Atomfeed for Sync 2.0

The Sync Parent will publish an atom feed that each of the Children will read. An entry in the feed represents an event in the system - a resource to be synced. Sync 2.0 will have its Parent node use an Atom feed to publish events which Children will poll for and synchronize. Children will retrieve urls pointing to updated items from the feed and then use FHIR (or for some of the metadata REST) to retrieve the data. Moreover, each Child will also publish its own feed. Children will read their own feeds and synchronize data with Parent based on the events from the feed.

More about the protocol is described here:

https://github.com/ict4h/simplefeed

Since Sync 2.0 is not the only party that can benefit from these feeds, it is best if the feed logic resides in a module separate of Sync. This module can be a dependency of Sync

Code to reuse:

ICT4H already implemented an Atomfeed module for OpenMRS, which is using their atomfeed implementation:

https://github.com/ICT4H/openmrs-atomfeed

A Bahmni fork of the module exists:

https://github.com/Bahmni/openmrs-atomfeed/

The module uses AOP to register advises that update the feed after updates are made. In Bahmni, all of the advices extend a BaseAdvice class which allows posting the urls pointing the updated resources. The base class provides a mechanism that controls for which resources events are published in the feed - it is controlled by settings (formerly global properties from platform 1.9 upwards), which take the form of i.e.: atomfeed.publish.eventsForPatientProgramStateChange

More about settings here

Bahmni also allows overriding the url template for resources using global properties. Currently it registers advice beans for the following resources:

- Patient
- Relationship (person)
- PatientProgram
- Encounter (emr-api)

The Bahmni fork of the module adds the following resources:

- Location

Atomfeed Client

Sync 2.0 Children will have to make use of an Atomfeed client to read the Parents feed. Logic for this will have to be added to the Sync 2.0 module. The ICT4H atomfeed library provides client code that can be used to consume the feed. The Sync 2.0 module will require the following:

- Set up settings which Sync will use to control the properties of the atomfeed client defined here:


- The Parent feed location - controlled by settings
- The following class will have to be implemented to process feed events:


Sync will have to use this event worker to delegate to its own processors. The feed event should be preserved when passed down the chain - processors that can be injected by implementers should have access to the event.

A factory based approach should be used, since multiple feed readers will be in use - Sync Children will reading both the Parent feed as well as their own.
Children consuming their own feed

Sync 2.0 Children will need to consume their own feeds, so that they can act on it and push changes to the Parent.

Publishing the feed will be exactly the same as the Parent feed. The Child will set up a secondary client responsible for reading its own feed and act upon it. Sync will determine which of these feed events require synchronization and will push the relevant data to the Parent.

For the sake of being reusable, the Child can use it’s own FHIR server/REST representations to keep the process of Syncing similar to reading a Parent feed - Sync will read the data from it’s own FHIR server, then forward it to the Parent.

Switching the feed implementation to an event based listener

The current Bahmni implementation is based on advices, written separately for each of the classes being recorded in the feed. This implementation is not ideal, since it requires manually adding new classes for every class that we wish to synchronize.

It would be better to use an event based approach that triggers for every BaseOpenMrsObject being persisted in the database, an approach using Hibernate interceptors underneath. Since this would trigger for all objects being persisted, we would not need to add additional advices whenever a new domain object is to be added to Sync. Whether an object should be part of the feed alongside all of the templates required for the feed should be configured through a json configuration file in the "feedConfigurations" section. The "feedFilterBeans" section is used in order to choose specific filter strategies (Catchment Strategies). Sample configuration:

```json
{
  "feedFilters" : [ "beanName1", "beanName2" ],
  "feedConfigurations" : [ {
    "openMrsClass" : "org.openmrs.Obs",
    "enabled" : false,
    "title" : "Observation",
    "category" : "observation",
    "linkTemplates" : [ {
      "rest" : "/ws/rest/v1/obs/{uuid}?v=full",
      "fhir" : "/ws/fhir/Obs/{uuid}"
    },
    "feedWriter" : "custom.PatientWriter"
  }]
}
```

Using this approach will allow to add new classes to the feed without the need to write any additional Java code that will listen in on saves or updates to the class.

It should be possible to defined multiple link templates for a a class - so that it can point both to a rest representation and a FHIR representation.

It should still be possible to inject a custom processor that will overwrite the generic logic if they wish - using the optional feedWriter property. The writer interface should take the object to write to the feed, the config object for it (from the json above) and any necessary atomfeed plumbing classes if required.

By default the "{uuid}" will be replaced by the UUID of the processed object. Sometimes in the link template you need also some additional data for instance UUID of the parent object in e.g. PersonName you also need the UUID of Person. The link template looks like:

```
... "rest" : "/ws/rest/v1/person/{parent-uuid}/name/{uuid}?v=full"
...```

To handle that kind of specific mapping you can create the specific converter. For instance for PersonName exist PersonNameFeedBuilder.java which extends the SubResourceFeedBuilder.java.

For choosing the specific builder is responsible that method:
private static FeedBuilder getCustomFeedBuilder(OpenmrsObject openmrsObject) {
    String feedBuilderBeanId = AtomfeedConstants.FEED_BUILDER_BEAN_ID_PREFIX +
        openmrsObject.getClass().getSimpleName() + AtomfeedConstants.FEED_BUILDER_BEAN_ID_SUFIX;
    return getRegisteredComponentSafely(feedBuilderBeanId, FeedBuilder.class);
}

where FEED_BUILDER_BEAN_ID_PREFIX = "atomfeed." and FEED_BUILDER_BEAN_ID_SUFIX = "FeedBuilder".

Steps for Sync 2.0

- Use the Atomfeed module (prefer Bahmni over ICT4H) to build an Atomfeed and make it a dependency of Sync 2.0
- Clean up the code and make sure it supports Sync needs
- Change the default urls to point to FHIR resources
- Update the module for generic support for OpenMRS data
- Sync on startup and on global property changes should verify if synchronization is enabled for an entity without a feed published. In such a case warnings should be logged and signalled by the module.
  - Note: Since the configuration for the feed will be separate, it should be possible to have feeds for resources that are not being synced. However Sync should not be configured for entities that don’t have the feed enabled. In that case:
  - Add the resources - currently the number of resources that publish the feed is limited, for Sync 2.0 we will want to add multiple additional resources.
- The feed should contain enough data for a client (Child) to decide whether it wants to read it (or push if it is reading its own feed). Only data that is being synced should be transferred outside of the feed.
- Implement an Atomfeed that client that the sync module (and others) will be capable of using.