PIACERE

Deliverable D3.7

PIACERE IDE - v1

Editor(s):	Marc Gil, Alfonso de la Fuente, Ismael Torres
Responsible Partner:	Prodevelop
Status-Version:	Final – v1
Date:	23.12.2021
Distribution level (CO, PU):	Public

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	PIACERE IDE – v1
Due Date of Delivery to the EC	30.11.2021

Work package responsible for the Deliverable:	WP3 - Plan and create Infrastructure as Code
Editor(s):	Ismael Torres (PRODEVELOP)
Contributor(s):	Alfonso De la Fuente, Marc Gil and Ismael Torres (PRODEVELOP), Elisabetta Di Nito (POLIMI), Leire Orue-Echevarria (TECNALIA), Matija Cankar (XLAB)
Reviewer(s):	Laurentiu Niculut, Lorenzo Blasi (HPE)
Approved by:	All Partners
Recommended/mandatory readers:	WP2, WP3, WP4, WP5, WP6 and WP7

Abstract:	This deliverable is the output of Task 3.5 and it will also be iterative. 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, THEIA, EMF
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	01.10.2021	First draft version. Including desktop version of the IDE	PRODEVELOP
V0.2	29.11.2021	second draft version. Including web version of the IDE	PRODEVELOP
V0.3	07.12.2021	Comments and suggestions received by consortium partners	POLIMI, TECNALIA and XLAB,
V0.5	12.12.2021	Internal review 1	7BULLS, TECNALIA
V0.6	12.12.2021	Updated version with review 1 comments	PRODEVELOP
V0.9	20.12.2021	Final version of the deliverable	PRODEVELOP
V1.0	23.12.2021	Ready for submission	TECNALIA

Table of contents

List of tables

List of figures

Terms and abbreviations

API	Application Programming Interfaces
ATL	Atlas Transformation Language
AWS	Amazon web services
CSS	Cascade Style Sheet
DI	Dependency Injection
DOML	DOML DevSecOps Modelling Language
DOML-E	DOML Extensions
DSL	Domain Specific Language
EMF	Eclipse Modeling Framework
GA	Grant Agreement
GLSP	Graphical Language Server Platform
IaC	Infrastructure as Code
ICG	Infrastructural Code Generator
IDE	Integrated Developed Environment
JDT	Java Development Tools
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
KR	PIACERE key Result
LSP	Language Server Protocol
QVT	Query/View/Transformation)
RCP	Rich Client Platform
REST API	Representational State Transfer Application Programming Interface
SaaS	Software as a Service
SVN	Apache Subversion
UML	Unified Modelling Language
VS Code / VSC	Visual Studio Code
VT	Verification Tool
XML	eXtensible Markup Language

Executive Summary

This deliverable is the output Task 3.5 and it will be iterative. The current version of the deliverable is the v1 and contains the output of the Task 3.5 after 12 months of work.

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. This deliverable is composed of the software prototype of the PIACERE IDE and the associated technical information.

This document starts with the analysis of possible technologies that could be used to develop the PIACERE IDE, with an explanation of why Eclipse Theia, a recent promising technology, was chosen for the implementation of the IDE. The key requirements considered for selecting the best technological option were on one hand. the possibility to offer the tool as a SaaS and on the other, to provide support to UML models and profiles.

The main contribution of this deliverable is the development of the initial version of the PIACERE IDE together with the technological and functional description of the IDE. This initial version is available in the PIACERE Gitlab repository along with instructions on how to install it.

The following versions v2 and v3 will be delivered in months 24th and 30th respectively. In these deliverables, new versions of the IDE will be presented together with the corresponding updated technical information. Those new versions of the IDE, will provide support for the new versions of the DOML and will integrate the rest of the PIACERE Key Results (KR).

1. Introduction

1.1. About this deliverable

This deliverable is the output of task 3.5 and describes the status of the Integrated Development Environment (IDE) after 12 months of work. This is the first version of the deliverable. The following versions are planned for months 24th and 30th respectively. In these deliverables, new or updated versions of the IDE will be presented.

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. This deliverable describes the analysis of possible technologies evaluated to develop the PIACERE IDE, with an explanation of why Eclipse Theia, a recent promising technology, was chosen for the implementation of the IDE.

The main contribution of this deliverable is the development of the initial version of the PIACERE IDE together with the technological and functional description of the IDE. This initial version is available in the PIACERE repository along with instructions on how to install it.

1.2. Document structure

This document is structured in the following sections:

1. **Introduction:** In this section an overall description of the delivery and its main goal is provided.
2. **Analysis of possible technologies to develop an IDE:** This section describes the candidate technologies for developing an IDE. Technology is evolving very fast, and in the last few years new technologies have appeared with which to develop IDEs. In this section, traditional versions have been analysed along with the newer ones that allow collaborative online work.
3. **PIACERE IDE implementation.** This section describes the architecture and the current status of the PIACERE IDE.
4. **Delivery and usage.** This section contains the installation and the user manuals.
5. **Conclusions and future work:** Where a summary of insights gained through this deliverable are presented briefly.
6. **References:** Where relevant additional documentation is presented as citations.

2. Analysis of possible technologies to develop IDEs

This section describes the candidate technologies for developing the PIACERE IDE. As its name indicates, the tool to be developed is an Integrated Development Environment and therefore it must incorporate the typical functionalities offered by this type of tools nowadays.

As indicated in the GA, a first candidate will be Eclipse, but other options may be considered. In particular, a special attention will be posed to the possibility to exploit a web-based approach, that will allow the IDE to be made available as a service, for exploitability and sustainability.

The two most important features to take into account when selecting the technology to build the IDE have been, 1) the possibility to offer the tool as a SaaS and 2) to have support UML models and profiles. These models will be used by other PIACERE Key Results within the distinct applications of the PIACERE framework. In addition, it must ensure that these models can be connected in any way to follow the lifecycle of a PIACERE solution.

The different tools analysed are described below, and then some tool pairs are compared. The section concludes with the technology selected to develop the PIACERE IDE.

2.1. IDE Candidates

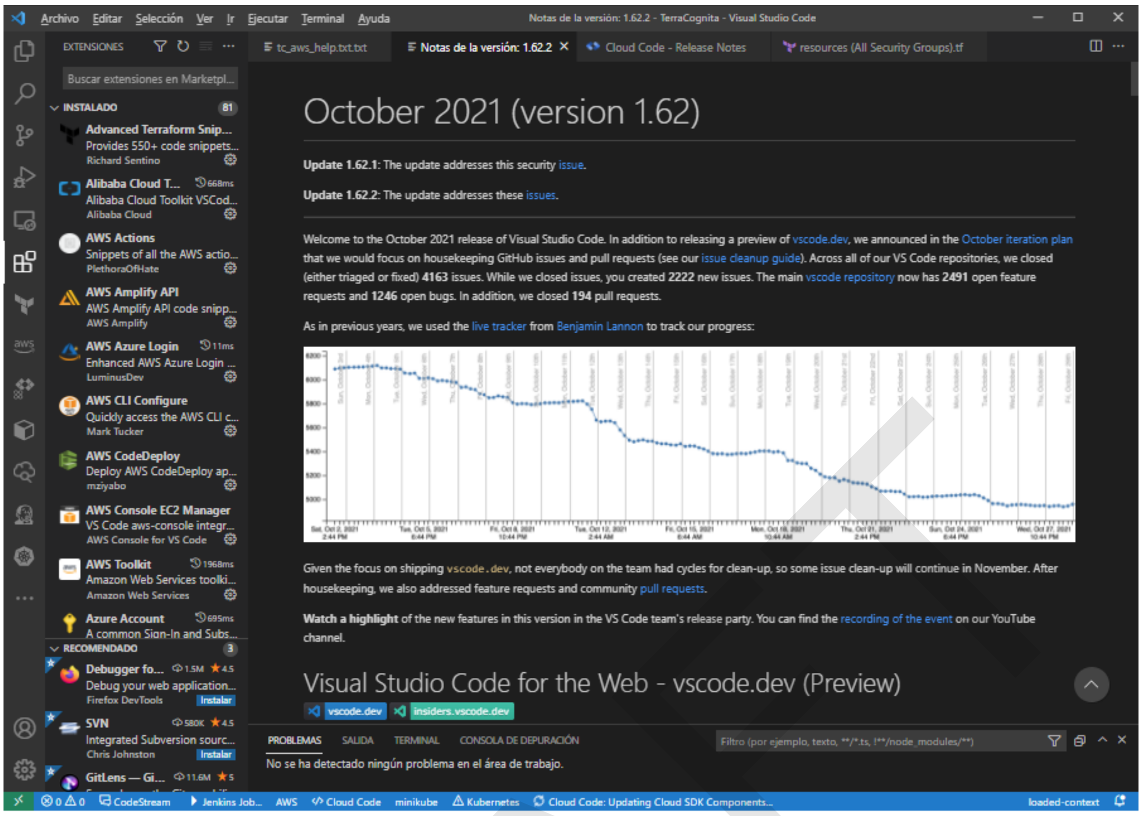
Part of the information used to describe the different IDEs analysed comes from the official website of the products.

2.1.1. VS Code

Visual Studio Code (VSC) IDE [1] is a desktop application mainly developed by Microsoft. It is an extensible code editor heavily based on keyboard commands and web technology but deployed on the Electron framework as a desktop application. It is written in Typescript and HTML, and it is available for all main operating systems.

