Project Number 732223

D7.6 IDE Integration Services - Final Version

Version 1.0
22 December 2018
Final

Public Distribution

FrontEndART

Project Partners: Athens University of Economics & Business, Bitergia, Castalia Solutions, Centrum Wiskunde & Informatica, Eclipse Foundation Europe, Edge Hill University, FrontEndART, OW2, SOFTEAM, The Open Group, University of L'Aquila, University of York, Unparallel Innovation

Project Partner Contact Information

Athens University of Economics & Business Diomidis Spinellis Patision 76 104-34 Athens Greece Tel: +30 210 820 3621 E-mail: dds@aueb.gr	**Bitergia** José Manrique Lopez de la Fuente Calle Navarra 5, 4D 28921 Alcorcón Madrid Spain Tel: +34 6 999 279 58 E-mail: jsmanrique@bitergia.com
Castalia Solutions Boris Baldassari 10 Rue de Penthièvre 75008 Paris France Tel: +33 6 48 03 82 89 E-mail: boris.baldassari@castalia.solutions	**Centrum Wiskunde & Informatica** Jurgen J. Vinju Science Park 123 1098 XG Amsterdam Netherlands Tel: +31 20 592 4102 E-mail: jurgen.vinju@cwi.nl
Eclipse Foundation Europe Philippe Krief Annastrasse 46 64673 Zwingenberg Germany Tel: +33 62 101 0681 E-mail: philippe.krief@eclipse.org	**Edge Hill University** Yannis Korkontzelos St Helens Road Ormskirk L39 4QP United Kingdom Tel: +44 1695 654393 E-mail: yannis.korkontzelos@edgehill.ac.uk
FrontEndART Rudolf Ferenc Zászló u. 3 I./5 H-6722 Szeged Hungary Tel: +36 62 319 372 E-mail: ferenc@frontendart.com	**OW2 Consortium** Cedric Thomas 114 Boulevard Haussmann 75008 Paris France Tel: +33 6 45 81 62 02 E-mail: cedric.thomas@ow2.org
SOFTEAM Alessandra Bagnato 21 Avenue Victor Hugo 75016 Paris France Tel: +33 1 30 12 16 60 E-mail: alessandra.bagnato@softeam.fr	**The Open Group** Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org
University of L′Aquila Davide Di Ruscio Piazza Vincenzo Rivera 1 67100 L′Aquila Italy Tel: +39 0862 433735 E-mail: davide.diruscio@univaq.it	**University of York** Dimitris Kolovos Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325167 E-mail: dimitris.kolovos@york.ac.uk
Unparallel Innovation Bruno Almeida Rua das Lendas Algarvias, Lote 123 8500-794 Portimão Portugal Tel: +351 282 485052 E-mail: bruno.almeida@unparallel.pt	

Table of Contents

1 Introduction 2

2 Technical documentation 3
2.1 Scenarios . . . 3
2.1.1 Pull-based Protocols . . . 3
2.1.2 Push-based Protocols . . . 4

3 Implementation 5
3.1 Relevant Participants of Integration . . . 6
3.2 Abstract Layers of Integration . . . 6
3.2.1 Common REST API Specification . . . 7
3.2.2 Common REST API Implementation . . . 7
3.2.3 Shared JSON Serialization Logic . . . 7
3.2.4 Shared Communication Data Model Implementation . . . 7
3.3 Features of CROSSMINER Eclipse IDE Plug-in Currently Relying on the Integration Services 7
3.3.1 Library Search Feature . . . 7
3.3.2 Code recommendation feature . . . 8
3.3.3 API documentation and Q&A posts . . . 9
3.3.4 Integration Related Settings . . . 10
3.4 Technical details . . . 10
3.4.1 Using OpenAPI standard as a Common REST API Specification . . . 11
3.4.2 Automatic Generation of REST API library for CROSSMINER Eclipse IDE Plug-in . 15
3.4.3 Description of the Generated API Client's Functionality . . . 15
3.4.4 Usage of the Generated API Client's Functionality . . . 19
3.5 Integration with the Continuous Integration System of CROSSMINER . . . 19
3.5.1 Using Tycho to Build the CROSSMINER Eclipse IDE Plug-in . . . 19
3.5.2 Framework for Automatic Update Site Deployment . . . 20

4 Further Steps 21

5 Conclusion 22
5.1 Technical requirements . . . 22
5.2 Use case requirements . . . 22
5.3 Next development steps . . . 22

Document Control

Version	Status	Date
0.5	Initial version	29 November 2018
0.8	Draft ready for internal review	3 December 2018
0.9	First pass corrections	19 December 2018
1.0	Final version	22 December 2018

Executive Summary

This document presents the deliverable D7.6 (IDE Integration Services - Final Version) of the CROSSMINER project. The deliverable is the final implementation of the IDE Integration Services, presented in task T7.3 of WP7 as part of the CROSSMINER Eclipse IDE Plug-in. IDE Integration Services enables the connection between the Eclipse IDE and the Knowledge Base.

The deliverable covers 100% of the plug-in integration related technology and 100% of the plug-in integration related use case requirements defined in deliverable D1.1 (Project Requirements). These covered requirements include the use of a common API description file and framework, the handling of JSON format, using the proper character encoding method, and connecting the CROSSMINER server to query certain recommendations.

1 Introduction

One of the main functionalities of the CROSSMINER Eclipse IDE Plug-in is to provide an interface for the end users through which they can access the functionality of CROSSMINER Server. The Integrated Development Environment itself acts as a client in this setting; most of the work is done by the CROSSMINER server where the knowledge is stored and the "answers" are calculated.

Figure 1 shows how the CROSSMINER Eclipse IDE Plug-in is connected to the CROSSMINER platform. It is connected to the Logic Layer (the CROSSMINER server) through the common CROSSMINER API. In the previous deliverable D7.5, we designed and planed the high level architecture of the integration services. The main goal of this deliverable was to define low level technical details based on these plans and describe implemented functionalities; by doing so, we enable the IDE to connect directly to the CROSSMINER server through the CROSSMINER API.

Section 2 gives an overview of the used protocol designs. The implementation of the connectivity features of the CROSSMINER Eclipse IDE Plug-in is described in Section 3. The planned further steps towards other related features of the CROSSMINER Eclipse IDE Plug-in is described in Section 4, and Section 5 links the current implementation status to the project requirements.

