Periscope

- Introduction
- Architecture
- Sub/Related Systems
  - Ranking Neural Network
  - Speech Recognition / Vocal Commands
  - Image Recognition
- Periscope SearchResultSuppliers & SearchResults
  - RestSearchResultSupplier
- PeriscopeOperations

Introduction

Periscope is essentially created to satisfy the lack of search capabilities within Magnolia. Initial concept had ideas such as speech recognition, vocal commands, image recognition and ranking search results via Machine learning/Deep learning.

It provides API to plugin so called Result suppliers easily into the search result set as well as possibility to create own vocal commands etc. In order to describe what's the current situation and plan **MGNLPER-8** - Getting issue details... STATUS was created.

More information can be found regarding user stories at: Find bar stories

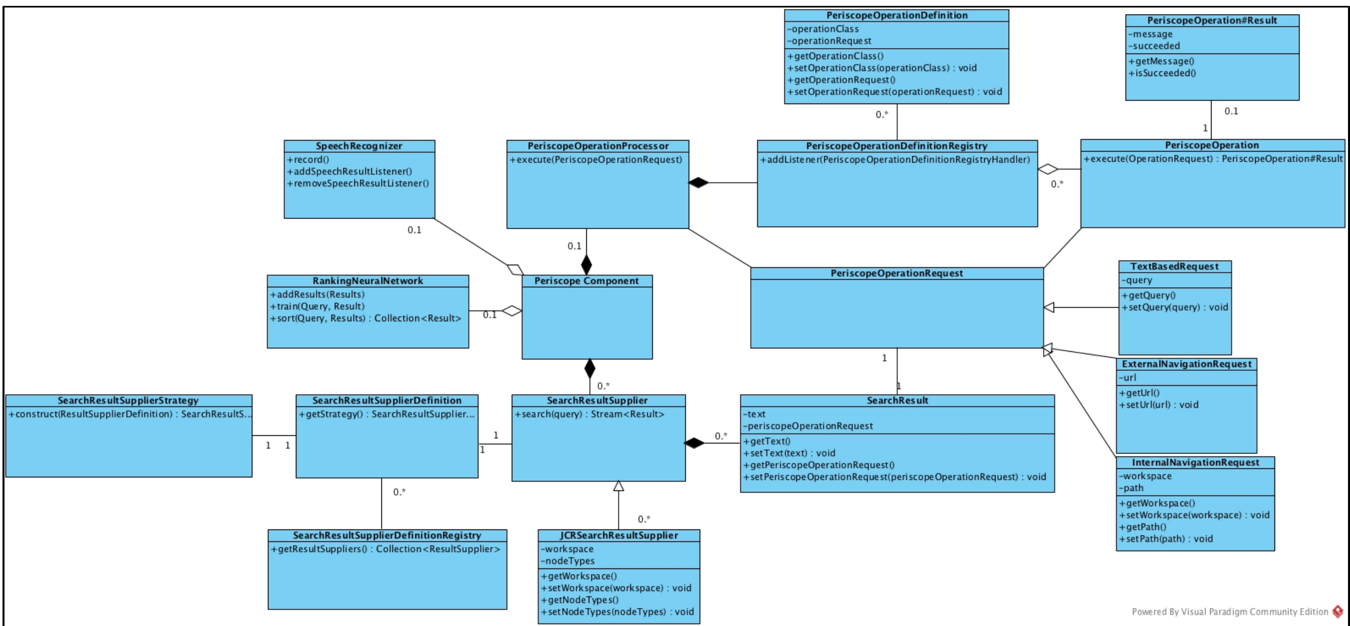
Also details of ResultSuppliers can be found at: Configure search results

It's scope and further discussion about various ResultSuppliers were discussed in here: 2018-03-01 Periscope scope and Solr discussion

