Antonio García-Domínguez
15th Transformation Tool Contest
STAF 2023, Leicester, UK

The Epsilon Solution to the KMEHR to FHIR Case

General approach

M2M transformation

- Epsilon Transformation Language is similar to ATL
- Transformation was only specified as ATL code and author is unfamiliar with KMEHR and FHIR
- Decided to translate the ATL code to ETL

Focus: traceability

- ETL is an interpreted language - overhead expected compared to ATL's compiled execution
- Focus is more on traceability, and making it easier to relate input and output models

Differences between ATL and ETL

Assignment operators and helpers

ATL "←" vs ETL "::="

- Both mean "assign to l-value result of transforming r-value"
- "::=" cannot be used with values not transformed
- ETL lacks "mapsTo" to limit who is "equivalent": filtering is needed

Helpers vs context ops

- ATL helpers were turned into EOL context ops
- ETL does not have the . / → distinction from OCL
- EOL @cached controls memory spend to save time / produce consistent results (e.g. `uuid()`)

Number of source objects per rule

Listing 2: Excerpt of the Posology rule in ATL

```
rule Posology {
  from
    f : KMEHR!FolderType,
    tx : KMEHR!TransactionType,
    i : KMEHR!ItemType,
    s : KMEHR!PosologyType (
      i.posology = s and
      tx.item->includes(i) and
      f.transaction->includes(tx) and
      i.isMedication
    )
  to
    t : FHIR!MedicationStatement mapsTo s (
      // ...
    ),
    // ...
}
```

Listing 1: Excerpt of the Posology rule in ETL

```
1  rule Posology
2    transform s: KMEHR!PosologyType
3    to t: FHIR!MedicationStatement, msid: FHIR!Id,
4        /* ... */
5  {
6    var i = s.eContainer();
7    var tx = i.eContainer();
8    var f = tx.eContainer();
9    // ...
10 }
```

- ETL only allows one source (avoids "cartesian product" cost of pattern matching: ATL 4.8.0+ uses local search)
- No issue for this tx - just focus on innermost object and use eContainer()

Lazy rules and rule inheritance

Lazy rules

- Original tx used them heavily: most translated into ETL lazy rules, and some into EOL operations (e.g. `FhirString`)
- Lazy rules slow down ETL: would have needed to redesign tx to avoid them

Rule inheritance

- Original ATL had base non-abstract rule + extra rules which extended it
- ATL will only generate one set of objects across base rule + subrules
- ETL will produce separate objects across rules

Rule inheritance: ATL vs ETL

- ATL had `SumEHRTransaction` base rule + 3 extensions of it (one extension combined `WithAuthor` and `WithCustodian`)

- ETL just has one rule with two `if` statements in its body
  - Personally, I think this is easier to understand...

```
1  rule SumEHRTransaction {
2    from
3      f : KMEHR!FolderType,
4      s : KMEHR!TransactionType (
5        f.transaction->includes(s) and
6        s.cd->exists(cd | cd.value = 'sumehr')
7      )
8    to /* ... */}
9
10 rule SumEHRTransactionWithAuthor
11   extends SumEHRTransaction {
12   from
13     f : KMEHR!FolderType,
14     s : KMEHR!TransactionType (
15       not s.txAuthor.oclIsUndefined()
16     )
17   to /* ... */}
18
19 rule SumEHRTransactionWithCustodian
20   extends SumEHRTransaction {
21   from
22     f : KMEHR!FolderType,
23     s : KMEHR!T
24     i : KMEHR!It
25     s.item->inclu
26       i.cd->exist
27     )
28   to /* ... */}
```

Listing 4: Adapting multiple non-abstract rule inheritance of ATL to ETL

```
1  rule SumEHRTransaction
2    transform s: KMEHR!TransactionType
3    to t: FHIR!Composition,
4        cid: FHIR!Id,
5        patRef: FHIR!Reference,
6        cStatus: FHIR!CompositionStatus,
7        dateTime: FHIR!fhir::DateTime
8  {
9    guard: s.cd.exists(cd| cd.value = 'sumehr')
10   /* .... */
11   if (s.txAuthor().isDefined()) { /* ... */}
12   if (s.custodianItem().isDefined()) { /* ... */}
13 }
```

Enumeration literals in ATL and ETL

```
helper def : genderMap : Map(                                    1
  KMEHR!CDSEXvalues,                                             2
  FHIR!AdministrativeGenderEnum) =                               3
  Map {                                                          4
    (#changed, #other),                                          5
    (#female, #female),                                          6
    (#male, #male),                                              7
    (#unknown, #unknown),                                        8
    (#undefined, #other)                                         9
  };                                                             10
```