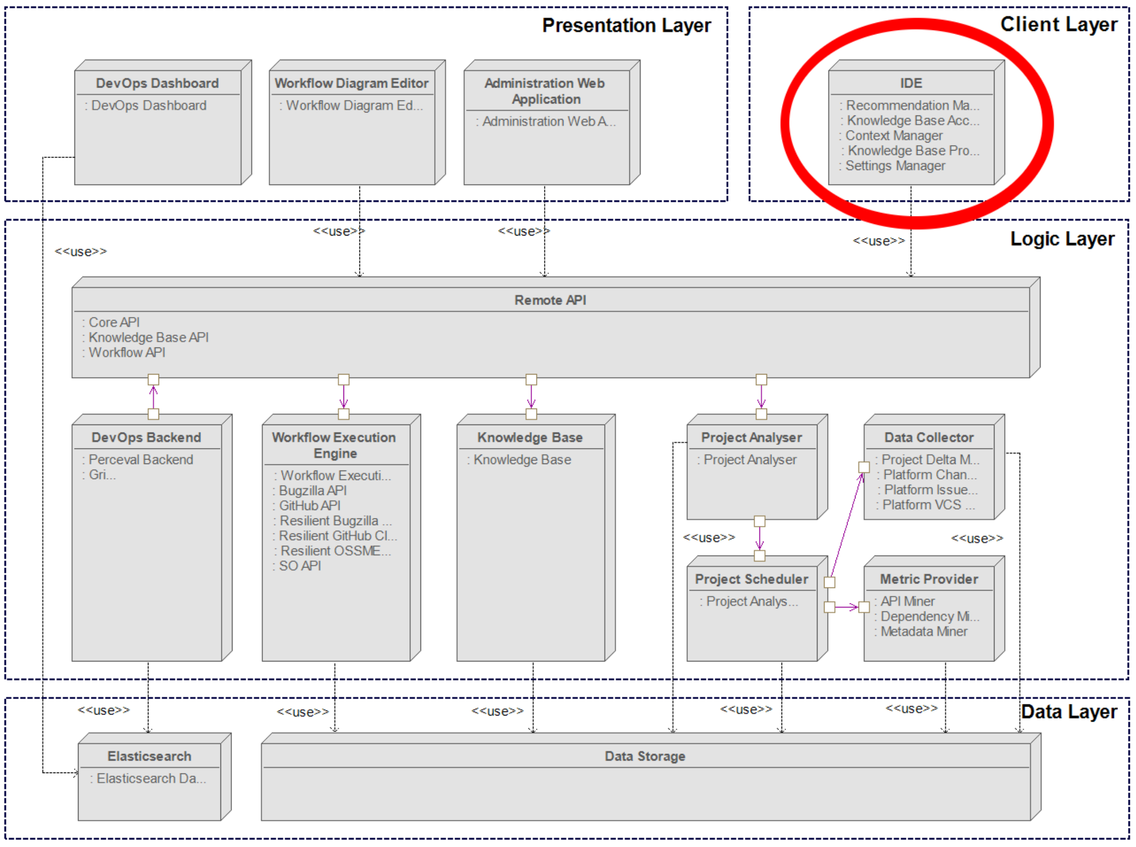


Figure 1: Location of the Integrated Development Environment in the CROSSMINER platform

2 Technical documentation

As mentioned above, CROSSMINER Eclipse IDE Plug-in is basically a user interface that enables the user to utilize the functionalities of the CROSSMINER knowledge base (although it implements other functionalities too). In this setting the Integrated Development Environment acts as a client which is connected to the CROSSMINER server. Most of the work is done by the server, that "answers" the "questions" the user asks using the CROSSMINER Eclipse IDE Plug-in. This requires a well-defined communication interface between the IDE and the CROSSMINER server, which must be used by the CROSSMINER Eclipse IDE Plug-in.

2.1 Scenarios

The communication between the CROSSMINER server and the CROSSMINER Eclipse IDE Plug-in could be classified by the main direction of the information flow. In both cases, it is the client that initiates the communication. In the case of recommendation retrieval, the client sends a query to the server which will send back the recommendations. For most of the features provided by the CROSSMINER Eclipse IDE Plug-in, it initiates a request by sending context and environment information to the CROSSMINER server. As a reply, the server sends back its recommendations related to the given situation. In the case of the user activity monitoring, the client simply pushes data to the server. For the first case, a pull-based communication protocol was designed, while, in the second case, the CROSSMINER Eclipse IDE Plug-in uses a push-based approach. These communication classes are illustrated in Figure 2.

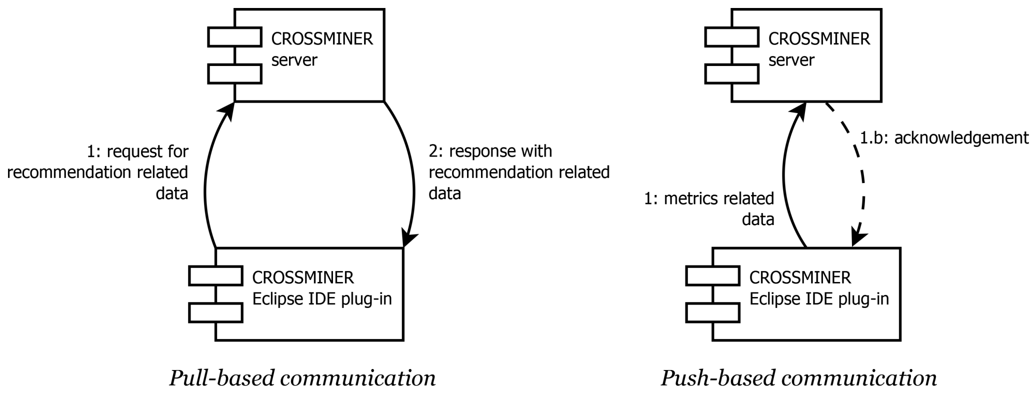

CROSSMINER server
1: request for recommendation related data
2: response with recommendation related data
CROSSMINER Eclipse IDE plug-in
Pull-based communication
CROSSMINER server
1.b: acknowledgement
1: metrics related data
CROSSMINER Eclipse IDE plug-in
Push-based communication

Figure 2: Overview of the CROSSMINER Eclipse IDE Plug-in Integration Related Components

2.1.1 Pull-based Protocols

When the user uses a feature in order to retrieve some information or recommendation, the CROSSMINER Eclipse IDE Plug-in initiates a request in which it sends all the necessary information collected on the client-side to the CROSSMINER server. For example, to check whether there are updates available for a certain library in the currently edited project, the CROSSMINER Eclipse IDE Plug-in collects the library related data and sends it to the CROSSMINER server. This is illustrated on the left side of Figure 2 by the arrow labeled by 1 that points from the CROSSMINER Eclipse IDE Plug-in to the CROSSMINER server. The server performs the given task and sends back the recommendation. This answer contains all the recommendation related data to enable the CROSSMINER Eclipse IDE Plug-in to successfully accomplish the requested feature. For example, the CROSSMINER server provides a list of accessible updates for the given library. This is shown on the left side of Figure 2 by the arrow labeled as 2, pointing from the CROSSMINER server to the CROSSMINER Eclipse IDE Plug-in.

2.1.2 Push-based Protocols

When a feature of the CROSSMINER Eclipse IDE Plug-in, whose primary activity is to send some data to the server, is activated, the plug-in simply sends the data to the CROSSMINER server, as illustrated on the right side of Figure 2 by the arrow, which starts from the CROSSMINER Eclipse IDE Plug-in and points to the CROSSMINER server. So, the client pushes the data to the server without expecting any data in return. However, due to prevent possible information loss, the server has to acknowledge that the current values of the metrics arrived. For example, to send the previously collected user activity metrics, the CROSSMINER Eclipse IDE Plug-in uses the method described above, for further details see *D7.7: Developer Activity Monitoring (Final Version)*.

This solution is more feasible for this kind of client-to-server data transfer than a pull-based method. In this setting, the CROSSMINER Eclipse IDE Plug-in collects data on the “client” side independently from the CROSSMINER server, thus, the server has no information on whether the data is ready or not. A pull-based method (when the communication is initiated from the server side) could work only in a timed way. However, it would require the server to actively maintain a list of its clients, and the success of the request would not be guaranteed as the CROSSMINER Eclipse IDE Plug-in could actually not being executed at the client side when the request is initiated.