The VSC IDE is free of charge, so it can be downloaded and used for free, and it is released as open source. Nevertheless, not all the product is open source, because it contains some parts that are proprietary. VS code´s multi-purpose plug-in system greatly extends its functionality. The software runs on macOS, Windows 10, and various Linux distributions.

VSC provides a very efficient and easy to use extension model, proven by the thousands of extensions available for it. The users/developers of VSC have a marketplace where they can share their work and obtain embedded extensions to expand the functionality of the editor so to meet their particular needs.


Archivo
Editar
Selección
Ver
Ir
Ejecutar
Terminal
Ayuda
Notas de la versión: 1.62.2 - TerraCognita - Visual Studio Code
EXTENSIONES
tc_aws_help.txt.txt
Notas de la versión: 1.62.2
Cloud Code - Release Notes
resources (All Security Groups).tf
Buscar extensiones en Marketpl...
INSTALADO 81
Advanced Terraform Snip...
Provides 550+ code snippets...
Richard Sentino
Alibaba Cloud T... 668ms
Alibaba Cloud Toolkit VSCod...
Alibaba Cloud
AWS Actions
Snippets of all the AWS actio...
PlethoraOfHate
AWS Amplify API
AWS Amplify API code snipp...
AWS Amplify
AWS Azure Login 11ms
Enhanced AWS Azure Login ...
LuminusDev
AWS CLI Configure
Quickly access the AWS CLI c...
Mark Tucker
AWS CodeDeploy
Deploy AWS CodeDeploy ap...
mziyabo
AWS Console EC2 Manager
VS Code aws-console integr...
AWS Console for VS Code
AWS Toolkit 1968ms
Amazon Web Services toolki...
Amazon Web Services
Azure Account 695ms
A common Sign-In and Subs...
RECOMENDADO 3
Debugger fo... 1.5M 4.5
Debug your web application...
Firefox DevTools
Instalar
SVN 580K 4.5
Integrated Subversion sourc...
Chris Johnston
Instalar
GitLens — Gi... 11.6M 5
Sat, Oct 2, 2021 2:44 PM
Tue, Oct 5, 2021 6:44 PM
Fri, Oct 8, 2021 10:44 PM
Tue, Oct 12, 2021 2:44 AM
Fri, Oct 15, 2021 6:44 AM
Mon, Oct 18, 2021 10:44 AM
Thu, Oct 21, 2021 2:44 PM
Sun, Oct 24, 2021 6:44 PM
Wed, Oct 27, 2021 10:44 PM
vscode.dev
insiders.vscode.dev
PROBLEMAS
SALIDA
TERMINAL
CONSOLA DE DEPURACIÓN
Filtro (por ejemplo, texto, **/*.ts, !**/node_modules/**)
0 0
CodeStream
Jenkins Job...
AWS
Cloud Code
minikube
Kubernetes
Cloud Code: Updating Cloud SDK Components...
loaded-context

October 2021 (version 1.62)

Update 1.62.1: The update addresses this security issue.

Update 1.62.2: The update addresses these issues.

Welcome to the October 2021 release of Visual Studio Code. In addition to releasing a preview of vscode.dev, we announced in the October iteration plan that we would focus on housekeeping GitHub issues and pull requests (see our issue cleanup guide). Across all of our VS Code repositories, we closed (either triaged or fixed) **4163** issues. While we closed issues, you created **2222** new issues. The main vscode repository now has **2491** open feature requests and **1246** open bugs. In addition, we closed **194** pull requests.

As in previous years, we used the live tracker from Benjamin Lannon to track our progress:

Given the focus on shipping `vscode.dev`, not everybody on the team had cycles for clean-up, so some issue clean-up will continue in November. After housekeeping, we also addressed feature requests and community pull requests.

Watch a highlight of the new features in this version in the VS Code team's release party. You can find the recording of the event on our YouTube channel.

Visual Studio Code for the Web - vscode.dev (Preview)

No se ha detectado ningún problema en el área de trabajo.

Figure 1: VSC IDE screenshot (source: VSC)

This IDE includes some innovations and is continuously under development thanks to its large user's community, most of them from Microsoft, that takes care of maintaining and improving it.

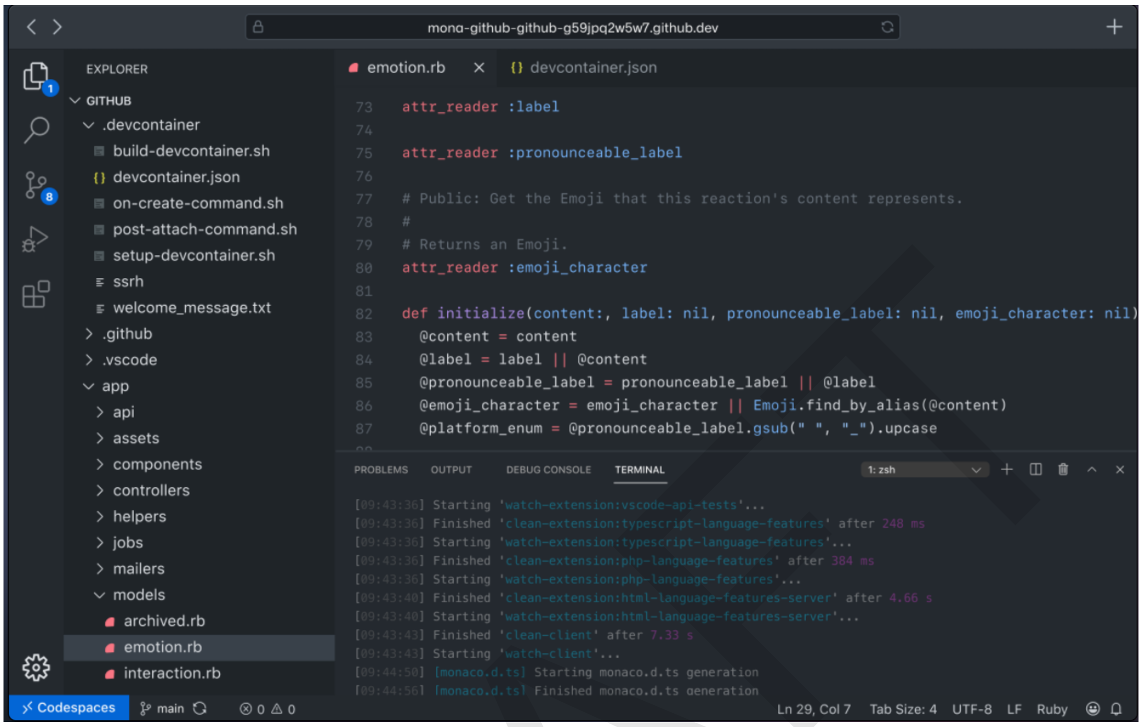
One of the most important components in VSC is the text editor, named Monaco Code Editor [2], that is part of the Visual Code Project, and free to use. This editor was embedded in IDEs other than VS Code, for instance: VS Codium [3] (completely free and open source), or Eclipse Theia [4].

Conclusions of VSC

VS Code is one of the most used IDEs by programmers today because of its large number of extensions and the possibility to be used with many languages and purposes. However, VS Code is not a tool intended to make tailor-specific and customized IDEs and does not support Eclipse Modeling Framework (EMF), which is a set of Eclipse plug-ins which can be used to model a data model and to generate code or other output based on this.

2.1.2. GitHub Codespaces (VS Code Online)

GitHub Codespaces [5] is an online version of Visual Studio Code, hosted by Microsoft in Azure, their Cloud Platform. It runs fully within a software container in the server where it's deployed.



```ruby
attr_reader :label

attr_reader :pronounceable_label

# Public: Get the Emoji that this reaction's content represents.
#
# Returns an Emoji.
attr_reader :emoji_character

def initialize(content:, label: nil, pronounceable_label: nil, emoji_character: nil)
  @content = content
  @label = label || @content
  @pronounceable_label = pronounceable_label || @label
  @emoji_character = emoji_character || Emoji.find_by_alias(@content)
  @platform_enum = @pronounceable_label.gsub(" ", "_").upcase
```

```
[09:43:36] Starting 'watch-extension:vscode-api-tests'...
[09:43:36] Finished 'clean-extension:typescript-language-features' after 248 ms
[09:43:36] Starting 'watch-extension:typescript-language-features'...
[09:43:36] Finished 'clean-extension:php-language-features' after 384 ms
[09:43:36] Starting 'watch-extension:php-language-features'...
[09:43:40] Finished 'clean-extension:html-language-features-server' after 4.66 s
[09:43:40] Starting 'watch-extension:html-language-features-server'...
[09:43:43] Finished 'clean-client' after 7.33 s
[09:43:43] Starting 'watch-client'...
[09:44:50] [monaco.d.ts] Starting monaco.d.ts generation
[09:44:56] [monaco.d.ts] Finished monaco.d.ts generation
```

Figure 2: Github Codespaces screenshot (source: Github.com)

This IDE includes the editor, terminal, debugger, version control, settings sync, and the entire ecosystem of extensions. It provides a lightweight Visual Studio Code experience entirely in the browser. The web-based editor allows to browse source code repositories from GitHub and to make lightweight code changes. It can open any repository, fork, or pull request in the editor, which has many of the features of VS Code, including search and syntax highlighting. For running or debugging the code, it is however, necessary to switch to the cloud-hosted environment or the VS Code desktop.

Itis not possible to access freely to GitHub Codespaces because itis in a beta phase where only some GitHub users have access. GitHub Codespaces will not be free of use. Its cost will vary, depending on the development needs [39].

