Towards API 2.0

The OpenMRS 2.0 UI is undergoing a lot of improvement and the API should garner an equal amount of attention.

There is a possibility of turning the current api into a jar or omod file. This creates space for the API to be completely rewritten without having to deal with backwards compatibility

Package Name Options

If the pojos and services are going to be changing drastically, then the package for the 2.0 api will have to be different. Keep in mind that the current package name is org.openmrs.api

- org.openmrs.api2 - Ben Wolfe
- org.openmrs.core.api - Ben Wolfe
- org.openmrs2.* – Burke Mamlin (since not only API is affected)
- org.xopenmrs.* or org.openmrsx.* – Nyoman Ribeka (following the java way)

Pojo Change Wishlist

The objects in the api (Patient, Person, Obs, etc) follow a common theme. Here is a list of things that I've either dreamed up or heard from others about things to change in them

- Make the objects more DTO-like (less hibernate magic) - Ben Wolfe
- Remove all calls to the database in pojos (no calls to Context) - Ben Wolfe
- Remove ability to traverse entire graph (creator object attribute becomes creatorId integer attribute) - Ben Wolfe
- Never use Sets as the Collection implementation as an attribute on an object (use List) - Ben Wolfe
- Most properties (especially lists) immutable by default – Burke Mamlin
- Avoid empty constructors & don't allow invalid objects to be created: if "stubs" needed, make them explicit (i.e., a separate object) – Burke Mamlin
- Use uuids for .equals and .hashcode - Ben Wolfe (see

  Error rendering macro 'jira'

  Unable to locate Jira server for this macro. It may be due to Application Link configuration.

)
- Move audit information out of Pojos – Burke Mamlin
  - Collections should not contain voided (deleted) items; rather, clients should have to specifically ask for deleted content or access an audit service to get to the history (changes & deletions) of objects.
  - Change voided to be deleted
- Change retired to be disabled (? Not sure on the name of this one)
  - -1 from Burke Mamlin, since "disabled" suggests a more temporary state than "retired" (enabling sounds good; unretiring sounds bad... which is what we want)
- Inside every pojo, we need to have constants for the property of pojo. This way we can refer this property in the hibernate criteria using this constants instead of using String. – Nyoman Ribeka
  - example: PersonName will have public static final String GIVEN_NAME = "givenName", this way we can access them with PersonName. GIVEN_NAME in the hibernate criteria.
- Pojos should reflect the real world instead of the data model – e.g., an Obs should be able to have multiple values or be a group of other Obs. -- Burke Mamlin

Service Change Wishlist

Similar to the previous list, here is a list of things that people would like to change about our current services (PersonService, PatientService, etc)

- Lessen the hibernate 'magic'. Never return a hibernate magical object - Ben Wolfe
- Get rid of the need for sessions (opensession/closesession). This goes with previous point about hibernate magical objects. - Ben Wolfe
- Get rid of the static Context. Inject everything like standard spring apps in order to take advantage of other tools, mocks in unit tests, etc - Ben Wolfe
- Do *not* return retired metadata or voided data by default – Burke Mamlin
  - In API 1.x, metadata lists – e.g., getConceptAnswers() – include retired values by default, which can encourage use of values that are no longer valid.
- More use of `final` keyword for parameters – Burke Mamlin
- Attention to multithreading support – Burke Mamlin
- Use Long instead of Integer for internal identifiers -- Burke Mamlin
  - Part of our goal for supporting 2+ million patients in OpenMRS will require supporting tables that contain over 2 billion rows
- Use absolute times (i.e., UTC) *everywhere* within the API.
  - We can always provide convenient means for timestamps to be rendered in the current timezone; however, all stored values should be in UTC in order to eliminate issues with transferring data across timezones or implementation headaches in locations that observe DST.
- Introduce an Event Bus so not everything is done via AOP
- Follow lessons given byBloch's Google Tech Talk (slides) – Burke Mamlin
  - Don't let implementation details "leak" into API

    - Self-Explanatory, Consistency, Symmetry
    - Strong javadocs
    - Avoid long parameter lists (break up method or create helper class to hold parameters)
    - Avoid return values that demand exception processing (return zero-length array or empty value, not null)
    - Favor unchecked exceptions
- Clustering should be supported at the API level, this will be helpful as some implementations grow bigger and wish to run in clustered environments - Wyclif Luyima
- I would rather us to have a lot of methods with less parameters than one method with tons of parameters. (so we won't see a call getPerson(null, null, null, null, null) in our code) – Nyoman Ribeka
- Integration of a full text search engine and at least replace the existing ConceptWord engine - Wyclif Luyima
- To add on Win's request, we should have less methods too, we can replace some method parameters with a single map of optional parameters, i believe there can be other use cases for this – Wyclif Luyima
    - INSTEAD OF THIS:

```
ConceptServce.getConcepts(String phrase, Locale locale, Datatype includeDatatype, Datatype
excludeDatatype, ConceptClass includeClass, .... );
```

    - WE HAVE THIS:
        - **Method signature**: *The method declares the optional arguments via an @optionalParam annotation*

```
@OptionalParam(name='locale', type=Locale.class)
@OptionalParam(name="includeDatatype", type=Datatype.class)
@.......
//you can have a minimal number of required parameters for the method
public List<Concepts> getConcepts(String phrase, Map<String, Object> optionalParams){
    Locale locale = null;
    Datatype includeDatatype = null;
    .....
    if(optionalParams != null)
         //A call to a generic method in a utility ParamUtils class
         locale = ParamUtils.getParameter(optionalParams, 'locale', Locale.class);
         includeDatatype = ParamUtils.getParameter(optionalParams, 'includeDatatype',
Datatype.class);
         .......
       }
       dao.getConcepts(phrase, locale, includeDatatype, ..............)
}
```

    - **Calling the method**:

```
Map parameters = new HashMap();
parameters.put('locale', Locale.ENGLISH);
parameters.put('datatype', Context.getConceptService().getDatatype(1));
List<Concept> concepts = Context.getConceptService().getConcepts('malaria', parameters);
```

    - Agreed, but I would go one step further and provide constants for the parameter names – e.g., `ConceptService.PARAM_LOCALE = "locale";` – external clients can use the strings, but internal code could benefit from using the constants. – Burke Mamlin
-