D5.5 Methodology Tool Support

The DECODER project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 824231

Project Acronym	DECODER
Project Title	DEveloper COmpanion for Documented and annotatEd code Reference
Project Number	824231
Project Type	Research and Innovation Action (RIA)
Thematic Priority	ICT-16-2018 Software Technologies
Start Date of Project	January 1st, 2019
Duration of Project	36 months

Deliverable type	DEM
Deliverable ref. number	DS-05-824231 / D5.5 / V1.0
Work Package	**WP5**
Task	**T5.3**
Due Date	September 2021 – M33
Submission Date	October 6th, 2021
Dissemination Level	PU

Responsible organisation:	UPV	
Author(s)	Nacho Mansanet	UPV
	Victoria Torres	UPV
	Miriam Gil	UPV
Reviewer(s)	Pablo de Castro	TREE
	Gilles Mouchard	CEA
	Frank Vedrine	CEA
	Pierre-Yves Gibello	OW2

Version history

0.1		Draft	UPV
0.2		First Complete Version Reviewed	UPV
0.3		Changes applied due to process engine invocations procedure	UPV
1.0		Final version	UPV

Table of Contents

List of Figures

Executive summary

This deliverable accompanies a software that represents the tool support for the methodology of the DECODER platform. This software instantiates the methodology proposed in D5.4 and previous and represents a first version of the integration mechanisms of all the tools being developed in DECODER and the Graphical User Interface (GUI). Considering that this set of tools may be extended during the project and also even once the project finalizes in Dec. 2021, the orchestration tools proposed at this point may vary to accommodate any change in this respect.

Contributing tasks of this WP	T5.3
Related deliverables of this WP	D5.2, D5.3, D5.4
Input from other WP(s)	WP1, WP2, WP3, WP4
Output to other WP(s)	WP6

Keywords
Tool orchestration, software development methodology, Tools Integration.

Acronyms and Abbreviations

AMT	Artefact Management Tools
APT	Artefact Processing Tools
BPMN	Business Process Model and Notation
DOM	Document Object Model
GUI	Graphical User Interface
IDE	Integrated Development Environment
JSON	Javascript Object Notation
SRS	Software requirements specification
SUT	System Under Test
PKM	Persistent Knowledge Monitor
PU	PKM Utilities
UML	Unified Modeling Language
PE	Process Engine
API	Application Programming Interfaces

1 Introduction

One of the main objectives stated in the DECODER project is to “significantly increase the software development and maintenance efficiency”. To achieve such goal, we propose changing how software projects are carried out by generating detailed materials at all stages of the development process, from early stages where requirements are set, to late stages where the different project artefacts need to be maintained. For example, precise requirements will lead to a quick understanding of the problem, precise specifications will leave little ambiguities and can be referred to at any later stage, precise designs will leave little room for discussions as to the structure of the application and its communications, precise bugs will allow engineers to quickly identify defects, etc. To this end, we propose a methodology that assists the stakeholder with (1) an instantaneous access to project documentation, abstract models and verification processes, (2) an access to a virtual expert in order to produce code that is functionally correct (including the parameterization of the underlying libraries), free of run-time errors, and with a control on memory space and execution time, (3) tools that verify requirements, and (4) tools that produce user documentation conformant to requirements.

1.1 Purpose

This deliverable instantiates the tools and integration mechanisms described in D5.4 and previous reports, and provides a brief description of the software developed to integrate all the tools present in the DECODER platform with the GUI.

Although the DoA mentioned Eclipse ecosystem as the basis for the tool support, given the proliferation of web-based tools, it was decided to move all the tool support and integrate it with the web GUI. This deviation was indicated in a previous report and agreed by the project consortium.

What we present in this report is a central piece of software known as Process Engine (PE), which acts as intermediary between the DECODER set of tools, the GUI and the PKM (see Figure 1).