3 Implementation

During the design of the CROSSMINER Eclipse IDE Plug-in, we identified several layers of integration among various CROSSMINER components. The most important components of the client and server side regarding the communication are shown in Figure 3. The blue arrow represents the information flow between the components.

As can be seen, there are several common modules used on both the server and client side of the connection. One goal of the integration is to use the same, common implementation of these modules in all components in which it is possible. To do this, we should be aware of the integration-related CROSSMINER components. These components, their connections, and the status of the integration at the different layers are described briefly in the following subsections.

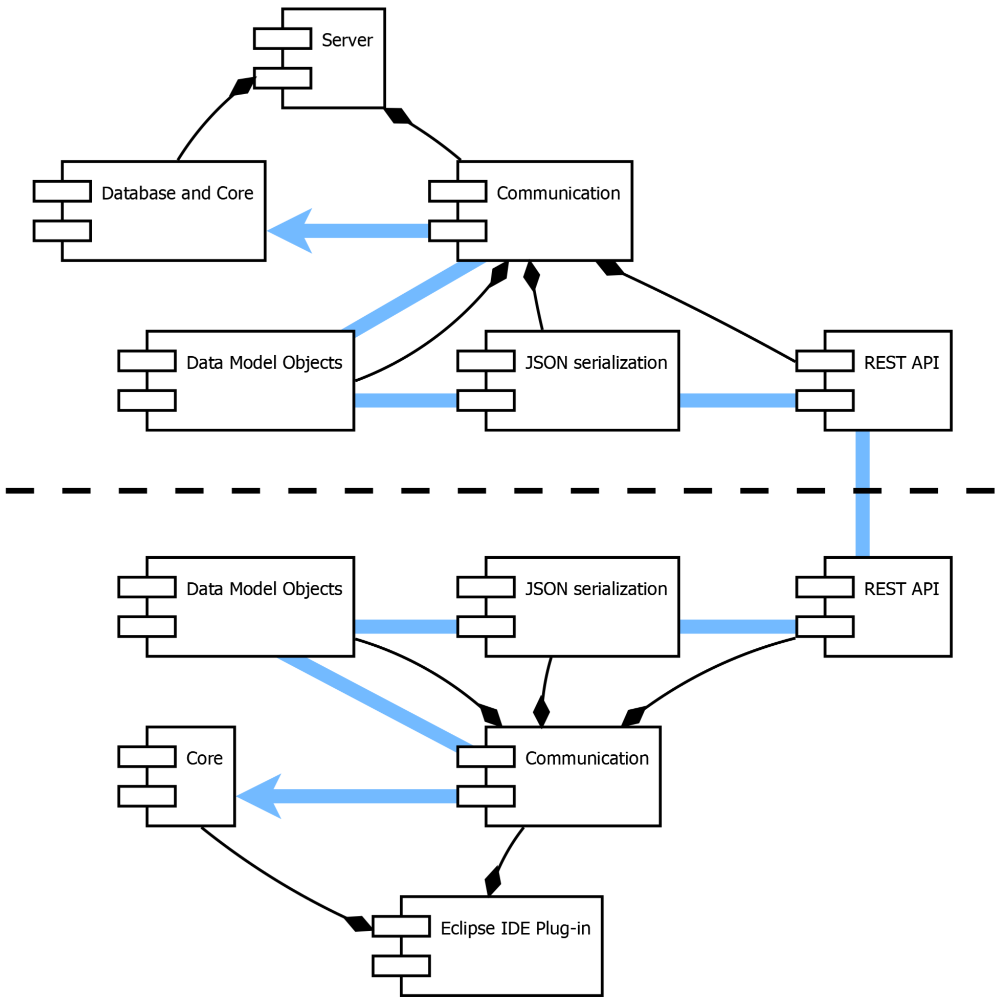


Figure 3: Overview of the CROSSMINER Eclipse IDE Plug-in Integration Layers

3.1 Relevant Participants of Integration

In Figure 4 the CROSSMINER components that are relevant to the integration of the CROSSMINER Eclipse IDE Plug-in into the CROSSMINER platform are shown. In this document, we are concentrating on the CROSSMINER Eclipse IDE Plug-in.

The CROSSMINER Eclipse IDE Plug-in has two main functionalities: to help the developers query and utilize the data stored in the knowledge base, and to collect user activity data and metrics. Both functionalities require the plug-in to communicate with the CROSSMINER server.

The CROSSMINER server has two main components that are targeted by the requests of the plug-in: the CROSSMINER Knowledge Base and the CROSSMINER Metric Provider. The Knowledge Base is the component that provides recommendations for the CROSSMINER Eclipse IDE Plug-in. A significant part of the communication targets this component. The Metric Provider is in charge of processing (among others) the user activity data collected by the CROSSMINER Eclipse IDE Plug-in. It will also provide aggregated metric values to the Web-based Dashboard.

As can be seen, the plug-in will use some functionalities of the Web-based Dashboard component. Namely, the plug-in itself lacks the functionality to show metrics, instead, it refers to the Web-based Dashboard when the user requires so. Thus, the plug-in should be able to assemble URLs that refer to the proper page of the dashboard server.

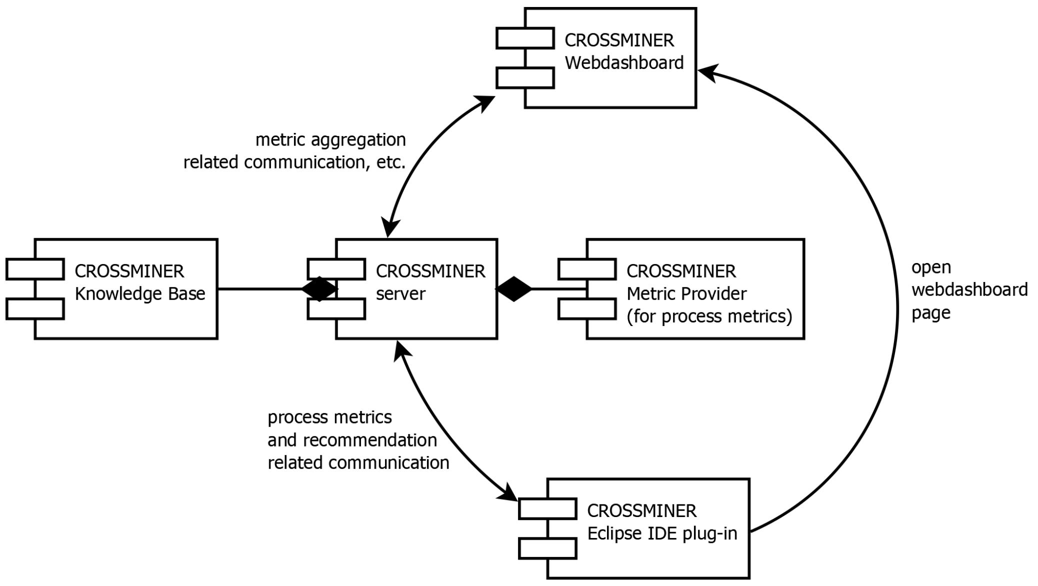

CROSSMINER Webdashboard
metric aggregation related communication, etc.
CROSSMINER Knowledge Base
CROSSMINER server
CROSSMINER Metric Provider (for process metrics)
open webdashboard page
process metrics and recommendation related communication
CROSSMINER Eclipse IDE plug-in

Figure 4: CROSSMINER Integration Related Components

3.2 Abstract Layers of Integration

To ensure the seamless communication between the CROSSMINER server and the CROSSMINER Eclipse IDE Plug-in, it is necessary to use a shared specification of the REST API along with a common serialization method for the shared data models when transmitting data from one component to the other. In the following subsection, we will elaborate on the integration of these two components.

