2019 GSoC XWiki Project Proposal

Contact Information

- Name: Zhijun Chen (Frank Chen)
- Email Address: frankcookie233@gmail.com
- IRC messenger name: @frankchen:matrix.org
- My Resume

Background & Education

- School: Southern University of Science and Technology, China
- Major: Computer Science
- Grade: Sophomore

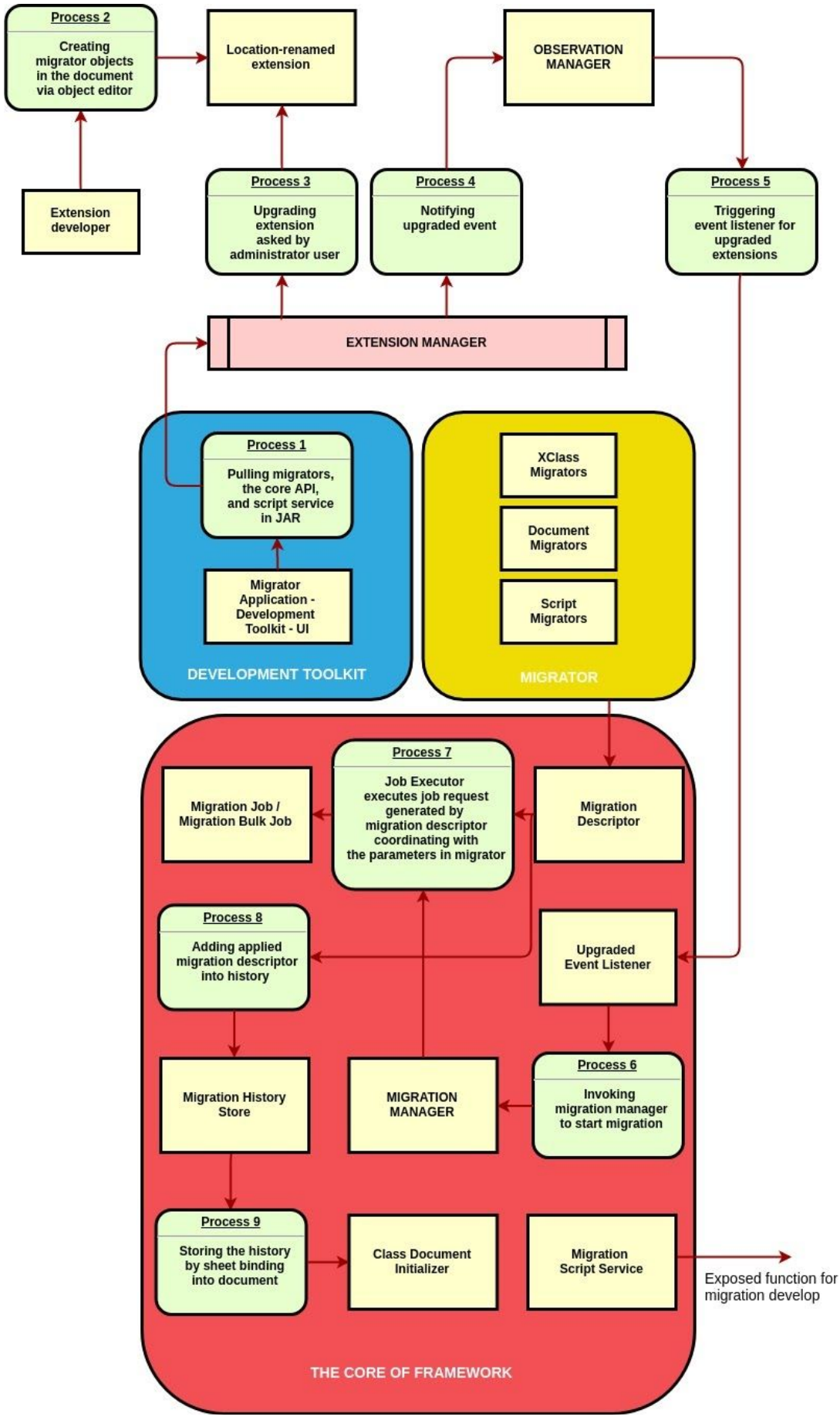
Project Information

Project Name: Data Migration Framework (DMF)

Project Description

During the updating, some extensions, which change the xwiki doc location in the new version, need to **Migrate its document, Xclass, Xobject, Xproperty** into the coordinate environment. For instance, when application-recruitment upgrades from v1.x to v2, it needs the migration process since its space Recruitment and RecruitmentCode has renamed to JobArea and RecruitmentApp, and the same situation happened to application-task. After updating by the Extension Manager (EM), both version pages are stored but the old data related with old version cannot be retrieved again.

