Deliverable D3.9

PIACERE IDE – v3

Editor(s):	Eliseo Villanueva
Responsible Partner:	Prodevelop
Status-Version:	Final – v1.0
Date:	01.06.2023
Distribution level (CO, PU):	Public

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	PIACERE IDE – v3
Due Date of Delivery to the EC	31.05.2023

Workpackage responsible for the Deliverable:	WP3 - Plan and create Infrastructure as Code
Editor(s):	Eliseo Villanueva (PRODEVELOP)
Contributor(s):	Eliseo Villanueva, Jose Climent and Ismael Torres (PRODEVELOP)
Reviewer(s):	Laurentiu Constantin Niculut (HPE)
Approved by:	All Partners
Recommended readers:	WP2, WP3, WP4, WP5, WP6 and WP7

Abstract:	This deliverable is the last deliverable of the task 3.5. The deliverable will be composed of a software prototype and a technical design document. This outcome will present the IDE resulting from the integration of KR1,KR3 - KR8. The software will be accompanied by a Technical Specification Report
Keyword List:	PIACERE IDE, ECLIPSE
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	03.05.2023	Table of contents	PRODEVELOP
v0.2	03.05.2023	First draft version	PRODEVELOP
v0.3	03.05.2023	Comments and suggestions received by Nacho	UPV
v0.4	03.05.2023	Update content	PRODEVELOP
v0.5	18.05.2023	Review ready	HPE, HPECDS
v0.6	29.05.2023	Candidate version, comments addressed.	PRODEVELOP
v1.0	01.06.2023	Final quality check. Ready for submission.	TECNALIA

Table of contents

List of tables

List of figures

Terms and abbreviations

API	Application Programming Interface
DevOps	Development and Operation
DevSecOps	Development, Security and Operations
DoA	Description of Action
DOML	DevSecOps Modelling Language
DOML-E	DevSecOps Modelling Language-Extensions
DSL	Domain Specific Language
EMF	Eclipse Modeling Framework
IaC	Infrastructure as Code
ICG	Infrastructural Code Generator
IDE	Integrated Development Environment
IEC	Infrastructural Elements Catalogue
IEM	IaC Executor Manager
IOP	IaC Optimization
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
KR	PIACERE key Result
PRC	Piacere Runtime Controller
QVT	Query/View/Transformation
RCP	Rich Client Platform
REQ	Requirement
REST API	Representational State Transfer Application Programming Interface
SVN	Apache Subversion
UC	Use Case
UML	Unified Modelling Language
VT	Verification Tool
WP	Work Package

Executive Summary

This deliverable provides updates to deliverable D3.8 [1], describing the progress and improvements of third iteration of the PIACERE Integrated Development Environment (IDE), developed in Task 3.5 related to WP3.

In the previous deliverable D3.8, it was described the progress of integration of the different KRs in the IDE, based on Eclipse IDE + EMF, through APIs. In the same way, this deliverable describes the changes made to adapt the modifications of each of the APIs to introduce improvements in the operation of the different tools; the alignment with the newest version of the DOML; the visual representation of the DOML Infrastructure Layer and, finally, the modification of the proxy server configuration. As usual, the deliverable is composed of a software prototype and a technical design document.

This is the latest version of the IDE so if there is any change until the end of the project it should be minimal, although some new update is not ruled out to support some new version of the DOML.

1 Introduction

This deliverable describes the current status of PIACERE Integrated Development Environment (IDE)-KR2 at M30. It provides updates to the earlier deliverable D3.8 [1] describing the main changes of IDE during this period (M25-M30).

The main activities have been focused on updating the other KRs APIs to adapt them to the improvements developed by KR owners towards their integration in the IDE; the alignment with the DOML version; the visual representation of the DOML Infrastructure Layer and, finally, to solve some problems arising from some of the use cases such as changing the use of the proxy server.

1.1 About this deliverable

This deliverable is the third and last one related to the IDE. It aims to show the evolution and changes produced since the earlier deliverable D3.8 [1]. The main improvements have been carried out in the modification of the APIs of the different KRs to be able to align them with the improvements introduced by their owners; the development of the visual representation part of the DOML infrastructure has also been completed so that the user can have a simpler vision of their infrastructure; lastly, the requirement from the Slovenian Ministry of Public Administration (SIMPA) use case with the use of the proxy from its internal network has been solved, being a government entity, it needs a very high level of security.

1.2 Document structure

The document is organized as follows:

- Section 1 introduces the contents and the objectives of this document.
- Section 2 provides a general overview of the IDE, of their changes compared to the version reported in D3.8, and of their main innovations.
- Section 3 provides an overview of the preliminary tests made with the tool.
- Section 4 summarizes the lessons learnt and the plan for future development.
- Section 5 concludes the deliverable.

The deliverable is accompanied by Appendix which includes the detailed specification of the implementation and indications about delivery and usage. This part of the deliverable is included for the shake of completeness and with the objective of producing a self-contained last version of the D3.8, even if it has not suffered many changes with respect to the content included in D3.8.

2 KR 2- IDE overview

The central element of the PIACERE Framework is the IDE, by which it is possible to specify the infrastructure of the application using a model-driven engineering approach. From there it is possible to call to the other KRs to execute their functionalities and return the results for the end-user view. IDE let the users use and show results of every tool easily, without having to access each of them separately.

2.1 Changes in v3

Since the last deliverable D3.8 [1], some functionalities have been added:

- Compatibility with the different versions of the DOML language, that have been updated, has been added. It currently supports DOML v3.0 as shown in the requirements table in section 2.2, in requirements 28 and 76.
- A graphical editor has been implemented that allows the visualization of a subset of the entities that make up the infrastructure layer of the DOML as shown in Figure 3.
- To address a usability issue in the SIMPA use case, the IDE Proxy settings have been changed.
- A configuration file "*deploymentConfig.json*" has been added when a new PIACERE project is created (once the user creates a PIACERE project this has an empty DOML and this JSON file as shown in Figure 1). In this file (Figure 2) the user should add the security credentials that are necessary to perform its deployment because, for security reasons, this type of information (credentials) must not be included in the DOML file.

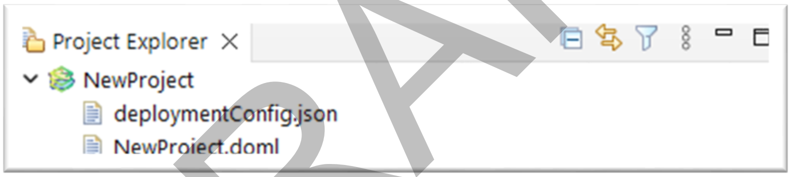
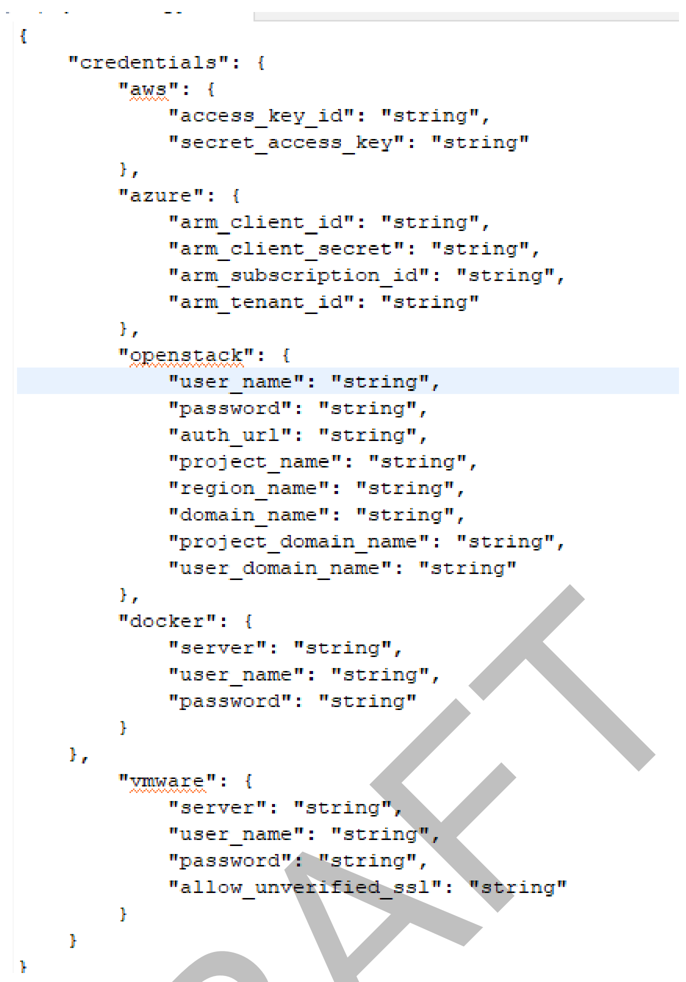


Figure 1: Files in a new PIACERE project



```
{
    "credentials": {
        "aws": {
            "access_key_id": "string",
            "secret_access_key": "string"
        },
        "azure": {
            "arm_client_id": "string",
            "arm_client_secret": "string",
            "arm_subscription_id": "string",
            "arm_tenant_id": "string"
        },
        "openstack": {
            "user_name": "string",
            "password": "string",
            "auth_url": "string",
            "project_name": "string",
            "region_name": "string",
            "domain_name": "string",
            "project_domain_name": "string",
            "user_domain_name": "string"
        },
        "docker": {
            "server": "string",
            "user_name": "string",
            "password": "string"
        }
    },
        "vmware": {
            "server": "string",
            "user_name": "string",
            "password": "string",
            "allow_unverified_ssl": "string"
        }
    }
}
```

Figure 2: Screenshot of "deploymentConfig.json" content when created

- Connection with IEM and PRC. The Error message has been enhanced with the logs or link to the logs and propagate it to the IDE/PRC.
- To adapt to the situation in which the user wants to use IaC files that he/she has previously developed, the possibility to include and encapsulating them in a ZIP file has been enabled. Then this .zip file is sent to the ICG and the PRC (not at same time).
- All PIACERE tools are continuously being updated and some of them have modified their APIs.
- The call to the different tools has been updated to make it without having to do the earlier step of converting the DOML files to DOMLX.