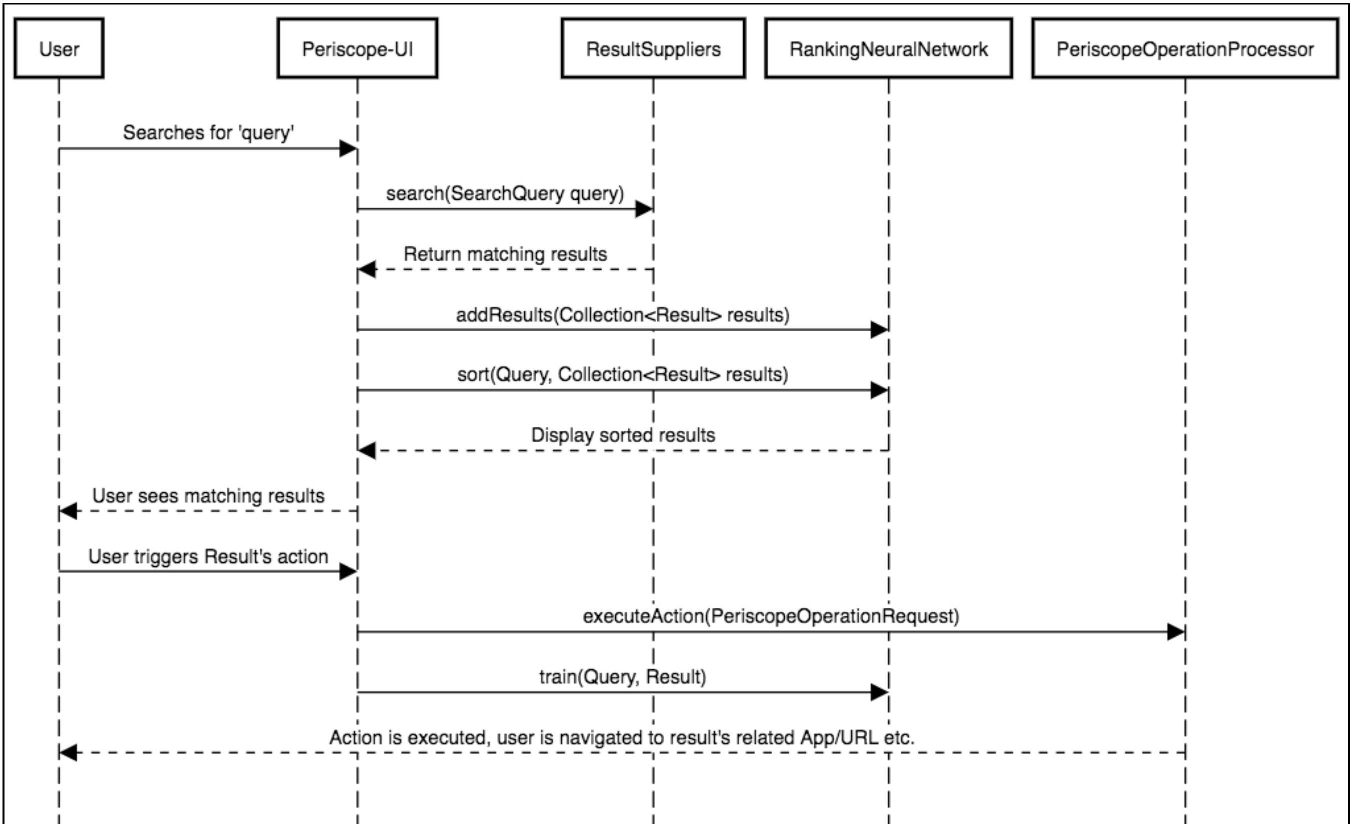
An app has been built to showcase the current situation in periscope-core and it can be found at: https://git.magnolia-cms.com/users/ilgun/repos/periscope-app/browse

Architecture

Overall architecture is as follows:



Typical Sequence diagram would look like:



Sub/Related Systems

Ranking Neural Network

A neural network has been introduced MGNLPER-3 - Getting issue details... STATUS to rank search results provided by so called ResultSuppliers. It uses DL capabilities and a custom algorithm to do the ranking.