3.2.1 Common REST API Specification

User requirement “U15: API uses a description file for the API specification” of the CROSSMINER project requires the API to be defined in an API description file. To satisfy this requirement, the CROSSMINER REST API is defined in a file using the OpenAPI[1] specification, a common REST API specification standard.

3.2.2 Common REST API Implementation

It is important that the server and the client use the same REST API, and the common REST API specification file enables us to do so. We use this file on the CROSSMINER Eclipse IDE Plug-in side as a basis of the implementation. It is also used on the server side. As it specifies all the available API calls, both the client and the server side implementations use the same REST paths with the right parameters, with the same character encoding, and proper headers and bodies for the queries.

3.2.3 Shared JSON Serialization Logic

The shared object serialization logic is essential for a proper communication between two components. It is a crucial part of the communication layer, which ensures that the two participants can understand each other. JSON is a well established and widely used data description language. Objects described in JSON can be easily converted to programming language objects in most of the object-oriented programming languages, including Java. Thus, we decided to use the JSON object serialization model to transform data into streams, transfer them through the REST API calls and convert them back to objects on the other side.

3.2.4 Shared Communication Data Model Implementation

Another fundamental part of the communication layer is the set of shared Data Models. Distinct components have to use the same objects to represent their states in order to ensure the success of the serialization components. We use the same interface for the data models on both the server and client side, as defined on the OpenAPI specification of the CROSSMINER API.

3.3 Features of CROSSMINER Eclipse IDE Plug-in Currently Relying on the Integration Services

3.3.1 Library Search Feature

The CROSSMINER Eclipse IDE Plug-in provides an interface, where the user can easily search for various projects or libraries that can be used in their project. To initiate a query like that, a short description of a wanted feature or at least a part of the name of the needed project must be given. The returned results will include the projects that have been already analyzed by the CROSSMINER

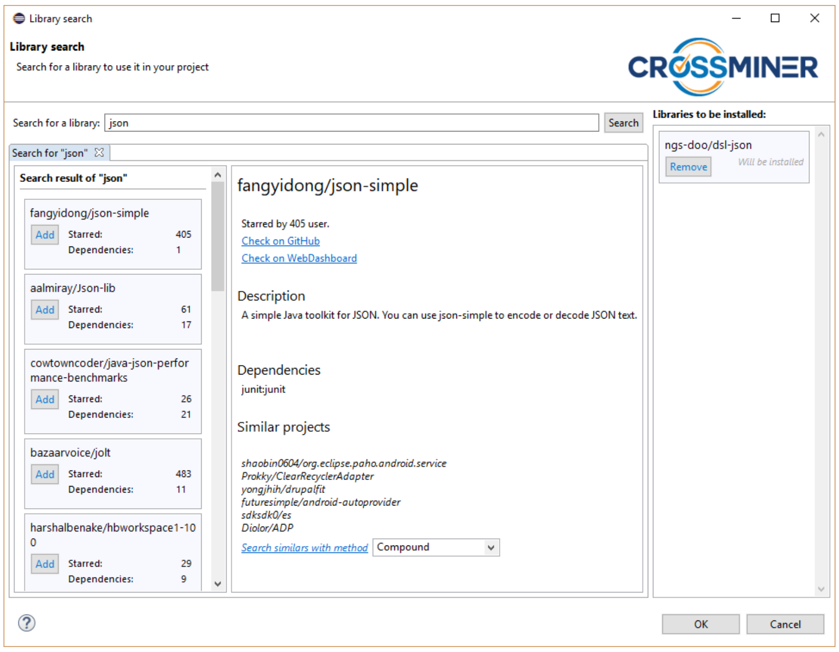
[1] `https://www.openapis.org/`

server and match the given criteria. The user can easily navigate among their searches, by switching between the search tabs, where each of them refers to a particular search or by requesting another list of projects which are similar to a previously selected result, where the method of similarity is determined by the user.

The CROSSMINER server is able to suggest some additional libraries to be used in our project beyond the currently used ones. For this query, the CROSSMINER Eclipse IDE Plug-in gathers the necessary information from our project definition files, like the `pom.xml` file, which consists of definitions of libraries that our project is dependent on among other MAVEN related data. After that, CROSSMINER Eclipse IDE Plug-in sends a particular set of the collected libraries, selected by the user to which the search should base on. The CROSSMINER server responds with another set of libraries that could be used beside the present ones. The user has the ability to easily install these libraries to their project via the provided interface.

All of the received results are processed, so they can be used by the CROSSMINER Eclipse IDE Plug-in in a meaningful way. For example, they can be listed on the graphical user interface in a way, that the user can learn some specific information about the given projects or libraries.

The mentioned interface after a search can be seen on the Figure 5.


Library search

Library search
Search for a library to use it in your project

Search for a library: json

Search for "json"

Search result of "json"

fangyidong/json-simple
Starred: 405
Dependencies: 1

aalmiray/Json-lib
Starred: 61
Dependencies: 17

cowtowncoder/java-json-performance-benchmarks
Starred: 26
Dependencies: 21

bazaarvoice/jolt
Starred: 483
Dependencies: 11

harshalbenake/hbworkspace1-100
Starred: 29
Dependencies: 9

fangyidong/json-simple

Starred by 405 user.
Check on GitHub
Check on WebDashboard

Description
A simple Java toolkit for JSON. You can use json-simple to encode or decode JSON text.

Dependencies
junit:junit

Similar projects

shaobin0604/org.eclipse.paho.android.service
Prokky/ClearRecyclerAdapter
yongjhih/drupalfit
futuresimple/android-autoprovider
sdksdk0/es
Diolor/ADP