2.1.1 Graphical editor

The graphical editor is the last tool apported in the last version of the IDE that helps to modeling the deployment of the elements of an infrastructure in a visual way.

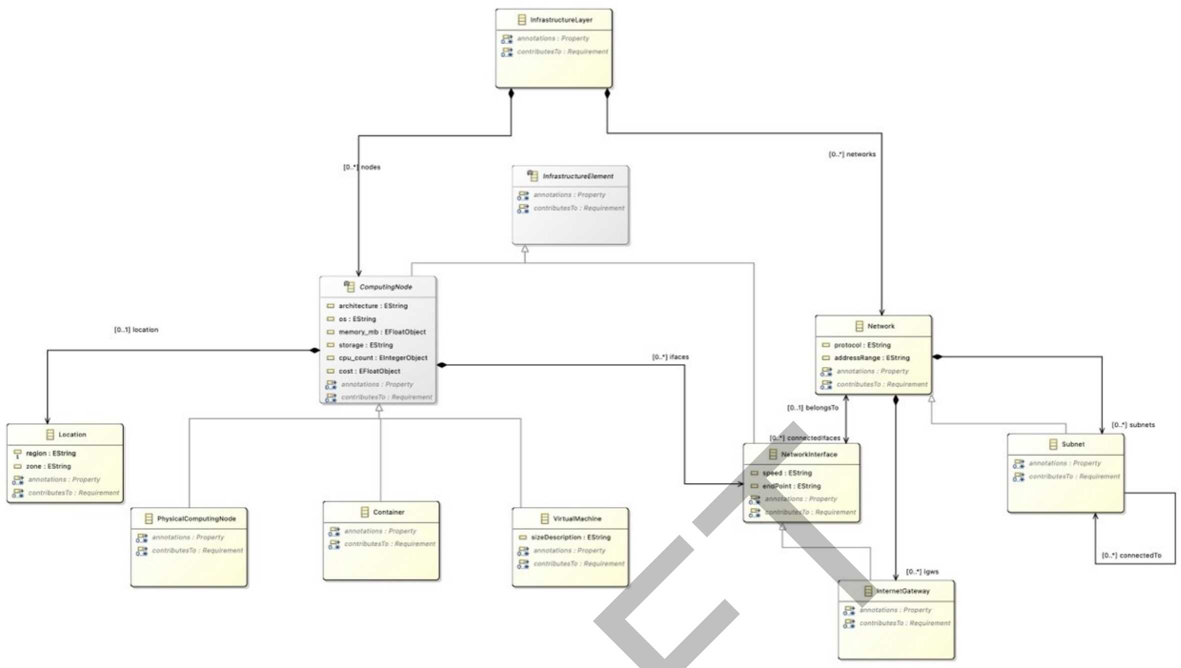
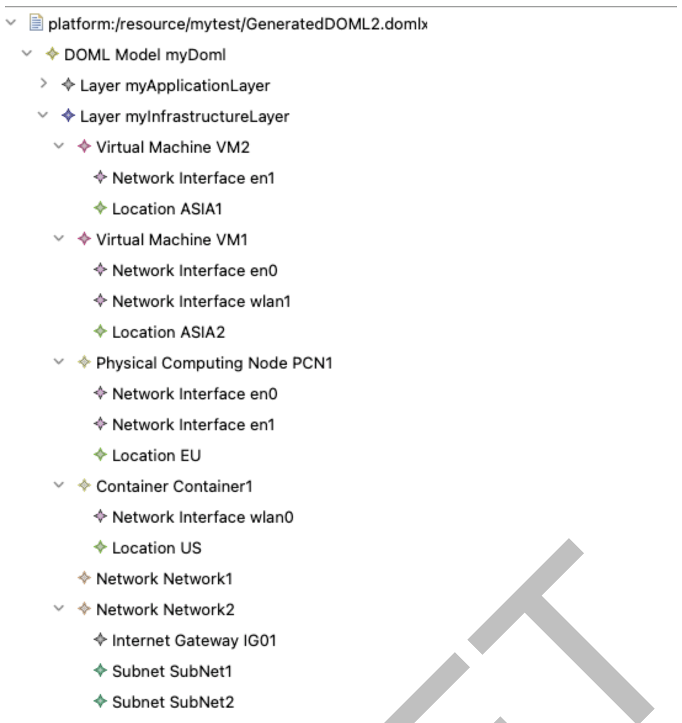


Figure 3: Subset of DOML elements to be represented

For example, in the third figure, the entities related to the computational nodes will be represented, as well as the networks and sub-networks to which they are connected, and the locations where these nodes are located.

To exemplify editing and viewing capabilities, we will use the following DOML example (Figure 4):



```
platform:/resource/mytest/GeneratedDOML2.domlx
  DOML Model myDoml
    Layer myApplicationLayer
    Layer myInfrastructureLayer
      Virtual Machine VM2
        Network Interface en1
        Location ASIA1
      Virtual Machine VM1
        Network Interface en0
        Network Interface wlan1
        Location ASIA2
      Physical Computing Node PCN1
        Network Interface en0
        Network Interface en1
        Location EU
      Container Container1
        Network Interface wlan0
        Location US
      Network Network1
      Network Network2
        Internet Gateway IG01
        Subnet SubNet1
        Subnet SubNet2
```

Figure 4: DOML Example

The graphical representation obtained from this DOML fragment would be as follows in Figure 5. It can be obtained using the new representation file option placed in the pop-up menu:

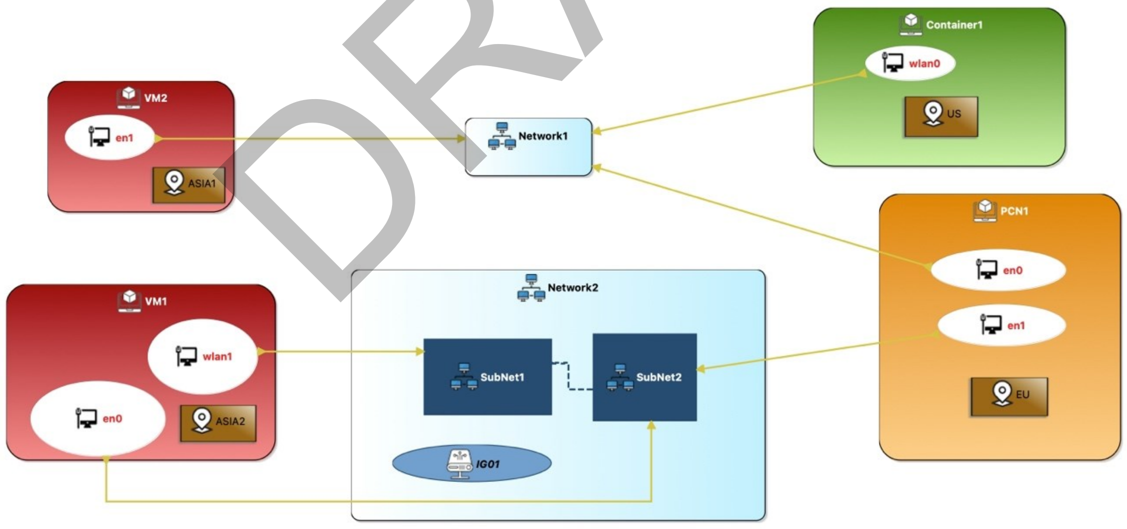
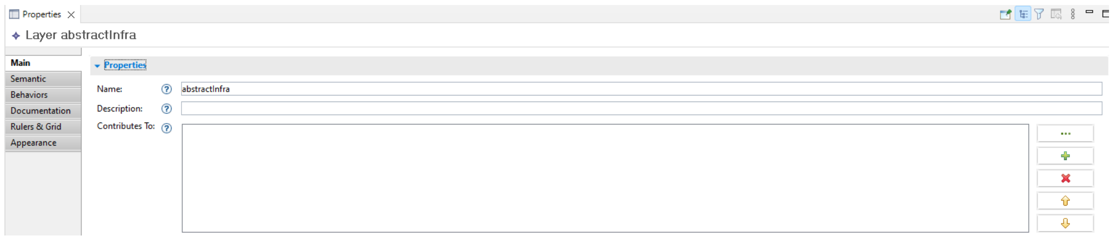


Figure 5: DOML Graphical Representation

Editing capabilities have been incorporated that allow modifying the parameters and relationships of the different entities, whose changes will be reflected in the textual specification of the DOML. To avoid some problems with the synchronization of text editor and graphic editor, the graphic editor only supports the visualization. If you want to edit some properties, you can use the tabs that we enabled for it. .