GitHub Codespaces Extensions
GitHub Codespaces has an extension model called **GitHub Codespaces extensions.**

GitHub Codespaces extensions provide a defined API [14] which empowers the developer to write plugins that use it to basically influence how the tool behaves. This offers the following possibilities:

- Add menu entries
- Hook into events (file save, etc)
- Add compilers
- Add language servers
- Add new views

The extensions run in a separated isolated process, so it cannot harm the stability of the tool. Extensions can be installed at runtime and used immediately.

GitHub Codespaces provides multi-purpose cloud-hosted development environments. There exists a marketplace where extensions can be downloaded and installed within GitHub Codespaces. There exists a lot of extensions available to download, including the same ones than VS Code for desktop has. Codespaces can be connected from VS Code or a browser-based editor that has been made accessible from any online location.

The terms of use of the VS Code marketplace do not allow to install extensions from other applications that are not VS Code.

Workspaces

The workspaces of the IDE and their configuration are hosted in the cloud, in the containers cluster where the IDE is installed.

There exist some options that allow the developer to add and install shell scripts which execute and install all the necessary tools needed for the development on each workspace.

Live sharing, where users can connect to instances of the tool and do a pair programming with other developers, is supported. That is the collaborative coding approach.

License

- MIT license, governed by Microsoft. They decide who can contribute or not
- More than 400 contributors to the project. However, top contributors are affiliated with Microsoft.

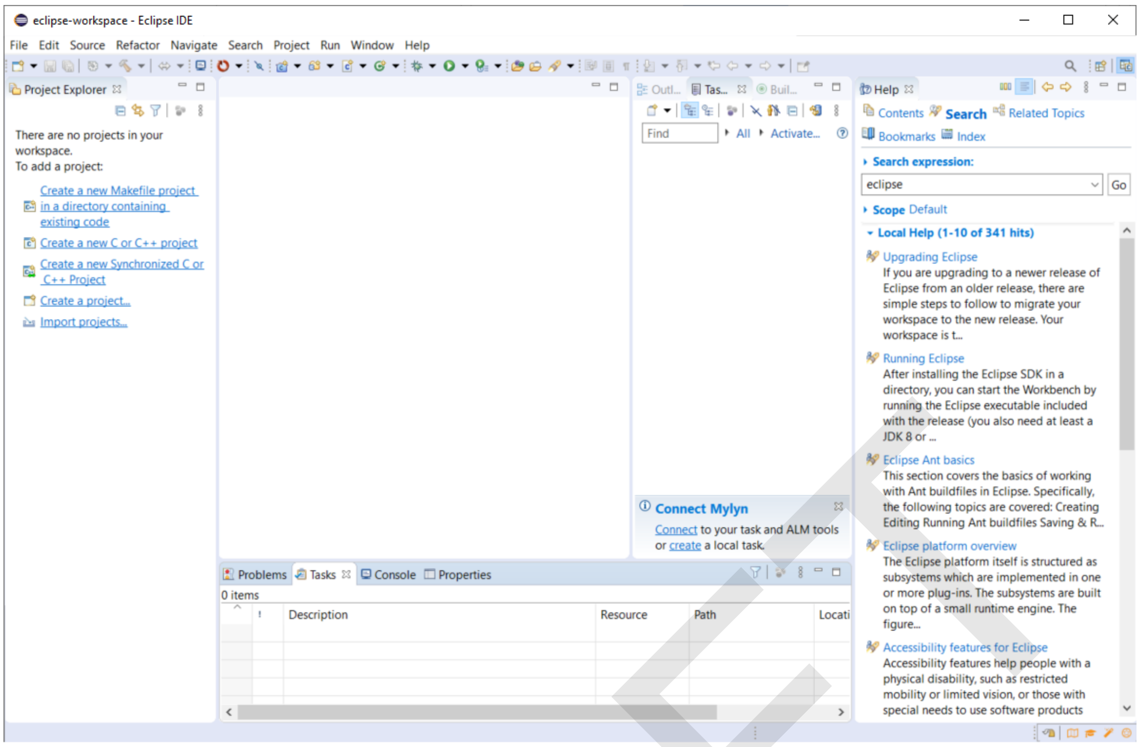
Conclusions of GitHub Codespaces

GitHub Codespaces [5] is an online version of Visual Studio Code, and it has the same issues as VC code for developing an ad-hoc IDE based on and for models.

2.1.3. Eclipse IDE + Eclipse Modeling Framework

The Classical Eclipse IDE [6] 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 3: Eclipse IDE screenshot (source: Eclipse)

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 such as SVN or GIT.

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

Eclipse is an IDE that can be used to create ad-hoc IDEs/products. In turn, these products can be extended with other plugins in order to fulfil the developers' needs. For example, Papyrus is a product created with Eclipse. Papyrus can be therefore downloaded and used as modelling application, but additional plugins to develop applications in Java and others can also be installed.

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 why 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. It is possible to 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**: itis a marketplace where the developer 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 the developer 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. However, this is not considered 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 it 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 modelling 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.

emf
ECLIPSE MODELING FRAMEWORK

Eclipse Modelling Framework [7] 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 very interesting to consider. It also provides transaction and validation on the models.

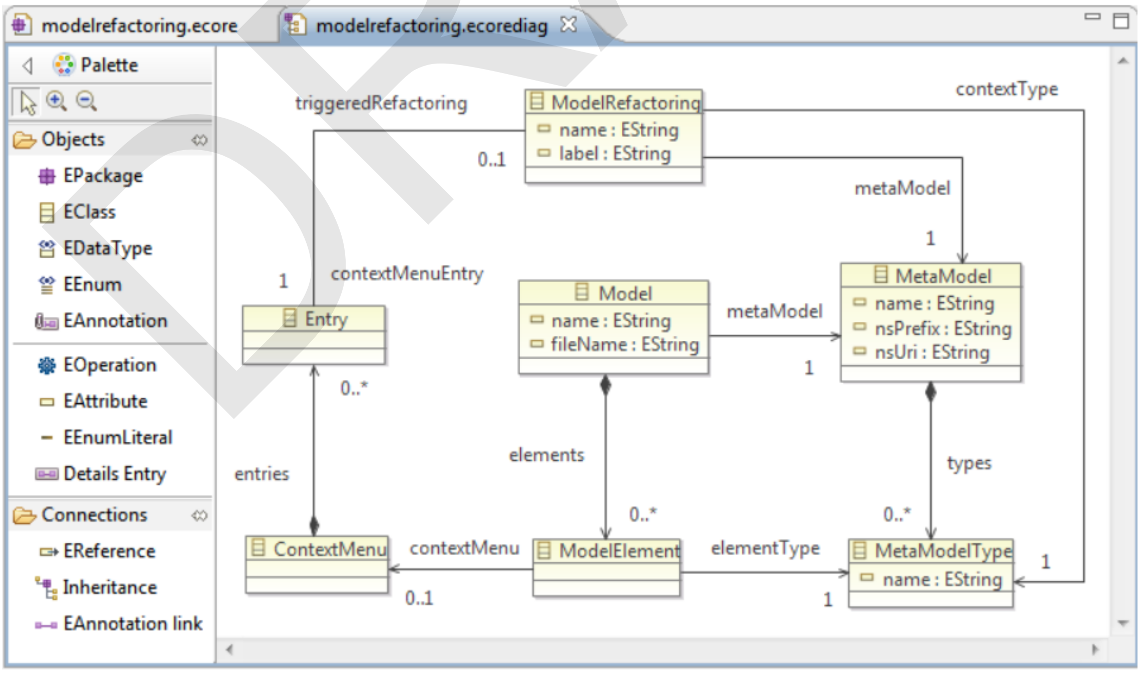


Figure 4: Eclipse Modeling Framework screenshot (source: Eclipse Modeling Framework)

EMF is considered one of the best and more expanded modelling languages, that is the base of the models. On top of it, other languages bring support for graphical editors, like GMF [24] or Graphiti [25]. 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 [28], QVT [29], XPAND [36], or Xtext [34].

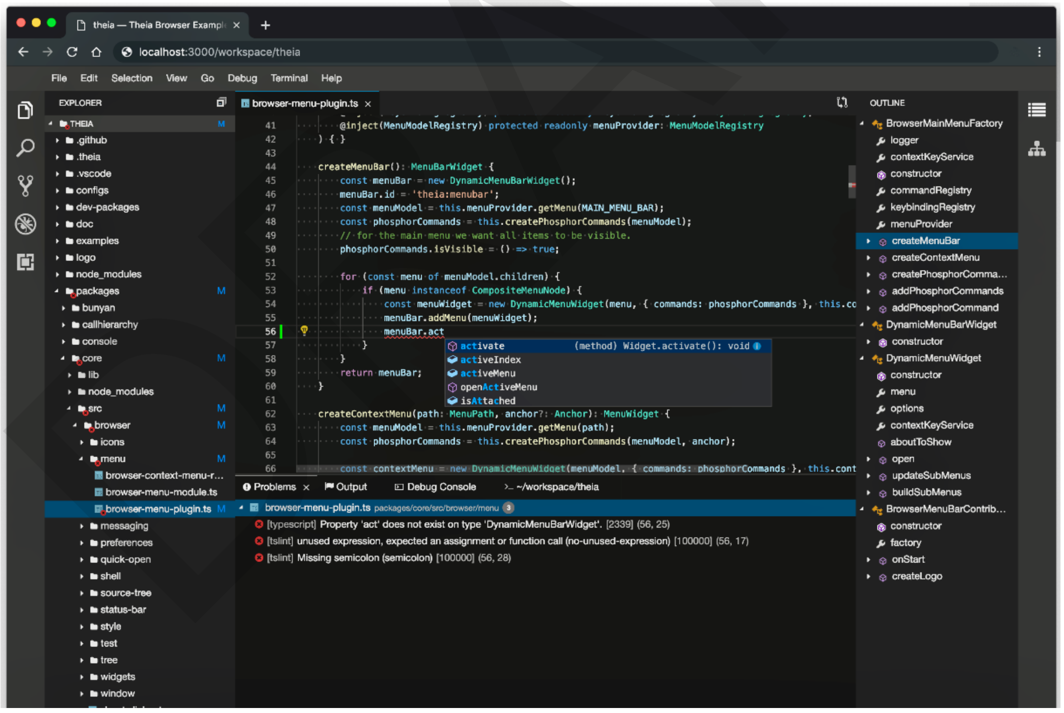
License
Eclipse IDE is free, open source and uses the Eclipse Public License.