Search similars with method: Compound

Libraries to be installed:
ngs-doo/dsl-json
Will be installed

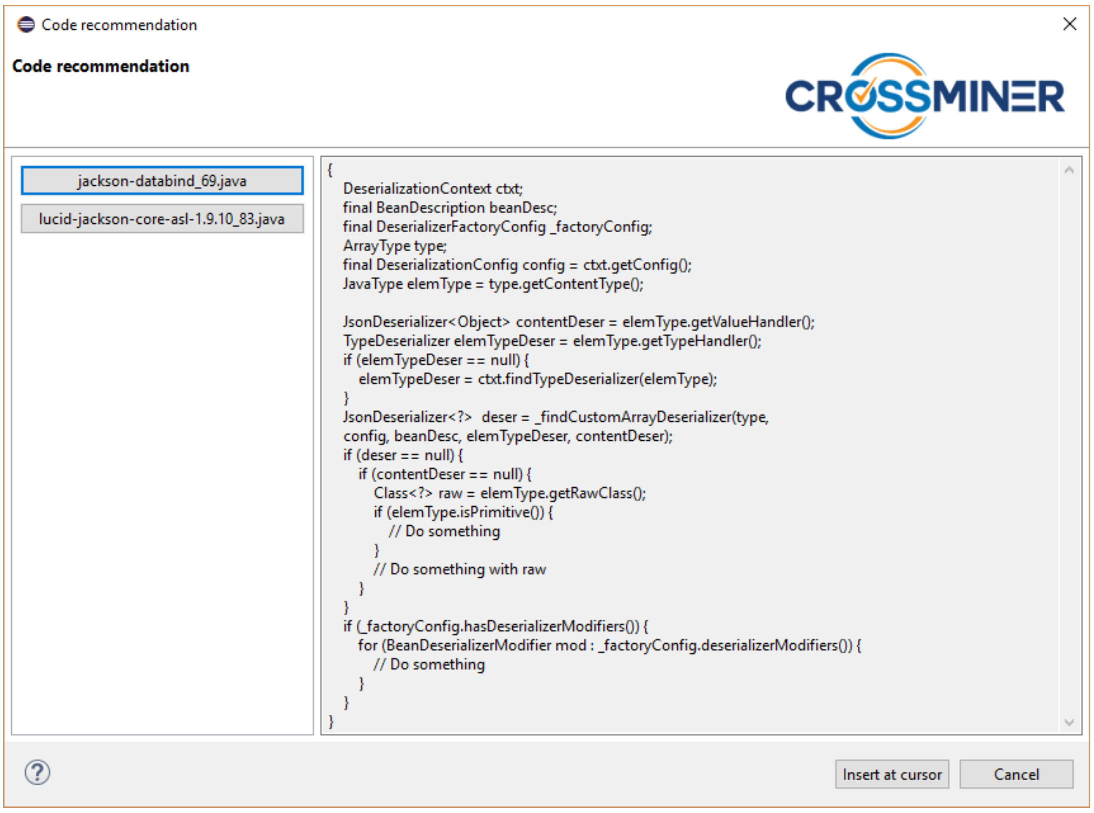
Figure 5: Results of Library Search in CROSSMINER Eclipse IDE Plug-in

3.3.2 Code recommendation feature

The CROSSMINER server is able to provide some source code related recommendations. One of these is the Code Recommendation which is used to provide the user with a pattern of the usage of the given code elements and API calls. To initiate a query for this, the CROSSMINER Eclipse IDE Plug-in sends the server a code snippet, that can be provided by the user via a selection in the Java

Source Code Editor Eclipse View and the server will use the given snippet as a base definition for our query. As a result the CROSSMINER Eclipse IDE Plug-in receives a collection of suggested patterns which can be directly inserted into the source code, however, this is not intended. The user should have an understanding of the given patterns and should use them only as a reference for an implementation. Currently, there is no guarantee for the semantic correctness of these patterns.

An example of the Code Recommendation interface with results can be seen on the Figure 6.



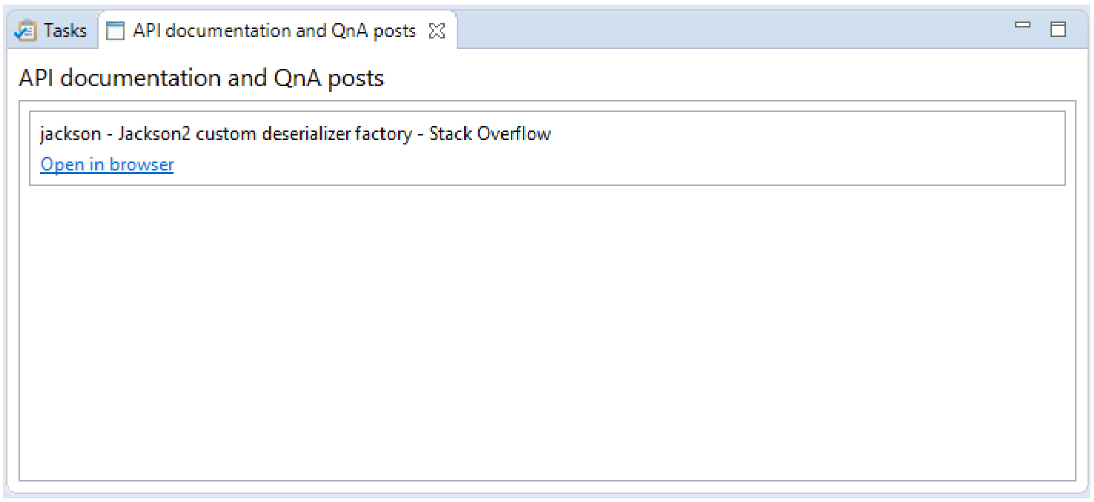
```
{
    DeserializationContext ctxt;
    final BeanDescription beanDesc;
    final DeserializerFactoryConfig _factoryConfig;
    ArrayType type;
    final DeserializationConfig config = ctxt.getConfig();
    JavaType elemType = type.getContentType();

    JsonDeserializer<Object> contentDeser = elemType.getValueHandler();
    TypeDeserializer elemTypeDeser = elemType.getTypeHandler();
    if (elemTypeDeser == null) {
        elemTypeDeser = ctxt.findTypeDeserializer(elemType);
    }
    JsonDeserializer<?>  deser = _findCustomArrayDeserializer(type,
    config, beanDesc, elemTypeDeser, contentDeser);
    if (deser == null) {
        if (contentDeser == null) {
            Class<?> raw = elemType.getRawClass();
            if (elemType.isPrimitive()) {
                // Do something
            }
            // Do something with raw
        }
    }
    if (_factoryConfig.hasDeserializerModifiers()) {
        for (BeanDeserializerModifier mod : _factoryConfig.deserializerModifiers()) {
            // Do something
        }
    }
}
```

Figure 6: Results of Code Recommendation in CROSSMINER Eclipse IDE Plug-in

3.3.3 API documentation and Q&A posts

The CROSSMINER Eclipse IDE Plug-in can help the user to get a better understanding of a function or API call by showing related Q&A posts or documentation about the given call. The CROSSMINER server receives an arbitrary source code chunk and responds with a set of recommended website URLs. Then, these results are displayed on the dedicated API Documentation and Q&A Posts view, where the users can browse them or request to open them in their default browser.

This interface can be seen on the Figure 7.