For editing the elements, the following tabs have been enabled, and allows to edit the fields that the editor can modify for a concrete element of the infrastructure. This tabs can automatically opened by clicking in the element that you like to edit


Properties
Layer abstractInfra

Main
Semantic
Behaviors
Documentation
Rulers & Grid
Appearance

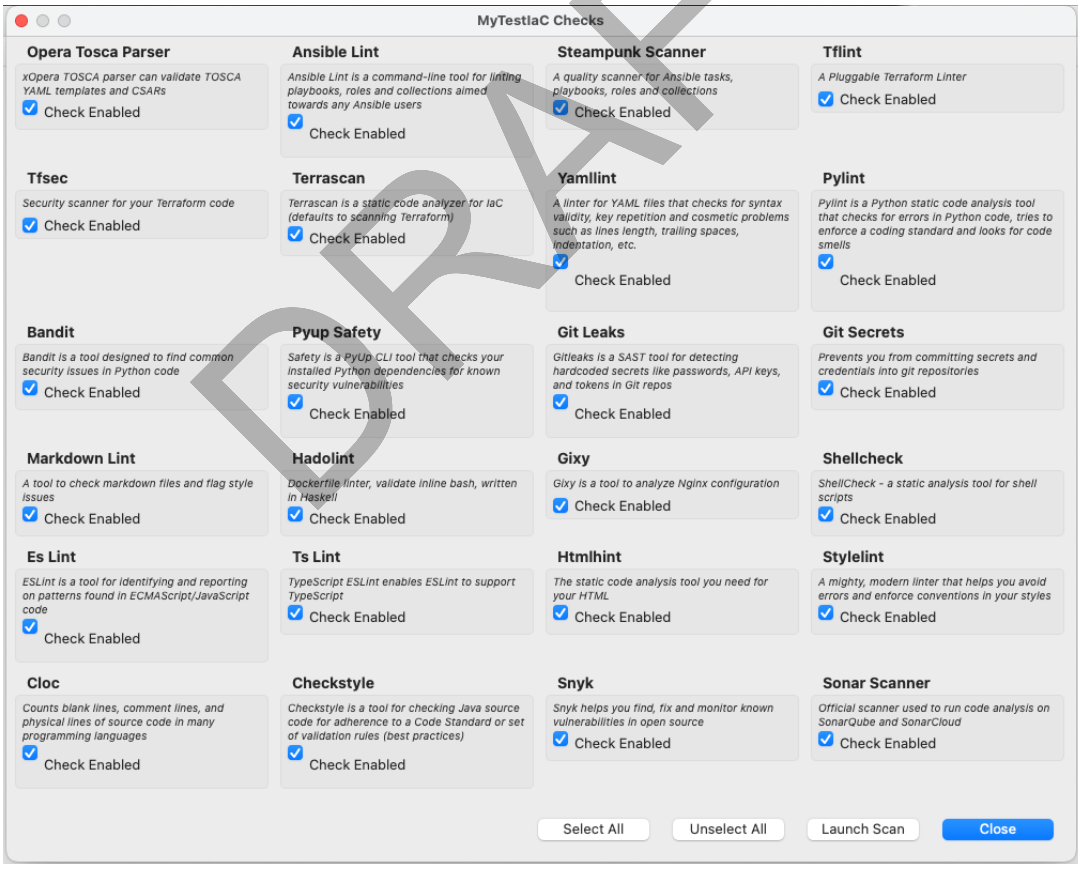
Properties
Name: abstractInfra
Description:
Contributes To:

Figure 6: Editing Tabs

2.1.2 APIs update

As mentioned above, all PIACERE tools are under continuous development. For this reason, some of them have modified their APIs to add new functionalities or modify the existing ones and this implied the need to also modify the IDE so that everything works correctly.

- The API of the IaC scanner was updated and the IDE was modified so that the end user could select the different types of analysis in a simpler, more intuitive, and user-friendly way showed in Figure 7.


MyTestIaC Checks

Opera Tosca Parser
xOpera TOSCA parser can validate TOSCA YAML templates and CSARs
☑ Check Enabled

Ansible Lint
Ansible Lint is a command-line tool for linting playbooks, roles and collections aimed towards any Ansible users
☑ Check Enabled

Steampunk Scanner
A quality scanner for Ansible tasks, playbooks, roles and collections
☑ Check Enabled

Tflint
A Pluggable Terraform Linter
☑ Check Enabled

Tfsec
Security scanner for your Terraform code
☑ Check Enabled

Terrascan
Terrascan is a static code analyzer for IaC (defaults to scanning Terraform)
☑ Check Enabled

Yamllint
A linter for YAML files that checks for syntax validity, key repetition and cosmetic problems such as lines length, trailing spaces, indentation, etc.
☑ Check Enabled

Pylint
Pylint is a Python static code analysis tool that checks for errors in Python code, tries to enforce a coding standard and looks for code smells
☑ Check Enabled

Bandit
Bandit is a tool designed to find common security issues in Python code
☑ Check Enabled

Pyup Safety
Safety is a PyUp CLI tool that checks your installed Python dependencies for known security vulnerabilities
☑ Check Enabled

Git Leaks
Gitleaks is a SAST tool for detecting hardcoded secrets like passwords, API keys, and tokens in Git repos
☑ Check Enabled

Git Secrets
Prevents you from committing secrets and credentials into git repositories
☑ Check Enabled

Markdown Lint
A tool to check markdown files and flag style issues
☑ Check Enabled

Hadolint
Dockerfile linter, validate inline bash, written in Haskell
☑ Check Enabled

Gixy
Gixy is a tool to analyze Nginx configuration
☑ Check Enabled

Shellcheck
ShellCheck - a static analysis tool for shell scripts
☑ Check Enabled

Es Lint
ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code
☑ Check Enabled

Ts Lint
TypeScript ESLint enables ESLint to support TypeScript
☑ Check Enabled

Htmlhint
The static code analysis tool you need for your HTML
☑ Check Enabled

Stylelint
A mighty, modern linter that helps you avoid errors and enforce conventions in your styles
☑ Check Enabled

Cloc
Counts blank lines, comment lines, and physical lines of source code in many programming languages
☑ Check Enabled

Checkstyle
Checkstyle is a tool for checking Java source code for adherence to a Code Standard or set of validation rules (best practices)
☑ Check Enabled

Snyk
Snyk helps you find, fix and monitor known vulnerabilities in open source
☑ Check Enabled

Sonar Scanner
Official scanner used to run code analysis on SonarQube and SonarCloud
☑ Check Enabled

Figure 7: New IaC Scanner menu

- As mentioned in a previous previously, the ICG has been updated to support the different scenarios that have arisen, and this has entailed modifying its API and the way the IDE communicates with.

- Another API that has been updated requiring modification of the IDE is that of the IOP, which has modified the way in which error messages are captured to make them easier to understand and, to this end, it is necessary for these messages to be displayed to the user.
- Finally, from the Infrastructural Elements Catalogue (IEC) we have enabled the possibility of adding new customized elements that are different from the services we already had, so these added resources have been treated separately in the database. This has meant a modification of the API to obtain which resources the catalogue contains.

2.2 Functional description and requirements coverage

In this section, it is supplied a summary of the PIACERE IDE requirements (functional, non-functional and business requirements). For each requirement, an explanation of how the requirement has been addressed.

Table 1: Requirements related with the IDE

Req ID	Description	Status	Requirement Coverage at M30
30REQ28	DOML should support the modelling of containerized application deployment	Fully	The DOML supports the modelling of containerized application deployment. The current version of the IDE provides full support for DOML 3.0.
REQ40	The IDE should provide a visual diagram functionality to visualise the different assets defined through the DOML and DOML Extensions.	Fully	The current version of the IDE provides a tree editor to manage the DOML instances. In this version a visual editor has been added to the IDE. The editor let the user visualize main DOML Infrastructure Layer, but only properties are editable.
REQ41	The IDE should be extensible through the plugin mechanism. Not only to support PIACERE assets (ICG, VT) but also for third party collaborators.	Fully	See comments in D3.8 [1].
REQ42	The IDE should be implemented using open-source software.	Fully	See comments in D3.7 [2].
REQ43	The IDE should be easily updatable to newer software versions.	Fully	See comments in D3.8 [1].
REQ44	The IDE could provide an import mechanism to automatically fulfil partial DOML.	Not implemen ted	Not provided but it will be interesting to study this functionality in further versions of the IDE.
REQ62	DOML must support different views.	Fully	See comments in D3.7 [2].
REQ64	The IDE should provide a text-based representation of DOML to ease version control.	Fully	See comments in D3.8 [1].
REQ76	DOML should allow the user to model each of the four considered DevOps activities (Provisioning, Configuration, Deployment, Orchestration).	Fully	The DOML supports DevOps activities. The second version of the IDE provides full support for the DOML version 3.0.
REQ99	IDE to integrate with both local and remote Git repositories.	Fully	Public Git repositories and local IaC files can be used.
REQ101	IDE should allow to create and edit a graphical DOML model.	Fully	The current version of the IDE provides a tree editor to manage the DOML instances. In this version a visual

	Possibly starting from a palette of supported components that can be drag & drop in the graphical model		editor has been added to the IDE. The editor let the user visualize main DOML Infrastructure Layer, but only the edition of the properties is allowed.
REQ111	The user could have the possibility to add external custom own IaC.	Fully	A new functionality has been added in the IDE to allow the user to use his own IaC by allowing him to attach it together with the DOML file in a ZIP to send it to the ICG.