Conclusions of Eclipse IDE + Eclipse Modeling Framework

Eclipse + EMF have been for years the favourite technologies to build IDEs based on models. EMF was designed with the specific purpose of developing customized IDEs, as it has native support for models and has a series of utilities and plugins for this purpose. This technology has been on the market for almost 20 years and is well established. The solutions based on these technologies can be offered as desktop solutions. It is not allowed to be offered in SaaS mode in the cloud.

2.1.4. Eclipse Theia + EMF.cloud

Eclipse Theia [4] is a completely open-source project hosted by Eclipse Foundation with a modular and flexible architecture. Theia is not a tool per se, but by definition, itis more of an open-source platform for building web-based tools and IDEs and developing under vendor-neutral open-source governance, which supports the VS Code Extension protocol.



```
        @inject(MenuModelRegistry) protected readonly menuProvider: MenuModelRegistry
    ) { }

    createMenuBar(): MenuBarWidget {
        const menuBar = new DynamicMenuBarWidget();
        menuBar.id = 'theia:menubar';
        const menuModel = this.menuProvider.getMenu(MAIN_MENU_BAR);
        const phosphorCommands = this.createPhosphorCommands(menuModel);
        // for the main menu we want all items to be visible.
        phosphorCommands.isVisible = () => true;

        for (const menu of menuModel.children) {
            if (menu instanceof CompositeMenuNode) {
                const menuWidget = new DynamicMenuWidget(menu, { commands: phosphorCommands }, this.co
                menuBar.addMenu(menuWidget);
                menuBar.act
            }
        }
        return menuBar;
    }

    createContextMenu(path: MenuPath, anchor?: Anchor): MenuWidget {
        const menuModel = this.menuProvider.getMenu(path);
        const phosphorCommands = this.createPhosphorCommands(menuModel, anchor);

        const contextMenu = new DynamicMenuWidget(menuModel, { commands: phosphorCommands }, this.cont
```

```
activate                 (method) Widget.activate(): void
activeIndex
activeMenu
openActiveMenu
isAttached
```

```
browser-menu-plugin.ts packages/core/src/browser/menu 3
[typescript] Property 'act' does not exist on type 'DynamicMenuBarWidget'. [2339] (56, 25)
[tslint] unused expression, expected an assignment or function call (no-unused-expression) [100000] (56, 17)
[tslint] Missing semicolon (semicolon) [100000] (56, 28)
```

Figure 5: Eclipse Theia screenshot (source: Eclipse)

It is intended to be the basis for a developer to implement tools, regardless of whether those tools are domain-specific, adapted code, or even unrelated to the source files.

Eclipse Theia supports both local and online deployment modes. Local installations are possible using Electron. Theia consists of a frontend, running on a browser or in the local desktop application, and a backend running on any host or locally within the desktop application. The frontend and backend communicate through JSON RPC over web sockets. In this case, itis more accurate to talk about a platform to build tools, and not of a product. An example of that is Eclipse Che [8], that comes with a default IDE that is based on Eclipse Theia.

Eclipse Theia cannot be downloaded and used directly but one can download the source code and build a desktop application with Electron, deploy it on a docker container, or try it directly from Gitpod.

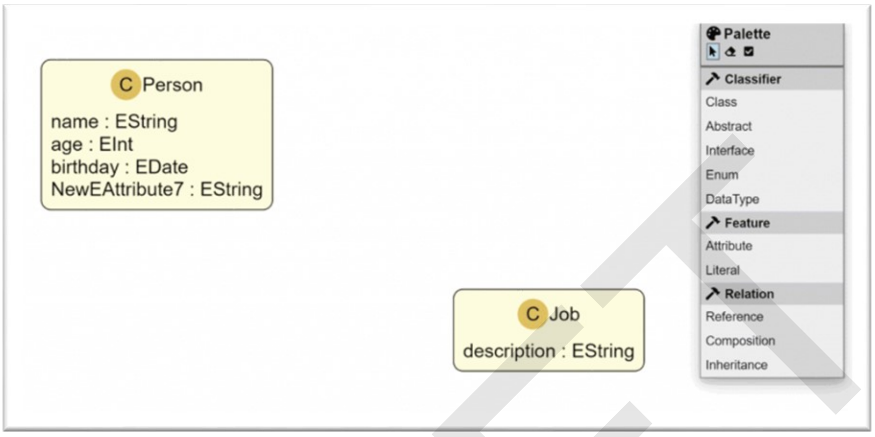
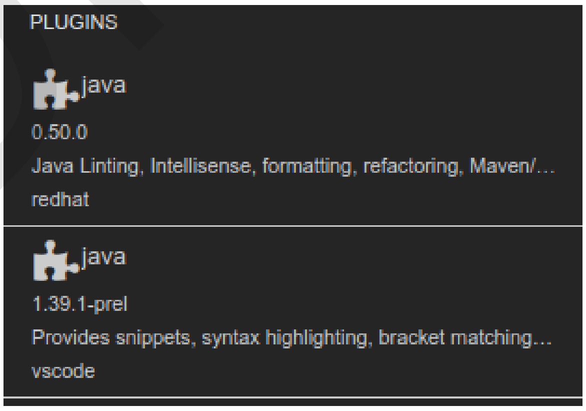


Figure 6: Eclipse Theia Model Example (source: Eclipse)

Theia Extensibility

Eclipse Theia extensions and plugins integrate textual language frameworks such as LSP, Monaco, Eclipse JDT/CDT, Git and Xtext to support web-based and cloud-based IDEs for generic programming languages as well as custom textual Domain-Specific Languages. This includes features such as syntax highlighting, autocompletion, refactoring, formatting, and quick fixes.

The extensions are part of the core of Eclipse Theia [15]. Everything included in the core platform, is an extension. There is no difference between extensions that the user adds by hand and components that were already installed. All the extensions' communications are handled via dependency injection. The extensions installed in Theia are accessible through the Plugins view (see Figure 7**¡Error! No se encuentra el origen de la referencia.**)



PLUGINS

java
0.50.0
Java Linting, Intellisense, formatting, refactoring, Maven/...
redhat

java
1.39.1-prel
Provides snippets, syntax highlighting, bracket matching...
vscode

Figure 7: Theia plugins list

This way of extending the platform has the advantage that the developer can do whatever the core platform can do, like accessing all the available APIs inside the core, overriding some barriers for a dependency injection, and customizing or adapting the tool that the developer is building.

Theia extensions cannot be installed at runtime. The use case for Theia Extensions is to build some extension as a product, to configure it, and then to deploy it, rather than letting developers install their own extensions. However, recent implementations of Eclipse Theia allow to incorporate an API to install also VS Code extensions, but not using the VS Code marketplace, as there is another place where the developers can publish their extensions to allow them to be installed within Eclipse Theia. This site is OpenVSX [16]. The terms of use of this marketplace are more relaxed than VS Code marketplace.

License

- **EPL license, governed by Eclipse Foundation. Itis an independent foundation where** everybody can contribute
- More than 100 contributors. Top contributors are related to different companies

EMF.cloud

EMF.cloud [21] is an Eclipse-based modelling framework and code generation facility for building tools and other applications based on a structured data model. It can be said that EMF.cloud is the web and cloud version of the classic EMF. Many of the EMF plugins have been adapted for this new web version, although new plugins have also appeared specifically for the web version.

This solution is licensed under the Eclipse Public License and the MIT License, therefore free of charge. Out of its scope is any software component that is specific to a particular application domain, including specific modelling languages or code generators for modelling browser- or cloud-based applications.

Conclusions of Eclipse Theia + EMF.cloud

Eclipse Theia + EMF.cloud are new technologies that run in the cloud and fill a gap in the market that is not covered by Eclipse + EMF. These technologies have the handicap that they are not yet fully mature, however with their current state it is possible to develop tools needed in the PIACERE project, as they are also in continuous development and have a very active community.

2.1.5. Eclipse Che

Eclipse Che [8] is not strictly speaking an IDE, it is more of a workspace server "Kubernetes-Native" for open-source developers that allows collaborative/multi-user work. Eclipse Che integrates multiple kinds of IDEs for developers. Eclipse Theia is the default option, but competes with others like VS Code, Jupyter, Classic Eclipse, or IntelliJ. From a hosting point of view, it is more flexible because it can be hosted in any Kubernetes cluster. To achieve that it is just needed to configure a Kubernetes installation in a server and run Eclipse Che on it. Each workspace corresponds with a container in the Kubernetes cluster and can be shared with multiple developers.