API documentation and QnA posts

jackson - Jackson2 custom deserializer factory - Stack Overflow
Open in browser

Figure 7: Results of a request for Q&A Posts and API Documentation in CROSSMINER Eclipse IDE Plug-in

3.3.4 Integration Related Settings

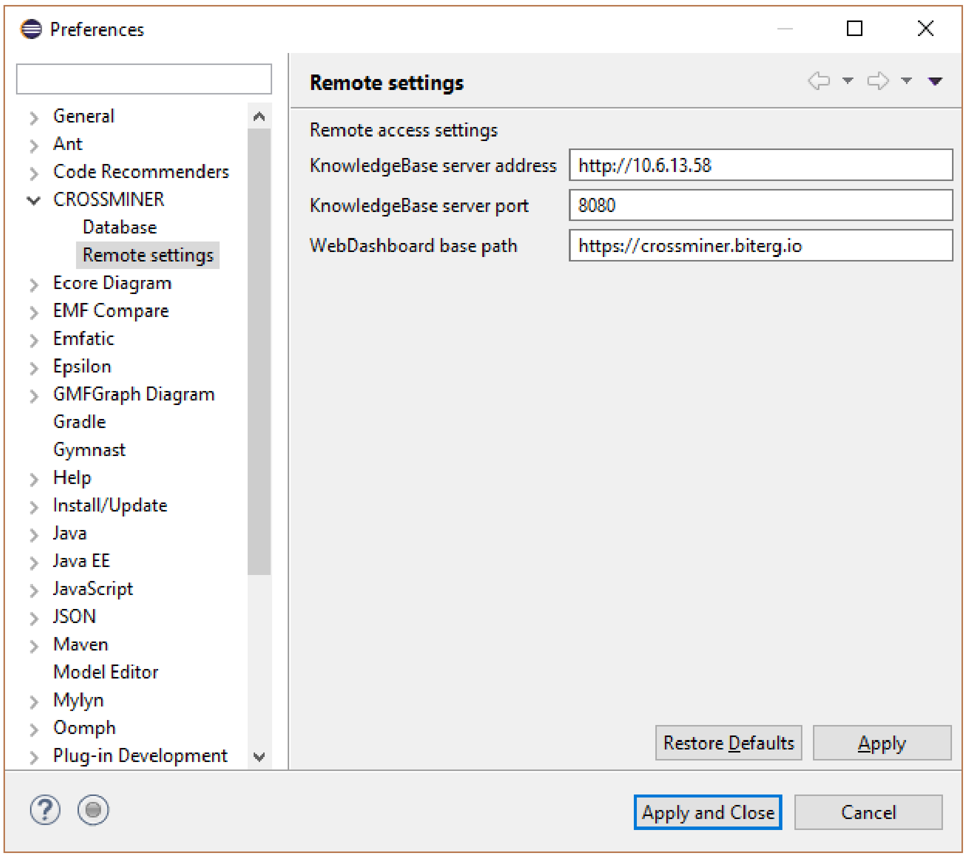
The CROSSMINER Eclipse IDE Plug-in supports the integration and the communication of the CROSSMINER server and the client by providing a settings interface for the user to give her the ability to customize the properties of the connection. This interface can be reached by opening the regular Eclipse Preferences page from the Eclipse IDE Window menu and going into the CROSSMINER category's Remote Settings preference page. This settings interface can be seen on the Figure 8.

The user can change the address and port of the CROSSMINER server where the request is going to be sent to and the address of the Web-based Dashboard, which will be used by the Library Search feature to open the related page of a project. The CROSSMINER Eclipse IDE Plug-in does not require to be restarted after setting a new value for these properties, simply apply the changes and the next query will be sent using the newly set properties.

3.4 Technical details

As it is shown on Figure 9, we use the shared OpenAPI specification as a bridge between the server and the client side. At the current stage, the OpenAPI specification is provided and maintained by the CROSSMINER server.

We use the Swagger Codegen (it will be described in Section 3.4.2), which can generate an implementation of the REST API handling client methods based on the description in the OpenAPI specification. This generated client can be used in the CROSSMINER Eclipse IDE Plug-in to provide a single façade over the communication between the components.



Preferences

- General
- Ant
- Code Recommenders
- CROSSMINER
  - Database
  - Remote settings
- Ecore Diagram
- EMF Compare
- Emfatic
- Epsilon
- GMFGraph Diagram
- Gradle
- Gymnast
- Help
- Install/Update
- Java
- Java EE
- JavaScript
- JSON
- Maven
- Model Editor
- Mylyn
- Oomph
- Plug-in Development

Remote settings

Remote access settings

KnowledgeBase server address	http://10.6.13.58
KnowledgeBase server port	8080
WebDashboard base path	https://crossminer.biterg.io

Figure 8: Remote Settings preferences page in CROSSMINER Eclipse IDE Plug-in

3.4.1 Using OpenAPI standard as a Common REST API Specification

OpenAPI is a standard in the industry for API documentation and specification. It is widely used by different companies all around the world. In the past few years, it has become one of the best ways to describe an API.

One of its main features is that one can create a very detailed description of an API call (including its parameters and return value). Type constraints, the method of requests, or the data structures received or sent all could be defined.

We use this standard as a common descriptor to specify the CROSSMINER server's behavior and to implement our client. The current CROSSMINER REST API specification file contains a detailed description of the available CROSSMINER server API calls and transferred objects. Listing 1 shows an example, namely the Artifact Search API call path.

```
...
"/api/artifacts/search/{artifact_query}": {
    "get": {
        "produces": [ "application/json" ],
        "parameters": [
                {
                    "name": "artifact_query",
                    "in": "path",
```

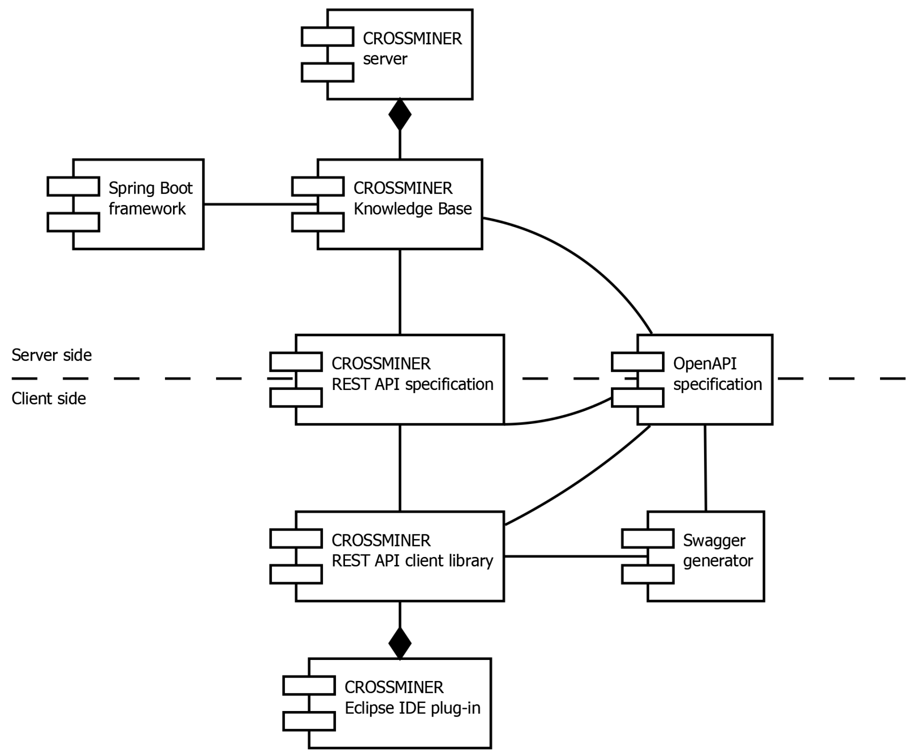

CROSSMINER server
Spring Boot framework
CROSSMINER Knowledge Base
Server side
Client side
CROSSMINER REST API specification
OpenAPI specification
CROSSMINER REST API client library
Swagger generator
CROSSMINER Eclipse IDE plug-in

Figure 9: Current State of CROSSMINER Eclipse IDE Plug-in Integration Related Components

```
                        "description": "artifact_query",
                        "required": true,
                        "type": "string"
                    },
                    ...
            ],
            "responses": {
                "200": {
                    "description": "OK",
                    "schema": {
                        "type": "array",
                        "items": {
                            "$ref": "#/definitions/Artifact"
                        }
                    }
                },
                ...
            }
            ...
        }
}
...
```

Listing 1: Example snippet from the CROSSMINER server's OpenAPI specification

The defined API calls could be grouped into the following major categories according to their functionality. These calls use several custom-made data structures during the communication, which are also defined using OpenAPI standard and could be grouped.

