In-page localization Backend design

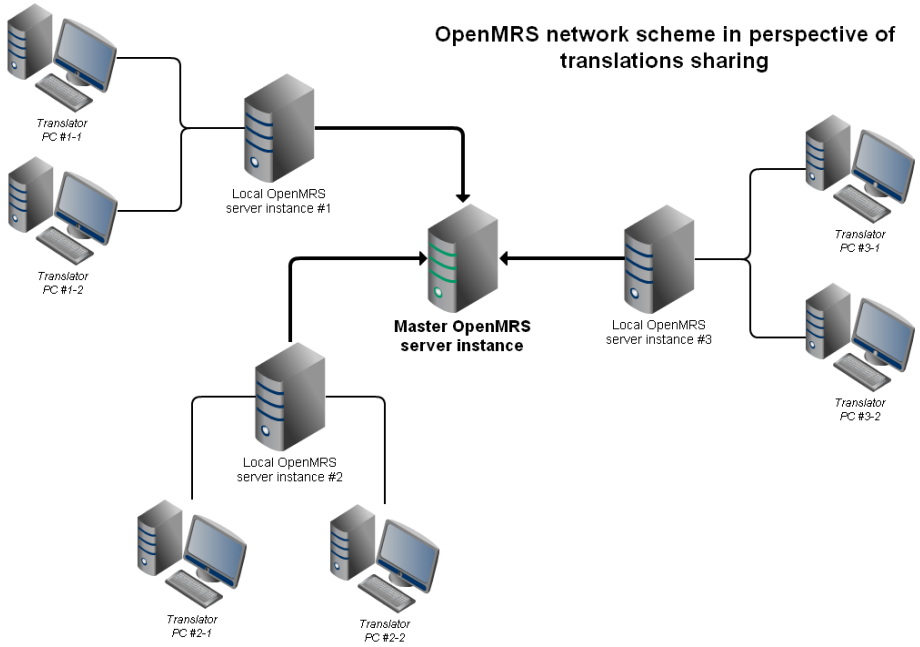
Abstract

The server side part of OpenMRS in-line localization tool should be designed as simple and sophishicated service, which can be easily accessed from client side part via dynamic AJAX requests. It will be built on top of custommessages module API.

Conception

> Note! The conception, given below, is true as for saving/exporting and as for committing translations. Basically, committing functionality will come later and currently it is beyond the scope of the project.

The very simplified idea of *translations sharing* is the major abstraction. This idea assumes that there will be two types of OpenMRS server instances: *client* and *master*. Translations will be shared between them just like regular metadata. In the most common case, there will be only *one master* OpenMRS instance and *amount of local client* instances (see picture below this paragraph). Each local client instance will know who is her master. Substantially, all translations will be being created and collected with local client instances. Any client instance even can don't have a permanent internet connection. All producing translations will be being stored on local server. Later, when connection will be available, these translations will be submitted to master instance. Also, person, who makes translations will have opportunity to export translations of his local instance to file.

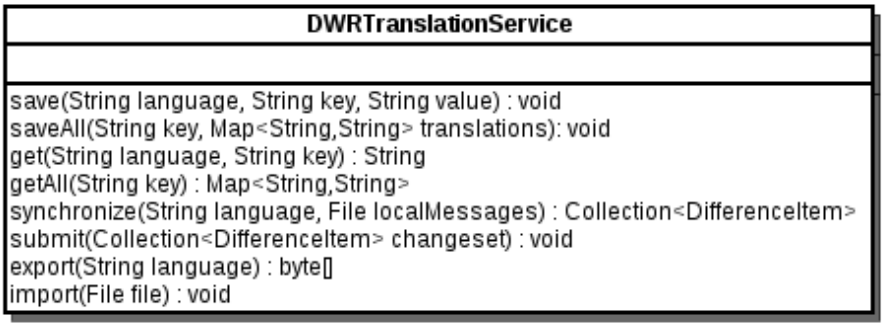


Next points outline (in more concrete example) the idea given above:

1. When user (let's say *Translator PC #1-1*) does any in-line translations on any pages, he can immediately store all outputs, made him so far. In this case, client side component of in-page localization tool will post new changeset to *local OpenMRS server instance #1*;
2. This server, having corresponding handler, will save new translations into message properties file;
3. In case of successfully saving of new changeset, the person, who makes translations, will be notified with corresponding message that says: *"Thank you! Your translations have been succesfully saved to local server, but was not committed to the main server yet. If you want to commit them ..."*
4. Then, translator can choose to commit local changes. *Local server #1 instance*, which receives commit requests, will check the connection with *master OpenMRS server instance*. If the connection can be established, it will make something like svn diff of message properties files on both local and master servers. After creating the diff, *local server #1* will respond this diff in user-friendly format to *Translator PC #1-1* client. Client may review it, correct any conflicts and approve commit operation. In fact, *local server #1* will send approved diff to *master OpenMRS server instance*, which will try to apply received changeset. In case of succeed, master sends corresponding aknowledge to local server and person, who made these translations, will be notified with success message.

Design

Technically, the server side of in-page localization tool will add corresponding DWR service into custommessages module. DWR service will be responsible for AJAX requests handling. It will be implemented with using of existing *org.openmrs.module.custommessage.service.CustomMessageService* service class. To implement the conception explained in previous section this DWR based service should have corresponding method:



- **save(String language, String key, String value) : void** - saves translation value string by given *key* into message properties file, specified by *lan guage* parameter (if translation with given *key* for this *language* already exists, then it will be replaced with new *valu*e);
- **saveAll(String key, Map<String,String> translations) : void** - saves *translations* by given *key* for multiple languages passed as keys within map (if translation with given *key* for certain *language* already exists, then it will be replaced with new *value*);
- **get(String language, String key) : String** - gets translation string for given *language* by *key;*
- **getAll(String key) : Map<String,String>** - gets all available translations for all languages by given *key;*
- **synchronize(String language, File localMessages) : Collection<DifferenceItem>** - provides collection of differences between local and master message properties file for given language;
- **submit(Collection<DifferenceItem> changeset) : void** - submits new *changeset* with difference items to master OpenMRS server instance;
- **export**(**String language) : byte[]** - exports all new translations for given language into dynamic message properties file and responds this file to cl ient browser, so user can store this translations onto his file system;
- **import(File file) : void** - merges translations from given file with corresponding message properties file and dynamically reloads resouce bundle;

When DWR service implementation will doing synchronize and submit operations, it will interacting with java servlet on master server (in will be added into OpenMRS web/ project). Programmatically, this servlet will work by *doPost()* method, the distinguishing between synchronize and submit operations will be done by request parameter.