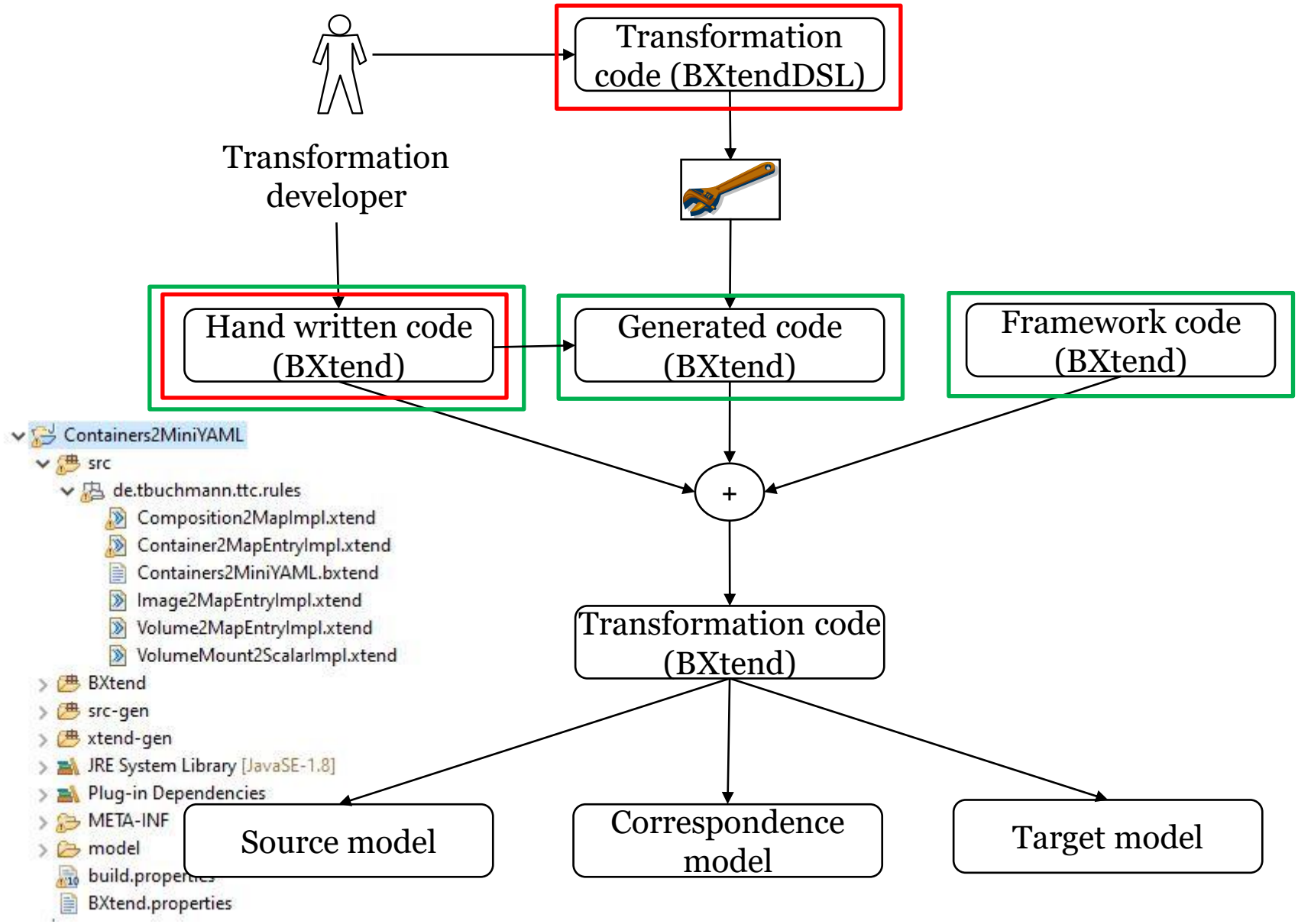
A BXtendDSL Solution for the Containers to MiniYAML Case