3.4.1.1 Multiply Entries Retrieving Calls There are a lot of requests which respond with a set or collection of elements not just with a single entity. This behavior is well described in the OpenAPI specification, and an example for this can be seen at the Listing 2.

```
...
"responses": {
        "200": {
                "description": "OK",
                "schema": {
                        "type": "array",
                        "items": {
                                "$ref": "#/definitions/Artifact"
                        }
                }
        },
        ...
}
...
```

Listing 2: Example for multiple entries result value snippet from the CROSSMINER server's OpenAPI specification

With this definition, we can ensure that the client and the server will understand each other when a collection is sent over the communication channels. It is essential to prepare both sides for this scenario.

3.4.1.2 Single Entry Retrieving Calls The other major group of calls is the ones that respond with a single entry. These are for example the recommendation queries. They respond with a single Recommendation element which is described at the definition of the API call. An example of this can be seen in Listing 3.

```
...
"200": {
        "description": "OK",
        "schema": {
                "$ref": "#/definitions/Recommendation"
        }
},
...
```

Listing 3: Example for Recommendation element result value snippet from the CROSSMINER server's OpenAPI specification

3.4.1.3 Data Models The definition of the previously mentioned calls heavily relies on the definition of model elements. The definition of these models is also included in the CROSSMINER API specification since both the client and the server have to know what kind of data is about to be transferred. The OpenAPI standard provides a precise way to describe these models. An example of these can be seen at the Listing 4.

```
...
"Recommendation": {
        "type": "object",
        "properties": {
                "recommendationItems": {
                "type": "array",
                        "items": {
                                "$ref": "#/definitions/RecommendationItem"
                        }
                }
        },
        "title": "Recommendation"
},
"RecommendationItem": {
        "type": "object",
        "properties": {
                "apiCallRecommendation": {
                        "$ref": "#/definitions/ApiCallResult"
                },
                "apiDocumentationLink": {
                        "type": "string"
                },
                "artifact": {
                        "$ref": "#/definitions/Artifact"
                },
                "recommendationType": {
                        "type": "string"
                },
                "recommendedLibrary": {
                        "$ref": "#/definitions/RecommendedLibrary"
                },
                "relatedTo": {
                        "type": "object"
                },
                "significance": {
                        "type": "number",
                        "format": "double"
                }
        },
        "title": "RecommendationItem"
},
...
```

Listing 4: Example for Recommendation element model definition snippet from the CROSSMINER server's OpenAPI specification

These defined models then can be used several times, since they can be referenced by other model elements even in a hierarchical way.

3.4.2 Automatic Generation of REST API library for CROSSMINER Eclipse IDE Plug-in

Swagger Codegen[2] is an open source solution for Java REST API client generation from OpenAPI specification. It is able to create an implementation of the client side of the communication from the API specification given in a JSON file according to the OpenAPI format.

In our case, Swagger Codegen generates a Java method for each API call, which can then be easily invoked. It also generates an implementation for every object defined by the specification; these can then be used in the communication either as input parameters or return values. These objects can be hierarchically nested into each other, which enables the definition of complex data structures. The ability to generate shared data models from the specification allows us to use the same objects on both sides of the communication.

A code snippet that uses the *ArtifactsRestControllerApi.getProjectUsingGET* CROSSMINER API method is shown in Listing 5. The implementation of the method was automatically generated by the Swagger Codegen, as well as the implementation of the return type *io.swagger.client.model.Artifact* which represents a project, that has been already analyzed by the CROSSMINER server.

A detailed description of the Swagger Codegen can be found on its website: `https://github.com/swagger-api/swagger-codegen`.

```
String queryString = inputField.getText();

ArtifactsRestControllerApi artifactsRestControllerApi =
        new ArtifactsRestControllerApi();

List<Artifact> artifactList = artifactsRestControllerApi
        .getProjectUsingGET(queryString, null, null, null);

return artifactList;
```

Listing 5: The use of a Swagger Codegen-generated CROSSMINER API method

3.4.3 Description of the Generated API Client's Functionality

Inside the generated client, the earlier mentioned *ArtifactsRestControllerApi.getProjectUsingGET* CROSSMINER API method is built upon the *ArtifactsRestControllerApi.getProjectUsingGETCall* method, which constructs the desired API call to be initiated later. This method takes care of the passed parameters to be included in the query in a proper way and attaches a progress listener to the call to be notified when a response arrives from the CROSSMINER server.

A simplified version of this is illustrated in the 6 listing.

```
public Call getProjectUsingGETCall(String artifactQuery,
        Object page, ..., ProgressListener progressListener) {

        Object localVarPostBody = null;

        ...
```

[2] `https://swagger.io/tools/swagger-codegen/`

```
        String localVarPath = "/api/artifacts/search/{artifact_query}"
                .replaceAll("\\{" + "artifact_query" + "\\}",
                apiClient.escapeString(artifactQuery.toString()));

        ...

        if (page != null)
                localVarQueryParams.addAll(
                        apiClient.parameterToPair("page", page));

        ...

        final String[] localVarAccepts = {
                "application/json"
        };

        ...

        final String[] localVarContentTypes = {

        };

        ...


        if(progressListener != null) {
                apiClient.getHttpClient().networkInterceptors()
                        .add(new Interceptor() {
                        @Override
                        public Response intercept(Chain chain) {

                                ...

                        }
                });
        }

        ...

        return apiClient
                .buildCall(localVarPath, "GET", localVarQueryParams, ...);
}
```

Listing 6: Building a call to retrieve a project list from the Server

Other methods like the *RecommenderRestControllerApi.getApiCallRecommendationUsingPOST* CROSSMINER API method also have a call builder under the hood. This request is going to transfer its parameters to the CROSSMINER server by POST method, which requires them to be encapsulated in the body of the request, instead of just being passed in the path of it. The *RecommenderRestControllerApi.getApiCallRecommendationUsingPOSTCall* method provides this functionality while having a lot of similarities with the previously discussed method.