In order to break ground, Data Migration Framework aims to help those space-renamed extensions correct and fresh document references, XClass names, XProperty values and run script service during migration. The DMF is divided into three components: the core APIs, migrators and development toolkit. And these three components work in the following way (with figure below).


Process 2
Creating migrator objects in the document via object editor
Location-renamed extension
OBSERVATION MANAGER
Extension developer
Process 3
Upgrading extension asked by administrator user
Process 4
Notifying upgraded event
Process 5
Triggering event listener for upgraded extensions
EXTENSION MANAGER
Process 1
Pulling migrators, the core API, and script service in JAR
Migrator Application - Development Toolkit - UI
DEVELOPMENT TOOLKIT
XClass Migrators
Document Migrators
Script Migrators
MIGRATOR
Process 7
Job Executor executes job request generated by migration descriptor coordinating with the parameters in migrator
Migration Job / Migration Bulk Job
Migration Descriptor
Process 8
Adding applied migration descriptor into history
Upgraded Event Listener
Migration History Store
MIGRATION MANAGER
Process 6
Invoking migration manager to start migration
Process 9
Storing the history by sheet binding into document
Class Document Initializer
Migration Script Service
THE CORE OF FRAMEWORK
Exposed function for migration develop

- Firstly, the development toolkit imports API reference for EM before extensions upgrade.
- After upgrading, UpgradedExtensionEventListener is triggered by observation manager while UpgradedExtensionEvent is notified.
- Finally the core API of Data Migration Framework-migration manager apply migration by executing job request.
- As a consequent, those data stored in old document can be converted to the latest version document by the process of migration.

Why Am I Interested in DMF Project？

Data Migration Framework is an useful in **Against-Data-Loss** and **Wide-Usage Potentiality** that appeals to me.

I highly precise this DMF project because it will benefit the XWiki community and make XWiki more extension-developer-friendly by avoiding data loss during extension upgrading process:

- For instance, it can help migrate User Interface Extension Point, so that custom-defined content will not be risked during upgrading.
- Additionally, the DMF can also conduct migrating on Sheet Files so that customized configuration will be safe.
- The DMF can also help make it possible to migrate and change some script contents such as attachment- and object- reference so that script attachments and target objects can work well in the new version renamed extension.

What is more, instead of designing one single extension for GSoC project, I prefer to artisan one framework which all extensions can benefit from. Currently, extension developers who demand migration have to realize it in their own xar files. It is the DMF that will integrity the whole migration process and provide developers with reusable API. With this framework, a wide range of extensions can reduce their development cycle for migration process and do not need focus on the trivial migrating facets.

Deliverables with Detailed Design

XProperty Migrator

Description

The existing XProperty Migration in the Class Migration Executor is the simple copy for the old values:

```
newObject.set(classPropertiesMapping.get(mappingEntry),
((BaseProperty) oldObject.safeget(mappingEntry)).getValue(),
xWikiContext);
```

In some case, the values of properties need to transfer to suit into the new-version extension after upgrading. For instance, Computed Pseudofield is the data model which carries scripts in the XProperty. Object- and attach- references can appear in some complicated Computed Pseudofield.

Due to space rename, these references need to be migrated to avoid script malfunction. For example, during the migration, 'Job.IssueClass' changes to 'XWikiJob.IssueClass', and then its original script property

```
#set($obj = $doc.newObject("Job.IssueClass"))
```

fails and needs a transfering to

```
#set($obj = $doc.newObject("XWikiJob.IssueClass"))
```