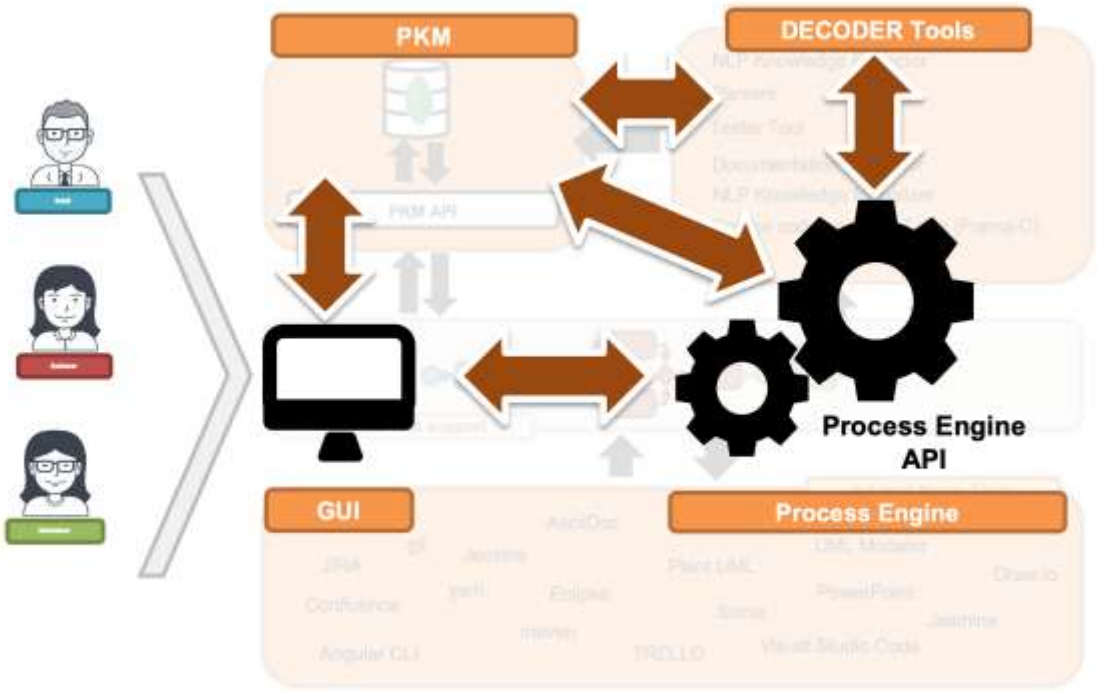

Figure 1: Interaction between the Process Engine, GUI and DECODER set of tools

Specifically, this deliverable is focused on describing the API proposed for the PE to ease the integration with the tools and the DECODER GUI. We divided it in 2 different parts:

1. *Tools:* The Tools API is the one proposed to retrieve all the information needed to get the list of available tools (depending on the phase or task where the development process is) and the invocation mechanisms for each tool (Tools API descriptions).
2. *Invocations:* The invocations API is used to set and retrieve all the tools invocations information and the results obtained for each invocation.

1.2 Overall organization of the document

The remaining of the document has been organized as follows. Section 2 presents the main contribution of this deliverable, i.e., the two proposed APIs for the PE. Section 3 explains the workflow designed to create and check new tool invocations using the proposed methodology and PE to support the interaction between the Tools and the GUI. Finally, Section 4 summarizes the work developed in this deliverable and also draws and discusses some conclusions.

2 Process Engine API

In this section we explain in detail the API proposed for the PE to achieve the integration with the set of tools being developed in DECODER and the GUI developed.

2.1 Tools API

As mentioned previously, the tools APIs descriptions should be stored in the PKM to ease the extensibility of the platform. In this way, and to provide a manner for the GUI to make the tools available to the user, we have designed and developed an API to allow the GUI to retrieve all the information needed about the tools to build the invocations that are required at each different stage of the development process. Next subsections describe the two different endpoints of the Tools API.

2.1.1 Tool API

```
GET /decoder/pe/tools/<dbName>/<toolID>
```

This endpoint retrieves the information about a tool which is specified as **<toolID>** in the path and that is available in the project **<dbName>**.