The workspaces can be configured in the following terms:

- **Editor**: an Administrator should decide which editor is used on each workspace from those that are available

- **Plugins**: the Administrator decides which plugins are installed in the workspace, meaning those that are most useful for the developers

- **Projects**: A Team Manager can configure multiple projects in a workspace, meaning those where their developers are required to work on. A workspace project normally is a Git project. Just only copying the URL of the Git project is enough to get started.

Eclipse Che makes the works with Docker and Kubernetes simpler by providing a working environment that avoids having to run local installations of Docker and Kubernetes.

Extensibility

Eclipse Che extensions are called plugins. Plugins are installed in the context of a workspace, and there exists a reduced set of official plugins available to be installed. Available plugins also depend on the selected editor for the workspace.

Eclipse Che uses the same syntax as Visual Studio Code for plugins, which means that most of it can be used in both tools without modification.

Despite of this, theoretically it is possible to install any extension if it is available on GitHub, because it is possible to configure it in the Eclipse Che workspace advanced configuration.

Workspaces

The workspaces of Eclipse Che and their configuration are hosted in the cloud, in the containers cluster where the IDE is installed.

Though the "stack management" it is possible to configure and manage stacks, providing a template for the developers.

- Java with Spring Boot projects
- Javascript projects
- Python projects
- ...

Workspaces can be shared simply by creating a one-click URL, configuring a GIT repository to be checked out and the respective stack to be used by the new developers that joins these workspaces.

License

It is completely free , open source and uses the Eclipse Public License.

Conclusions of Eclipse Che

Eclipse Che is not an IDE *per se*, but as it integrates with several IDEs included in this section, its use will be discussed in the coming months, in case it is required to have and IDE with a real multi-user feature.

2.2. Workspaces management

The situation with GitHub Codespaces, Eclipse Theia deployed remotely, or Eclipse Che, is that there exists a container that runs on a remote location that holds all the business logic, the source files, the compiler, the checker, the debugger, etc. Basically, everything that is headless and that interacts with files while consuming some computation power.

This is called a workspace container. The Browser IDE is the User Interface that connects to this workspace container. Therefore, the client application only displays the information, while the server makes the calculations.

When the developers start working, they need to log in into the system and then somehow need to decide which workspaces they are working on. The responsibility of choosing the proper workspace is for the workspace server. It hosts all the workspaces for the developers. It also has a sort of dashboard where developers can configure the workspaces of choice, start and stop them, share with other people, etc. All of these are possible because these containers are deployed on clusters with solutions like Kubernetes [9] or Docker [10].

2.3. IDE baseline selection

This section starts with a comparison of the IDEs described in the previous sections. The considered characteristics for the comparison were:

- **Free of use:** If any type of fee is needed for using the tool or not
- **Open source**: If it is open source or not
- **License type**: The type of license
- **Online**: If the IDE can be accessible via work or locally
- **Share workspaces:** If the workspaces are shared and it is allowed that several users utilise the same workspace.
- **Extendable:** If the IDE could be extended using any extensibility mechanism or not.
- Open developer community. If the IDE has a developer community or not.
- **Support for UML:** If the IDE provides support for UML
- **Maturity:** since which year the IDE is available.

The following table shows a comparison of the different characteristics of the technologies analysed.

Table 1: IDEs Features Comparison (source: PIACERE's own contribution)

	VS Code	Github Codespaces	Eclipse Theia+ EFM.cloud (Selected)	Eclipse Che	Eclipse IDE+ EMF
Free of use	Yes	No	Yes	Yes	Yes
Open source	Not 100%	No	Yes	Yes	Yes
Licence type	MIT	Private	EPL	EPL	EPL
Online	No	Yes	Yes/No	Yes	No
Share workspaces	No	Yes	Yes	Yes	No
Extendable	Yes	Yes	Yes	Yes	Yes
Open developer community	Yes	No	Yes	Yes	Yes
Support for UML	No	No	Yes	No	Yes
Maturity	2015	2020	2017	2016	2001

Depending on the specific aim of the project, one product or the other can be the best choice. Some considerations are described below: .

- **VS Code**: if it is preferable a downloadable application for developing applications.
- **VS Code Online**: if it is preferable an online application for developing applications, and Azure or GitHub are available to the developing group.
- **VS/Github Codespaces**: if use VS Code Online is the preferred option, or if the use of other features than the ones offered by default is not needed.
- **Eclipse IDE**: if a full IDE that supports any kind of development is desired, including modelling software approach, and if developing online is not a real need. Also supports UML modelling and UML Profiles, and contains a lot of plugins to generate code and models
- **Eclipse Che**: if the desired option is to self-host the workspaces in the user's own Kubernetes servers. Also, if Eclipse Theia is chosen as the default editor.
- **Eclipse Theia**: if an online IDE that supports any kind of development is the preferred choice, including modelling software approach. Thanks to EMF.cloud it supports UML modelling and UML Profiles.

After analysing the different technologies that could be used to develop the PIACERE IDE, the viable options if we want the solution to incorporate support for models would be **Eclipse Theia** and **Eclipse IDE**. Both technologies provide support thanks to EMF which is a set of Eclipse plug-ins along with a modelling framework for building tools based on a structured data model in Eclipse. EMF (core) is a common standard for data models.

On one hand, Eclipse IDE + EMF are mature technologies with stable, well-documented versions and a complete set of plugins that would allow us to develop the IDE with minimal risk. The solutions based on these technologies can be offered as desktop solutions. SaaS mode on the cloud is however not supported.

On the other hand, Eclipse Theia + EMF.cloud are new technologies. At the time of writing this document, they are not fully mature but are evolving very fast with a very active community. These technologies allow to develop tools in a SasS mode in the cloud.

In order to have more confidence in using Eclipse Theia, at the conference Eclipsecon 2021 [17], we had the opportunity to ask the Eclipse software architects in the session "Building (web-based) tools with Eclipse" whether it would be possible to develop the PIACERE IDE using Theia. Their recommendations were:

- If it is a new development, do it directly in Theia + EMF.cloud.
- Start developing the parts of the IDE that use the more mature Theia + EMF.cloud technologies and postpone the development of the less mature Theia components. The road map of these technologies includes several important releases in the following months.
- We can contact the community and the developers to raise needs to be included in the roadmap of the products.
- The idea of Eclipse is to evolve towards the web. So, if we want to make a tool that lasts over time, the most sensible thing to do is to make it web-based.
- Follow an incremental approach and start with the headless development. It can be reused in case we have to change to the classic Eclipse IDE.

Although Eclipse Theia + EMF.cloud are not yet complete mature technologies, they are constantly evolving and are the future in the short term. Even if it costs us more effort to develop the IDE with Theia, we are convinced that using it we can develop the IDE for PIACERE.

In the following section, the PIACERE IDE technical details will be explained, including the information related to the Theia version (our first choice) and Eclipse IDE version that the current version uses. This second option has been defined in case we find any blocking issue with Theia.

3. PIACERE IDE Implementation

3.1. Functional description

The central element of the PIACERE DevSecOps Framework (KR13) is the Integrated Development Environment "IDE" (KR2), by which the technical user (Architect, DevOps engineer...) specifies the infrastructure of the application using a model-driven engineering approach. To support this activity, the PIACERE IDE integrates the DOML, a Modeling language used for describing the infrastructure of the application using different levels of abstraction [40].

When developing the IDE for the PIACERE project, the biggest challenge has been in the choice of the technology to be used. On the one hand, there was the possibility of developing a desktop IDE based on the classic version of Eclipse, a mature technology in which several project partners had a lot of successful experience developing IDEs for both R&D and industrial projects. On the other hand, there was the possibility of realising the IDE using very promising emerging Eclipse web technologies, which are not yet consolidated and with the aggravating factor that the consortium lacked experience at the beginning of the project.

After analysing the pros and cons (see previous section), it was finally decided to develop the IDE using Eclipse Theia. This section describes the solution used to develop the IDE with Theia. The alternative solution to be used in case we prefer the classic Eclipse has been added as an annex. This second solution was designed in case we encounter any technological impediment that would prevent us from continuing with the web version solution.

3.1.1. Requirements for the PIACERE IDE

The requirements for the PIACERE IDE are presented as part of Deliverable D2.1. They have been collected through multiple workshops with technical partners.

In this section, It is provided a summary of the PIACERE IDE requirements grouped in three tables, the first one contains the functional requirements, the second one contains the non-functional requirements and the last one contains the business requirements. For each requirement, an explanation of how the requirement is addressed or planned to be addressed is provided.

