Migration algorithm Koppeltaal 2.0

Version:	1.0
Datum:	May 5th, 2023

Note 1: This document has been created in cooperation with IT participants. More information regarding KT2 and the preparation of the migration can be found on the website **Documentatie | Koppeltaal**.

Note 2: This document provides guidelines and patterns. The exact algorithm will vary from one situation to another. Therefore, each party should check how to implement the guidelines and patterns for their situation.

Koppeltaal

maakt digitale
zorg toegankelijk

This document describes how, for migration to Koppeltaal 2.0, we can populate the FHIR Resource provider in Koppeltaal 2.0 while keeping the referential integrity of Koppeltaal 1.3 "resources". Filling is done using a number of technical steps which can be programmed. We call this sequence of technical steps the migration algorithm. We will describe the organizational steps needed for migration in another document: the migration action plan.

This document focuses on keeping the referential integrity between Koppeltaal resources. In the "Mapping Koppeltaal 1.3 to Koppeltaal 2.0" we will describe the mapping of Koppeltaal 1.3 attributes to Koppeltaal 2.0 attributes.

Contents

Description of the challenge

Data is exchanged in Koppeltaal between application instances. In existing Koppeltaal 1.3 implementations, the exchange of data between application instances is based on FHIR STU 1.3. This is a message-based protocol, where application instances send messages to each other via an enterprise servicebus. The Koppeltaal 1.3 facility 'forgets' these messages. In Koppeltaal 2.0, application instances exchange data via the FHIR R4 RESTFull api of a Koppeltaal FHIR Resource Provider, as part of the Koppeltaal 2.0 facility.

However, for historical and current "records" which are spread across various application instances, just writing resources in the Koppeltaal FHIR Resourceprovider do not contain the interrelationships between these resources. This break in cross application-instance data integrity is unacceptable.

For example: An EHR and an eHealthPlatform are both active in a Koppeltaal domain. Both contain a Patient with an KT 1.3 logical identification PPP. If, as part of the migration, the EPD creates a resource Patient in the Koppeltaal FHIR Resource provider, this Patient is given a new logical KT 2.0 identifier, for example 111 (see KOP-KT-003). When a eHealthPlatform subsequently retrieves the Patient this application instance does not recognize the Patient and creates a new Patient within the eHealthPlatform. But recognition is necessary! Because the Patient has already been transferred in the past in Koppeltaal 1.3 and is known in the eHealthPlatform. But it is known under number KT 1.3 logical identification PPP and not under the new logical KT 2.0 identifier number 111. So without countermeasures, the relationship between Patient PPP in the EHR and Patient PPP in the eHealthPlatform falls away and the migrated data are not usable.

Outline of the solution

To know which KT2 resource represents the corresponding KT1 resource, **we introduce a specific business identifier**. This business identifier will have a to-be-defined "system" and its value is that of the "logical id" from the KT 1.3 context. **We will call the system of these introduced identifiers "migration". And we will call the corresponding business identifiers: "migration" business identifiers.** In this way, we can keep the interrelationships between resources across application-instances in an existing Koppeltaal domain. We use the "migration" business identifiers to hold and transfer the KT 1.3 logical identifiers through the Koppeltaal FHIR resource provider, so that the "receiving" application instance can recognize the resource by this identifier. The application instance can respond to it and it can rebuild the relationships.

For example (continued): The EHR creates the Patient with "migration" business identifier PPP in the Koppeltaal FHIR Resource provider. The eHealthPlatform retrieves the Patients. The eHealthPlatform also reads the "migration" business identifier with value PPP and can match the existing instance of the Patient in the eHealthPlatform with this value PPP. The eHealthPlatform stores the KT 2.0 logical identifier 111 with the Patient in the eHealthPlatform. So no new Patient is created, the existing Patient is just updated with the KT 2.0 logical identifier. The end result is that the KT 2.0 logical identifiers of the Patient in the EHR, Koppeltaal FHIR resource server and the eHealthplatform all contain the value 111. And that across the domain, the "migration" business identifier contains the value PPP.