The answer to this endpoint contains data structured as follows:

- **ToolID**
- **Tool Name** to be displayed
- **Tool Description** to be used by the GUI
- **Phases** where the tool can be launched
- **Tasks** where the tool can be launched
- **Server** where the tool is installed
- The **Endpoint** description that contains:
    - The server's **Path** where the tool attends
    - The **method** used to invoke the tool
    - The **Query Parameters**
    - The **Path Parameters**
    - The **Request Body** description

```
{
    "toolID": "javaParser",
    "toolName": "Java Parser",
    "description": "Generates one or several JSON file(s) from the
artefactID according to the generate parameter",
    "phases": [*],
    "tasks": [*],
    "server":"http://localhost:5001/",
```

```
    "endpoint":{
        "path":"/decoder/...",
        "method":"get",
        "queryParameters":[
            {
                "name": "generate",
                "description": "Generate JSON files just with a specific
type of annotations. If ommited, all files will be generated",
                "required": true|false,
                "type": "string|number|Boolean...",
                "allowedValues: [*]
            }
        ],
        "pathFields":[
            {
                "name": "dbName",
                "description": "The DB name where to find the source file",
                "required": true,
                "type": "string",
                "isDBName": true
            },
            {
                "name": "sourceFileName",
                "description": "The source file relative path",
                "required": true,
                "type": "string",
                "artifactDesc": {
                    "type:": "code",
                    "allowedExtensions": [*]
                }
            }
        ]
    }
}
```

The existing tools specifications are described and available in the project repository at https://gitlab.ow2.org/decoder.

2.1.2 Tools List API

```
GET /decoder/pe/tools/<dbName>?processTask=*&processPhase=*
```

This endpoint retrieves the tools list available in the project specified as **<dbName>** in the path. It accepts two optional parameters that allow the user to filter this list according to the task (processTask parameter) or the phase (processPhase parameter) where the process is at any moment.

The response of this endpoint is a JSON list of items like the one obtained in the previous endpoint as shown in the next listing.

```
[
    {
        "toolID": "javaParser",
        "toolName": "Java Parser",
        "phases": [*],
        "tasks": [*],
        "server": "...",
        "endpoint": "{...}"
    },
    {
        "toolID": "framac",
        "toolName": "Frama C",
        "phases": [*],
        "tasks": [*],
        "server": "...",
        "endpoint": "{...}"
    }
]
```

2.2 Invocations API

The Invocations API has two different purposes: (1) to create, store, and update new tools invocations, and (2) to retrieve all the invocations information. Next subsections describe the two different endpoints of the Invocations API.

2.2.1 Invocations List API

To retrieve the invocations list, a GET request method should be used as follows:

```
GET /decoder/pe/invocations/<dbName>?invocationStatus=*
```

This endpoint retrieves the invocations list available in the project specified as **<dbName>** In the path. It accepts an optional parameter **invocationStatus** that allows the user to filter the retrieved list attending to the status of the invocation (PENDING | RUNNING | COMPLETED | FAILED).

To create a new invocation, a POST request method should be used as follows:

```
POST /decoder/pe/invocations/<dbName>
```

This endpoint creates a new invocation in the project specified as **<dbName>** in the path.

The request body should contain all the information needed to invoke the tool such as:

- **Tool**
- **User**
- Tool Invocation **Configuration**[1]

```
{
    "toolID": "javaParser",
    "user":"jsmith",
    "invocationConfiguration":{
        "generate":"all",
       "dbName":"myThaiStar",
        "sourceFileName":"code/rawsourcecode/mythaistar/java/mtsj/api/src/main/java/com/devonfw/application/mtsj/bookingmanagement/service/api/rest/BookingmanagementRestService.java"
    }
}
```

The answer to this call contains the new **Invocation ID**.

```
{
    "invocationID":"734128cjajdsasd"
}
```

2.2.2 Invocation API

To retrieve the invocation information, GET request method should be used

```
GET /decoder/pe/invocations/<dbName>/<invocationID>
```

This endpoint retrieves the information about an invocation **<invocationID>** available in the project **<dbName>**.

The answer to this endpoint contains data structured as follows:

- **Invocation ID**: which identifies uniquely a tool invocation performed during the development process.
- **Tool**: The tool (toolID) invoked
- **Tool Configuration**: The tool configuration sent within the tool invocation
- **User**: The user who has sent the tool invocation

[1] Only artifacts that are going to be processed by the invoked tool will be specified in this field (Compilation Units, Classes, annotations files, etc.)

