The TTC 2023 KMEHR to FHIR case

Dennis Wagelaar[1]

[1] Corilus nv, Gaston Crommenlaan 4, 9050 Gent

Abstract
The history of medical information systems has seen many twists and turns, and while there has long been a global standardization body in the form of HL7, it only recently gained a lot of traction with the FHIR standard. As a result, a number of countries have developed their own medical data interchange formats over the years, which now need to be realigned with the global FHIR medical data interchange format. One such example is the Belgian KMEHR format. For this TTC case, we will focus on a specific kind of medical data interchange, namely the Patient Summarized Medical Record. In KMEHR, this is called the Summarized Electronic Health Record (SumEHR). In FHIR, this is called the International Patient Summary (IPS). The primary purpose of such a record is to provide an emergency "cheat sheet" to healthcare providers who don't normally see the patient in question, e.g. a hospital's emergency department. Especially when a patient is abroad, the capability to exchange such data is important, as it will often be the only source of medical background data. This TTC case will require you to translate between the Belgian SumEHR format and the international FHIR IPS format.

1. Introduction

This Transformation Tool Contest case concerns the transformation between two medical data interchange formats: the Belgian "Kindly Marked-up Electonic Healthcare Record" (KMEHR) standard [1], and the international "Fast Healthcare Interoperability Resources" (FHIR) standard by Health Level 7 (HL7) [2].

For this TTC case, we will focus on a specific kind of medical data interchange, namely the Patient Summarized Medical Record. In KMEHR, this is called the Summarized Electronic Health Record (SumEHR) [3]. In FHIR, this is called the International Patient Summary (IPS) [4]. The primary purpose of a Patient Summarized Medical Record is to provide an emergency reference sheet to healthcare providers who don't normally see the patient in question, e.g. a hospital's emergency department, or a different doctor than your regular doctor. Especially when a patient is abroad, the capability to exchange such data is important, as it will often be the only source of medical background data.

All resources for this case are available on Github[1]. Please follow the link in the footnote and create a pull request with your own solution.

The rest of the document is structured as follows: Section 2 describes the structure of the KMEHR to FHIR case. Section 3 describes the proposed tasks for this case. Section 4 mentions the benchmark framework for those solutions that focus on raw performance. Finally, Section 5 mentions an outline of the initial audience-based evaluation across all solutions, and the approach that will be followed to derive additional prizes depending on the attributes targeted by the solutions.

2. Case Structure

The case is intended to review the different approaches for bridging the gap between two medical data standards, KMEHR and FHIR, that use vastly different document structures and medical code systems. The metamodels for KMEHR and FHIR have been automatically generated from their published XML schemas[2] using the EMF XSD generator[3]. The resulting metamodels are too large to include in this paper, but they can be viewed online at the "kmehr-emf"[4] and "fhir-xml-emf"[5] Github projects. The metamodels consist of 297 and 1110 metaclasses, respectively. Please note that the original XSD files are also included in these GitHub repositories.

These metamodels aren't only large because they cover many concepts, but also due to accidental complexity that is caused mostly by the process of a large standardization body, and to a lesser extent by EMF's representation in Ecore of XML schemas. It is necessary for an industrial transformation tool to be able to process the technical space [5] that is used, and not require the end user to manually translate from the provided technical format to the modeling technology of choice. Having to bridge technical spaces is common for industrial data format standards, which often don't target modeling technology. It is also apparent that the level of accidental complexity is proportial to the size of the standardization body, as

TTC'23: 15th Transformation Tool Contest, Part of the Software Technologies: Applications and Foundations (STAF) federated conferences, Eds. A. Boronat, A. García-Domínguez, and G. Hinkel, 20 July 2023, Leicester, UK.
dennis.wagelaar@corilus.be (D. Wagelaar)

[1] https://github.com/dwagelaar/ttc2023-kmehr2fhir

[2] FHIR canonically uses JSON representation, but the standard also provides an XML representation.

