KMEHR to FHIR case solution with UML-RSDS

Kevin Lano, King's College London, UK
Alireza Rouhi, Azabaijan Shahid Madani University, Iran

- Analyse KMEHR to FMIR reference solution for quality flaws
- Define UML-RSDS solution which avoids some flaws
- Show this solution has effective performance
- Show how an inverse transformation can be derived from UML-RSDS solution.

Reference ATL solution

- Large scale transformation: 20 matched rules, 32 lazy rules, 42 helpers. Library package is 346 LOC, main transformation module is 973 LOC
- Quality aspects – Maximum OCL expression length (MEL)
- Excessive fan-out (EFO)
- Excessive parameter length (EPL)
- Excessive rule size (ERS)
- Magic numbers (MGN)
- Duplicated code (DC).

High MEL, ERS, EFO and EPL in a rule can hinder comprehension and testing. MGN and DC increase maintenance effort.

Quality issues in ATL transformation

Rule	*Issues*
Folder	c = 133, MEL (size = 110), EFO = 8
SumEHRTransaction	c = 124, MEL (size = 25), EFO = 12, MGN = 6
... *WithAuthor*	MGN = 3
... *WithCustodian*	MGN = 4, DC: *refPrefix* out-pattern is a parameterised clone.
Patient	c = 99, EFO = 6, MEL (size = 16)
PatientContact	DC: cloned *humanName* with *Patient* rule
Organization	MGN = 1, DC: cloned out-patterns with *Practitioner* rule

Quality issues in ATL transformation

Practitioner	MGN = 1
Medication	MEL (size = 15), cloned in *Vaccine* (DC)
Posology	c = 160, EPL = 11, MGN = 2
...WithUnitAndTakes	EPL = 6, MGN = 5
AllergyOrIntolerance	c = 111, MGN = 12
...WithCode	MEL (size = 17), expression cloned in *ProblemWithCode* (DC)
Problem	c = 108, EFO = 6, MGN = 7
Vaccine	c = 118, EPL = 7, MGN = 2, MEL (size = 15, DC)

Migrating from ATL to UML-RSDS

UML-RSDS expresses transformations as UML use cases + operations.

- ATL matched rules correspond to UML-RSDS rules (use case postconditions)
- ATL called/lazy rules correspond to UML-RSDS operations
- ATL rule inheritance translates to UML-RSDS *rule conjunction*.

For example, ATL rules:

```
rule A2B {
  from a : A
  to b : B
     ( y <- a.x )
}

rule A2C extends A2B {
  from a : A
  to b : B ( cs <- Set{c} ),
     c : C ( z <- a.x->size() )
}
```

Translate to:

```
A::
  B->exists( b | b.$id = self.$id & b.y = self.x )
```

```
A::
  B->exists( b | b.$id = self.$id &
    C->exists( c | c.$id = "c_" + $id &
      c.z = self.x->size() & b.cs = Set{c} ) )
```

`mapsTo` keyword of ATL indicates which input and output elements are linked by same identity.

`to`

```
  t : T mapsTo s ( ... )
```

for out variable t means $t.\$id = s.\$id$ instead of $t.\$id = self.\$id$.

Improved solution in UML-RSDS

- Re-expressed in UML-RSDS using similar main rules, but with factoring to reduce number of clones + exploit similarities between different rules.
- MEL reduced by fine-grain expression factoring
- MGN cases removed by introducing named constants
- Classes such as *FhirBoolean* and *FhirString* made into *value types*: only one instance for a given value.

Quality improvement

- More concise: matched rules part reduced to 45% of original length (LOC)
- Clones & similar processing steps replaced by calls of operations that factor out duplicated code
- Frequency of magic numbers & other flaws reduced.

Total MGN in matched rules reduced from 47 to 13.

MEL for *Folder* reduced from size = 110 to 6, MEL for *SumEHRTransaction* from size = 25 to 12.

The 4 exact clones and 1 parameterised clone of ATL version removed.

Rule	*ATL length*	*UML-RSDS length*
DocumentRoot	8	5
Folder	25	12
SumEHRTransaction	48	24
SumEHRTransactionWithAuthor	24	7
SumEHRTransactionWithCustodian	27	10
Patient	30	15
Address	11	7
Telecom	16	11
PatientContact	23	11
Organization	23	7

Practitioner	27	8
Medication	23	7
Posology	54	13
PosologyWithUnitAndTakes	35	15
AllergyOrIntolerance	45	22
AllergyOrIntoleranceWithCode	20	7
Problem	39	24
ProblemWithCode	19	7
Vaccine	44	14
Total	541	226

Performance

Input model	*Execution time (ms)*	*Output model size (KB)*	*Memory use (MB)*
1	31.3	77	64
10	56.3	440	457
100	194.3	4171	780
1000	3857	42560	1073

For largest model, stack size was increased to 8MB.

Inverse transformation

Based on inverting predicates, eg., assignment

$$t.g = Set\{s.f\}$$

inverts to $s.f = t.g \rightarrow any()$.

Inverse of rule

```
A::
  PCond(a) =>
    B->exists( b | b.$id = $id & SCond(b) & Succ(a,b) )
```

is

```
B::
  SCond(b) =>
    A->exists( a | a.$id = $id & PCond(a) & Succ~(a,b) )
```

Inverse transformation

Reconstructs KMEHR source information from FHIR model built using forward transformation.

Eg., in *Address* rule, assignment

```
addrx.postalCode = Set{FhirString.newFhirString(self.zip)}
```

inverts to:

```
self.zip = addrx.postalCode.any.value
```

fstr.any.value is inverse function of *FhirString.newFhirString*(*fstr*).

Some functions modified so they can be inverted, eg., *addressLine*() should be *tab*-separated concatenation of *street*, *housenumber* and *postboxnumber*.

Inverse transformation

Inverse of $\rightarrow collect(\ x\ \mid\ expr(x)\ )$ is
$\rightarrow collect$ of $expr^{\sim}$ values.

For example:

```
t.given =
  s.firstname->collect( fn | FhirString.newFhirString(fn))
```

inverts to

```
s.firstname = t.given->collect( gn | gn.value )
```

Inverse transformation

- Defined inverse rules for *Patient* rule and all related rules of *PersonType* to *Patient* mapping
- Can recover KMEHR *PersonType* information from an FHIR *Patient*
- We noticed some source information is not mapped to the target, eg., *text* of allergy or intolerance. So complete source information cannot be reconstructed from target.

Conclusions

- Described alternative solution to KMEHR to FHIR case, using UML-RSDS
- More concise, improved quality measures compared to original
- Efficiency is satisfactory
- Can be used as basis of inverse transformation from FHIR to KMEHR.