- **Timestamps**: Timestamps when the invocation is (1) requested, (2) started and (3) finalized
- **Invocation results**: The results generated by the tool invocation
- **Invocation status**: The current status of the tool invocation (Pending, Running, Completed or Failed)

Next listing illustrates this answer with real data related to the invocation of the javaParser tool.


```
{
    "invocationID":"734128cjajdsasd",
    "tool":"javaParser",
    "user":"jsmith",
    "timestampRequest":"20210614_082526",
    "timestampStart":"20210614_082527",
    "timestampCompleted":"20210614_082643",
    "invocationConfiguration":{
        "generate":"all",
       "dbName":"myThaiStar",
        "sourceFileName":"code/rawsourcecode/mythaistar/java/mtsj/api/src/main/java/com/devonfw/application/mtsj/bookingmanagement/service/api/rest/BookingmanagementRestService.java"
    }
    "invocationStatus":"COMPLETED",
    "invocationResults":[
        {
        "path":"/code/java/sourcecode/mythaistar/java/mtsj/core/src/main/java/com/devonfw/application/mtsj/usermanagement/logic/impl/UsermanagementImpl.java",
        "type":"code"

        },
        {
"path":"/code/java/comments/mythaistar/java/mtsj/core/src/main/java/com/devonfw/application/mtsj/usermanagement/logic/impl/UsermanagementImpl.java",
        "type":"code"

        },
        {
"path":"/code/java/annotations/mythaistar/java/mtsj/core/src/main/java/com/devonfw/application/mtsj/usermanagement/logic/impl/UsermanagementImpl.java",
        "type":"code"

        }
    ]
}
```


To update the invocations information, a PUT request method should be used as follows:

PUT /decoder/pe/invocations/<dbName>/<invocationID>

This endpoint allows to update the information regarding to an invocation **<invocationID>** available in the project **<dbName>**. The fields that can be updated are:

- Invocation **Status**
- Invocation **Result**


```
{
    "invocationStatus":"COMPLETED",
    "invocationResults":[
        {
"path":"/code/java/sourcecode/mythaistar/java/mtsj/core/src/main/java/com/d
evonfw/application/mtsj/usermanagement/logic/impl/UsermanagementImpl.java",
        "type":"code"
        },{
"path":"/code/java/comments/mythaistar/java/mtsj/core/src/main/java/com/dev
onfw/application/mtsj/usermanagement/logic/impl/UsermanagementImpl.java",
        "type":"code"
        },{
"path":"/code/java/annotations/mythaistar/java/mtsj/core/src/main/java/com/
devonfw/application/mtsj/usermanagement/logic/impl/UsermanagementImpl.java"
,
        "type":"code"
        }
    ]
}
```


The answer to this call contains the same information as the previous one but with the fields updated with the new values.

3 Supporting the Interaction

This section explains the process that should be followed to invoke the available tools in DECODER through the GUI by using the PE. Figure 2 shows the intermediary role played by the PE between the GUI, the DECODER tools and the PKM as well as the invocation order followed to support the execution of the tools selected at each stage of the process. This process is transparent for the user but is performed between the GUI and the PE in an automatic way.

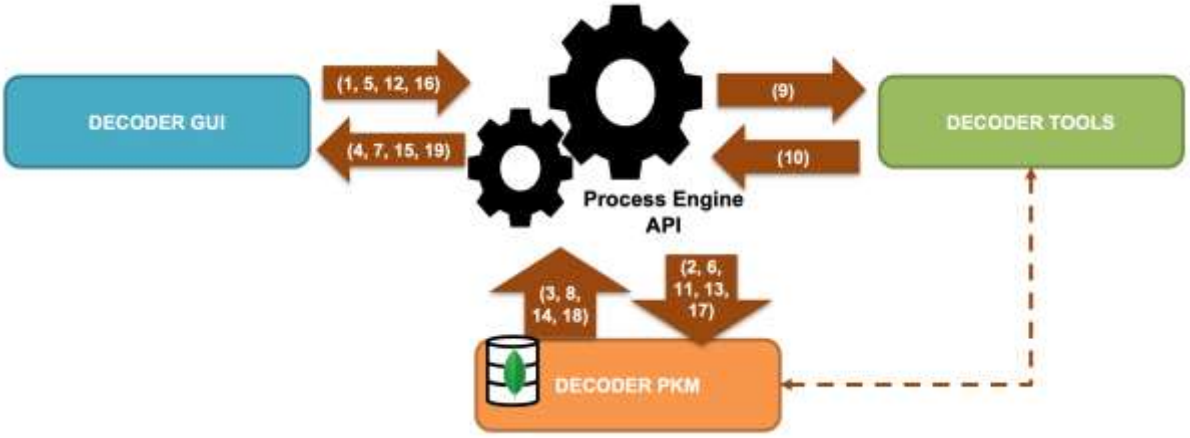

Figure 2: DECODER Tool Invocation Process

This process is performed in 3 phases:

- Phase 1: Retrieving Tool Invocation Information.
- Phase 2: Performing Tool Invocation.
- Phase 3: Retrieving Tool Invocation Results.

