

The TTC 2023 KMEHR to FHIR Case

Dennis Wagelaar
dwagelaar@gmail.com

April 22, 2023

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 Electronic Healthcare Record” (KMEHR) standard [2], and the international “Fast Healthcare Interoperability Resources” (FHIR) standard by Health Level 7 (HL7) [5].

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) [1]. 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 Github¹. 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 schemas² using the EMF XSD generator³. 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 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 upcoming 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 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.

The transformation translates a SumEHR document, which is effectively a KMEHR document with a header and a folder that contains a 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.

¹<https://github.com/dwagelaar/ttc2023-kmehr2fhir>

²FHIR 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%2Fxmlmod%2Fxmlmod.html>

⁴<https://github.com/dwagelaar/kmehr-emf>

⁵<https://github.com/dwagelaar/fhir-xml-emf>

Listing 1: Posology rule

```

1 rule Posology {
2     from
3         f : KMEHR!FolderType,
4         tx : KMEHR!TransactionType,
5         i : KMEHR!ItemType,
6         s : KMEHR!PosologyType (
7             i.posology = s and
8             tx.item->includes(i) and
9             f.transaction->includes(tx) and
10            i.isMedication)
11    to
12        t : FHIR!MedicationStatement mapsTo s (
13            id <- msid,
14            medication <- medCodRef,
15            status <- msstatus,
16            subject <- subRef,
17            effectivePeriod <- effectivePeriod,
18            dosage <- Sequence{dosage}),
19        msid : FHIR!Id (
20            value <- s.uuid),
21        medCodRef : FHIR!CodeableReference (
22            reference <- medRef),
23        medRef : FHIR!Reference (
24            reference <- thisModule.FhirString('Medication/' + i.uuid)
25            ),
26        msstatus : FHIR!MedicationStatementStatusCodes (
27            value <- #recorded),
28        subRef : FHIR!Reference (
29            reference <- thisModule.FhirString('Patient/' + f.patient.
30            uuid)),
31        effectivePeriod : FHIR!Period (
32            start <- thisModule.FhirDateTime(i.beginmoment),
33            end <- thisModule.FhirDateTime(i.endmoment)),
34        dosage : FHIR!Dosage (
35            timing <- timing),
36        timing : FHIR!Timing (
37            repeat <- repeat),
38        repeat : FHIR!TimingRepeat (
39            count <- thisModule.FhirPositiveInt(i.dayperiod->size()),
40            periodUnit <- periodUnit,
41            when <- i.dayperiod->collect(dp | thisModule.EventTiming(
42                dp))),
43        periodUnit : FHIR!UnitsOfTime (
44            value <- #d)
45    }

```

- **contactperson**: a key contact person of the patient, such as a family member or employer.
- **socialrisk**: a health-related risk originating from the patient’s social context.
- **risk**: a health-related general risk for the patient.
- **problem**: an ongoing condition the patient suffers from, or – if inactive – a historic condition.
- **medication**: currently prescribed medication for the patient.
- **vaccine**: a vaccine/immunization the patient has received in the past.
- **adr**: an adverse drug reaction that the patient suffers from.
- **allergy**: an allergy the patient suffers from.

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.

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.

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 [3], 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:

Listing 2: `solution.ini` file for the reference ATL solution

```
1 [build]
2 default=mvn package
3 skipTests=mvn package --skipTests=true
4
5 [run]
6 cmd=JAVA_OPTS="-Xms4g" \
7   java -cp target/reference-0.0.1-SNAPSHOT.jar:target/dependency/* \
8   ttc2023.kmehr2fhir.reference.Driver $SourcePath $TargetPath
```

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 project⁶, 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 2. In the `build` section, the `default` option specifies the command to build and test the solution, and the

⁶<https://github.com/dwagelaar/ttc2023-kmehr2fhir>

`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 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 imply 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?

References

- [1] eHealth-platform. *Summarised Electronic Healthcare Record v2.0*, December 2016. Online: <https://www.ehealth.fgov.be/standards/kmehr/en/transactions/summarised-electronic-healthcare-record-v20>.
- [2] eHealth-platform. *KMEHR standard (v1.38)*, April 2023. Online: <https://www.ehealth.fgov.be/standards/kmehr/en>.
- [3] Georg Hinkel. The TTC 2017 Outage System Case for Incremental Model Views. In *Proceedings of the 10th Transformation Tool Contest*, volume 2026, pages 3–12, Marburg, Germany, July 2017. CEUR-WS.org.

- [4] HL7 Community. *International Patient Summary Implementation Guide (v1.1.0)*, November 2022. Online: <http://hl7.org/fhir/uv/ips/STU1.1/>.
- [5] HL7 Community. *FHIR Specification (v5.0.0: R5)*, March 2023. Online: <https://hl7.org/fhir/R5/>.
- [6] Dennis Wagelaar, Massimo Tisi, Jordi Cabot, and Frédéric Jouault. Towards a general composition semantics for rule-based model transformation. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Proceedings of MoDELS 2011*, volume 6981 of *LNCS*, pages 623–637. Springer-Verlag, October 2011.