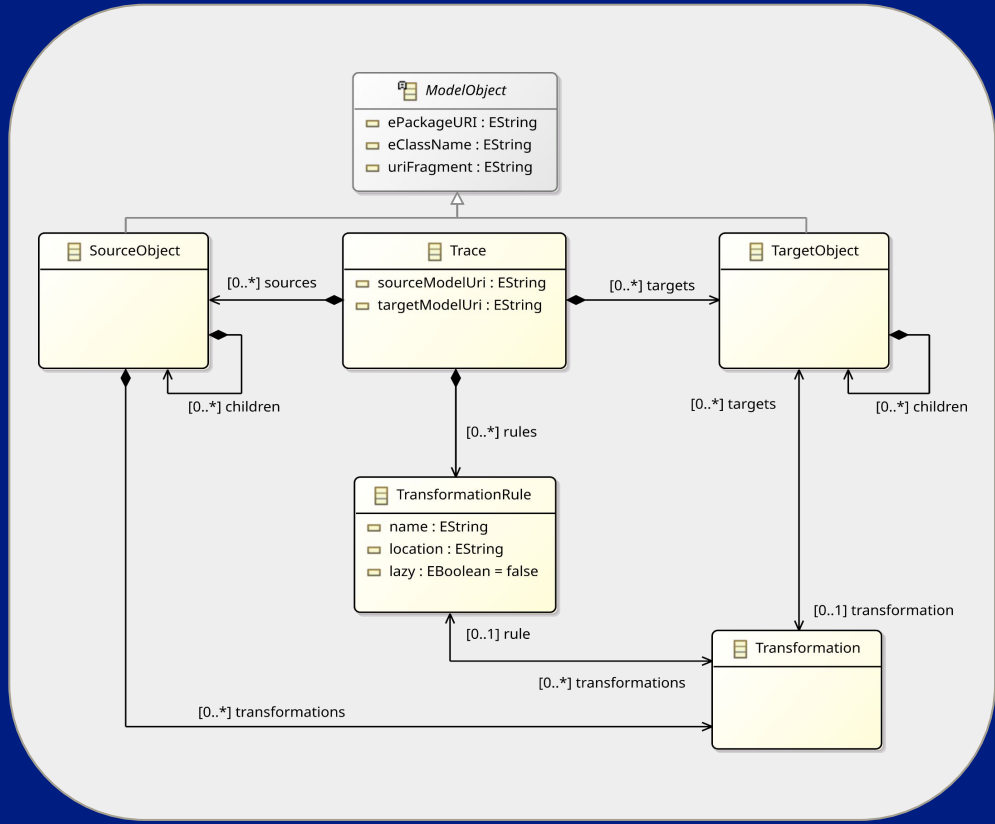
```
var genderMap = Map {                                            1
  KMEHR!CDSEXvalues#changed                                      2
    = FHIR!AdministrativeGenderEnum#other,                       3
  KMEHR!CDSEXvalues#female                                       4
    = FHIR!AdministrativeGenderEnum#female,                      5
  KMEHR!CDSEXvalues#male                                         6
    = FHIR!AdministrativeGenderEnum#male,                        7
  KMEHR!CDSEXvalues#unknown                                      8
    = FHIR!AdministrativeGenderEnum#unknown,                     9
  KMEHR!CDSEXvalues#undefined                                    10
    = FHIR!AdministrativeGenderEnum#other                        11
};                                                               12
```

- In `#changed`, ATL can guess enumeration from the context
- In Epsilon 2.4.0, you would need fully-qualified name (`KMEHR!CDSEXvalues#changed`)
- In Epsilon 2.5.0, it will be enough to have an unambiguous reference: `#changed` will work so long as there is no other enumeration literal with the same name
- Change was just merged, so we did not have time to work this into the solution

Generation and visualisation of transformation traces with Picto

Trace generation from ETL

- ETL produces trace but does not save it: users extract wanted info
- Java wrapper of ETL script has an algorithm to do this, based on custom metamodel (see figure)
- Trace models are standalone from source/target models due to lack of KMEHR/FHIR tree editors
- Containment forests from source/target models are reproduced, then pruned