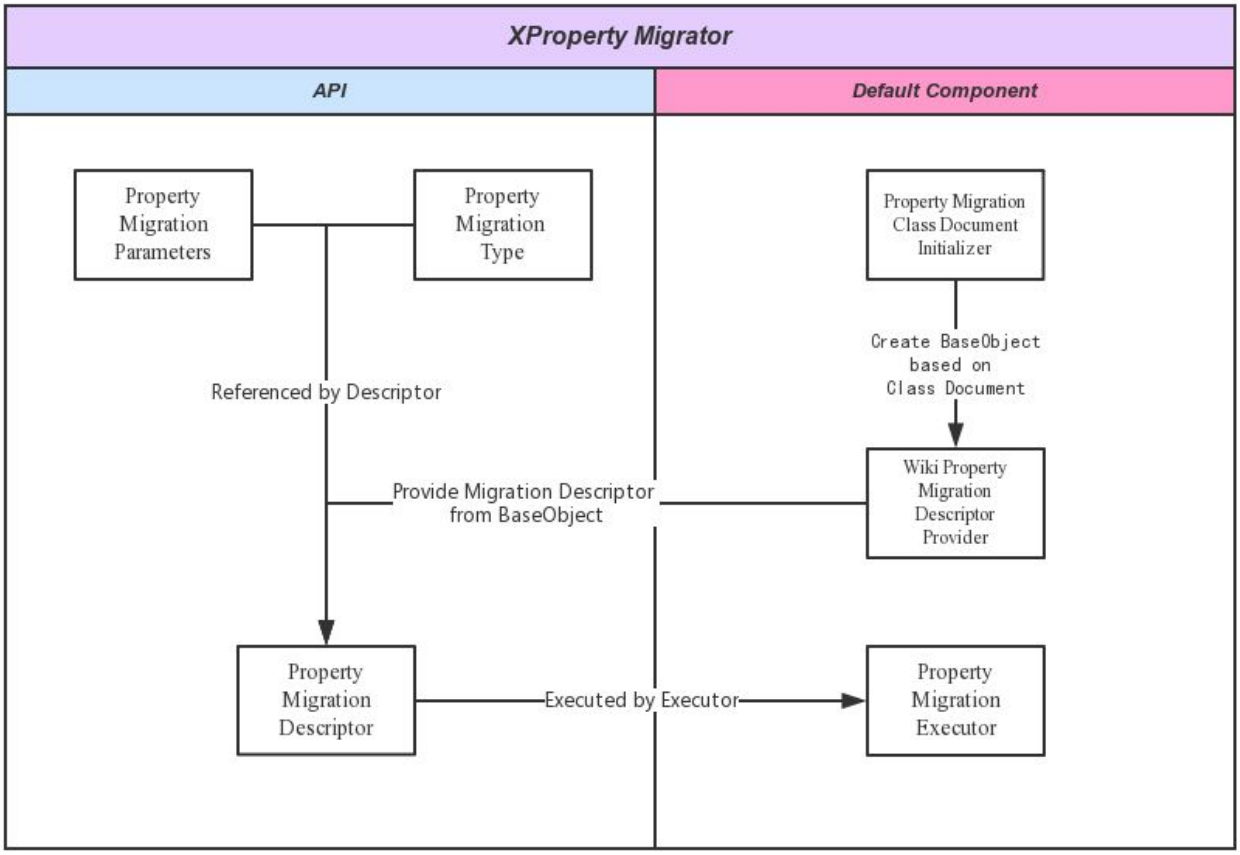
XProperty Migrator handles more comprehensive property migration such as changing scripts, and storing logs than the other Migrators do.

- By replacing the old reference in Computed Pseudofield with the new one, XProperty Migrator get script property updated during migration.
- On the other hand, XProperty Migrator can also be used to store customized migration log into certain property by appending property's value.

Deliverable Details

APIs are 'PropertyMigrationDescriptor', 'PropertyMigrationParameters' and 'PropertyMigrationType' which carry migration information, config parameters and

migrator type respectively. Default components are 'PropertyMigrationClassDocumentInitializer', 'PropertyMigrationExecutor', and 'WikiPropertyMigrationDescriptorProvider' which conduct initializing sheet, executing migration and defining descriptor property. The interactions between these classes are as follows:



- XProperty Migrator has the same component architecture with XClass Migrator and Document Migrator.
- API is mostly related to the configuration and information for migration, while default components play the role in create and execute migrations

'PropertyMigrationParameters' and 'PropertyMigrationExecutor' are significantly unique among XProperty Migrator classes compared with the other migrators. More proposal details for these two classes are below:

In 'PropertyMigrationParameters', example fields can be:

Field Name	Field Type	Description
oldObject	String	old object reference where properties are extracted from
newObject	String	property-received object during migration
transferComputedPseudofield	boolean	migrate Computed Pseudofield if it is true
logProperty	String	the property name in new object where logs append into