The above example is a simple one, but the mechanism is universally applicable. Based on this mechanism, we describe how we technically perform the migration in the migration algorithm.

Considerations in design of the migration algorithm

The migration algorithm considers the following:

- The migration algorithm is aimed at maintaining the referential integrity between resources and across applications and the FHIR service in an Koppeltaal domain. The mapping of Koppeltaal 1.3 attributes to Koppeltaal 2.0 attributes will be described in another document. Together these documents give guidance for realization of the migration scripts.

- We want the migration algorithm to be as simple as possible with as little additional programming as possible. So we will use "Regular" Koppeltaal 2.0” software as much as possible. Koppeltaal 2.0 uses data exchange based on FHIR R4 Restfull api's based on the Koppeltaal 2.0 FHIR profile (see TOP-KT-002, TOP-KT-009), including associated validation (see TOP-KT-010) and logging (see TOP-KT-010). We will use these api's in the migration algorithm to populate the Koppeltaal FHIR resource provider.

- Because Koppeltaal 1.3 is implemented differently in each domain and the application instances involved differ in each domain, the migration is per domain.

- In case Koppeltaal 2.0 does not function after some time in production, we do not create a rollback mechanism. We will mitigate the associated risks through testing and acceptance of Koppeltaal 2.0 and testing and acceptance of the migration algorithm before release to and use in production.

- In the migration algorithm, we take into account the principles of privacy by design.

- Migration will be batch based. This means that we will not use the subscribe and notify mechanism for the migration. One of the underlying reasons is that the current notification mechanism does not contain enough information and needs to be temporarily extended for the migration. For example, by including the logical identifiers or the entire resource in the payload of the notification.

Koppeltaal

Best practice for realization of the migration algorithm

For realization of the migration algorithm, the following best practice can be used:

- Mediator script: For migration, a "Mediator" script can be created that is positioned between the api of the application instance and the api of the FHIR service. This mediator script performs migration steps as a mediator between the application instance and the FHIR service. This way, the migration codebase remains decoupled from the application's codebase and can also be discarded after performing the migrations. This prevents confusion at a later time when reading through the application codebase. This is consistent with the general principle of separation of concerns.

The migration algorithm

Preconditions
In a Koppeltaal domain, the following technical requirements are met prior to starting the migration:

1. The buffers in the Koppeltaal 1.3 data stream MUST be empty. Thus, no Koppeltaal 1.3 messages are in transit or pending. And all messages in Koppeltaal 1.3 are fully processed. This requirement is necessary because we are migrating data from connected application instances in a Koppeltaal domain. Thus, there is no migration from the Koppeltaal 1.3 facility to the Koppeltaal 2.0 facility and thus no possibility to migrate "pending" messages.

2. Unwanted notifications from the Koppeltaal 2.0 FHIR Service and/or unwanted actions in application-instances as a response to these notifications MUST be switched off. This is evident.

3. Unwanted actions in application-instances, which can result from data migration MUST be switched off. E.g. sending unwanted emails. This is evident.

Koppeltaal

Algorithm for transferring a Patient

The sequence diagram below describes how a Patient is migrated. This example with a Patient also illustrates how to migrate other resources, without references to other resources.

Remarks:
• Migration_BI is a shorthand for a business identifier of the system "migration".
• KT1.3_ID is a shorthand for the attribute containing the "old" logical Koppeltaal 1.3 identifier
• KT2.0_ID is a shorthand for the attribute containing the "new" logical Koppeltaal 2.0 identifier

Situation referential integrity for existing Patients:
• FHIR.Patient not existent
• EPD.Patient.KT1.3_ID == eHealthplatform. KT1.3_ID == PPP
• EPD.Patient.KT2.0_ID == (empty)
• eHealthplatform.KT2.0_ID == (empty)

Sequence diagram algorithm for existing Patients:

1. The EHR Posts selects all Patients which are eligible for migration and post these Patients into the Koppeltaal FHIR Resource Provider. This post includes the Migration business identifier Patient.Migration_BI which contains the value from Ehr.Patient.KT1.3_ID (PPP).