Trace visualisation with Picto

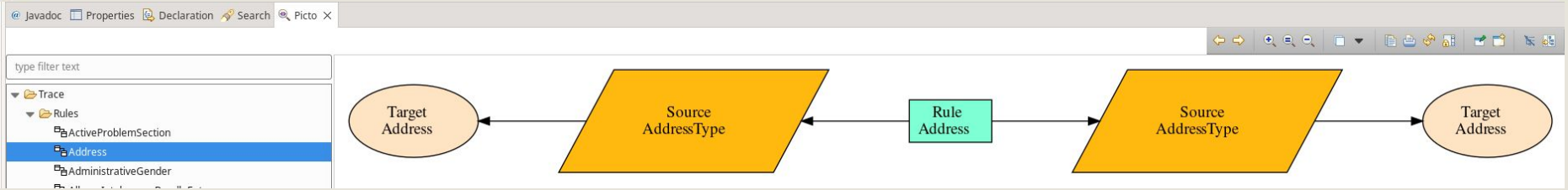
EGL and EGX

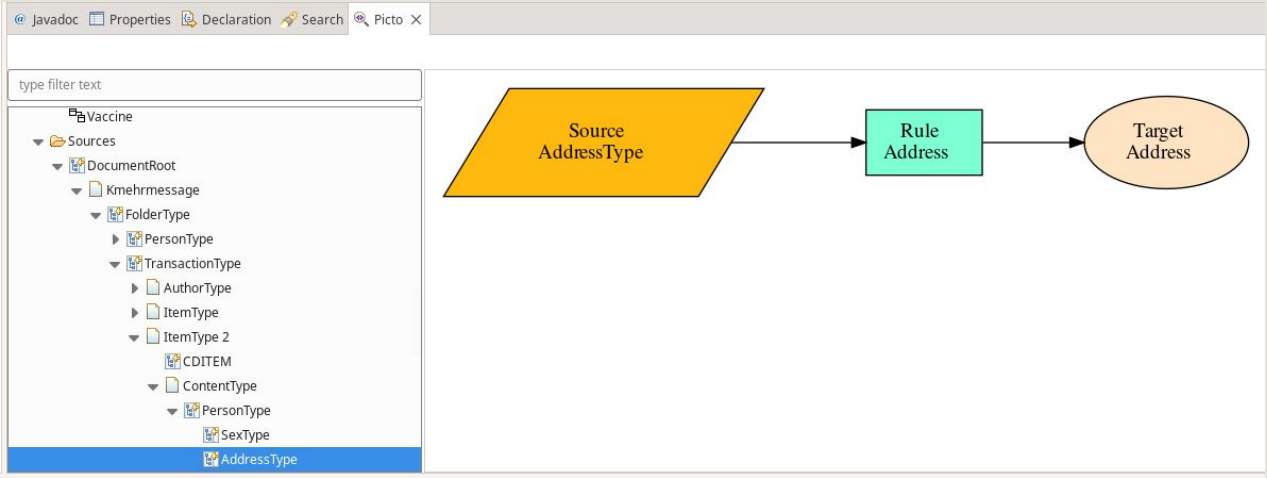
- EGL is an Epsilon language for writing model-to-text tx
- EGX is an orchestration language to decide which EGL scripts to run against which model elements

Using Picto

- Write EGX + EGL scripts which visualize the neighbourhood of an elem
- For trace file `x.trace`, add a `x.trace.picto` file pointing Picto to the EGX orchestration script
- Picto will do the UI for us

