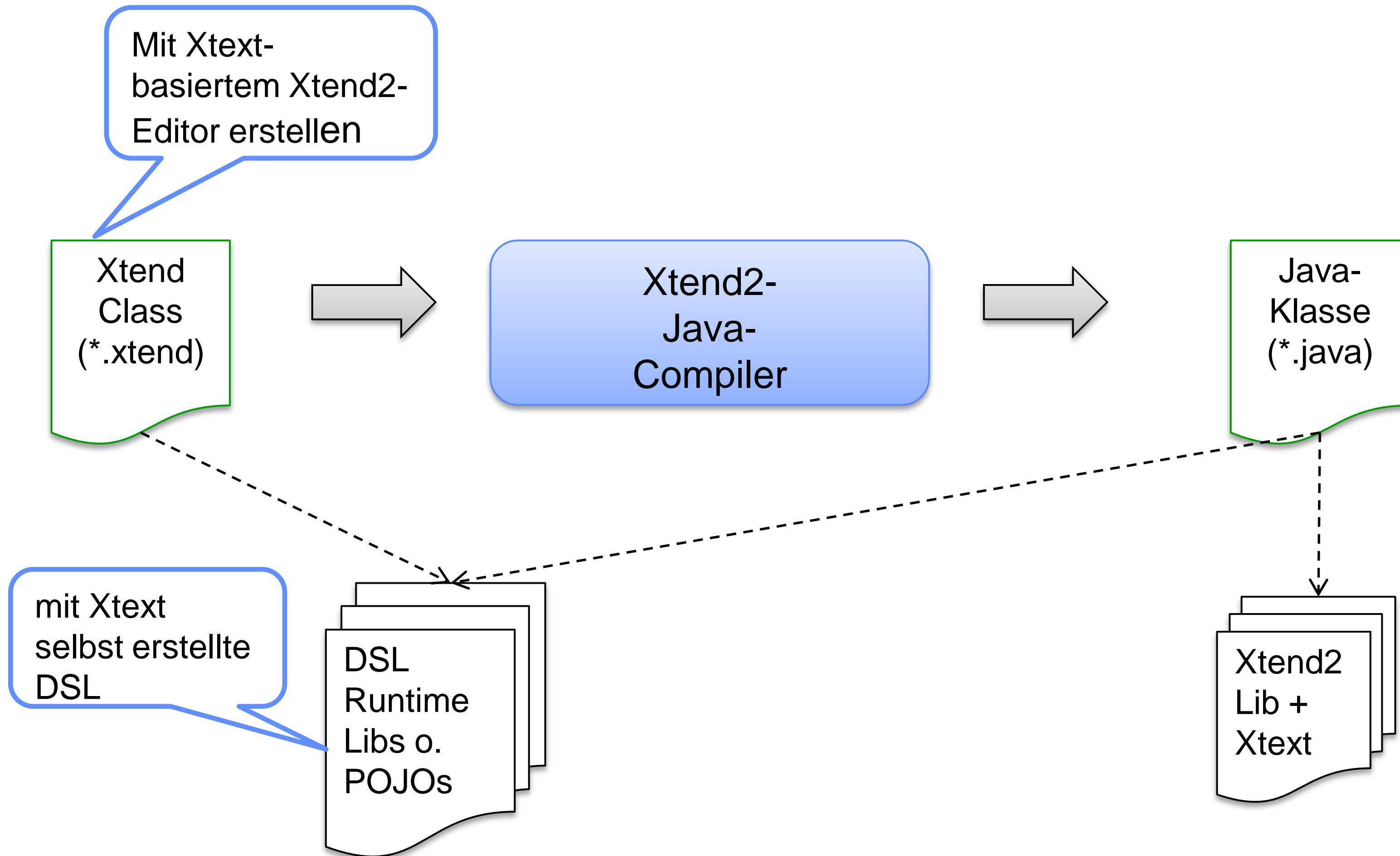
- Funktionale, objektorientierte Sprache für Codegeneratoren
  - Klassen
  - Closures
  - Extension Methods
  - Multidispatch
- Nachfolger von Xpand / Xtend / Check
- Basiert auf Xbase
  - an Java gebundene Basissprache mit Expressions + Closures
  - basiert auf Java (Typsystem etc.) + Xtext
- Compiliert zu Java
  - Unmittelbare Integration mit Java, s. Xbase
  - Xtend2 besteht aus Library + Compiler