More information will come with: MGNLPER-11 - Getting issue details... STATUS

Speech Recognition / Vocal Commands

Speech recognition is essential for Periscope because due to that we can have Vocal commands. Typically users can speak to Magnolia via Microphone and Speech recognition service is able to translate it into text.

Periscope from that point takes care of the rest and execute the text as if it is a normal text query from user.

Currently, the default Speech recognition implementation uses Web Speech API MGNLPER-2 - Getting issue details... STATUS however, although it's in the specification currently it's not really usable from other Web Browsers except Google Chrome.

As of today, Firefox and others have plans to have it in by the time that we release Periscope but there is certain chance that all browsers won't support it.

Therefore we are currently investigating other ways to capture users' voice and use external speech recognition services. There is a Google connector which does it for now and others like Amazon also will be offering a solution in that regard in foreseeable future.

Image Recognition

IMGREC-2 - Getting issue details... STATUS

Image recognition is not a direct component of Periscope.

However, Periscope can potentially benefit a lot from it hence our users. Once image recognition services is in, it'll recognise the images in Magnolia and insert the label as tags.

Those tags can be searchable within Periscope, therefore we can satisfy the need of user searching for an image within Magnolia.

As of today, we have a pre-trained local solution which can detect images with acceptable rate. This can/should be a module of its own.

In addition, Amazon, Google or any other service can be used to detect images. Especially Amazon solution is quite cheap now that we could also offer to our customers for free (Magnolia can take the cost and limit it to say 10 dollars per month).

This way our customers will have different options and choose what fits to their needs.

Periscope SearchResultSuppliers & SearchResults

Periscope provides SearchResultSupplier API for users to customise/configure/implement their own SearchResultSupplier.

SearchResultSupplier is responsible to provide results to the Periscope. To do so one has to implement the following interface:

```
public interface SearchResultSupplier {
    Stream<SearchResult> search(SearchQuery query) throws SearchException;
}
```

One can provide all kinds of results to periscope by simply extending this interface with choice of <T>.

As you see in the example below, results of the Suppliers are wrapped/presented in *SearchResult* object which contains text and a *PeriscopeOperationReq uest* which specific what should be done when user selects the particular result.

SearchResultSupplier are responsible to provide this information since they know what should be done with their provided resultSet. There is a section below which covers PeriscopeOperations in general.

```
public class SearchResult {

    private final String text;
          private final PeriscopeOperationRequest operationRequest;
}
```

and *operationRequest* is the request when user clicks on the desired result from the supplied result set in Periscope.

There is a registry for *SearchResultSuppliers* for convinience, therefore users can define their *SearchResultSuppliers* definitions via YAML and also benefit from other registry goodies.

For maximum convinience we have created *JcrSearchResultSupplier* which is responsible to search over defined JCR workspace and such. It has configurable capabilities to specific node types or doing full-text search.

One can easily define a *JcrSearchResultSupplier* via YAML as such:

```
workspace: website
nodeTypes:
- mgnl:page
fullTextSearch: true
class: info.magnolia.periscope.search.jcr.JcrSearchResultSupplierDefinition
```

All you have to do is to specify the class like in the example and specify what you are interested at. For instance here we care about website workspace and node type of mgnl:page. However, we could also define bunch of other node types here since it is a List.

In addition, we define fullTextSearch as true and as the name suggests *JcrSearchResultSupplier* then will be doing a full-text search within that workspace.

SearchResultSupplierDefinitionRegistry exposes *getResultSuppliers()* method in order to lazily populate various *SearchResultSuppliers* on demand which is typically done by Periscope-UI.