Next subsections detail the sequence of invocations designed to achieve the goals stated by each of these three phases.

3.1 Phase 1: Retrieving the Tool Invocation information

Prior to any tool invocation, the GUI should provide the user with detailed information about the set of tools that are available in DECODER at each stage of the development process and how these can be invoked (e.g., which parameters are required, or which configurations can be defined over them). This information is stored in the PKM and delivered by the PE as it is shown in Figure 3.

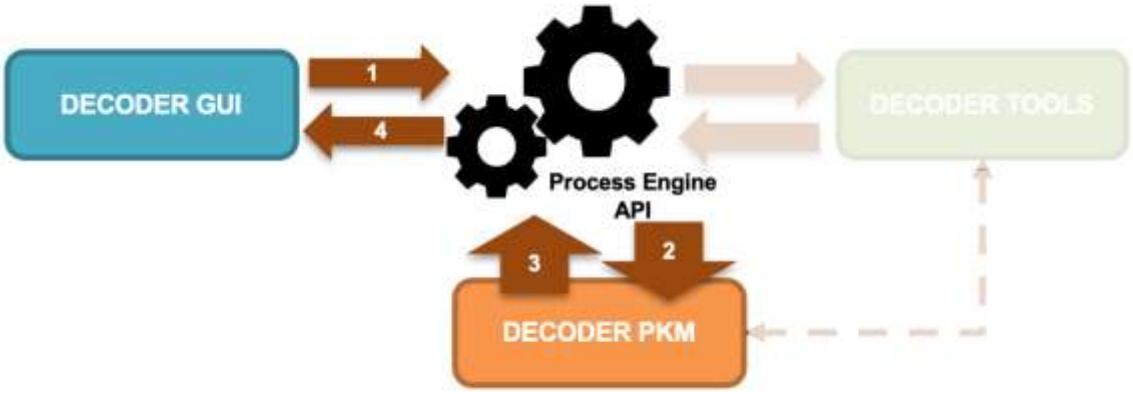

Figure 3: DECODER Tool Invocation Process - Phase 1

Specifically, the four steps performed in this phase are the following:

(1) **GUI** asks **PE** for **Tool's** specifications
(2, 3) **PE** searches in the **PKM** for **Tools** specifications
(4) **PE** answers **GUI** with **Tools** specifications

3.2 Phase 2: Performing the Tool Invocation

Once the user decides which tools are going to be invoked and how, this information is sent to the PE with two purposes, 1) to store in the PKM a trace over the invocation of the selected tools, and 2) to actually invoke the tools. These two tasks are performed by the PE (see Figure 4), which behaves as intermediary between the GUI, the PKM and the DECODER tools.

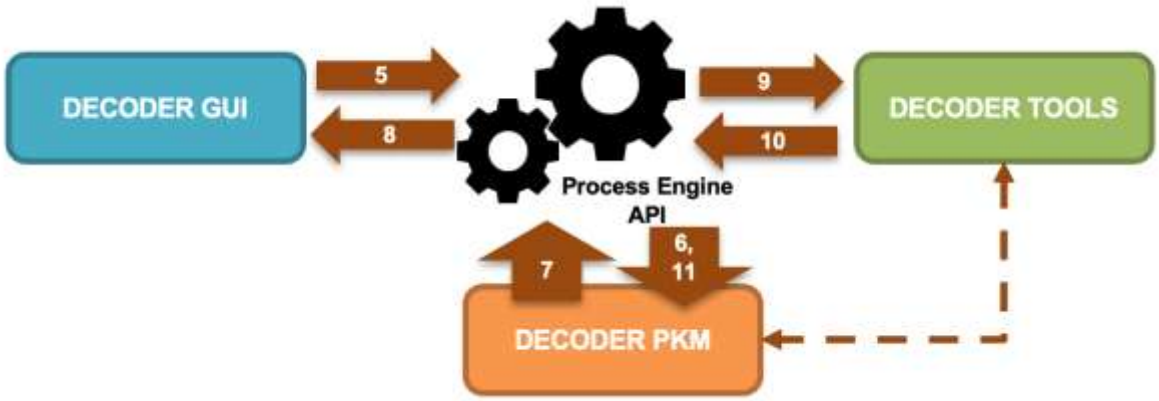

Figure 4: DECODER Tool Invocation Process - Phase 2

Specifically, the seven steps performed in this phase are the following:

(5) **GUI** creates and configures a new **Tool** invocation and sends it to the **PE**
(6) **PE** stores in the **PKM** the new invocation data
(7) **PKM** returns the new invocation ID
(8) **PE** returns to the **GUI** the new invocation ID
---
(9) **PE** invokes the tool with the specified configuration[2]
---
(10) Once the **Tool** finalizes, the tool will request the **PE** to store the execution results[3]
(11) **PE** stores in the **PKM** the invocation results

3.2.1 Tools Invocation Queue

There are some tools that only accept one single invocation at a time, that is, some tools do not accept parallel calls. Due to these restrictions introduced by the tools, a **queue** is needed to manage the different invocations to the tools. This queue will be implemented by a background process. The background process will check for pending invocations to tools that have no running invocations. In case an invocation that fills these restrictions is found, the process will send the execution request.

3.3 Phase 3: Retrieving the Tool Invocation Results

Finally, the GUI should provide the user information about the current status of the invocations performed to the selected tools. This information again is delivered to the GUI by the PE, which queries the PKM to retrieve such information (see Figure 5).

[2] It can be delayed depending on the invocations queue
[3] It can be delayed depending on the time spent performing the tool invocation.

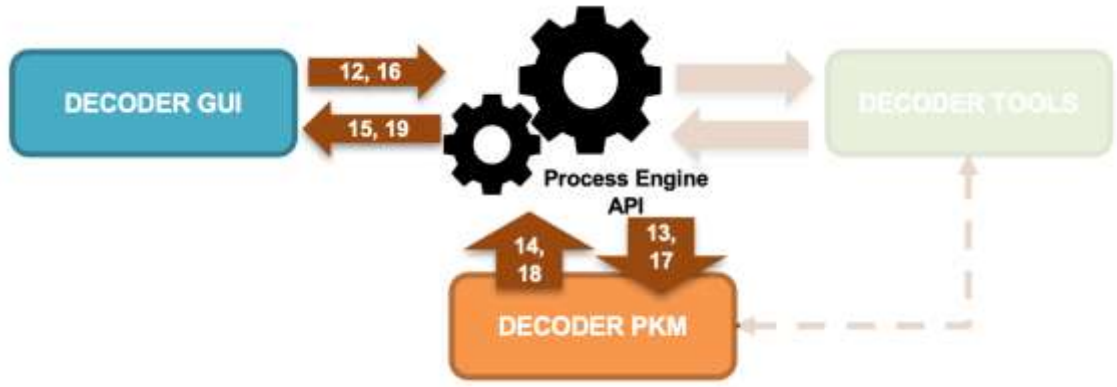

Figure 5: DECODER Tool Invocation Process - Phase 3

Specifically, the eight steps performed in this phase are the following:

(12) **GUI** asks the **PE** for the invocation results
(13, 14) **PE** searches in the **PKM** for the invocations results
(15) **PE** answers **GUI** with invocation results
(16) **GUI** asks the **PE** for specific invocation results.
(17, 18) **PE** searches in the **PKM** for the specific invocation results
(19) **PE** answers **GUI** with the specific invocation results

4 Summary, conclusion and next steps

In this document we presented the PE, which acts as intermediary between the DECODER set of tools, the GUI and the PKM to support the methodology of the DECODER platform.

This PE was implemented using a set of services described in the Section 2 of the document, that allow users (1) to retrieve information about the tools that are available in DECODER to support a specific task, and (2) to create and retrieve information about the invocations performed by the tools that were executed. On the other hand, the process described to perform a new invocation to the tools using the platform was described in Section 3 of the document. This process is divided into three different phases, which were designed to address the problem introduced by some tools that do not support parallel calls.