Thomas Buchmann

BXtendDSL

- Small and lightweight external DSL
- Using BXtendDSL the transformation developer essentially declares correspondences between elements of source and target models
- BXtendDSL is **intentionally** incomplete
  - Usually it is not possible to solve a transformation completely on the declarative level (as this would require a more expressive and comprehensive language)
  - Rather, from a transformation definition written in BXtendDSL code on top of the BXtend framework is generated
  - Subsequently, the generated code is extended with manually written imperative code


Transformation developer
Transformation code (BXtendDSL)
Hand written code (BXtend)
Generated code (BXtend)
Framework code (BXtend)
+
Transformation code (BXtend)
Source model
Correspondence model
Target model
Containers2MiniYAML
src
de.tbuchmann.ttc.rules
Composition2MapImpl.xtend
Container2MapEntryImpl.xtend
Containers2MiniYAML.bxtend
Image2MapEntryImpl.xtend
Volume2MapEntryImpl.xtend
VolumeMount2ScalarImpl.xtend
BXtend
src-gen
xtend-gen
JRE System Library [JavaSE-1.8]
Plug-in Dependencies
META-INF
model
build.properties
BXtend.properties

BXtendDSL Solution: Declarative Layer

```
1  sourcemodel "http://york.ac.uk/ttc/containers/1.0.0"
2  targetmodel "http://york.ac.uk/ttc/miniyaml/1.0.0"
3
4  rule Volume2MapEntry
5      src Volume v;
6      trg MapEntry me | filter;
7
8      v.name <--> me.key;
9
10 rule Image2MapEntry
11     src Image img;
12     trg MapEntry me | filter, creation;
13
14     img.image <--> me.value;
15
16 rule VolumeMount2Scalar
17     src VolumeMount vm;
18     trg Scalar sc | filter;
19
20     vm.path vm.volume --> sc.value;
21
22 rule Container2MapEntry
23     src Container c;
24     trg MapEntry me | filter;
25
26     c.name <--> me.key;
27     c.image c.replicas c.dependsOn {c.volumeMounts: VolumeMount2Scalar} --> me.value {me.value: VolumeMount2Scalar};
28     c.image c.replicas c.dependsOn <-- me.value;
29
30 rule Composition2Map
31     src Composition c;
32     trg Map m | filter, creation;
33
34     {c.nodes: Image2MapEntry, Container2MapEntry, Volume2MapEntry} --> m.entries {m.entries: Image2MapEntry, Container2MapEntry, Volume2MapEntry};
35     c.nodes <-- m.entries {m.entries: Image2MapEntry[img], Container2MapEntry[c], Volume2MapEntry[v]};
36
```

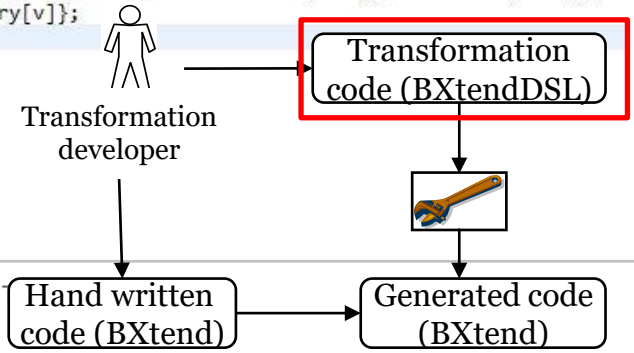
define source and target metamodels

Mapping of attributes / references

Specification of a transformation rule

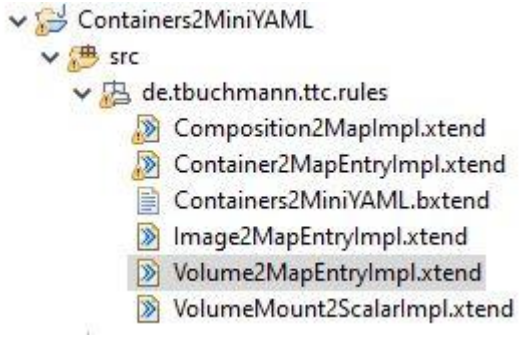
Modifiers for creating stubs on the imperative layer