anderScore
trust in competence

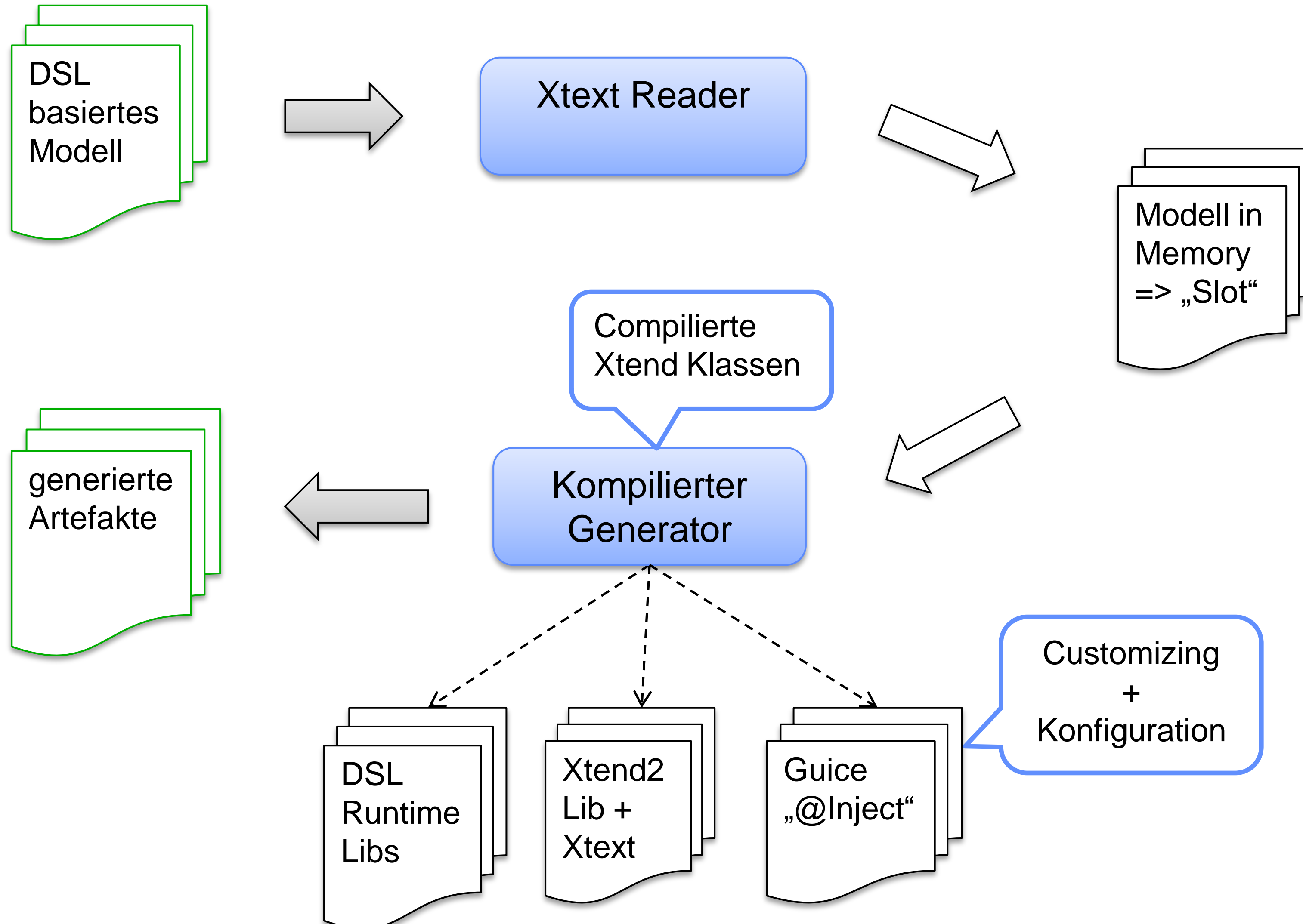
XTEND2 KONZEPTE

© 2012 anderScore GmbH

Generator erstellen



Codegenerierung



Xtend Klassen

anderScore trust in competence

- Sind Java-Klassen (nach Kompilierung)
  - Vererbung
  - Generics
  - Methoden
  - Attribute
  - …