[3] https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.emf.doc%2Ftutorials%2Fxlibmod%2Fxlibmod.html

[4] https://github.com/dwagelaar/kmehr-emf

[5] https://github.com/dwagelaar/fhir-xml-emf

can be seen in the size difference between the KMEHR and FHIR metamodels. FHIR simply has a much larger scope, and must be general enough to apply worldwide. It therefore also covers many more corner cases than KMEHR.

Figure 1 shows a small part of the FHIR metamodel that shows how primitive types, such as strings, integers, and booleans, are wrapped inside a container object. The KMEHR metamodel does not do this, and an additional translation step is necessary to bridge this – purely technical – gap. In this case, the Composition metaclass, which represents the body of the medical record summary, typically has a custodian. The custodian is the medical party that is responsible for keeping the medical record summary up to date, and is typically your general practitioner in Belgium. The custodian property is represented as a reference string in FHIR, which resolves to the type and identifier of a Resource (document attachment) inside the same FHIR Bundle (document).

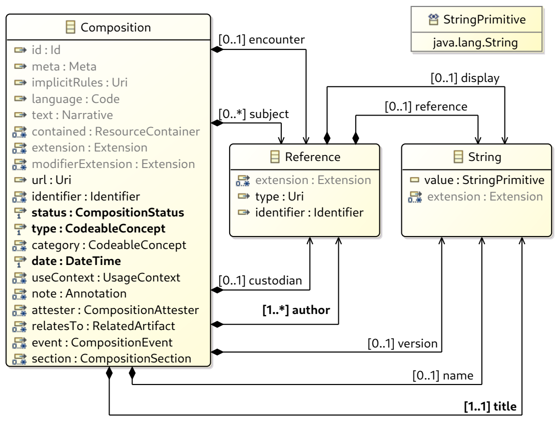

Figure 1: Extract of the FHIR metamodel

A transformation tool's ability to deal with this accidental complexity is an important factor for industrial acceptance. Not only should the transformation tool be able to process the accidental complexity, but it should also allow the user to hide away or otherwise modularize the part of the transformation code/specification that deals with this accidental complexity.

The reference transformation is written in ATL/EMFTVM [6] and comprises approximately 1300 lines of code, divided over the main `KMEHRtoFHIR.atl` transformation module and the `libKMEHRtoFHIR.atl` helper library. Both of these files can be found in the aforementioned case Github repository. It uses advanced features of the EMFTVM runtime, such as multiple rule inheritance and invocation of native Java code. It also relies on the local search compiler included with the 4.8.0 release of ATL, which allows for efficient execution of matched rules with many input element. For example, the Posology rule shown in Listing 1 uses four input elements, which would require iterating four times through the entire input model before ATL 4.8.0. As of ATL 4.8.0, the filter expressions on lines 7–9 are translated by the compiler into local search expressions for the `tx`, `i`, and `s` input elements. Only the `f` input element needs to be found by iterating over the entire input model. An additional **mapsTo** keyword is required to indicate that the output element `t` does not map to the entire collection of input elements, as is the default, but rather directly maps to the input element `s`. That way, whenever another transformed model element is assigned a reference to KMEHR!PosologyType `s`, ATL's implicit tracing mechanism will translate it to FHIR!MedicationStatement `t`.

The Posology rule is responsible for translating the statement of how and when medication must be administered – e.g. "once daily, after dinner" – from a KMEHR!PosologyType to a FHIR!MedicationStatement, as well as during which time period it is applicable (the `effectivePeriod`). Another transformation rule PosologyWithUnitAndTakes exists, which extends the Posology rule by adding a unit and dosage, e.g. "1 caps.". This rule is shown in Listing 2.