Table 2: Functional requirements related with the IDE (source D2.1)

REQ ID	Description	How IDE is addressing this requirement
REQ28	DOML should support the modelling of containerized application deployment	The DOML supports the modelling of containerized application deployment. The initial version of the IDE provides full support for the initial version of the DOML. If new properties/elements related with containerization are required and added to the DOML, the IDE will be updated accordingly.
REQ40	The IDE should provide a visual diagram functionality to visualise the different assets defined through the DOML and DOML Extensions.	The current version of the IDE provides a tree editor to manage the DOML instances. In the following versions a visual editor will be add to the IDE. Thanks to this editor, the user will be able to visualize main DOML assets. This editor will not have the capability of editing the DOML.
REQ41	The IDE should be extensible through the	The IDE is based on Theia and Theia extensions. The integrations of the PIACERE KR will be implemented using

REQ ID	Description	How IDE is addressing this requirement
	plugin mechanism. Not only to support PIACERE assets (ICG, VT) but also for third party collaborators.	Theia extension mechanism. Any third-party component/functionality can be integrated using the same approach.
REQ43	The IDE should be easily updatable to newer software versions.	The IDE is based on Theia and Theia extensions. All the work made to build the IDE follows the best Theia development practices including the Dependency Injection (DI) framework to wire up the different components. In the scripts for building/compiling the tool is it established the specific versions to be included. Moreover, a node utility called Yeoman is used for building the solution. This utility is similar to "Maven" and allow us to choose the libraries to be included in the project. For the above reasons, it can be stated that the IDE is ready to easily adapt to the new versions that appear
REQ44	The IDE could provide an import mechanism to automatically fulfil partial DOML.	The current version of the IDE does not support this functionality. In the first iteration of the IDE the efforts have been dedicated to providing support for creating new DOML instances. In the next release, the functionality for importing partial IaC specifications will be provided.
REQ62	DOML must support different views.	DOML allows models to be defined on a per-layer (view) basis. Layers represent different viewpoints on the system. The layers defined are the (i) application layer, (ii) the abstract infrastructure layer, (iii) concrete layer. Currently the IDE provides support for the different views included in the DOML
REQ76	DOML should allow the user to model each of the four considered DevOps activities (Provisioning, Configuration, Deployment, Orchestration).	The DOML supports DevOps activities. The initial version of the IDE provides full support for the initial version of the DOML. If new properties/elements related with DevOps activities are required and added to the DOML, the IDE will be updated accordingly.
REQ99	IDE to integrate with both local and remote Git repositories.	This requirement is not supported yet, but it is planned to support this requirement before the month 15 of the project (March 2022). To support this requirement, specific Git Theia extensions will be evaluated. After this evaluation, the most suitable one for the needs of the IDE will be integrated.
REQ GA	Develop and IDE using web-technologies.	This is a requirement not included in the list of requirements of the D2.1 ,but It has been taken from the description of the Task 3.5 in the Grant Agreement ***"a first candidate will be Eclipse, but other options will be considered. In particular, a special attention will be posed to the possibility to exploit a web-based approach, that will allow the IDE to be made available as-a-service "*** This is the requirement that has most conditioned the development of IDE, because after carrying out the initial

REQ ID	Description	How IDE is addressing this requirement
		study of possible technologies to develop IDE, it was decided to use the classic Eclipse. But after a second in-depth review of Eclipse Theia in which a prototype was developed, it was decided to use Eclipse Theia, which allows us to obtain a Web editor.

Table 3. Non-Functional requirements related to the IDE (source D2.1)

REQ ID	Description	How IDE is addressing this requirement
REQ42	The IDE should be implemented using open-source software.	All the software artifacts used by the IDE are open source.

Table 4. Business requirements related to the IDE (source D2.1)

REQ ID	Description	How IDE is addressing this requirement
REQ64	The IDE should provide a text-based representation of DOML to ease version control.	The IDE provides a textual representation of the DOML instances created. For each of these instances a JSON file is created. The user can create as many versions of an instance as it is needed. These files are stored in the workspace of the project, but in the future an integration with a git repository will be added.

3.1.2. 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). At the technological level, the IDE has been developed using the Eclipse Theia Framework, a technology used to create customized tools or IDEs.

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 will integrate the Verification Tool (VT) and the Infrastructural Code Generator (ICG). Thanks to the VT, it will be possible to validate the defined DOML instance. On the other hand, the ICG tool, when triggered from the IDE, will automatically generate the corresponding IaC for a specific target environment (e.g., Terraform, Ansible, TOSCA, …) from a DOML instance.

All the information produced at design time will be 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 will be also linked with the IDE. The runtime controller (PRC) will be invoked through the IDE. This component will be in charge of controlling the deployments and linking them with the Infrastructure Advisor components.

A detailed description of the PIACERE Design time and Run time workflows can be found in the Deliverable D2.1.

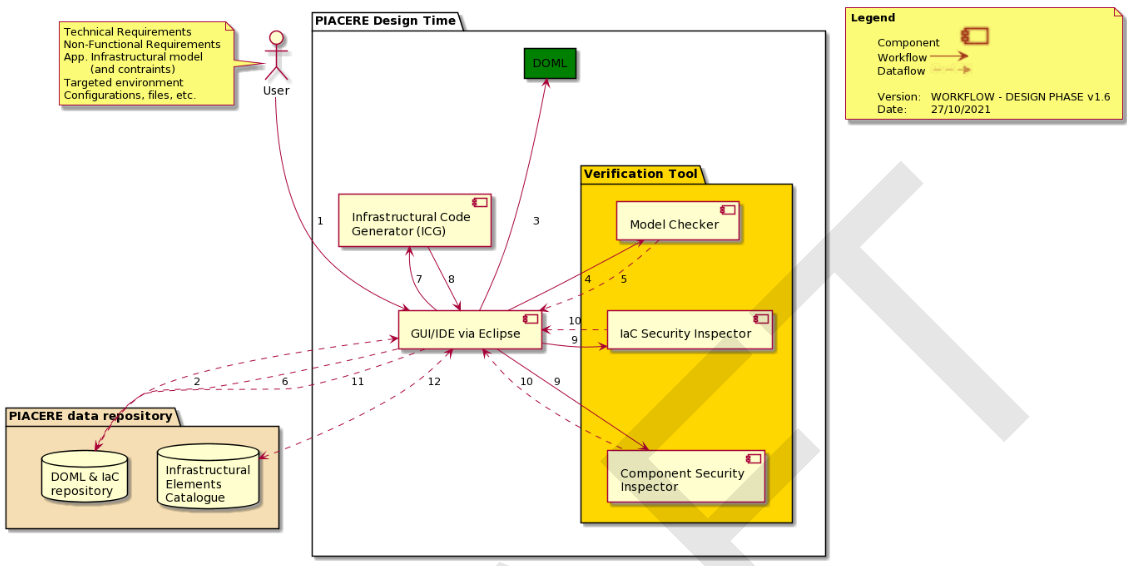


Figure 8: PIACERE Design time Components

3.2. Technical description

This section describes the PIACERE IDE using Eclipse Theia + EMF.cloud, the technologies finally chosen for developing the PIACERE IDE.

3.2.1. Prototype Architecture

Eclipse Theia [18,19] is a framework for developing multi-language Cloud and Desktop Integrated Development Environment (IDE)-like products. It is implemented in TypeScript and is based on Visual Studio Code and stands out for its extensibility.

Theia supports multiple languages through the Language Server Protocol as well as many VS Code extensions. The installation and deployment of Eclipse Theia depend on existing frameworks, such as Docker and Electron. Eclipse Theia needs to be deployed on cloud infrastructure providers [18].

Eclipse Theia has a frontend and a backend running. Their communication is through JSON RPC over Web Sockets. Theia allows to develop an IDE and run it on browsers or native desktop applications from a single source. It supports three architecture configurations [18], as illustrated in the following Figure:

- **Option1.** Native frontend "Desktop" and Remote backend.
- **Option 2.** Native frontend "Desktop", local backend.
- **Option 3.** Web Client, remote backend. Eclipse Theia renders a user interface on the web browser for the frontend. The backend runs on a host.

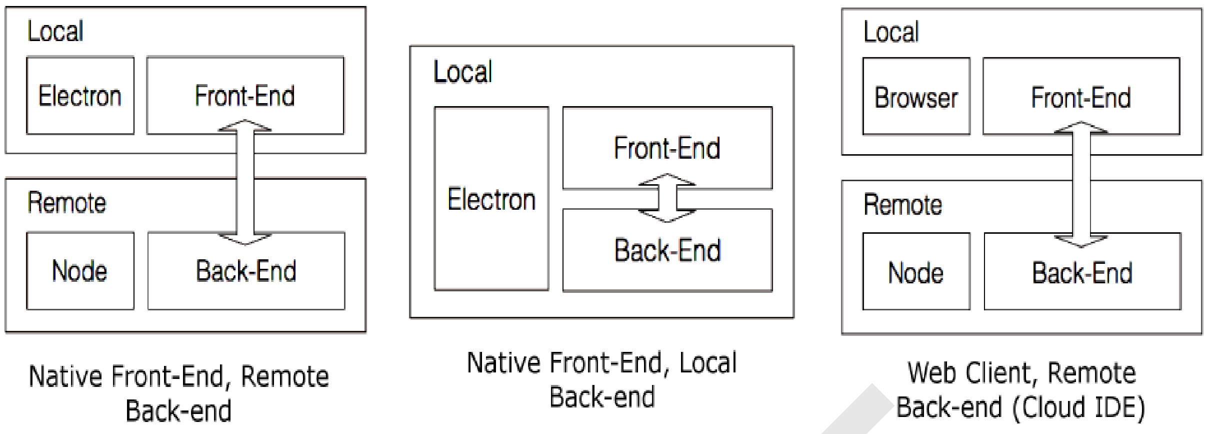


Figure 9: Theia Architecture configuration. (Source desosal.nl [18])

Theia is an extensible and flexible framework that has an architecture based on components. In Theia everything is an extension and the components are loosely coupled, allowing an easy integration of new /updated components without affecting the others.

THEIA CONTAINER VIEW

A container is an isolated element that provides a specific functionality. Each container is a separate deployable object or runtime environment. Theia is composed of five containers [19]:

- The **frontend** application provides a client User interface that runs as a single page application, which can be hosted in browsers and in an Electron Browser Window.
- The **backend** runs in Node.js and can be deployed locally or remotely, communicating with the frontend application.
- The **language server** dissociates language-related features and functions from the IDE. Theia provides different language servers for each supported programming language.
- **Debugger servers** extend debug UI and support debug function.
- The **file system** container provides access to the file system that controls how data is stored and retrieved.