- Bsp.:

```
class HelloWorld {
    def String sayHelloTo(String to)
      val date = new Date()
      var count = 1
      count++
      return "Hello "+to+"!"
    }
}
```

Extension Methods

anderScore trust in competence

- Verwendete Klassen/Typen im Scope einer Xtend-Class um zusätzliche Methoden erweitern
  - Methoden haben Parameter des zu erweiternden Typs als ersten Parameter
  - Methode in (anderer) Xtend-Class definiert
- Bsp.:

```
def String toPackageName (Type t) {
        switch (t.eContainer) {
                Package :  (t.eContainer as Package).name
                default:     ""
        }
}
….
        class PojoGenerator implements IGenerator {

            @Inject extension TypeExtensions

            def generatePojo (BusinessObject bo, IFileSystemAccess fsa) {
                val content = '''
                package «bo.toPackageName»;
```

Multidispatch

anderScore trust in competence

- Zur Laufzeit die am besten passende Methode für den Aufruf aussuchen
  - Parametertypen zur Laufzeit ausgewertet
  - Nur mit „dispatch“ gekennzeichnete Methoden
  - Java wertet Parametertypen zur Compiletime aus (kein Multidispatch)
    - Xtend-Verhalten, wenn Xtend-Methoden nicht mit „dispatch“ gekennzeichnet

Multidispatch Beispiel

anderScore trust in competence

- Bsp.: Type sei Basistype von BusinessObject

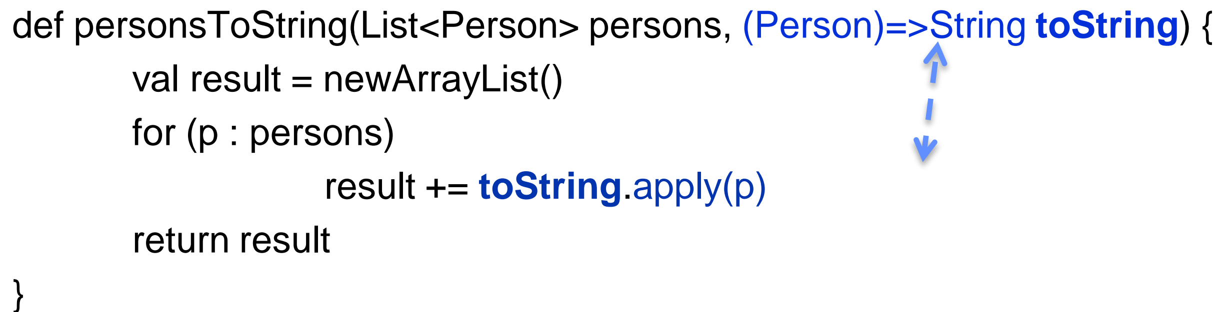
```
//Die Definition…

def dispatch toTypeName (Type t) {
}

def dispatch toTypeName (Enum t) {
}

def dispatch toTypeName (BusinessObject bo) {
}

… und der Aufruf …
def Iterable<String> getTypeNames (Package p) {
        return p.getTypes().map ( t | t.toTypeNa…
}
```

Polymorpher Methodenaufruf mit Multidispatch

Closures

- s. Groovy, Scala etc.
- Bsp.:

```
def closures_01(List<Person> persons) {
        persons.personsToString(p | p.name+", "+p.forename )
}
```



```
/** Funktion mit Closure als Parameter “toString” */
def personsToString(List<Person> persons, (Person)=>String toString) {
        val result = newArrayList()
        for (p : persons)
                result += toString.apply(p)
        return result
}
```

```
/** Funktionsaufruf mit Closure */
def closures_02(List<Person> persons) {
        Collections::sort(persons, [ a, b | a.name.compareTo( b.name ) ])
}
```

Alles ist ein Ausdruck

anderScore trust in competence

- Also kann alles Variablen zugewiesen oder als Parameter an Methoden übergeben werden …

- Bsp.:

```
val data = try {
    fileContentsToString('data.txt')
  } catch (IOException e) {
    'data.txt'
  }
```

