Fraunhofer
IEM

MOIS-ASSIST

IntelliJ IDEA Plugin

Oshando Johnson (oshando.johnson@iem.fraunhofer.de)
Goran Piskachev (goran.piskachev@iem.fraunhofer.de)

Fraunhofer IEM

Table of Contents

Fraunhofer IEM

1 Introduction

This document outlines the architecture and features of MOIS-Assist, an *IntelliJ* plugin that provides IDE support for the identification of configured methods for static analyses. Information about the structure of the plugin and how it will interface with other applications as well as the user experience are covered in this document.

1.1 Goals and Objectives

MOIS-Assist provides a GUI for the information provided by MOIS, a tool for the fully automated classification and categorization of android sources and sinks, by offering IDE support as a Plugin for IntelliJ. The plugin enables a developer identify sources, sinks, sanitizers and other category of methods easily and intuitively in the source code. These categorizations further lead to the grouping of methods according to Common Weakness Enumeration (CWE) - a list of popular software weaknesses. User-defined categorizations will be possible and the user will also be able to provide feedback about the generated classifications of methods. This feedback will be conveyed to MOIS so that the tool can better classify methods.

1.2 Project Overview and Scope

The plugin will fetch the results from MOIS as well as any user supplied input and will provide output in a similar file format. The plugin will then use this information to help the developer to easily recognize problematic methods in the program by providing different views and details. The developer will be able to modify the classifications of methods and the necessary properties of these methods.

1.3 Core Features

The application will have the following features.

Feature	Description	Priority
Import output JSON file from MOIS	The output JSON file from MOIS provides information about each method that was classified by the application as a source, sink, sanitizer, etc. This information will be loaded by the plugin and used to provide additional info about possible issues with the source code.	High
Summary and details view for methods	The summary view will be provided when hovering over a method name in the editor and also in a view adjacent to the editor. Similarly, the details view will be adjacent to the editor and will provide additional information about various properties of a particular method.	High
Method Reclassification and User Feedback	The user will be able to change the category and other properties of a method.	Medium
Export user reclassification and feedback to MOIS	User supplied information should be made available to MOIS so that the application will be better able to classify methods.	Medium

Import user-defined configuration file	The user will be able to load their own configuration file that follows the format of the output JSON file from MOIS. A form should also be available for the user to make updates.	High
Annotation of methods	The various properties of the methods will be used to annotate and group methods in the editor as well as the summary and details view.	High
Refresh MOIS	The user will be able to trigger a refresh of the categorizations generated by MOIS from the plugin. This refresh should be done on a separate thread so that other activities are not blocked.	Low

2 Data Design

MOIS-Assist's data will be mainly provided by MOIS's output JSON file or a similar file provided by the user. These files must use the same JSON schema and are used to capture details about how a method is categorized based on the static analysis that was done by MOIS or the user. The plugin will also export such a file that MOIS can use to refine its classification process.

A sample of the file layout is provided below.


```
{
   "methods": [
      {
         "name": "org.apache.xalan.xsltc.runtime.BasisLibrary.replace",
         "return": "java.lang.String",
         "parameters": [
            "java.lang.String",
            "java.lang.String",
            "java.lang.String[]"
         ],
         "dataIn": {
            "return": false,
            "parameters": [
               0,
               1,
               2
            ]
         },
         "dataOut": {
            "return": true,
            "parameters": []
         },
         "securityLevel": "none",
         "discovery": "manual",
         "framework": "apache",
         "link": "https://xml.apache.org/xalan-
j/apidocs/org/apache/xalan/xsltc/runtime/BasisLibrary.html",
         "cwe": [
            "cwe352",
            "cwe078",
            "cwe311",
            "cwe079",
            "cwe798",
            "cwe306",
            "cwe089",
            "cwe807"
         ],
```

```
    "type": [
       "sink",
       "sanitizer"
    ],
    "comment": "String replacement"
}. . . . .
```

The plugin may also rely on a XML configuration file that would store the settings for the plugin.

3 Architecture

The plugin will obtain the data it needs to provide IDE support from MOIS's output file or a user supplied file. Plugin settings stored in an XML file may also be necessary.

The plugin will load the contents of the output file to memory. This information would include details of the different properties of the methods that have been categorized as sources, sinks, sanitizers, etc. The plugin would then invoke the necessary interfaces in *IntelliJ* in order to highlight various methods that are problematic as well as the list methods and their categorizations.

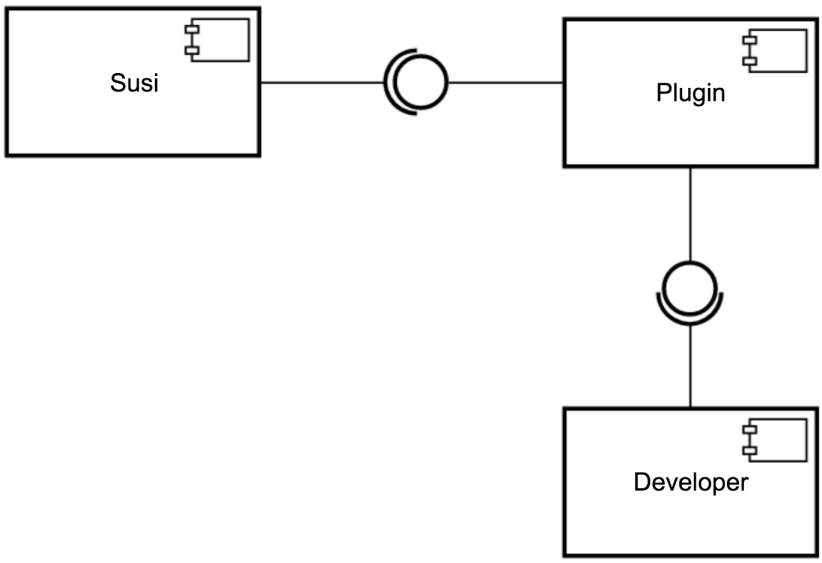

Figure 1: *Component-based Architecture of MOIS-Assist.*

4 User Experience

A mockup of the user interface is provided in the image below. The developer will be able to extract information about a particular method from the various sections highlighted in the diagram.