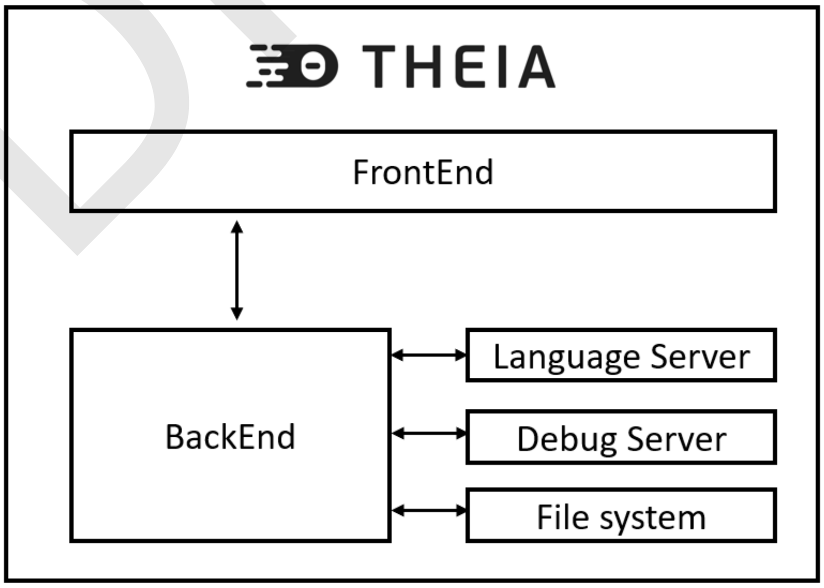


Figure 10: Theia containers (adopted from [19])

Eclipse Theia can be customised to offer tailor specific functionalities, which can be achieved by updating or adding new extensions to the frontend and/or backend containers.

THEIA COMPONENT VIEW

Theia containers are made up of components. Figure 11 shows the details of the components included in the different Theia containers. These components can be categorised into 4 types [19]:

- **Extension Component (green):** extends the IDE's functions. The extensions are done by a developer. It depends on the platform components.
- **Platform Component (yellow):** main components for the IDE platform. It depends on the Runtime components.
- **Runtime Component (red):** supports Theia's runtime environment.
- **External Resource Component (blue):** external resources needed by Theia.

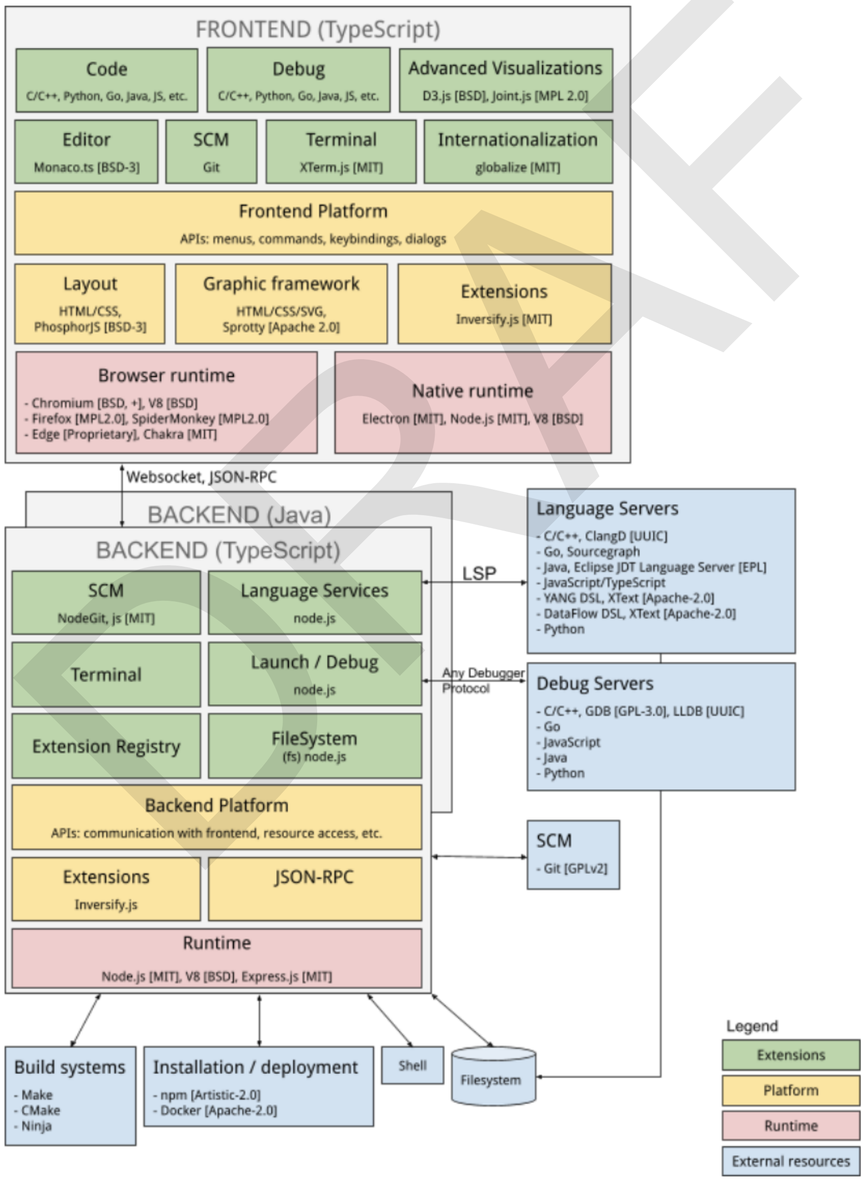


Figure 11: Theia components detail (Source desosal.nl [19])

THEIA CONNECTOR VIEWS

For the communication among containers connectors are used. The connector types supported on the container level are: JSON-RPC protocols, REST APIs, Language Server Protocol (LSP), and some debugger protocols.

On the component level, the components in Theia are all loosely coupled. To write up the different components, Eclipse Theia uses the Dependency Injection (DI) framework Inversify.js [37]. The dependencies are created by the DI on creation for each component, based on some configuration provided by the developer on start-up through so-called container modules.

THEIA EXTENSION MECHANISMS

Theia supports three extension mechanisms [20] to adapt the IDE to the needs of each project. These extension mechanisms, described next, can be used in isolation or combine them in the IDE.

- **VS Code extensions**: compatible with VS Code, limited to the VS Code extensions API, as some use cases are not possible due to API restrictions. They are used to add features to existing tools. Eclipse Theia provides the same extension API as VS Code, which make extensions compatible among each other.
- **Theia Extensions**: Installed at compilation time, full access to Theia internals is provided through the Dependency Injection, meaning almost no limitations in terms of accessible API. All Theia extension have access to the same API as the core extensions. This modularity allows to extend, adapt or remove almost anything in Theia according to the user's requirements. Theia project is entirely built using Theia extensions in a modular way. These extensions, however, cannot be used within VS Code
- **Theia plugins**: Like VS Code extensions, additional access to some Theia-specific APIs and the Frontend (frontend plugins). Theia plugins are a special type of VS Code extensions that only run in Eclipse Theia.

Figure 12 depicts the extension mechanisms in Theia. VS Code extensions and Theia plugins run in a dedicated process, can be installed at runtime and use a specific API. Theia extensions are added during compilation time, becoming a core part of the user's Theia application, and having access to the full Theia API.

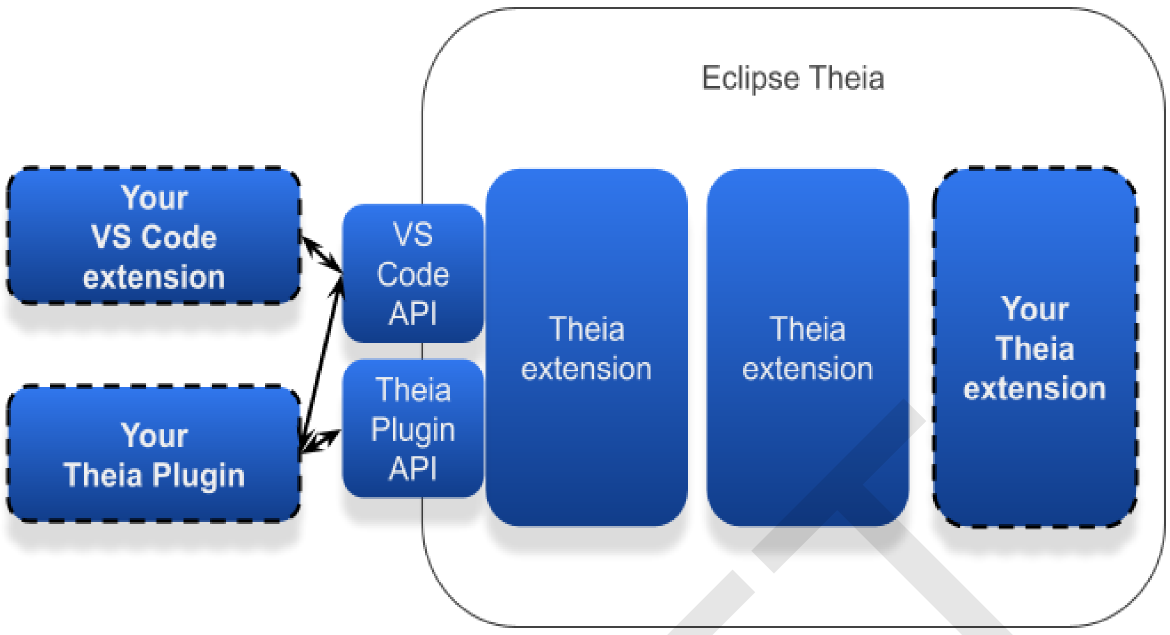


Figure 12: Theia Extension mechanisms (Source theia-ide.org [20])

3.2.2. Components Description

In the APPENDIX: PIACERE IDE based on Eclipse desktop, the plugins that would be used for the alternative desktop version of the IDE have been described. In Eclipse Theia the plugin concept also exists, but the plugins developed for the desktop version are not compatible with Theia and therefore a reprogramming process is needed in order to adapt them, especially those parts that are UI related [41]. Furthermore, many of these plugins do not have their equivalent in Theia, and, in addition, the Theia philosophy is different from the classic version in Eclipse. Fortunately, there is a web version of EMF, which is one of the core components needed for PIACERE, called EMF.cloud.