At this moment, the IDE supports all the requirements except requirement 44. This requirement hasn`t been addressed because it is a non-blocking requirement with minor priority and the requirements were ordered by priority. Currently req44 is under analysis to check if it's interesting to be addressed in the future versions of the IDE.

2.3 Main innovations

PIACERE Integrated Development Environment offers a standard and modern DevSecOps framework for DevOps to cover all related tools and workloads for manipulating DOML models. Naturally, Verification Tool (VT), IaC Optimizer Platform for selecting the best resources or Infrastructure Code Generator would be quite difficult to use and manage, if they were not integrated into a governance framework with a structured configuration and user-friendly interfaces.

Although partial solutions exist, such as Terraform, there is no single, standard, integrated, and holistic framework for working with IaC at each Cloud service provider that organises each stage within the cloud architecture development and operation processes.

There is no single, neat, security-focused DevOps methodology that cross-connects each of the tools currently in use. From both a customer and cloud service provider perspective, having a standard tool to address these challenges would offer substantial benefits to DevOps teams.

PIACERE IDE offers provisioning of incremental versions of the PIACERE framework, as well as a DevSecOps methodological loop that connects all the tools in the suite.

3 Overview of preliminary experiments

The main experiments or tests that have been carried out after testing the different versions of the tool have been focused on using them directly with the examples that the different tools had prepared. For example, the different DOML files that can be found in the repository were used to test the functionality of each of the tools and from which some improvement points emerged, not only for the IDE but also for the tools themselves, such as the need to improve the different error messages or to be more specific in the location of the errors that come from the DOML.

On the other hand, the different use cases had to learn how to use the tool to develop their infrastructures using the IDE and for this purpose training sessions were given with the use cases to explain the different functionalities and how to access them. This is where some of the functionalities that were not covered by the examples were tested and a problem appeared with the use of the proxy that was fixed in the next version of the IDE.

4 Lessons learnt and outlook to the future

With the use of the different versions of the IDE, feedback has been obtained from the different users who have tested it. For example, after starting development with Eclipse Theia [20], there were many problems in completing the development and the conclusion was to start again using EMF [4]. When a first functional version of the IDE was obtained, needs appeared that had not been contemplated, for example, possible changes in the way the results of the different tools were presented. After the second version, the need to change the properties of the way of using a proxy was demonstrated because a use case required it for its correct functioning. On the other hand, the use of the different tools has also shown the usefulness of a graphical representation that would allow the interrelationships of the different components to be seen.

With all this compilation of needs/updates, work has continued improving the IDE, although this is a work of continuous improvement that is always open to the diverse needs of the users.

5 Conclusions

As has been repeated throughout deliverables D3.7, D3.8 and D3.9, the IDE is the central piece because it is the entry point for the user to interact with the PIACERE framework. This IDE is the main tool that facilitates the use of the other PIACERE tools. Throughout the project, the development of this Integrated Development Environment has been carried out using consolidated tools such as Eclipse EMF that allow a simple integration of models such as those required by the DOML language and allow the use of REST APIs in an easy way; a fact that has been very useful because most of the tools of the project have been integrated in this way in the IDE.

As mentioned in the previous sections, all the requirements foreseen for the tool have been met apart from requirement 44 "The IDE could provide an import mechanism to automatically fulfil partial DOML", which will be considered for future developments outside the scope of this project. In addition, the use of the tool has been validated in a first validation phase and improvements have been added to the comments made by the different use cases.

The next steps to be taken are to help the user to fill in the different fields of a DOML file, the creation of a visual editor that allows interacting with all the elements and modifying their parameters from that view; and, finally, to wait for the conclusions of the second phase of the validation by the three use cases to listen to their requests and comments and integrate them as points of improvement for future versions.

6 References

[1] "PIACERE Deliverable D3.8 - PIACERE IDE-v2_v1.0," 2022. [Online]. Available: https://zenodo.org/record/7431234#.ZDZPEHZBz5p.

[2] "PIACERE Deliverable D2.1 - PIACERE DevSecOps Framework Requirements specification, architecture and integration strategy – v1," [Online]. Available: https://zenodo.org/record/6801782#.Y3tUo3bMIuW.

[3] "Eclipse IDE," [Online]. Available: https://www.eclipse.org/eclipseide/.

[4] "Eclipse Modeling Framework," [Online]. Available: https://www.eclipse.org/modeling/emf/.

[5] "Eclipse Graphical Modeling Framework," [Online]. Available: https://www.eclipse.org/gmf-runtime/.

[6] "Eclipse Graphiti," [Online]. Available: https://www.eclipse.org/graphiti/.

[7] "Eclipse ATL," [Online]. Available: https://www.eclipse.org/atl/.

[8] "Eclipse QVTo," [Online]. Available: https://www.eclipse.org/mmt/qvto.

[9] "Xpand," [Online]. Available: https://projects.eclipse.org/projects/modeling.m2t.xpand .

[10] "Xtext," [Online]. Available: https://www.eclipse.org/Xtext.

[11] "PIACERE Deliverable D3.7 - PIACERE IDE-v1_v1.0," [Online]. Available: https://zenodo.org/record/6821671#.Y3tJLXbMIuW.

[12] "Eclipse IDE Base plugins," [Online]. Available: https://download.eclipse.org/eclipse/downloads/.

[13] "Eclipse Sirius," [Online]. Available: https://www.eclipse.org/sirius/.

[14] "Eclipse Acceleo," [Online]. Available: https://www.eclipse.org/acceleo/.

[15] "Eclipse UML2," [Online]. Available: https://www.eclipse.org/modeling/mdt/?project=uml2.

[16] "Eclipse M2E," [Online]. Available: https://www.eclipse.org/m2e/.

[17] "Eclipse Papyrus," [Online]. Available: https://www.eclipse.org/papyrus/.

[18] "Eclipse EGit," [Online]. Available: https://www.eclipse.org/egit.

[19] "readthedocs," [Online]. Available: https://readthedocs.org/.

[20] "Eclipse Theia" [Online]. Avaliable: https://theia-ide.org/

APPENDIX: Implementation, delivery, and usage

1 Implementation

1.1 Fitting into overall PIACERE Architecture

The PIACERE IDE is a tool for modelling Infrastructure solutions based on the PIACERE DOML (DevOps Modelling Language) and DOML-E (DOML Extensions). The IDE, as the main interface for user's interaction, is connected with other PIACERE tools/components (Figure 8). The design time components are more tightly integrated with the IDE as they all belong to the design phase of the solution. The runtime components are less coupled with the IDE, but nevertheless the IDE interacts with these components.

Through the IDE, users can describe their system infrastructure according to the underlying metamodel, which in the case of PIACERE is the DOML. The IDE integrates the Verification Tool (VT) and the Infrastructural Code Generator (ICG). Thanks to the VT, it is able to validate the defined DOML instance. On the other hand, the ICG tool, when triggered from the IDE automatically obtain the corresponding IaC in a specific target environment (Terraform, Ansible, VMWare) from a DOML instance. Apart from this, now you can choose an external folder with the IaC code instead of generating from a DOML instance.

All the information produced at design time is stored into the PIACERE data repository, and after finalising the design time phase, a DOML specification will be completed, and the corresponding IaC will be generated.

The runtime components of the PIACERE are also linked with the IDE. The runtime controller (PRC) could be invoked through the IDE. This component is in charge of doing the deployments and link them with the Infrastructure Advisor components.

A detailed description of the PIACERE Design time and Run time workflows can be found in the Deliverable D2.2 [2].