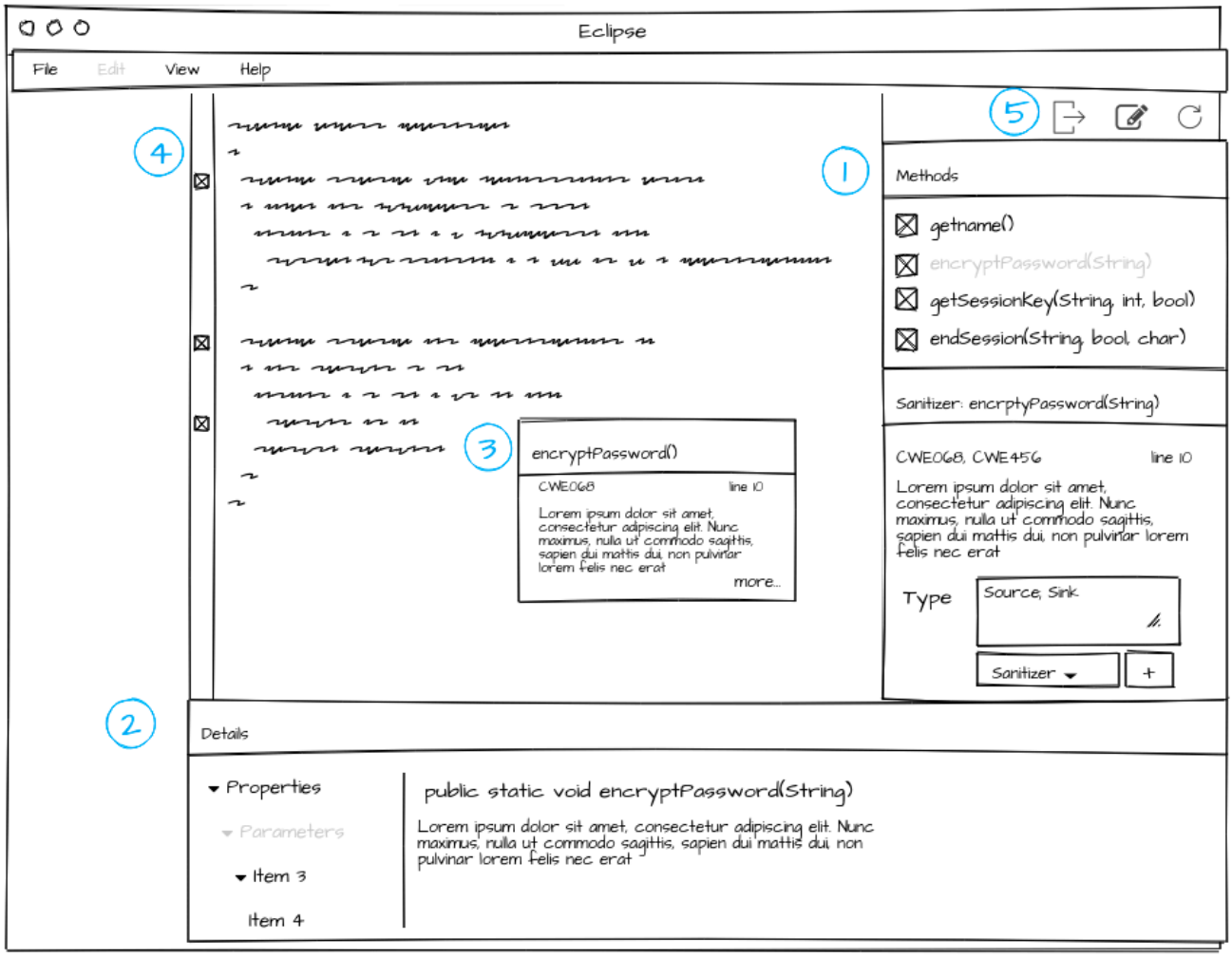

***Figure 2**: Mockup of MOIS-Assist's user interface.*

Section 1

This section will occupy its own view and will provide the user with important summary information about a method that is selected from the list. Icons will be used to represent the various categories of the methods and the user will be able to change the categorization of the menu from this view.

The menu at the top will allow the user to import/export configuration files, reload a file as well as other functions.

Section 2

This will be the details view that provides additional details about a methods properties and classification. As the developer browses the various properties, they will see the corresponding information. This view should also be editable.

Section 3

On mouse over information about a method will be available to the developer. For example, they will be able to see the category that a particular function may belong to.

Section 4

This section will provide annotation for particular lines in the editor. Various flags will be used in the margin to identify improper usages of methods (warnings and errors).

Section 5

This will act as the menu for the applications with options to refresh the categorizations provided by MOIS, load user-provided categories or to export the categorizations provided by MOIS.

5 Conclusion

The ultimate aim of this plugin is to provide the developer with important information about possible sources, sinks, sanitizers and other category of methods in a program. The tool will extend IntelliJ's framework such that the information about various methods is seamlessly integrated in the IDE.

Being provided with such information, the developer will be better able to manage and improve the program's source code. Ultimately, effectively using the feedback from MOIS provided using MOIS-Assist will help the developer to avoid some of the Common Weakness Enumeration (CWE) that MOIS aims to identify.