The TTC 2023 KMEHR to FHIR case

Dennis Wagelaar

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 Electronic Health Record (SumEHR) in KMEHR, this is called the Summarized Medical Record. 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 [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) [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 schemas using the EMF XSD generator1. The resulting metamodels are too large to include in this paper, but they can be viewed online at the "kmehr-emf"3 and "fhir-xml-emf"4 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 proportional to the size of the standardization body, as

Footnotes:

1https://github.com/dwagelaar/ttc2023-kmehr2fhir
2FHIR canonically uses JSON representation, but the standard also provides an XML representation.
3https://github.com/dwagelaar/kmehr-emf
4https://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](image1.png)

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 110KMEHRtoFHIR.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 therefore also covers many more corner cases than KMEHR.

![Figure 1: Extract of the FHIR metamodel](image1.png)

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 SumEHRR 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
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.
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.

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 project9, and should be submitted as pull requests.

Each solution wishing to use the benchmark 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

---

9http://unitsofmeasure.org
8https://www.samportal.be/
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 Garcia-Dominguez and Georg Hinkel [8] as a template.

References