2. The FHIR Resource provider creates the Patient and assigns an KT2.0 logical ID. E.g. with value 111.

3. The FHIR Resource provider responds and returns the created Patient with the KT 2.0 logical ID to the EHR. E.g. with value 111.

4. The EHR receives a Patient from the FHIR Resource Provider with the KT 2.0 logical ID (111) and MAY or MUST update the EHR.Patient with this KT2.0 logical ID (111). See the additional requirements, when MAY is allowed.

5. The eHealthPlatform gets all the Patients in the domain, including the Migration Business Identifiers of the Patients. E.g. with value FHIR.Patient.Migration_BI == PPP

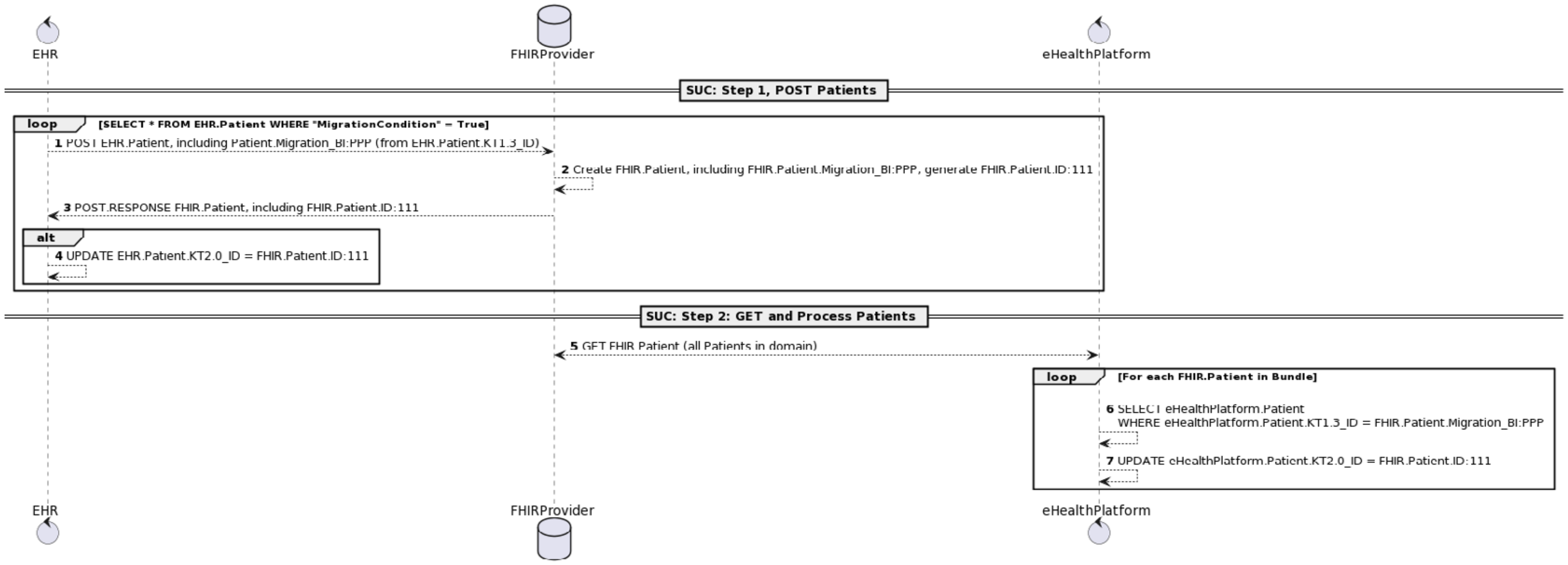
6. For each Patient in the responding bundle, the eHealthPlatform selects the eHealthPlatform. Patient with the help of the FHIR.Patient.Migration_BI. The eHealthPlatform.Patient.KT1.3_ID is used for matching.