Note how ATL modularizes the accidental complexity of wrapped primitive types in FHIR by delegating the creation of the wrapper objects to lazy rules, such as the FhirString rule shown in Listing 3. The Posology rule can now simply wrap the medication reference string inside a FHIR String by invoking the lazy rule. Note that the remaining medication reference wrapper objects, CodeableReference and Reference, could also have been modularized in a lazy rule, further increasing the clarity of the Posology rule. This has not been done here, because those wrapper objects are only used once in the entire transformation code. Instead, only the wrapper objects that must be created several times are extracted into lazy rules.

The transformation translates a SumEHR document, which is effectively a KMEHR document with a header and a folder that contains administrative patient data and a "sumehr" type transaction. The sumehr transaction contains a number of items of the following types:

- **gmdmanager**: the doctor that is responsible for keeping the patient's main medical record up-to-date. It is translated to a custodian Practitioner element in FHIR.
- **contactperson**: a key contact person of the patient, such as a family member or employer. It is translated to a list of contact CodeableConcepts in FHIR, embedded within the Patient element.
- **socialrisk**: a health-related risk originating from

Listing 1: Posology rule

```
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 (
   id <- msid,
   medication <- medCodRef,
   status <- msstatus,
   subject <- subRef,
   effectivePeriod <- effectivePeriod,
   dosage <- Sequence{dosage}),
  msid : FHIR!Id (
   value <- s.uuid),
  medCodRef : FHIR!CodeableReference (
   reference <- medRef),
  medRef : FHIR!Reference (
   reference <- thisModule.FhirString('
        Medication/' + i.uuid)),
  msstatus : FHIR!MedicationStatementStatusCodes
        (
   value <- #recorded),
  subRef : FHIR!Reference (
   reference <- thisModule.FhirString('Patient/'
         + f.patient.uuid)),
  effectivePeriod : FHIR!Period (
   start <- thisModule.FhirDateTime(i.
        beginmoment),
   end <- thisModule.FhirDateTime(i.endmoment)),
  dosage : FHIR!Dosage (
   timing <- timing),
  timing : FHIR!Timing (
   repeat <- repeat),
  repeat : FHIR!TimingRepeat (
   count <- thisModule.FhirPositiveInt(i.
        dayperiod->size()),
   periodUnit <- periodUnit,
   when <- i.dayperiod->collect(dp | thisModule.
        EventTiming(dp))),
  periodUnit : FHIR!UnitsOfTime (
   value <- #d)
}
```

Listing 2: Posology rule

```
rule PosologyWithUnitAndTakes extends Posology {
 from
  f : KMEHR!FolderType,
  tx : KMEHR!TransactionType,
  i : KMEHR!ItemType,
  s : KMEHR!PosologyType (
   not s.unit.oclIsUndefined() and
   not s.takes.oclIsUndefined()
  )
 to
  t : FHIR!MedicationStatement mapsTo s,
  doseAndRate : FHIR!DosageDoseAndRate (
   type <- thisModule.CodeableConcept(thisModule.
        CodingWithDisplay(
    'http://terminology.hl7.org/CodeSystem/dose-
          rate-type',
    'ordered',
    'Ordered'
   )),
   doseQuantity <- doseQuantity
  ),
  dosage : FHIR!Dosage (
   doseAndRate <- Sequence{doseAndRate}
  ),
  doseQuantity : FHIR!Quantity (
   system <- qSys,
   code <- qCode,
   unit <- thisModule.FhirString(s.unit.cd.
        toUnitsOfMeasureValue),
   value <- thisModule.FhirDecimal(s.takes.high)
  ),
  qSys : FHIR!Uri (
   value <- 'http://unitsofmeasure.org'
  ),
  qCode : FHIR!Code (
   value <- '1'
  )
}
```

the patient's social context. It is not included in a FHIR IPS document.