In 'PropertyMigrationExecutor', main methods are:

Method Name	Input	Return	Functionality
constructTransfer	--	Map<String, Object>	Invoked by 'contructProperiesMapping' when migrate XObject. Return the map of <property entry, new value of property>.
migrateComputedPseu-dofieldReference	map-Entry	--	Specific migration for Computed Pseudofield. According to input map entry, migrate its value.
migrateLogProperty	property-Name	--	Like the migrate for pseudofield, append migration log into this property's value

The reason why I choose using the name 'contructTransfer' rather than 'constructComputedPseudofieldTransfer' to name the method is to increase its reusability, since maybe more property value transfer is needed in the future.
In the future, with Script Migrator, more customized script can be added into migrateLogProperty method and more things can be logged into certain property.

XObject Migrator

Description

The existing XClass Migrator realizes object migration from old reference Class to the new one. However, XObject Migrator allows every object has its own target class. For example, due to some architecture change, the objects in original one class is needed to separate into two new class. 'Code.JobClass' is too complicated so that in the new extension, two classes, 'Job.RecruitmentClass' and 'Job.ClerkClass' are migrated. Original objects in 'JobClass' need to be migrated into the related one of two new classes.

In this case, XObject Migrator can handle migration at once, while XClass Migrator may need two or more times to migrate different XObjects between different pairs. Vice versa, XObject Migrator can also combine different objects into one class. As a result, object migration for extension is more flexible.

Deliverable Details

APIs are 'ObjectMigrationDescriptor', 'ObjectMigrationParameters' and 'ObjectMigrationType'. Default components are 'ObjectMigrationClassDocumentInitializer', 'ObjectMigrationExecutor', and 'WikiObjectMigrationDescriptorProvider'. These classes have the similar relationships as those in XProperty Migrator.

In 'ObjectMigrationParameters', one typical field is:

```
private Map<String, String> newClasses;
```

The object name is key and target class name is map's value.

In the 'ObjectMigrationExecutor', according to newClasses, migrate objects to different classes.

Mandatory Migration Trigger

Description

The whole migration process can be seen in Project Description Section and currently migration is triggered by upgraded event. A mandatory trigger way is beneficial for unit

test. What is more, it can also be exposed by script service so that facing to big migration, user can manually schedule it.

Design Details

Create script service model to invoke migration manager's certain related methods. In 'MigrationScriptService', add function:

Method Name	Input	Return	Functionality
applyMigrationsForVersion	extensionId	AbstractBulkMigration-JobStatus	manually trigger all migrations of certain extension
applyMigrations	Set<AbstractMigration-Descriptor>	AbstractBulkMigration-JobStatus	manually trigger all migrations provided by the set of descriptors
applyMigration	AbstractMigrationDes-criptor	AbstractMigrationJob-Status	manually trigger the migration provided by the the descriptor

Dependency Detect

Description

It is idealized that migration can be completed successfully. There are some other potential problems such as the dependency of migration. For example, User Interface Extensions highly depend on the location of User Interface Extension Point, which is recorded in another extension usually. In this case, the order of migration matters. To make sure the migration can be executed in the correct order, dependency checking is needed.

Design Details

In my pull request to MIGRATOR-2 issue, Dependency Manager is created. This detect function can be realized in the default Dependency Manager class by search in the dependency link in Dependency Manager.

Method Name	Input	Return	Functionality
checkDependency	AbstractMigrationDescriptor	boolean	check dependency for single migration
checkDependencies	Set<AbstractMigrationDescriptor>	boolean	detect dependencies for a group of migrations
checkDependencyForVersion	extensionId	boolean	check all the dependencies for migrations of one extension

The returned boolean value then can be used in script service to show extension developers the checking result. In 'MigrationScriptService', add function:

```
public boolean checkDependency()
```

Before users manually run migration script, this dependency checking script service can be used to check whether there are dependencies.

Migration Creation

Description

It is convenient if more ways can be used to create migration for example by script service. The migrator creation script service can help achieve this. In the creation of Migrators, the type and parameters are important. The existing way to get base object is through 'ClassMigrationClassDocumentInitializer'. In the script service, parameters can be received and then new one base object carrying these parameters.

Design Details

In 'MigrationScriptService', add function:

```
public String createMigration(String migratorType, Map<String,
String> migrationParameters)
```

The returned String is the name of migration descriptor, and then can be used to retrieval this migration from component manager.