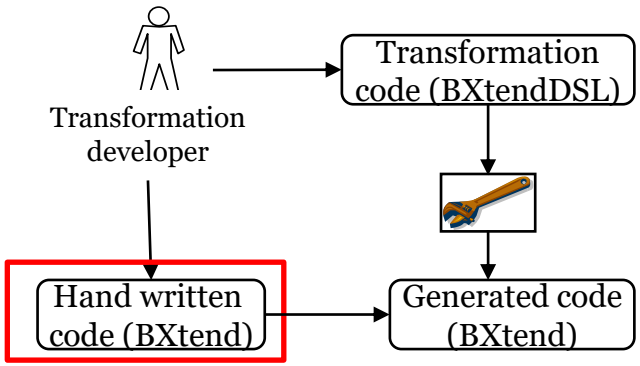
Map root containers of involved models



BXtendDSL Solution: Imperative Layer

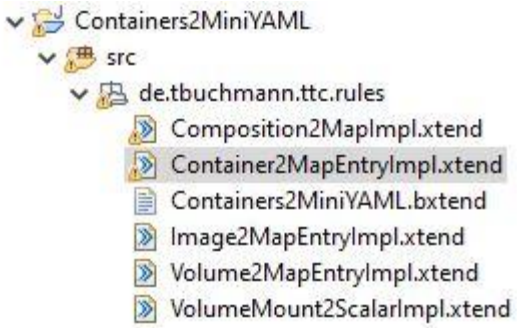
```
12  override protected filterMe(MapEntry me) {
13      (me.key != "services" || me.key != "version" || me.key != "volumes") &&
14      me.eContainer instanceof Map &&
15      me.eContainer.eContainer instanceof MapEntry &&
16      (me.eContainer.eContainer as MapEntry).key == "volumes"
17  }
```



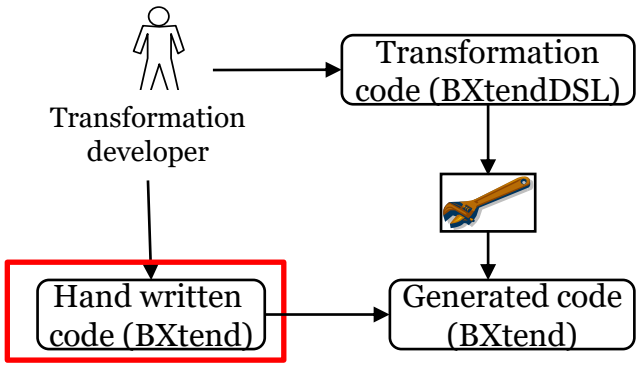


BXtendDSL Solution - Imperative Layer: Forward direction

```
22 rule Container2MapEntry
23     src Container c;
24     trg MapEntry me | filter;
25
26     c.name <--> me.key;
27     c.image c.replicas c.dependsOn {c.volumeMounts: VolumeMount2Scalar} --> me.value {me.value: VolumeMount2Scalar};
28     c.image c.replicas c.dependsOn <-- me.value;
```