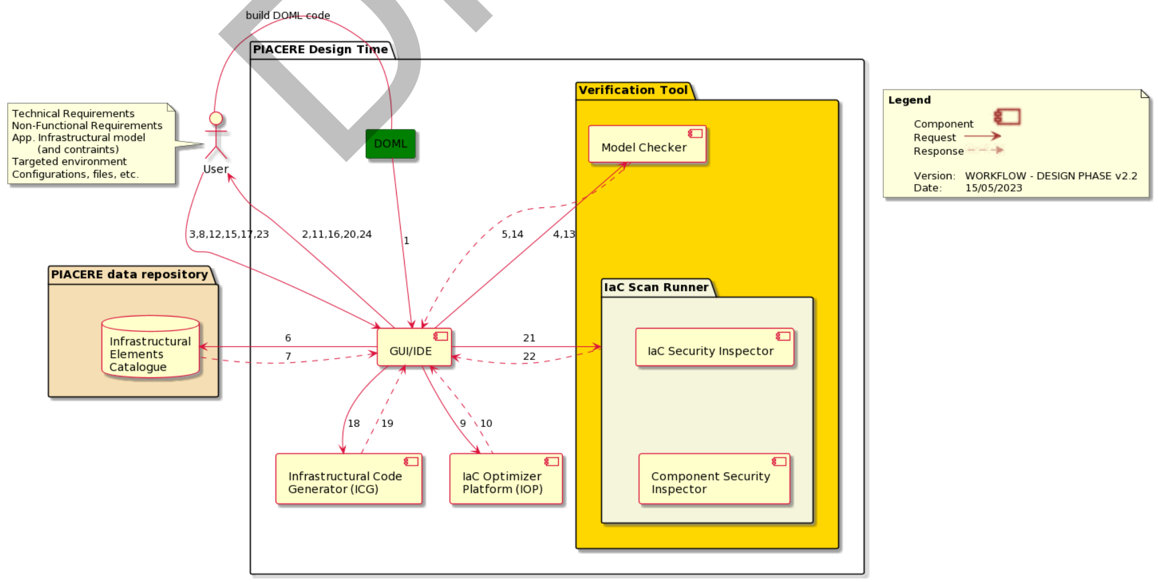
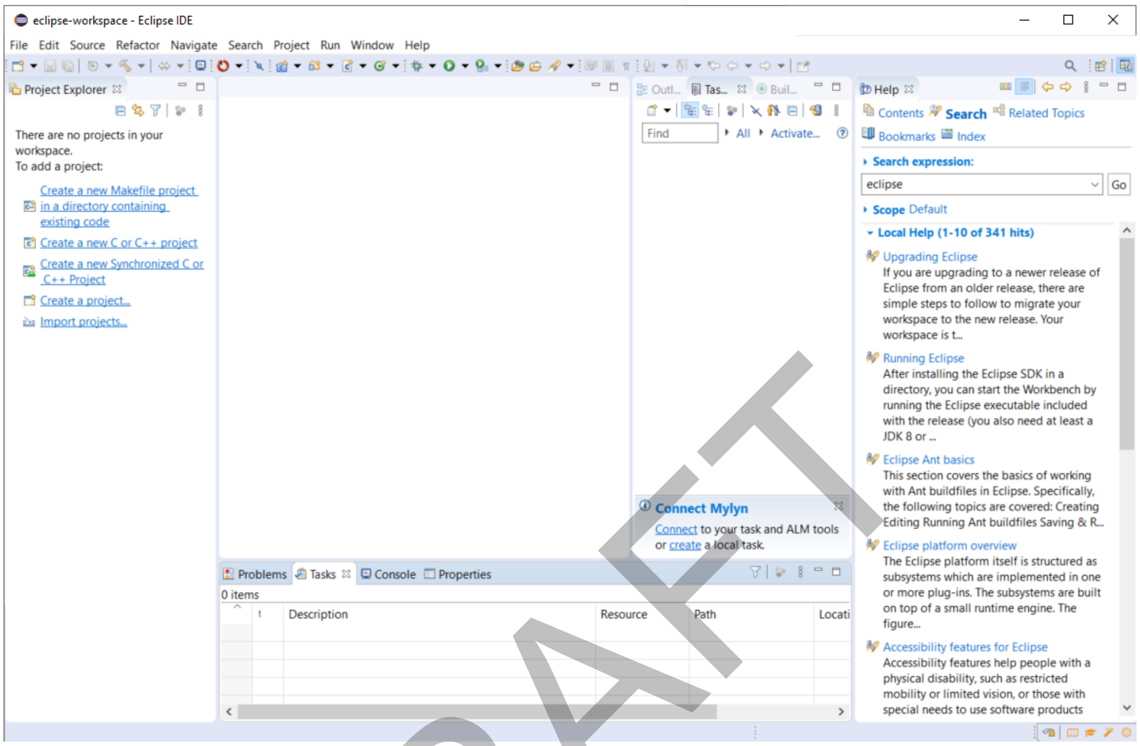


Figure 8: PIACERE Design time Components

1.2 Technical description

The Classical Eclipse IDE [3] is the most complete IDE available at this moment, completely free of use and more powerful than any other desktop IDE. It has more than 20 years of experience and improvements, with a lot of projects based on it, and a lot of plugins that contribute to it by extending its functionality and providing a lot of extra features.



There are no projects in your workspace.
To add a project:

Create a new Makefile project in a directory containing existing code
Create a new C or C++ project
Create a new Synchronized C or C++ Project
Create a project...
Import projects...

Connect to your task and ALM tools or create a local task.

Upgrading Eclipse
If you are upgrading to a newer release of Eclipse from an older release, there are simple steps to follow to migrate your workspace to the new release. Your workspace is t...

Running Eclipse
After installing the Eclipse SDK in a directory, you can start the Workbench by running the Eclipse executable included with the release (you also need at least a JDK 8 or ...

Eclipse Ant basics
This section covers the basics of working with Ant buildfiles in Eclipse. Specifically, the following topics are covered: Creating Editing Running Ant buildfiles Saving & R...

Eclipse platform overview
The Eclipse platform itself is structured as subsystems which are implemented in one or more plug-ins. The subsystems are built on top of a small runtime engine. The figure...

Accessibility features for Eclipse
Accessibility features help people with a physical disability, such as restricted mobility or limited vision, or those with special needs to use software products

Figure 9: Eclipse IDE screenshot

Eclipse IDE is a desktop application that must be installed locally. The workspaces are also stored in the local machine of the developer. It allows the integration with code version control systems like SVN or GIT.

Furthermore, the Eclipse IDE exhibits many powerful integration capabilities with UML (Unified Modelling Language) and Profiles. The other IDEs do not so well right now cover this part, and it is an important feature for the PIACERE project.

Eclipse is an IDE that can be used to create your own IDEs/products. In turn, these products can be extended with other plugins in order to fulfil the developers' needs. For example, Papyrus [12] is a product created with Eclipse. You can download Papyrus and use it as modelling application, but you can also install plugins to develop applications in Java and others.

Nevertheless, the more plugins there are installed in an Eclipse instance, the more memory is required for it to function, and the slower it becomes. Eclipse is not a light environment. Indeed, it is a very demanding environment in terms of memory consumption. This is the reason it is preferable to have several instances of Eclipse products, one per development environment, instead of only one instance with many plugins and features installed on it.

Eclipse is based on workspaces. You can have a single installation of Eclipse with multiple workspaces, each of them containing projects of distinct developments. This way, a developer can organise their projects separately and load only those that are currently being worked out.

Regarding modelling development in Eclipse, it has a long-standing experience and many plugins and features that make it easy and powerful against other IDEs.

Extensibility

Eclipse IDE extensions are called plugins too, and there exist multiple ways to install them:

- **Eclipse marketplace**: it is a marketplace where you can select and install any plugin available on that marketplace. This marketplace is hosted on Eclipse Foundation severs
- **Update site**: the developer can upload its solution with a certain structure and share the URL. With it you can install the plugins
- **Drop-ins**: the plugins can be installed locally by unzipping the content in the drop-ins folder of the Eclipse installation. This is not the best approach

Workspaces

Eclipse IDE workspaces are installed locally on each machine for each developer. There is no way to share the workspace with others and deploy in a distributed way. The user has access to all the workspaces available in its own hard drive.

The **Eclipse Modeling Framework** (EMF) project is a set of Eclipse plug-ins along with a modeling framework and code generation for building tools and other applications based on a structured data model in Eclipse. EMF (core) is a common standard for data models, which many technologies and frameworks are based on.

Eclipse Modeling Framework [4] is the core of the model generation in Eclipse. It eases building tools based on structured data models in Eclipse. EMF is provided by default on each Eclipse distribution, but there is a lot of extra functionality on editors and code generation that is interesting to consider. It also provides transaction and validation on the models.