Bessere Switch-Expressions

anderScore trust in competence

- Ähnlich „switch“ Statement in Java
- Aber:
  - Case-Ausdrücke mit Strings oder Expressions
  - Switch ist selbst ein Ausdruck
  - Case-Ausdrücke können Type-Guards haben
    - Nur bei passendem Typ des switch-Values ausgewertet
- Bsp.:

```
val Shape shape = ...
switch (shape) {
    Rectangle case shape.width == shape.height : "Square ("+shape.width+")"
    Rectangle :                         "Rectangle ("+shape.width+" x "+shape.height+")"
    Circle :    "Circle ("+shape.diameter+")"
    default :  "Don't know"
}
```

Switch Besipiele

anderScore trust in competence

```
/* mit Type Guards */
def switchExpression_01(Shape shape) {
    switch (shape) {
        Circle      :           'a circle : diameter='+shape.diameter
        Rectangle case shape.height == shape.width :
                                'a square : size='+shape.width
        Rectangle :             'a rectangle : width='+shape.width+', height='+shape.height
    }
}

def switchExpression_02(String value) {
    switch(value) {
        case 'foo' :            "it's foo "
        case 'bar' :            'a bar'
        default :               "don't know "
    }
}
```

Template Expressions – „Rich Strings“

anderScore trust in competence

- Strings mit Ausrücken, die beim Schreiben in die Datei ersetzt werden
- mit ''' ''' markiert, Ausdrücke stehen in «»
- Bsp.:

```
def writeLetterTo (Person p) '''
        Dear «p.forename»,

        bla bla foo

        Yours sincerely,

        Joe Developer

                «signature»
'''
```

- Einrückungen für Blockorientierte Zielsprachen berechnet

Ausdrücke in Rich Strings

anderScore trust in competence

- Eingebettete Bedingungen
  - Bedingte Evaluierung de jeweils zw. «IF » … «ELSE IF » … «ELSE» …«ENDIF» Eingeschlossenen Blöcke
  - Bsp.:
    - '''
    - Hello «IF name != null» «name» «ELSE» Mr. X«ENDIF» !
    - '''
- Eingebettete Schleifen
  - Schleife umBlock zwischen «FOR …» und «ENDFOR»
  - Bsp.:
    - «FOR attr : bo.attributes»
    - private String «attr.name»;
    - «ENDFOR»

Outlets

anderScore trust in competence

- Konfigurierbare Zielordner
  - Haben logischen Namen
  - Nutzer des Generators deklariert für jedes Outlet einen Basisordner

- Artefakte beim Generieren unterhalb eines passenden Zielordners ablegen
  - z.B. Java getrennt von XML

- Bsp.:
  - Im Generatorskript:

```
outlet = {
        outletName = 'Java'
        path = "src/generated/java"
 }
```

  - Im Template:

```
def generate (BusinessObject bo, IFileSystemAccess fsa) {
        fsa.generateFile (bo.toPojoFileName, "Java", content)
```

Konfiguration / @Inject

anderScore trust in competence

- Implementierungen ersetzen mit Guice => Subklassen bzw. Interface-Implementierung nutzen

Bsp.:

```
class UIModelGenerator {
        @Inject IImportsProvider impProv

        def generate (BusinessObject bo) '''
                toImports ( impProv.getImports(bo) )
        '''
}

class MyImportProvider implements IImportProvider {
        override getImports (BusinessObject bo) {
        }
}
```

Konfiguration / @Inject cont…

anderScore trust in competence

- Konfiguration erfolgt in Guice-Module

- für Xtext GeneratorComponent
  - Guice Module wird in einer Implementierung von ISetup angelegt
  - ISetups werden bei der GeneratorComponent registriert

- Oder einfach in Java main()

- Bsp.:

```
public class MyGeneratorModule extends AbstractModule {
    public void configure () {
        …
        bind (IImportProvider.class) .to (MyImportProvider.class);
        …
        }
    }
    …
```

Workflow Engine MWE2

- ANT ähnliches Scripting von Generatoraufrufen
- Basiert auf Xtext und dem alten MWE

- Workflow besteht aus Components
  - Reader zum Modell einlesen
  - Generator zum Generieren

- Workflows können wie Komponenten behandelt werden
  - Workflows können Workflows aufrufen

anderScore
trust in competence

HANDS ON

© 2012 anderScore GmbH