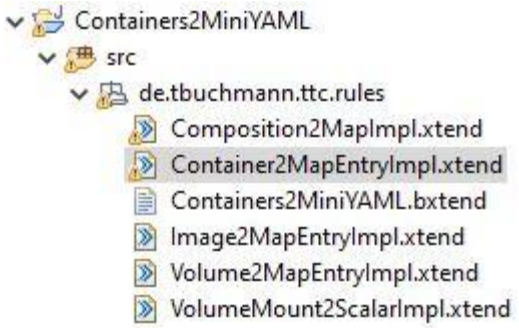



```
override protected valueFrom(Image image, int replicas, List<Container> dependsOn, List<Scalar> volSc, Value oldValue) {
    var entry = MiniyamlFactory.eINSTANCE.createMap()

    if (replicas > 1) {
        val me = MiniyamlFactory.eINSTANCE.createMapEntry()
        me.key = "replicas"
        me.value = MiniyamlFactory.eINSTANCE.createScalar() => [s | s.value = "" + replicas]
        entry.entries += me
    }

    if (image != null)
        entry.entries += (elementsToCorr.get(image).getTarget().get(0) as SingleElem).element as MapEntry

    if (!dependsOn.isEmpty) {
        val me = MiniyamlFactory.eINSTANCE.createMapEntry() => [m | m.key = "depends_on"]
        val list = MiniyamlFactory.eINSTANCE.createList()
        me.value = list
        for (Container c : dependsOn) {
            list.values += MiniyamlFactory.eINSTANCE.createScalar() => [s | s.value = c.name]
        }
        entry.entries += me
    }
    if (!volSc.empty) {
        val me = MiniyamlFactory.eINSTANCE.createMapEntry() => [m | m.key = "volumes"]
        val list = MiniyamlFactory.eINSTANCE.createList()
        me.value = list
        for (Scalar s : volSc)
            list.values += s
        entry.entries += me
    }

    // check if there are unmatched entries left
    if (oldValue != null && oldValue instanceof Map) {
        for (MapEntry me : ((oldValue as Map).entries)) {
            if (me.key == "restart") {
                val newME = MiniyamlFactory.eINSTANCE.createMapEntry() => [key = me.key]
                val newVal = MiniyamlFactory.eINSTANCE.createScalar() => [value = (me.value as Scalar).value]
                newME.value = newVal
                entry.entries += newME
            }
            if (me.key == "tmpfs") {
                val newME = MiniyamlFactory.eINSTANCE.createMapEntry() => [key = me.key]
                val newList = MiniyamlFactory.eINSTANCE.createList()
                newME.value = newList
                val oldMEValue = me.value
                for (Value v : (oldMEValue as miniyaml.List).values ) {
                    val newV = MiniyamlFactory.eINSTANCE.createScalar => [value = (v as Scalar).value]
                    newList.values += newV

                }
                entry.entries += newME
            }

        }
    }

    new Type4value(entry)
}
```

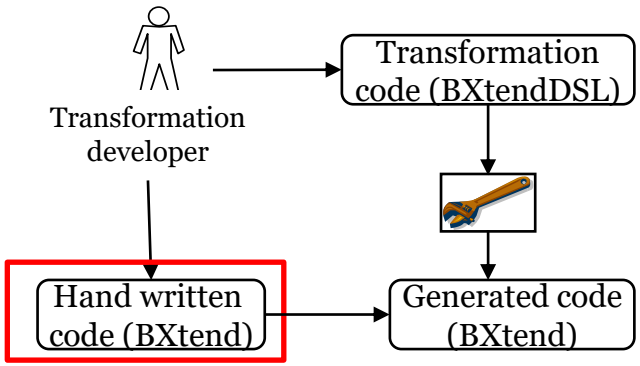


BXtendDSL Solution - Imperative Layer: Backward direction

```
22 rule Container2MapEntry
23     src Container c;
24     trg MapEntry me | filter;
25
26     c.name <--> me.key;
27     c.image c.replicas c.dependsOn {c.volumeMounts: VolumeMount2Scalar} --> me.value {me.value: VolumeMount2Scalar};
28     c.image c.replicas c.dependsOn <-- me.value;
```




```
91  override protected image_replicas_dependsOnFrom(Value value) {
92      val depends = newArrayList
93      var replicas = 1
94      if (value instanceof Map) {
95          for (MapEntry me : ((value as Map).entries)) {
96              if (me.key == "replicas") {
97                  replicas = new Integer((me.value as Scalar).value).intValue
98              }
99          }
100     }
101
102     return new Type4image_replicas_dependsOn(null, replicas, depends)
103 }
```



Evaluation

- Quantitative Analysis: LOC Metrics

Metric	BXtendDSL Declarative	BXtendDSL Imperative	Total
LOC	32	202	234
#Words	100	788	888
#Characters	862	5967	6829

- Qualitative Analysis: BXtendDSL solution passes all test cases where YAML key order is ignored
  - and all Batch FWD tests, where YAML key order is preserved
  - only 1 Incremental FWD test with preserving YAML key order fails (updReplicas)

- Performance Analysis: BXtendDSL solution is fast and scalable

Conclusion

- We used BXtendDSL – a DSL for specifying bidirectional and incremental model transformations
- The case was challenging and required a significant portion of code on the imperative layer
- The declarative language needs an extension to allow navigation to container elements
  - and a way to loosen strict checking of applied rules when assigning previously transformed elements to references
  - Code generator needs to be updated accordingly
- Find the solution on
  - https://github.com/tbuchmann/benchmarxTTC2023/tree/main

Thank you!

Any Questions?