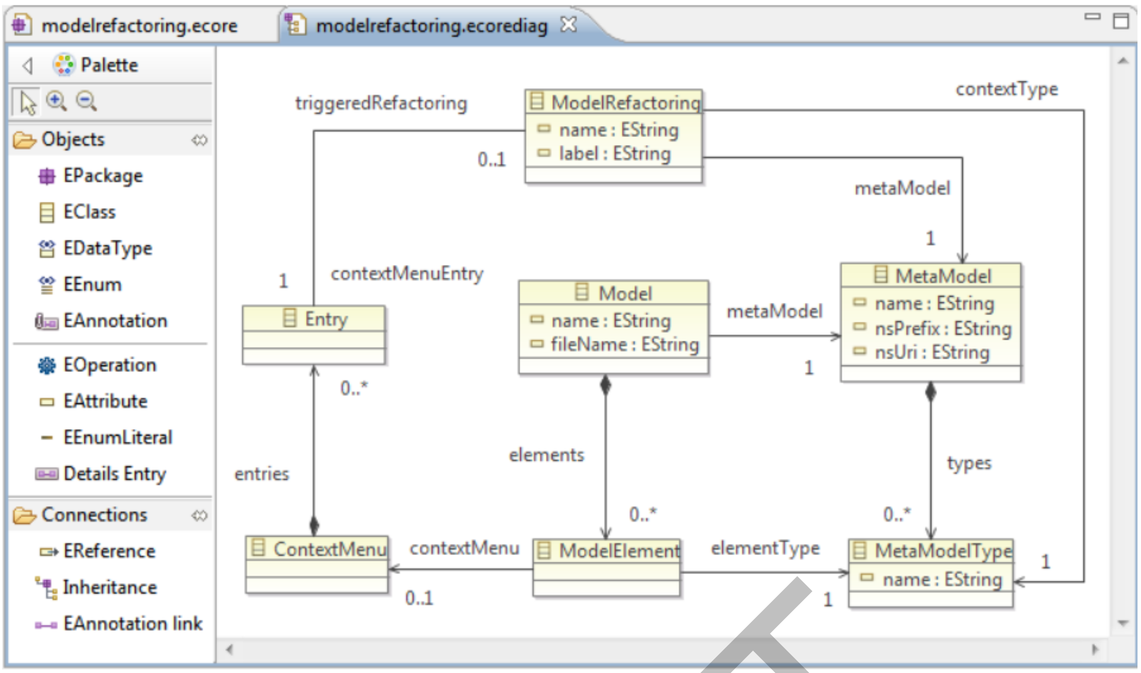


Figure 10: Eclipse Modeling Framework screenshot

EMF is considered one of the best and more expanded modelling languages, which is the base of the models. On top of it, other languages bring support for graphical editors, like GMF [5] or Graphiti [6]. There also exist model-to-model and model-to-text language generators that provide the functionality to convert models into other models, or into text or code, like ATL [7], QVT [8], XPAND [9], or Xtext [10].

Thanks to EMF, the IDE supports metamodels. In the case of the PIACERE IDE, DOML and DOML-E are the supported metamodels used for modelling application infrastructures.

1.2.1 Prototype architecture

The Eclipse IDE [3] is an Eclipse Rich Client Platform (RCP) application. The core functionalities of the Eclipse IDE are provided via plug-ins(components). The PIACERE IDE functionality is based on the concept of extensions and extension points. Thanks to EMF, the IDE will support metamodels. In the case of the PIACERE IDE, DOML and DOML-E are the supported metamodels used for modelling application infrastructures.

The other KRs of the project are integrated into the IDE using some of the integration mechanism that these technologies provide but not all in the same way. Some of these tools have been fully integrated into Eclipse as Eclipse plugins. On the other hand, other tools could be run in isolation and invoked through a REST service using a set of menus enabled to facilitate access to these KRs.

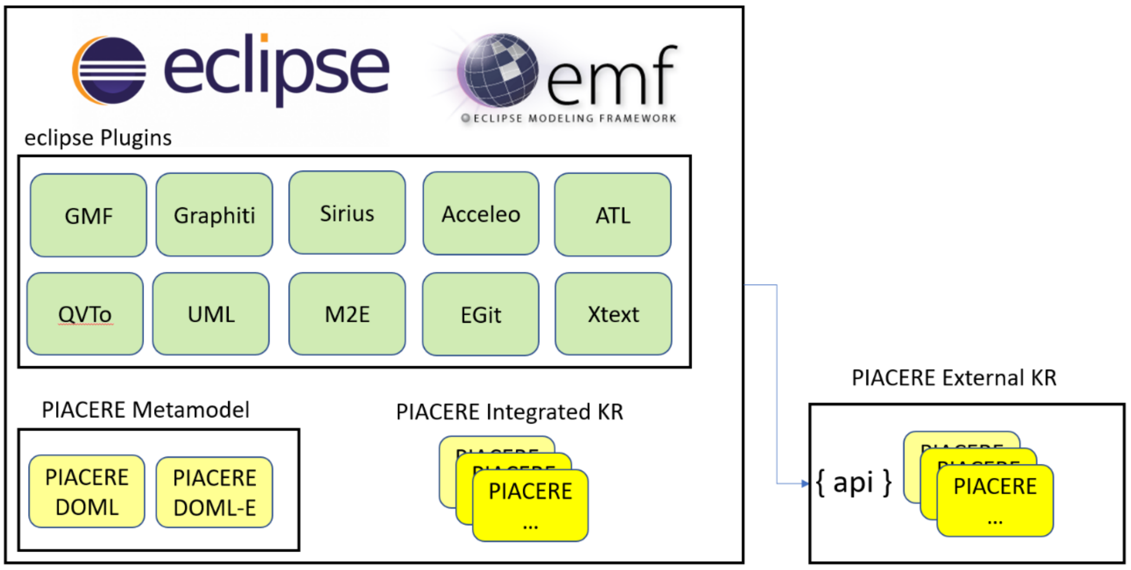


Figure 11: eclipse IDE components

1.2.2 Components description

This section is related with "Appendix: PIACERE IDE based on Eclipse desktop" of deliverable D3.7 [11] and lists some modelling plugins interesting in the building of PIACERE IDE.

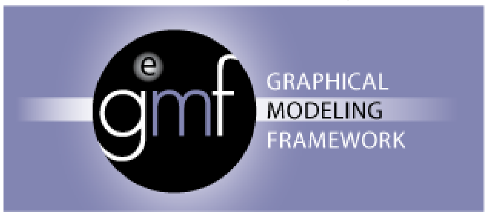
Eclipse base

These plugins are the core of Eclipse [12] and involve the minimum plugin set to build an executable desktop application. The current version of these plugins is 4.21.

EcoreTools: Ecore Diagram Editor

EcoreTools is a graphical modeler to create, edit and analyse Ecore models. It provides several a class diagram editor (similar to UML Class Diagrams), a package dependencies diagram editor and several table editors to design your Ecore model.

GMF Runtime



Graphical Modeling Framework [5] is the most used library to develop graphical editors on Eclipse. It is based on EMF models, and generates EMF based models too.

The current version of GMF is >= 1.9.0.

Graphiti

Graphiti [6] is another library for developing graphical editors on Eclipse, which enables rapid development of state-of-the-art diagram editors for domain models

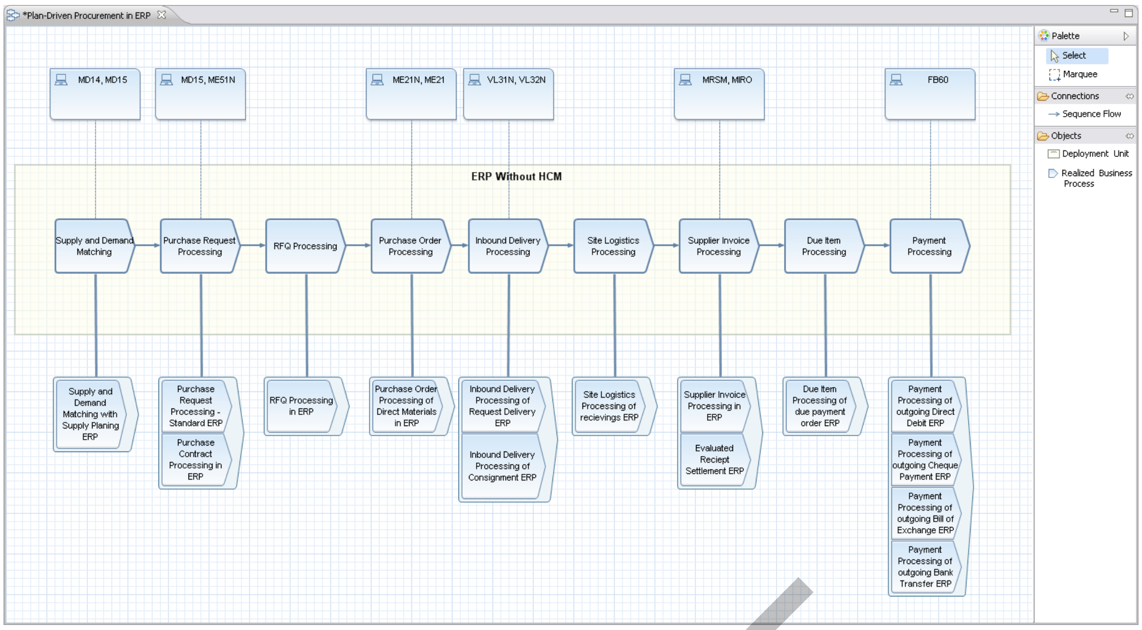

*Plan-Driven Procurement in ERP
Palette
Select
Marquee
Connections
Sequence Flow
Objects
Deployment Unit
Realized Business Process
MD14, MD15
MD15, ME51N
ME21N, ME21
VL31N, VL32N
MRSM, MIRO
FB60
ERP Without HCM
Supply and Demand Matching
Purchase Request Processing
RFQ Processing
Purchase Order Processing
Inbound Delivery Processing
Site Logistics Processing
Supplier Invoice Processing
Due Item Processing
Payment Processing
Supply and Demand Matching with Supply Planing ERP
Purchase Request Processing - Standard ERP
Purchase Contract Processing in ERP
RFQ Processing in ERP
Purchase Order Processing of Direct Materials in ERP
Inbound Delivery Processing of Request Delivery ERP
Inbound Delivery Processing of Consignment ERP
Site Logistics Processing of recievings ERP
Supplier Invoice Processing in ERP
Evaluated Reciept Settlement ERP
Due Item Processing of due payment order ERP
Payment Processing of outgoing Direct Debit ERP
Payment Processing of outgoing Cheque Payment ERP
Payment Processing of outgoing Bill of Exchange ERP
Payment Processing of outgoing Bank Transfer ERP

Figure 12: Graphiti Editor Example

The current version of Graphiti is 0.12.0.