A simplified version of this is illustrated in Listing 7.

```
public Call getApiCallRecommendationUsingPOSTCall(Query query,
        ProgressListener progressListener, ...) {

        Object localVarPostBody = query;

        String localVarPath =
                "/api/recommendation/recommended_API_call";

        ...

        final String[] localVarAccepts = {
                "application/json"
        };

        ...

        final String[] localVarContentTypes = {
                "application/json"
        };

        ...

        if(progressListener != null) {
                apiClient.getHttpClient().networkInterceptors()
                        .add(new Interceptor() {
                        @Override
                        public Response intercept(Chain chain) {

                                ...

                        }
                });
        }

        ...

        return apiClient.buildCall(localVarPath, "POST", ...);
}
```

Listing 7: Building a call to retrieve a code recommendation from the Server

These kinds of methods usually rely on some specialized compound data structures to encapsulate their results. For example consider the sample in Listing 8, which is used to wrap up multiple RecommendationItems while being transferred over the network.

```
public class Recommendation {

        @SerializedName("recommendationItems")
        private List<RecommendationItem> recommendationItems = null;

        ...

        public List<RecommendationItem> getRecommendationItems() {
                return recommendationItems;
        }

        public void setRecommendationItems(List<RecommendationItem>
                recommendationItems) {
                this.recommendationItems = recommendationItems;
        }

        ...

        @Override
        public String toString() {
                ...
        }
}
```

Listing 8: Recommendation Datatype used during Communication

There are some other data structures that are used as parameters for these methods. These are also specified by the CROSSMINER API and generated on the client side by the Swagger Codegen. Consider the code snippet in Listing 9, which shows a simplified version of the implementation of the Query parameter data structure. It is usually used to describe the current state of the project the user is working on, and sent to the CROSSMINER server, so it can rely on while suggesting some recommendations.

```
public class Query {

        @SerializedName("annotations")
        private List<String> annotations = null;

        @SerializedName("classDependencies")
        private List<Dependency> classDependencies = null;

        ...

        public List<String> getAnnotations() {
                return annotations;
        }

        public void setAnnotations(List<String> annotations) {
                this.annotations = annotations;
        }


        ...

        public List<Dependency> getClassDependencies() {
                return classDependencies;
        }

        public void setClassDependencies(List<Dependency>
                classDependencies) {
                this.classDependencies = classDependencies;
        }
        ...

        @Override
        public String toString() {
                ...
        }
}
```

Listing 9: Query Datatype used during Communication

3.4.4 Usage of the Generated API Client's Functionality

For concrete usage of the functionalities elaborated in the previous section consider Listing 5 and Listing 10. In the latter we use the Query parameter data structure mentioned in Listing 9 to encapsulate the selected source code sample. After the request, the received answer is processed, so the recommended API calls are collected into a list and passed back to the CROSSMINER Eclipse IDE Plug-in. This functionality is used for the Code Recommendation feature described in Section 3.3.2.

```
String currentMethodCode = activeJavaEditor.getSelectedText();

RecommenderRestControllerApi recommenderRestController =
        new RecommenderRestControllerApi();

Query query = new Query();
query.setCurrentMethodCode(currentMethodCode);

Recommendation recommendation = recommenderRestController
        .getApiCallRecommendationUsingPOST(query);

List<ApiCallResult> results =
        recommendation.getRecommendationItems()
        .stream().map(i -> i.getApiCallRecommendation())
        .collect(Collectors.toList());

return results;
```

Listing 10: The use of getApiCallRecommendationUsingPOST(), a Swagger Codegen-generated CROSSMINER API method

3.5 Integration with the Continuous Integration System of CROSSMINER

3.5.1 Using Tycho to Build the CROSSMINER Eclipse IDE Plug-in

We have adapted the project of CROSSMINER Eclipse IDE Plug-in to be compatible with the Tycho Build System. Tycho is a Maven plug-in for building Eclipse Plug-ins. Currently, this system provides the functionality to build the update sites of the CROSSMINER Eclipse IDE Plug-in, which can be used to easily install it into an Eclipse IDE.

Because of the well-defined building process of Maven, we were able to inject the inclusion of dependencies used by the CROSSMINER Eclipse IDE Plug-in into the building process. To achieve this, there is a *org.eclipse.scava.plugin.dependencies* project, which is responsible to gather all the required Maven dependencies used by the plug-in. Then, during the building process, this project is assembled into a single JAR file and included into the project folder of CROSSMINER Eclipse IDE Plug-in. This way we can choose third-party libraries for the project from a wider range.

Another significant benefit of this system is that it provides the ability to effortlessly include the whole building process into an Automatic Integration service. Because Maven forms the basis of this system, any automatic building service which is compatible with it can be used over our system.

3.5.2 Framework for Automatic Update Site Deployment

To help the process of building Update sites for CROSSMINER Eclipse IDE Plug-in, we have developed several scripts and modules that can automate the repetitive phases of the deployment.

3.5.2.1 OpenAPI specification updater This module can help the development by fetching the newest OpenAPI specification of the CROSSMINER API directly from the CROSSMINER server and saving it into a file in the building system for a later use.

3.5.2.2 Client builder This script is responsible for generating a REST client with the formerly discussed Swagger Codegen generator and compiling it with the Maven build system. It relies on the OpenAPI specification produced by the previously described module in Paragraph 3.5.2.1. As a result of running this script, the generated client will be installed into our local Maven repository.

3.5.2.3 Dependency builder The building of *org.eclipse.scava.plugin.dependencies* project described in Section 3.5.1 is done by this module. Its only goal is to build the dependency-project and inject it into the CROSSMINER Eclipse IDE Plug-in's project folder to solve the dependencies.

3.5.2.4 Update site builder The purpose of this script is to launch the whole Update Site building process, which, as a result, will produce a completely standalone package of the CROSSMINER Eclipse IDE Plug-in that can be shared even over the network and provides the ability for the users to easily install it into her Eclipse IDE and to start using it right away.

4 Further Steps

As noted earlier, CROSSMINER Eclipse IDE Plug-in represents one of the front-ends of CROSSMINER, thus almost all features are related to integration to some degree. It does not mean that all of these features are implemented at this stage, but the integration service layer is ready to be used during further development.

In the future, our main task will be to support all the provided functionality of the CROSSMINER server in the CROSSMINER Eclipse IDE Plug-in. This may need further conciliation related to the API calls to make sure the server can support every expected functionality.

For example, we also plan to implement some additional functionality, like error handling in the client side, to make it more flexible against different operational conditions. Functionality that enables the client to provide feedback on the received recommendations will also be implemented. We also would like to further polish the UI for the CROSSMINER Eclipse IDE Plug-in.

5 Conclusion

In this deliverable, an interim version of the IDE plug-in was prepared that uses the final version of the integration services, i.e. communicates with the server through a well defined final interface.

In the following subsections we show how the technical (Section 5.1) and use case (Section 5.2) requirements related to the CROSSMINER Eclipse IDE Plug-in Integration Services and defined in the Project Requirements document (deliverable D1.1) are covered by the interim version of the plug-in. In the last column of the tables an empty circle (○) denotes that the requirement is minimally (or not) covered, a half-filled circle (◐) denotes that it is only partially covered, and a filled circle (●) denotes that it is mostly (or fully) covered.

5.1 Technical requirements

D74	The IDE shall provide a settings interface to the user, where the different properties of the CROSSMINER IDE plugin (like server address and port, global settings for recommendation queries, etc.) can be checked and changed. So the user can configure the plugin.	SHALL	●
D98	The IDE shall be able to send developer activity data (as controlled by the user settings) to the CROSSMINER server.	SHALL	●
D138	The CROSSMINER REST API shall use UTF-8 encoding for all kind of data sent or received in text mode.	SHALL	●

5.2 Use case requirements

U12	Able to obtain the API results in JSON format	SHALL	●
U13	Able to use the API over REST	SHALL	●
U15	API uses a description file for the API specification	SHALL	●
U18	API is utilized by all UIs (dashboard, IDE plugin)	SHALL	●

5.3 Next development steps

We will continue with the implementation of the more general features loosely related to integration services, which are not fully implemented in this deliverable version.