Timeline

Add dependency detect & Testing (May 06-27,2019)

- Week 1 (May 06-13,2019): Further discussion about proposal with community and promote the completion of migration dependency jira issues with mentors
- Week 2 (May 13-20,2019): Add dependency detect in dependency manager.
- Week 3 (May 20- 27,2019): Add test for detect method of dependency manager.

Milestone I - XProperty Migrator (May 27 -Jun 29,2019)

- Week 4 (May 27-June 03,2019): Implement the Property Migrator base architecture.
- Week 5 (June 03-June 10,2019): Realize the Computed Pseudofield transfer method for Property Migrator.
- Week 6 (June 10-June 17,2019): Implement the value transfer for log purpose.
- Week 7 (June 17-June 24,2019): Create Property Migrate Executor.
- First Evaluation time:(June 24 -29,2019)

Milestone II - XObject Migrator(Jul 01- 26,2019)

- Week 8 and Week 9 (July 08-July 15,2019): Design the base architecture for XObject Migrator
- Week 10 (July 15-July 22,2019): Allow object migration in XObject Migration Executor.
- Second Evaluation time:(July 22 -26,2019)

Milestone III - Script Services(Jul 26 -Aug27,2019)

- Week 11 (July 26-Aug 2,2019): Add manually-invoked migration script service to the framework.
- Week 12 (Aug 2-Aug 9,2019): Add create migration script service.
- Week 13 (Aug 9-Aug 16,2019): Create dependency detect script service.
- Week 12 (Aug 16-Aug 27,2019): Submit

Reference

- Migrator Application
- Computed Pseudofield Data Model
- Script Service
- Script Get Object
- JAR Attachment Location
- XClass, XProperty, XClass Editor Model and Sheet
- Script Service for manually trigger migration

Technical skills and Project Experience

- Java/JavaScript/Web Skills: Familiar with Java Spring Framework, ajax asynchronous request and live media based on Nginx.
- Database Skills: Database Migration Script Experience on Laravel 5, as well as Database Design Experience on MySQL for E-Commercial Wechat Applets.
- Project experience: Project summary for Food Cat
- XWiki issue experience:MIGRATOR-2 pull request to help create dependencies by dependency manager

Open Source Experience

I am very interested in open source because open source development is a flexible and convenient way to cooperate on the application and project. The process of learning new architecture and discussion with community appeals to me. I have experience on pulling request and reporting issues for internship company's github repository, and I also actively explore open source such as Django, OpenCV, and Hisi Camera SDK etc. During these days engaging into XWiki community, I report bug on Blockly Editor and XWiki-debug-eclipse in the community IRC room. And I succeed to pull request on MIGRATOR-2 issue. I would like to keep contributing on open source projects in the future.

Summer Plans

For the coming summer, I am planning to take an online course about network penetration on Kali, which is supposed to be finished in the first week of June. Besides

it, I do not have any other plans. So I will stay in my university and focus on working on GSoC project. And I am sure I could spend at least 30 hours one week for GSoC project.

GSoC Experience

Why I decided to apply for the Google Summer of Code?

First, I think GSoC is a great approach for me to get deep into open sources organizations and work with passionate and qualified developers around the world, which helps me committed to coding and contributing.

Second, I respect GSoC mentorship. Due to my personal experience, I find out GSoC mentors are all helpful to give useful guidance and support whenever I ask for help. So now I better understand values of community and I would like to help build up.

Last, as known, GSoC is challenging and intensive in terms of its unique and real world project tasks. However, I enjoy challenging and I believe it helps develop not only my technical skills but also my personality.

Why I decided to apply for an XWiki project?

It is XWiki's wide range of utilization opportunity in the daily lives and work that appeals to me. XWiki is a second-version Wiki platform which provides users with co-operation on different kinds of page artisan and guarantees on the access rights of users. It has more use case fitting into different service systems and daily lives thanks to its support for not only text but also timetable, codebase, CMS etc.

What’s more, I am quite familiar with the language and component architecture XWiki use. XWiki mostly use Java as its development framework and main design pattern is component-oriented design combined with annotations. With the experience of Java, it is quite fast for me to engage in the development.

Why I should be taken?

During these months engaging into XWiki community, I have the knowledge and skills on XWiki architecture, data model, and other facets from the developer guidance. I will try my best to coordinate all of these technique with my previous project experience to artisan the Data Migration Framework with mentors and XWiki community.