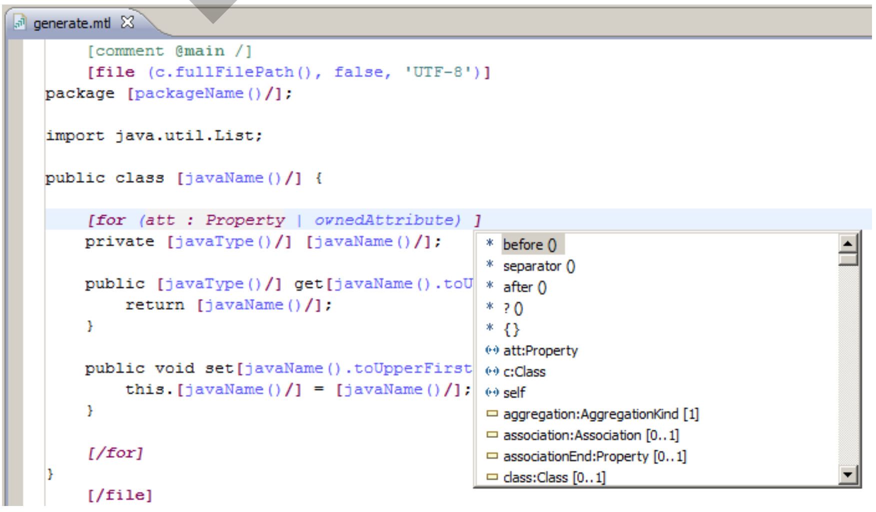
Sirius

Sirius [13] is a tool based on EMF and GMF that allows to easily create graphical modelling workbenches. A modelling workbench created with Sirius is composed of a set of Eclipse editors (diagrams, tables, and trees) which allow the users to create, edit and visualize EMF models.

The current version of Sirius is 6.5.1.

Acceleo

Acceleo [14] is a template-based technology to create code generators from an EMF model, which has been designed to be customizable, interoperable, and easy to kick-start.


generate.mtl

```
    [comment @main /]
    [file (c.fullFilePath(), false, 'UTF-8')]
package [packageName()/];

import java.util.List;

public class [javaName()/] {

    [for (att : Property | ownedAttribute) ]
    private [javaType()/] [javaName()/];

    public [javaType()/] get[javaName().toU
        return [javaName()/];
    }

    public void set[javaName().toUpperFirst
        this.[javaName()/] = [javaName()/];
    }

    [/for]
}
    [/file]
```

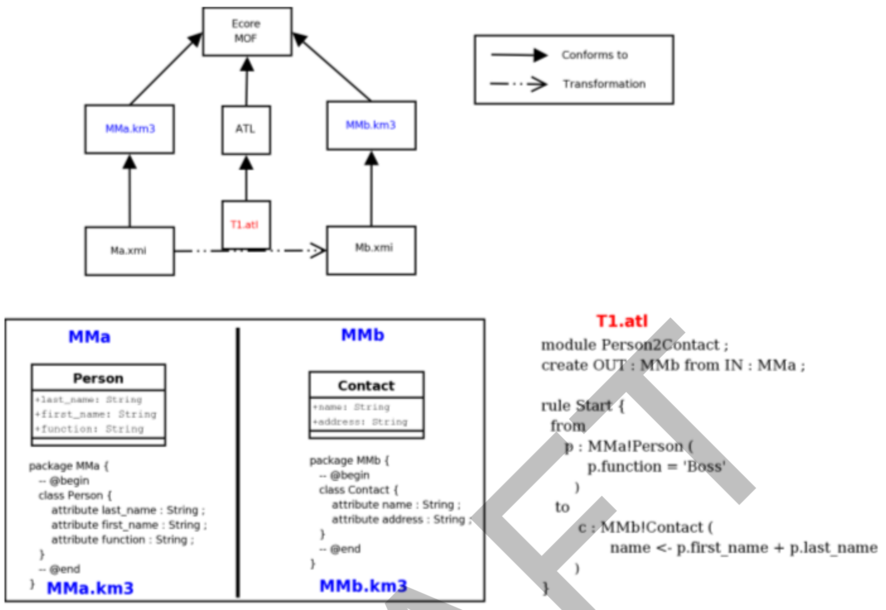
```
* before ()
* separator ()
* after ()
* ? ()
* {}
att:Property
c:Class
self
aggregation:AggregationKind [1]
association:Association [0..1]
associationEnd:Property [0..1]
class:Class [0..1]
```

Figure 13: Acceleo template example

The current version of Acceleo is 3.7.

ATL

ATL [7] is a model transformation-oriented language that helps to convert a model defined in a Domain Specific Language (DSL) into another model defined in a distinct (or not) DSL. These include some sample ATL transformations, an ATL transformation engine, and an IDE for ATL.



```
package MMa {
  -- @begin
  class Person {
    attribute last_name : String ;
    attribute first_name : String ;
    attribute function : String ;
  }
  -- @end
}
```

```
package MMb {
  -- @begin
  class Contact {
    attribute name : String ;
    attribute address : String ;
  }
  -- @end
}
```

```
module Person2Contact ;
create OUT : MMb from IN : MMa ;

rule Start {
  from
    p : MMa!Person (
      p.function = 'Boss'
    )
  to
    c : MMb!Contact (
        name <- p.first_name + p.last_name
    )
}
```

Figure 14: ALT transformation Example

The current version of ATL is 4.5.0.

QVTo

QVTO [8] is another language that supplies features to implement transformations between models. Eclipse QVTo is the only actively maintained QVTo implementation, and so conversely, QVT 1.2 has evolved to resolve issues uncovered by Eclipse QVTo and its users.

The current version of QVTo is 3.10.5.

UML2

Eclipse UML2 [15] is a set of plugins based on EMF that brings support to UML OMG Metamodel in Eclipse. Although UML2 supplies the metamodel, it does not provide UML modelling tools itself. It is the base of other important projects like Papyrus, which incorporates these kinds of tools.

The current version of UML2 is 5.5.2.

M2E

The M2Eclipse, or M2E [16] plugin provides Apache Maven functionality into Eclipse. It allows building projects based on Maven within Eclipse, as well as integrated dependency management and other features.

The current version of M2E is 1.19.0.

Papyrus

Papyrus [17] is the most popular open-source environment for editing UML models based on EMF for Eclipse. It provides many editors such as the Class diagram editor, Activity diagram editor, State Machine diagram editor, Components diagram editor, Profile diagram editor, etc.

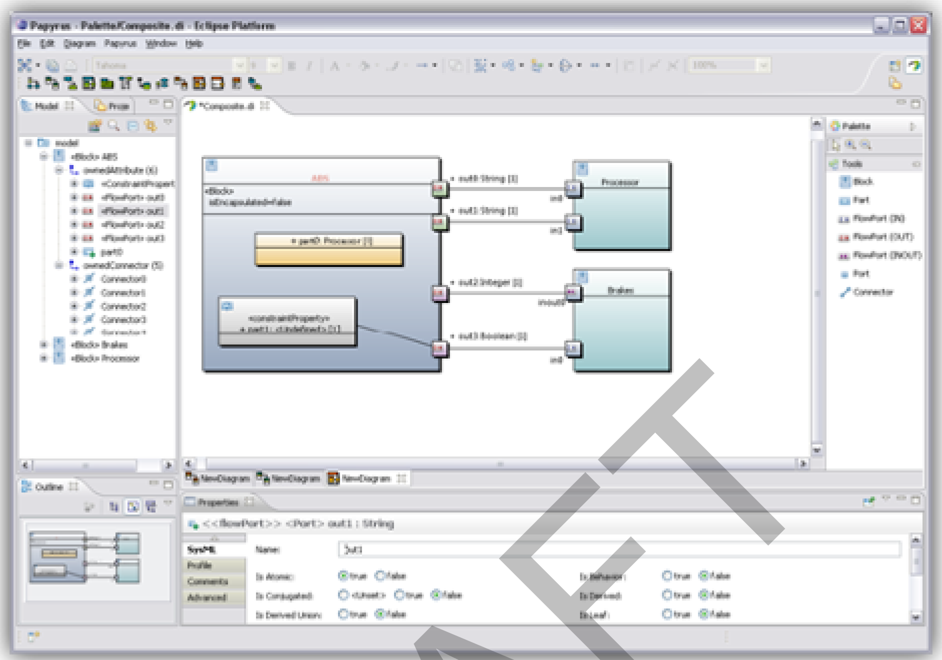


Figure 15: Papyrus instance example

The current version of Papyrus is 5.2.0.

EGit

EGit [18] is a set of plugins that enables to connect to GIT source code repositories.

The current version of EGit is 5.13.0.

Xtext

Xtext [10] is a framework for development of programming languages and domain-specific languages. With Xtext you define your language using a powerful grammar language. As a result, you get a full infrastructure, including parser, linker, type checker, compiler as well as editing support for Eclipse, any editor that supports the Language Server Protocol and your favourite web browser.