There registry knows how to construct a particular *SearchResultSuppliers* because *SearchResultSupplierDefinition* simply exposes its construction strategy.

```
/**
 * Definitions for {@link SearchResultSupplier Search result suppliers}.
 *
 * <p>
 *     Each definition exposes its strategy via {@link #getStrategy()} method which know how to construct a
{@link SearchResultSupplier}
 *     from {@link SearchResultSupplierDefinition}.
 *
 *     That method later utilised to create {@link SearchResultSupplier Search result suppliers} on the fly
 *     by {@link SearchResultSupplierDefinitionRegistry}.
 * </p>
 */
public interface SearchResultSupplierDefinition {
    SearchResultSupplierStrategy getStrategy();
}
```

```
/**
 * Knows how to construct {@link SearchResultSupplier SearchResult supplier}.
 *
 * @param <S> result supplier.
 * @param <D> result supplier definition.
 */
public interface SearchResultSupplierStrategy<S extends SearchResultSupplier, D extends
SearchResultSupplierDefinition> {
    S construct(D definition);
}
```

This way users easily create their own implementations of *SearchResultSuppliers* via *SearchResultSupplierDefinition* and *SearchResultSupplierStrategy*.

RestSearchResultSupplier

TODO:

PeriscopeOperations

MGNLPER-10 - Getting issue details... STATUS

Periscope operations are built with the idea of customisability in mind. Those operations are completely configurable by users and there are also easy to write/customise.

Essentially they look like this:

```
public interface PeriscopeOperation<T extends PeriscopeOperationRequest> {

    Optional<Result> execute(T operationRequest);

    /**
     * Result for {@link PeriscopeOperation} which merely contains information regarding how the operation went.
     */
    class Result {

        private final boolean succeeded;
        private final String message;

        public Result(boolean succeeded, String message) {
            this.succeeded = succeeded;
            this.message = message;
        }

        public Result(boolean succeeded) {
            this.succeeded = succeeded;
            this.message = null;
        }

        public boolean isSucceeded() {
            return succeeded;
        }

        public Optional<String> getMessage() {
            return Optional.ofNullable(message);
        }

        public static Result successfulResult() {
            return new Result(true, null);
        }
    }
}
```

This is the generic interface which should be implemented in order to introduce a new *PeriscopeOperation.* It can be thought as commans/actions, it merely takes a type of *PeriscopeOperationRequest* which specifies what the request is and they are handled by *PeriscopeOperationProcessor.*

There is a dedicated registry for those operations and thus user can benefit from its functionality out of the box such as hotswapping definitions on the fly and etc. That being said *PeriscopeOperationProcessor* populates *PeriscopeOperation* which are bound to a specific *PeriscopeOperationRequest.* With this in mind let's see how to define a *PeriscopeOperation in YAML*

```
operationClass: info.magnolia.periscope.operation.FindContentOperation
operationRequest: info.magnolia.periscope.operation.request.ImperativeOperationRequest
```

It is as easy as to define an operation class and an operation request class. That way we know in *PeriscopeOperationProcessor* which *PeriscopeOperation* is responsible for which *PeriscopeOperationRequest.*

Currently we have five types of *PeriscopeOperation*s:

InternalNavigationOperation, ExternalNavigationOperation, FindContentOperation, FindNodeOperation, OpenAppOperation. First two operations have their own PeriscopeOperationRequests and the latter three are bound to *ImperativeOperationRequest.*

ImperativeOperationRequest is quite special because it only contains a text such as ''open pages app' which is essentially the user query and each *PeriscopeOperation* is responsible to know if they can handle a particular *PeriscopeOperationRequest* or not.

For instance *OpenAppOperation* only handles query if it exactly matches 'open pages app', otherwise it will not be doing anything.

Periscope UI generates a *ImperativeOperationRequest* upon every user query and tries to execute it via *PeriscopeOperationProcessor,* if there is an *PeriscopeOperation* which is responsible for that particular query then it's executed. If not we simply delegate to *SearchResultSuppliers.*

That is why ImperativeOperationRequest is special because so far it's the only *PeriscopeOperationRequest* which is not created via *SearchResultSuppliers* . *Others* are simply created by *SearchResultSuppliers* and there each *SearchResultSuppliers* is responsible to generate a *SearchResult* with *PeriscopeOperationRequest.*

That allows users to write their own operations/actions/commands and easily execute them via Periscope.