- **risk**: a health-related general risk for the patient. It is not included in a FHIR IPS document.
- **problem**: an ongoing condition the patient suffers from, or – if inactive – a historic condition. It is translated to a Condition element in FHIR.
- **medication**: currently prescribed medication for the patient. It is translated to a Medication element in FHIR, with the embedded KMEHR Posology being translated to a MedicationStatement element in FHIR.
- **vaccine**: a vaccine/immunization the patient has received in the past. It is translated to an Immunization element in FHIR.
- **adr**: an adverse drug reaction that the patient suffers from. It is translated to an AllergyIntolerance element of type "intolerance" in FHIR.
- **allergy**: an allergy the patient suffers from. It is translated to an AllergyIntolerance element of type "allergy" in FHIR.

Listing 3: Posology rule

```
lazy rule FhirString {
 from
   s : String
 to
   t : FHIR!"fhir::String" (
     value <- s
   )
}
```

More types of information could be included in both SumEHR and FHIR IPS, but for the purpose of the TTC case, the types are limited to the ones listed.

In addition, the reference model transformation does not (fully) translate between the medical coding systems used in KMEHR and FHIR for the sake of simplicity, and simply embeds the KMEHR medical codes within the FHIR document. Submissions to this case should follow the same strategy.

That said, a full translation would have to map additional medical coding systems in order to be complete, notably units of measure[6] and vaccine indication codes[7]. Finally, a translation of the Belgian CNK codes for medication to the international ATC standard would be very useful, but that would require incorporating a database in the transformation, such as SAM[8].

3. Task Description

There is a mandatory task and an optional task in this case:

- The mandatory task is to re-implement or improve the original transformation itself, in a way that lends itself better to after-the-fact consistency checking. Your transformation tool may have better support for this, or ATL could be made to deal better with larger versions of this model.
- The optional task is to define the reverse transformation that translates the generated FHIR IPS document back to SumEHR.

Solutions can focus on efficiency, conciseness, or clarity of presentation to the user. Clarity of presentation is key for this kind of transformation, as domain experts must typically validate the correctness of the transformation logic by reviewing the code.

[6]http://unitsofmeasure.org
[7]https://www.ehealth.fgov.be/standards/fhir/vaccination/ValueSet-be-vs-vaccine-code.html
[8]https://www.samportal.be/

4. Benchmark Framework

If focusing on performance, the solution authors should integrate their solution with the provided benchmark framework. It is based on that of the TTC 2017 Smart Grid case [7], and supports the automated build and execution of solutions. For this specific case study, the visualisation of the results is currently disabled.

The benchmark consists of three phases:

1. **Initialization**, which involves setting up the basic infrastructure (e.g. loading metamodels). These measurements are optional.
2. **Load**, which loads the input models.
3. **Run**, which runs the consistency checking, finding a number of consistency violations in the mutated DocBook model.

4.1. Solution requirements

Solutions should be forks of the main Github project[9], and should be submitted as pull requests.

Each solution wishing to use the benchmarking framework should print to the standard output a line with the following fields, separated by semicolons (";"):

- **Tool**: name of the tool.
- **Source**: base name of the input KMEHR model (e.g. "sumehr_example10.kmehr").
- **Target**: base name of the output FHIR model (e.g. "output.fhir").
- **RunIndex**: index of the run of this combination of tools and inputs.
- **PhaseName**: name of the phase being run. It may be **Initialization**, **Load**, or **Run**.
- **MetricName**: the name of the metric. It may be the **Memory used (b)** in bytes, the wall clock **Runtime (ns)** spent in integer nanoseconds, or the number of Bundle **Entries** found in the output FHIR model.

To enable automatic execution by the benchmark framework, solutions should add a subdirectory to the `solutions` folder of the benchmark with a `solution.ini` file stating how the solution should be built and how it should be run. As an example, the `solution.ini` file for the reference solution is shown on Listing 4. In the `build` section, the `default` option specifies the command to build and test the solution, and the `skipTests` option specifies the command to build the solution while skipping unit tests. In the `run` section, the `cmd` option specifies the command to run the solution.