The current version of Xtext is 2.25.0.

1.2.3 Technical specifications

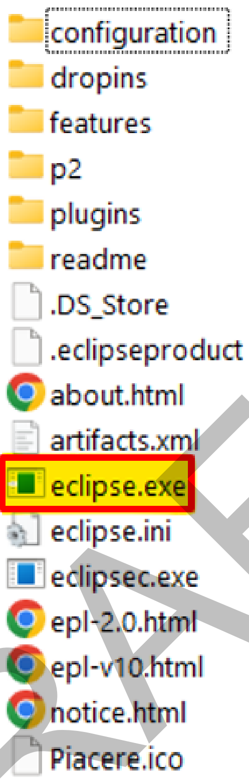
As explained in deliverables D 3.7 and D3.8, and in the previous section Components description in detail, Components description IDE has been developed using Eclipse using some of the large number of plugins available that are very helpful to add functionalities to the tool.

2 Delivery and usage

2.1 Installation instructions

See 2.4 Download section.

Once the IDE is downloaded (for the corresponding OS), the user only needs to execute it to start using PIACERE IDE (for example with double clicking .exe file in Windows).



configuration
dropins
features
p2
plugins
readme
.DS_Store
.eclipseproduct
about.html
artifacts.xml
eclipse.exe
eclipse.ini
eclipsec.exe
epl-2.0.html
epl-v10.html
notice.html
Piacere.ico

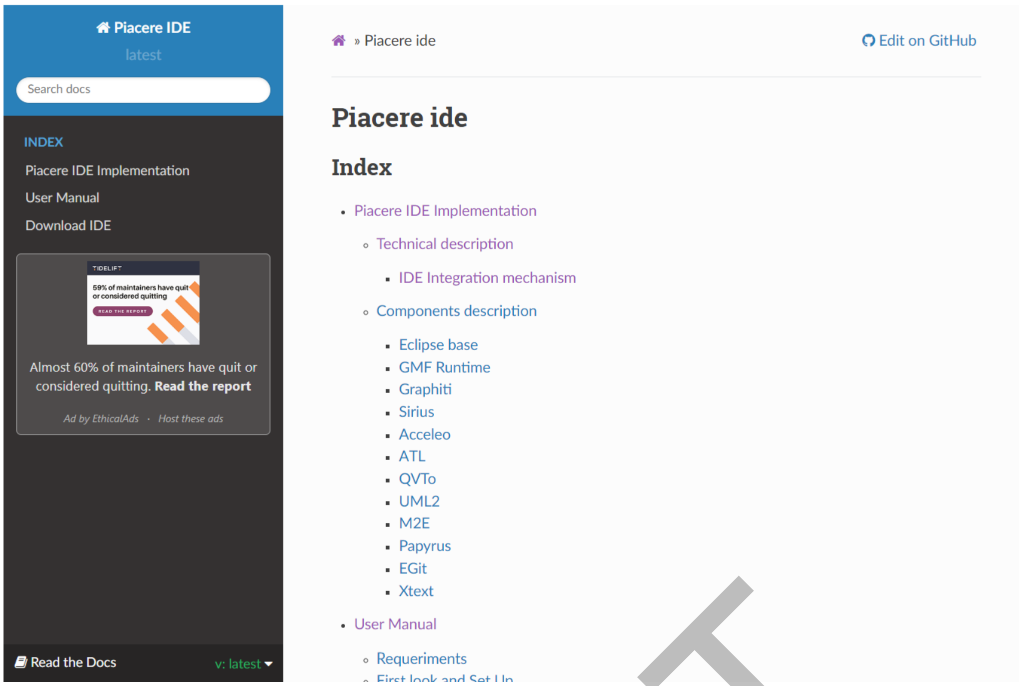
Figure 16: Contents of the ZIP file for Windows.

2.2 User Manual

The documentation to help users has been improved and this information has been placed in "Read the Docs" (an open-sourced free software documentation hosting platform) [19], to make access more convenient.

https://piacere-ide-docs.readthedocs.io/en/latest/

The information contained in the link describes the implementation of the PIACERE IDE with a technical description and components description. A user manual with guidelines to use the tool and, finally, a link to the download site.



Piacere IDE
latest
Search docs
INDEX
Piacere IDE Implementation
User Manual
Download IDE

TIDELIFT
59% of maintainers have quit or considered quitting
READ THE REPORT

Almost 60% of maintainers have quit or considered quitting. **Read the report**

Ad by EthicalAds · Host these ads

Read the Docs v: latest

» Piacere ide Edit on GitHub

Piacere ide

Index

- Piacere IDE Implementation
  - Technical description
    - IDE Integration mechanism
  - Components description
    - Eclipse base
    - GMF Runtime
    - Graphiti
    - Sirius
    - Acceleo
    - ATL
    - QVTo
    - UML2
    - M2E
    - Papyrus
    - EGit
    - Xtext
- User Manual
  - Requeriments
  - First look and Set Up

Figure 17: Main page of PIACERE IDE in Read the Docs.

2.3 Licensing information

The PIACERE IDE is licensed under license Apache 2.0 (https://www.apache.org/licenses/LICENSE-2.0).

2.4 Download

The PIACERE IDE is hosted in the git repository of PIACERE under the project name “IDE”. In the repository, there is a README.md file which explains how download the tool. The IDE are available for different OS: WINDOWS, MAC, and LINUX and for different DOML versions.

The zipped Eclipse IDEs are stored in a Git Large File Storage (LFS) repo (https://git-lfs.github.com/) to minimize the network overload because of this direct download has been excluded. The procedure for a PIACERE IDE user for the installation is the following:

- install git-lfs on your platform
- clone the repo
- choose the IDE for your operative system
- download the selected IDE

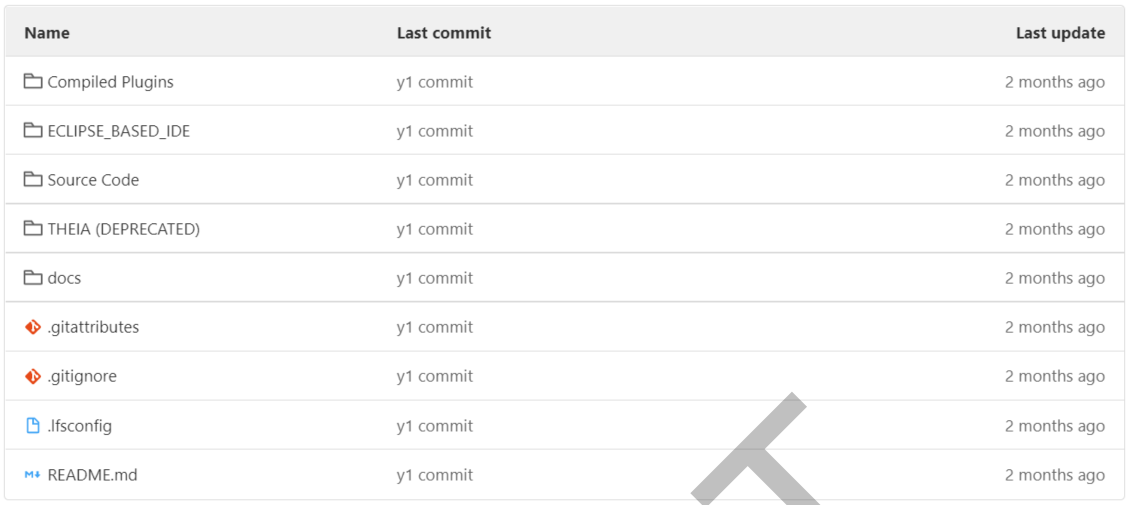
For instance, these are the steps to download the version 2.1 for Windows:

```
install git-lfs

git config lfs.activitytimeout 120

git clone https://git.code.tecnalia.com/piacere/private/ide_tool.git

cd ide_tool

ls -l "ECLIPSE_BASED_IDE/DOML 2.1/WIN/Piacere.zip"

git lfs pull --exclude= --include "ECLIPSE_BASED_IDE/DOML
2.10/WIN/Piacere.zip"
```

```
ls -l "ECLIPSE_BASED_IDE/DOML 2.1/WIN/Piacere.zip"
```

The URL of the project is:

https://git.code.tecnalia.com/piacere/public/the-platform/ide



Name	Last commit	Last update
Compiled Plugins	y1 commit	2 months ago
ECLIPSE_BASED_IDE	y1 commit	2 months ago
Source Code	y1 commit	2 months ago
THEIA (DEPRECATED)	y1 commit	2 months ago
docs	y1 commit	2 months ago
.gitattributes	y1 commit	2 months ago
.gitignore	y1 commit	2 months ago
.lfsconfig	y1 commit	2 months ago
README.md	y1 commit	2 months ago

Figure 18: Folder structure of IDE in public repo.

The folder structure is as follows: on the one hand there are the compiled plugins, the second folder is where the IDE versions for the different versions of the DOML are, and within them there are folders for each supported operating system (be careful selecting branch "y3").There is also a source code folder and another one named Theia (deprecated) and a folder with documentation that contains the same information that is presented in Read the Docs.