7. The eHealthPlatform updates the KT2.0 logical ID of the Patients in the eHealthPlatform with the KT2.0 logical ID as received from the FHIR Resource service. E.g. with the value 111.

End situation: All the Patients in the EHR, FHIR Resource Service and eHealthPlatform are now connected via a double link:
• EPD.Patient.KT1.3_ID == FHIR.Patient.Migration_BI == eHealthplatform.KT1.3_ID == PPP
• EPD.Patient.KT2.0_ID == FHIR.Patient.ID == eHealthplatform.KT2.0_ID == 111

In the case the EPD is the original creator of the Patient, the link between EPD.Patient.KT2.0_ID and FHIR.Patient.ID is optional. See additional requirements.

Koppeltaal


EHR
FHIRProvider
eHealthPlatform
SUC: Step 1, POST Patients
loop
[SELECT * FROM EHR.Patient WHERE "MigrationCondition" = True]
1 POST EHR.Patient, including Patient.Migration_BI:PPP (from EHR.Patient.KT1.3_ID)
2 Create FHIR.Patient, including FHIR.Patient.Migration_BI:PPP, generate FHIR.Patient.ID:111
3 POST.RESPONSE FHIR.Patient, including FHIR.Patient.ID:111
alt
4 UPDATE EHR.Patient.KT2.0_ID = FHIR.Patient.ID:111
SUC: Step 2: GET and Process Patients
5 GET FHIR.Patient (all Patients in domain)
loop
[For each FHIR.Patient in Bundle]
6 SELECT eHealthPlatform.Patient
WHERE eHealthPlatform.Patient.KT1.3_ID = FHIR.Patient.Migration_BI:PPP
7 UPDATE eHealthPlatform.Patient.KT2.0_ID = FHIR.Patient.ID:111
EHR
FHIRProvider
eHealthPlatform

maakt digitale
zorg toegankelijk

Koppeltaal

Algorithm for transferring a Task

The sequence diagram below describes how a Task of a Patient is migrated. This example with a Task also illustrates how to migrate other resources, with references to other resources.

Remarks:
- Migration_BI is a shorthand for a business identifier of the system "migration".
- KT1.3_ID is a shorthand for the attribute containing the "old" Koppeltaal 1.3 identifier
- KT2.0_ID is a shorthand for the attribute containing the "new" logical Koppeltaal 2.0 identifier
- KT2.0_PatientID is a shorthand for the attribute containing the "new" logical Koppeltaal 2.0 PatientID identifier in "Task".

Start situation: Patients have already been migrated, so:
- eHealthPlatform.Patient exists
- eHealthPlatform.Patient.KT1.3_ID == "PPP"
- eHealthPlatform.Patient.KT2.0_ID == 111

Before migration, the existing Tasks in eHealth platform and in the module only contain a reference to a Patient based on the KT1.3_ID. So the start situation for existing Tasks is:
- FHIR.Task not existent
- eHealthPlatform.Task.KT1.3_ID == Module.Task.KT1.3_ID == "TTT"
- eHealthPlatform.Task.KT1.3_PatientID == Module.Task.KT1.3_PatientID == "PPP"
- eHealthPlatform.Task.KT2.0_ID == (empty)
- eHealthPlatform.Task.KT2.0_PatientID = (empty)
- Module.Task.ID == (empty)
- Module.Task.PatientID = (empty)

Sequence diagram algorithm for existing Tasks:

1. The eHealthPlatform updates all Tasks in the eHealthPlatform which are eligible for migration with the correct KT2.0 logical identifiers of the Patients in the eHealthPlatform. So eHealth-Platform.Task.PatientID will now contain the correct value (111). This is why the patients must be migrated before the tasks.

2. The eHealthPlatform selects all Tasks which are eligible for migration and post these Tasks into the Koppeltaal FHIR Resource Provider. This post includes the Migration business identifier Task.Migration_BI which contains the value from eHealthPlatform.Task.KT1.3_ID (TTT). It also contains the correct Task.PatientID's with the correct value (111).