The repetition of executions as defined in the benchmark configuration is done by the benchmark. For 3

[9]https://github.com/dwagelaar/ttc2023-kmehr2fhir

Listing 4: `solution.ini` file for the reference ATL solution

```
[build]
default=mvn package
skipTests=mvn package --skipTests=true

[run]
cmd=JAVA_OPTS="-Xms4g" \
 java -cp target/reference-0.0.1-SNAPSHOT.jar:
      target/dependency/* \
 ttc2023.kmehr2fhir.reference.Driver $SourcePath
      $TargetPath
```

runs, the specified command will be called 3 times, passing any required information (e.g. run index, or input model name) through environment variables. Solutions must not save intermediate data between different runs: each run should be entirely independent.

The name and absolute path of the input model, the run index and the name of the tool are passed using environment variables `Tool`, `SourcePath`, `TargetPath`, and `RunIndex`. Solution authors are suggested to study the reference solution on how to use these values to run their transformation.

4.2. Running the benchmark

The benchmark framework only requires Python 3.3 to be installed. Furthermore, the solutions may require additional frameworks. We would ask solution authors to explicitly note dependencies to additional frameworks necessary to run their solutions.

If all prerequisites are fulfilled, the benchmark can be run using Python with the command `python scripts/run.py`. Additional options can be queried using the option `--help`. The benchmark framework can be configured through the `config/config.json` file: this includes the input models to be evaluated (some of which have been excluded by default due to their high cost with the sample solution), the names of the tools to be run, the number of runs per tool+model, and the timeout for each command in milliseconds.

5. Evaluation

The evaluation will operate on several dimensions:

- How efficient is the approach in time and space (memory)?
- How understandable is the transformation code for domain experts to review and validate?

Acknowledgement

This paper used the TTC 2019 Live Case paper by Antonio García-Domínguez and Georg Hinkel [8] as a template.

References

[1] KMEHR standard (v1.38), eHealth-platform, 2023. Online: https://www.ehealth.fgov.be/standards/kmehr/en.

[2] FHIR Specification (v5.0.0: R5), HL7 Community, 2023. Online: https://hl7.org/fhir/R5/.

[3] Summarised Electronic Healthcare Record v2.0, eHealth-platform, 2016. Online: https://www.ehealth.fgov.be/standards/kmehr/en/transactions/summarised-electronic-healthcare-record-v20.

[4] International Patient Summary Implementation Guide (v1.1.0), HL7 Community, 2022. Online: https://hl7.org/fhir/uv/ips/STU1.1/.

[5] J. Bézivin, Model driven engineering: An emerging technical space, in: R. Lämmel, J. Saraiva, J. Visser (Eds.), Proceedings of GTTSE 2005, Revised Papers, volume 4143 of *LNCS*, Springer-Verlag, 2005, pp. 36–64. doi:10.1007/11877028_2.

[6] D. Wagelaar, M. Tisi, J. Cabot, F. Jouault, Towards a general composition semantics for rule-based model transformation, in: J. Whittle, T. Clark, T. Kühne (Eds.), Proceedings of MoDELS 2011, volume 6981 of *LNCS*, Springer-Verlag, 2011, pp. 623–637. doi:10.1007/978-3-642-24485-8_46.

[7] G. Hinkel, The TTC 2017 Outage System Case for Incremental Model Views, in: Proceedings of the 10th Transformation Tool Contest, volume 2026, CEUR-WS.org, Marburg, Germany, 2017, pp. 3–12. URL: https://ceur-ws.org/Vol-2026/paper1.pdf.

[8] A. García-Domínguez, G. Hinkel, The TTC 2019 Live Case: BibTeX to DocBook, in: Proceedings of the 12th Transformation Tool Contest, volume 2550, CEUR-WS.org, Eindhoven, The Netherlands, 2019, pp. 61–65. URL: https://ceur-ws.org/Vol-2550/paper8.pdf.