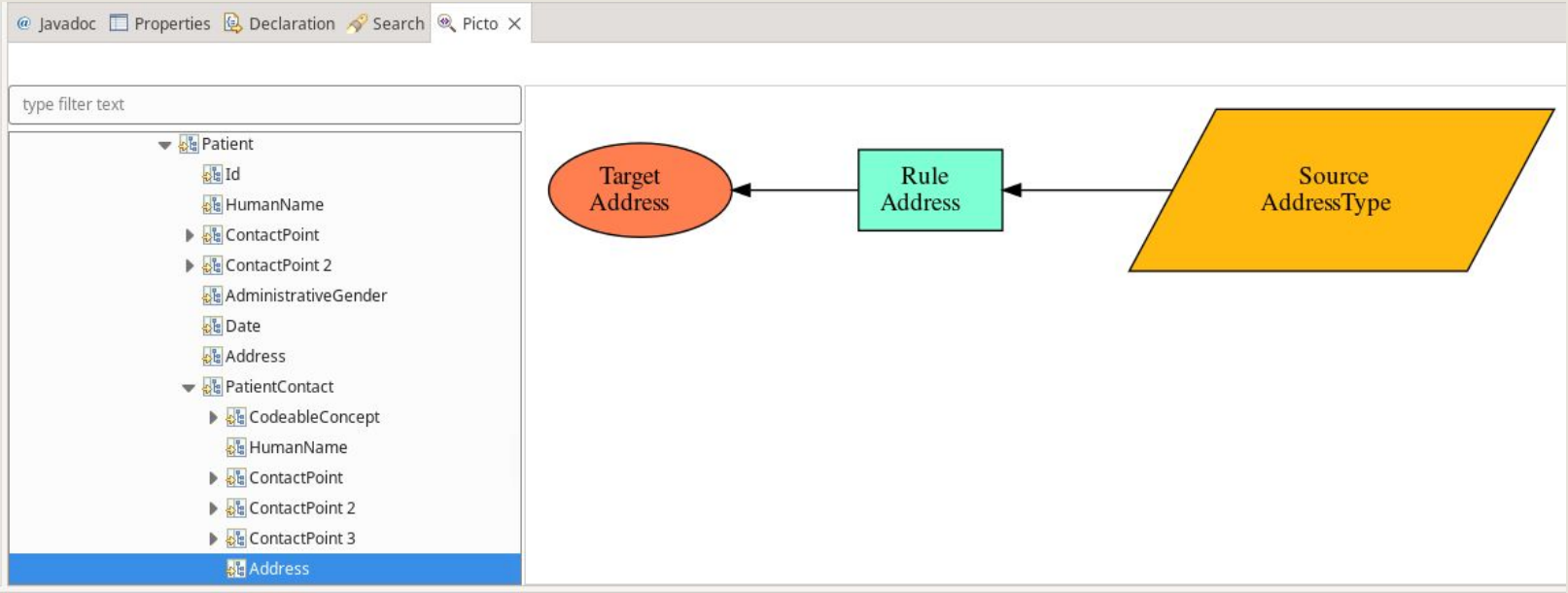
Picto visualisations: rule and source





- Visualisations are based on Graphviz in its "circo" mode
- Visualisations are interactive - you can click on a diagram element and jump to its visualisation

Picto visualisations: target



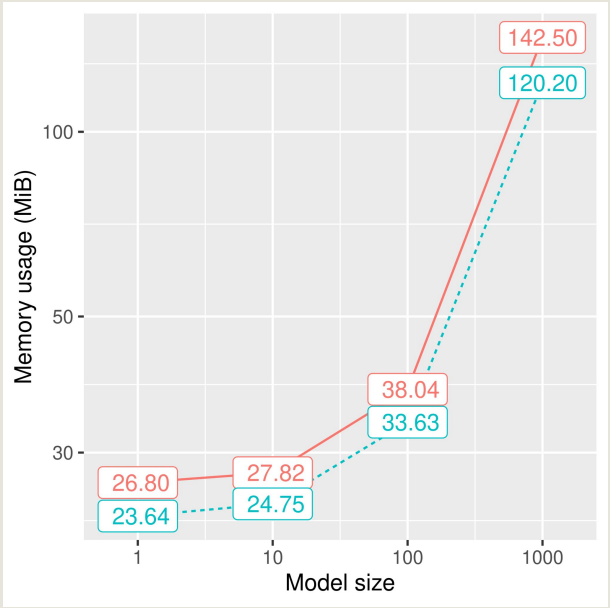
Visualisation helped find the "orphan" objects not within the target DocumentRoot, and point to the rules that needed improvement - this fixed one bug in the ETL script causing test failures.

Benchmark results and conclusion

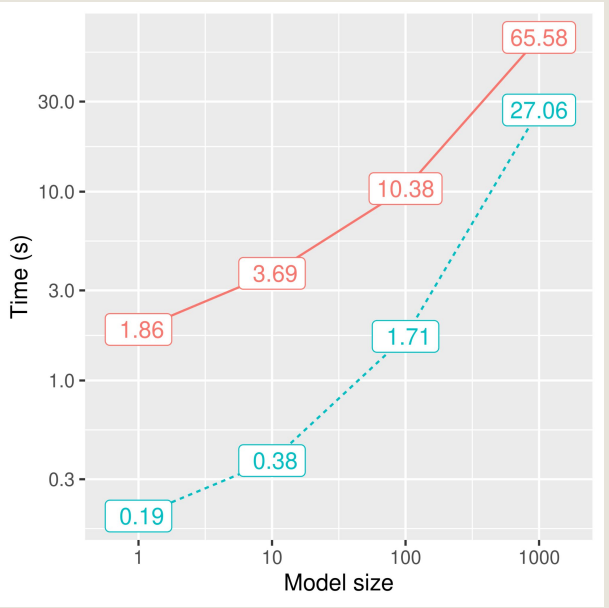
Benchmark results

ETL is solid line, ATL is dashed line

Memory usage



Performance



Conclusions

Overall comparison

- ETL took fewer lines of code (1096 lines vs 1319)
- ETL used similar memory but was slower - want to refactor tx to avoid lazy rules (should be faster!)
- Picto-based trace viz was easy and helped fix bug

Changes in Epsilon

- Optimised ETL internal trace data structure (from flat list to Guava Network)
- Easier enum literals from Epsilon 2.5.0
- Fixed thread deadlock in Picto from GTK/Linux

Thank you!

a.garcia-dominguez@york.ac.uk