3. The FHIR Resource provider creates the Task and assigns an KT2.0 logical ID to the Task. E.g. value 222.

4. The FHIR Resource provider responds and returns the created Task with the KT2.0 logical ID of the task to the eHealthPlatform. E.g. with value 222. It also updates the Task.PatientID, which is the reference to Patient.ID in the FHIR ResourceProvider. E.g. with value 111.

5. The eHealthPlatform receives the Task with the KT2.0 logical ID and MUST update the Task with this KT2.0 logical ID in the eHealthPlatform. E.g. with value 222.

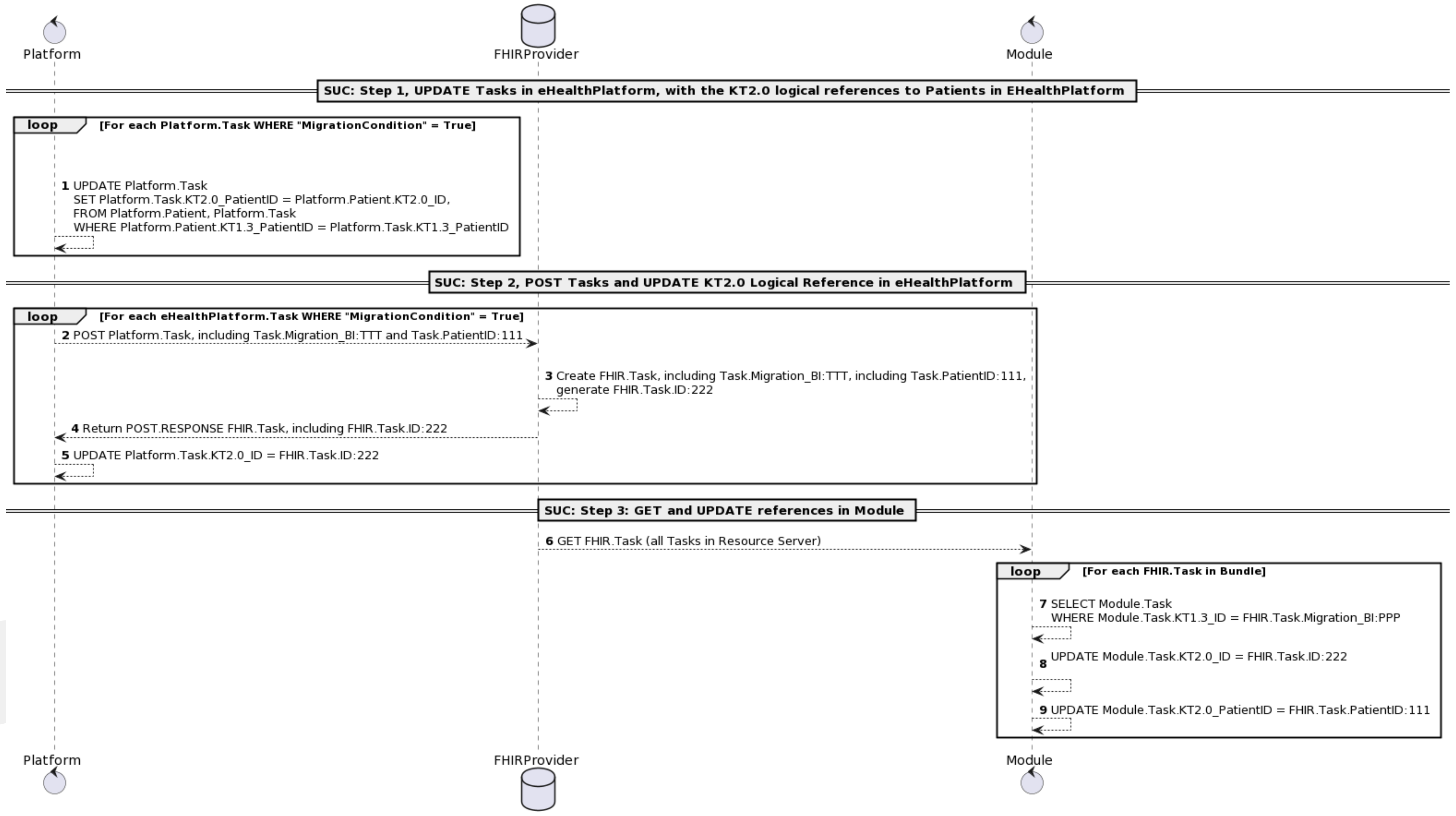
6. The Module gets all the Tasks in the domain, including the migration business identifiers of the Tasks. E.g. with value FHIR.Task.Migration_BI = TTT

8. For each Task in the responding bundle, the module selects the Module.Task with the help of the FHIR.task.Migration_BI. The module.Task.KT1.3_ID is used for matching (e.g. on TTT).

7. The Module updates the KT2.0 logical ID of the Tasks in the Module with the KT2.0 logical ID of the tasks as received from the FHIR Resource service. E.g. with the value 222. It also updates the Logical Reference Module.Task.PatientID with the value retrieved from the FHIR Resource Service. E.g. with value 111.

End situation: All the Tasks in the eHealthPlatform, FHIR Resource Service and Module are now connected via a double link:
- eHealthPlatform.Task.KT2.0_ID == FHIR.Task.ID == eHealthplatform.Task.KT2.0_ID == 222
- eHealthPlatform.Task.KT1.3_ID == FHIR.Task.Migration_BI == Module.Task.KT1.3_ID == TTT

And the references of the logical Patients ID's are correct
- eHealthPlatform.Task.KT2.0_PatientID == FHIR.Task.PatientID == Module.KT_2.0_PatientID == 111
- eHealthPlatform.Task.KT1.3_PatientID == Module.KT1.3_PatientID == PPP (which was already the case)


Platform
FHIRProvider
Module
SUC: Step 1, UPDATE Tasks in eHealthPlatform, with the KT2.0 logical references to Patients in EHealthPlatform
loop [For each Platform.Task WHERE "MigrationCondition" = True]
1 UPDATE Platform.Task
SET Platform.Task.KT2.0_PatientID = Platform.Patient.KT2.0_ID,
FROM Platform.Patient, Platform.Task
WHERE Platform.Patient.KT1.3_PatientID = Platform.Task.KT1.3_PatientID
SUC: Step 2, POST Tasks and UPDATE KT2.0 Logical Reference in eHealthPlatform
loop [For each eHealthPlatform.Task WHERE "MigrationCondition" = True]
2 POST Platform.Task, including Task.Migration_BI:TTT and Task.PatientID:111
3 Create FHIR.Task, including Task.Migration_BI:TTT, including Task.PatientID:111,
generate FHIR.Task.ID:222
4 Return POST.RESPONSE FHIR.Task, including FHIR.Task.ID:222
5 UPDATE Platform.Task.KT2.0_ID = FHIR.Task.ID:222
SUC: Step 3: GET and UPDATE references in Module
6 GET FHIR.Task (all Tasks in Resource Server)
loop [For each FHIR.Task in Bundle]
7 SELECT Module.Task
WHERE Module.Task.KT1.3_ID = FHIR.Task.Migration_BI:PPP
8 UPDATE Module.Task.KT2.0_ID = FHIR.Task.ID:222
9 UPDATE Module.Task.KT2.0_PatientID = FHIR.Task.PatientID:111
Platform
FHIRProvider
Module

Koppeltaal

Additional requirements

The following additional MUST or MAY be applicable:

- The Migration Business Identifier MUST be the old logical identifier from Koppeltaal 1.3 and becomes available, wherever needed, in a domain.

- A resource in Koppeltaal 2.0 can have multiple BusinessIdentifers. The BusinessIdentifer for migration MUST therefore be given a separate distinctive "system". We call this system "migration". See also action 1.

- If an application instance gets resource data from the FHIR Service and this application instance is not the original creator of this data, then the data, including the new Koppeltaal 2.0 logical identifiers (KT2.0_ID), MUST be stored in the database of the application instance.

- If an application instance receives resource data from the FHIR Service after posting this data to the FHIR Service and this application instance is the original creator of this data, then the included new Koppeltaal 2.0 logical identifiers (KT2.0_ID), MAY be stored in the database of the application instance.

- The old logical identifiers (KT1.3-ID) and the Migrations BusinessIdentifer (Migration_BI) MUST be retained by the application instance and FHIR resource service after migration. We will make additional arrangements at a later date on how long to store these identifiers. We suspect that these identifiers will play an important role in recovery use cases. See action 2.

Bijlage - PlantUML Patient

```
@startuml
autonumber

control     EHR             as E
database    FHIRProvider    as F
control     eHealthPlatform as P

== SUC: Step 1, POST Patients ==
Loop SELECT * FROM EHR.Patient WHERE "MigrationCondition" = True
  E --> F: POST EHR.Patient, including Patient.Migration_BI:PPP (from EHR.Patient.KT1.3_ID)
  F --> F: Create FHIR.Patient, including FHIR.Patient.Migration_BI:PPP, generate FHIR.Patient.ID:111
  E <-- F: POST.RESPONSE FHIR.Patient, including FHIR.Patient.ID:111
  alt
    E --> E: UPDATE EHR.Patient.KT2.0_ID = FHIR.Patient.ID:111
  end alt
end loop

== SUC: Step 2: GET and Process Patients ==

F <--> P: GET FHIR.Patient (all Patients in domain)
Loop For each FHIR.Patient in Bundle
  P --> P: \nSELECT eHealthPlatform.Patient\nWHERE eHealthPlatform.Patient.KT1.3_ID = FHIR.
Patient.Migration_BI:PPP
  P --> P: UPDATE eHealthPlatform.Patient.KT2.0_ID = FHIR.Patient.ID:111
End loop

@enduml
```

Bijlage – PlantUML Task

```
@startuml
autonumber
control     Platform as P
database    FHIRProvider    as F
control     Module          as M
== SUC: Step 1, UPDATE Tasks in eHealthPlatform, with the KT2.0 logical references to Patients in EHealthPlatform ==
Loop For each Platform.Task WHERE "MigrationCondition" = True
  P --> P: \n\n\nUPDATE Platform.Task\nSET Platform.Task.KT2.0_PatientID = Platform.Patient.KT2.0_ID,\nFROM Platform.Patient, Platform.Task\nWHERE Platform.Patient.KT1.3_PatientID = Platform.Task.KT1.3_PatientID
end loop
== SUC: Step 2, POST Tasks and UPDATE KT2.0 Logical Reference in eHealthPlatform ==
Loop For each eHealthPlatform.Task WHERE "MigrationCondition" = True
  P --> F: POST Platform.Task, including Task.Migration_BI:TTT and Task.PatientID:111
  F --> F: \nCreate FHIR.Task, including Task.Migration_BI:TTT, including Task.PatientID:111,\ngenerate FHIR.Task.ID:222
  P <-- F: Return POST.RESPONSE FHIR.Task, including FHIR.Task.ID:222
  P --> P: UPDATE Platform.Task.KT2.0_ID = FHIR.Task.ID:222
end loop
== SUC: Step 3: GET and UPDATE references in Module ==
F --> M: GET FHIR.Task (all Tasks in Resource Server)
Loop For each FHIR.Task in Bundle
  M --> M: \nSELECT Module.Task\nWHERE Module.Task.KT1.3_ID = FHIR.Task.Migration_BI:PPP
  M --> M: UPDATE Module.Task.KT2.0_ID = FHIR.Task.ID:222\n
  M --> M: UPDATE Module.Task.KT2.0_PatientID = FHIR.Task.PatientID:111
End loop
@enduml
```