The plugins used in the PIACERE IDE are described below:

EMF.cloud

EMF.cloud [21] is the web version of EMF. Eclipse EMF.cloud comprises a set of components that facilitate and simplify the adoption of the Eclipse Modeling Framework (EMF) in cloud-based applications. Some of the EMF components such as the EMF-JSON Jackson mapper and Tree editor have been used to develop the IDE. These components will be described below.

LSP

LSP [38] is a language for defining the information to be exchanged between an editor and the corresponding "language server" used by the editor. Some of the characteristics/features of the editors (for instance: autocompletion, find references) are provided by LSP. Theia is based on the Language Server Protocol (LSP) and thanks to LSP, it supports a variety of programming languages. It can be used as a desktop application, a web application or a hybrid application with separate front-end and back-end.

GLSP

GLSP (Graphical Language Server Platform) [22] is an extensible open-source framework to build custom diagram editors in the web/cloud. It can be fully integrated into Eclipse Theia. GLSP defines a language server protocol (LSP) for diagrams and integrates well with an existing tool chain and business logic.

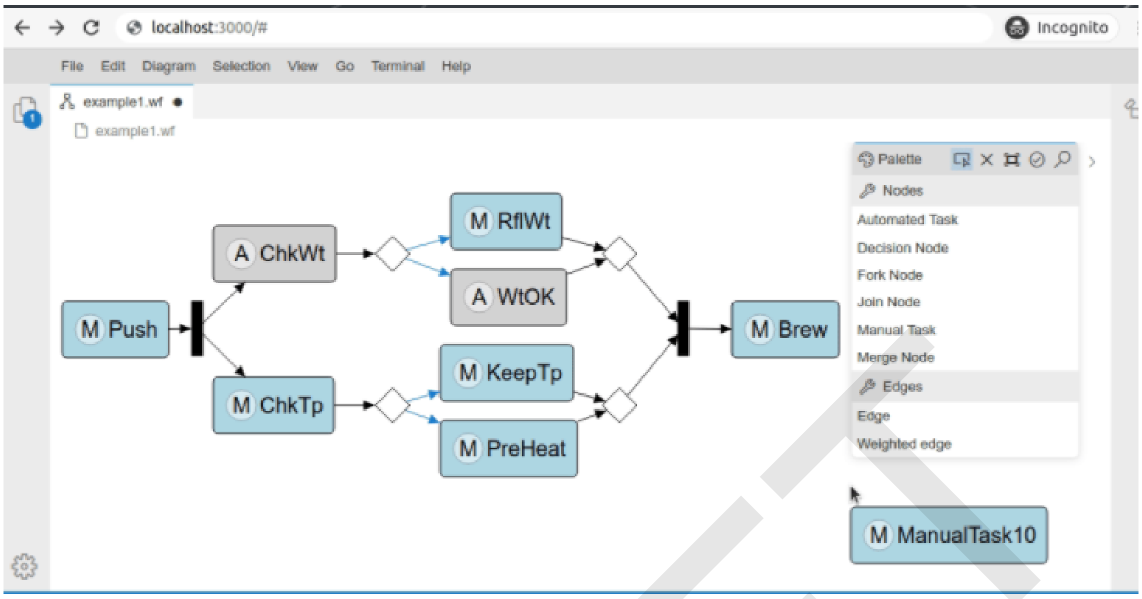


Figure 13: Example of a diagram using GLSP (Source eclipse.org [22])

GLSP provides diagram editors, including edit functionality, layouting (via CSS), shapes, palettes and all other functionalities to build diagrams. In the case of the PIACERE IDE, this visual editor will be used, in principle, in a read only mode, to visually represent the main concepts of the application infrastructure. The model cannot be editable through GLSP.

Eclipse CHE

Eclipse Che is not strictly speaking an IDE, it is more of a workspace server that allows collaborative/multi-user work. Due to the fact that Eclipse Che includes Theia as one of the supported IDEs, the PIACERE IDE could be easily integrated in Che in the future, if the need of a collaborative work arises.

EMF-JSON Jackson mapper

JSON binding for EMF models allows serialization and deserialization of EMF Resources in JSON. For this the file format to store the model instances needs to be changed. Alternatively, the JSON serialisation to transfer model data, e.g., to a web client can be used. The EMF-JSON mapper [21] is used by the EMF.cloud model server but it can be used independently too.

Tree Editor

The EMF.cloud tree editor framework [21] allows to build data-centric editors in Eclipse Theia. The framework provides the basic requirements of a tree editor that should be complemented with the particularities of a specific domain, e.g., how to build the hierarchy of the domain elements represented in the tree or the icons used for each element (see Figure 17) .

3.2.3. Technical specifications

As explained in the previous section, Theia provides three possible architecture configurations. In the case of the PIACERE IDE, the option 3 was selected**:** a Web Client front-end and a remote back-end. PIACERE IDE renders a user interface on the web browser for users. The backend runs on a host.

As an IDE, the key component for Theia is the editor, which is used to create instances of a model. In the case of PIACERE, the instances are hardware infrastructures representations. These instances must satisfy the underlaying metamodel, which in PIACERE's case is DOML (KR1) and DOML-E (KR4). To do this verification, some grammar checkers/validators will be used "Verification Tool" (KR5). These instances can be transformed in other type of models or languages. In the case of PIACERE, a code generator from DOML to text will be provided ("Infrastructural Code Generator" [KR3]). A control version system based on GIT Technology will be included in the IDE in the future version. The specific control version extension to be added in not decided at this stage of the project (M12).

JSON Schema is the format used by Theia components. As a result, the editor used in the PIACERE IDE (see Figure 14), to create DOML infrastructure instances, should work internally with JSON and JSON Schema formats. In order to obtain the JSON Schema from the DOML Ecore and vice versa, a transformation between both formats has to be done (step 3 -4 in Figure 14). In the current version of the PIACERE IDE, the equivalent JSON Schema has been obtained manually. In the future versions it is planned to use EMF.cloud / JACKSON toolset to transform the Ecore to EMF.cloud/Json and vice versa.

The representations of the instances in these formats (XML, EObject and JSON) are equivalent, so any extension / Key Result can use any of these formats to work with the instances. For example, the Infrastructural Code Generator could use a JSON instance of the infrastructure to execute the transformation. While the Verification tool could use an equivalent instance in an XML format to run the validation.

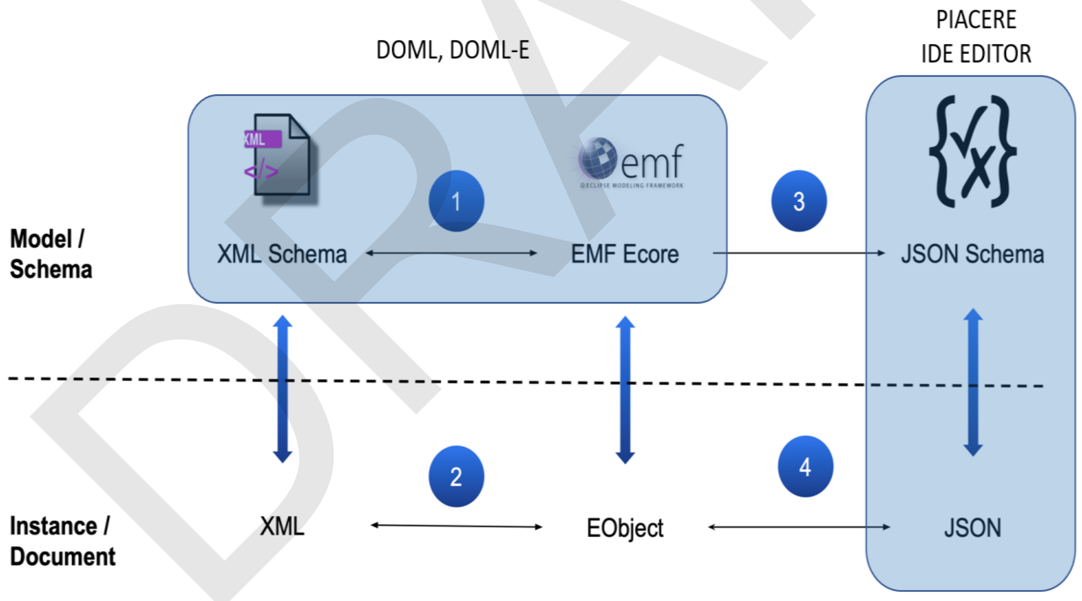


Figure 14: DOML transformations (source: PIACERE's own contribution)

The extension mechanisms used in the PIACERE IDE is the so-called "Theia Extension", which embeds the extension within the tool at compile time. This option has been selected because it is the most powerful one since it uses the same API as the core extensions.

In the PIACERE IDE v1, three extensions have been created:

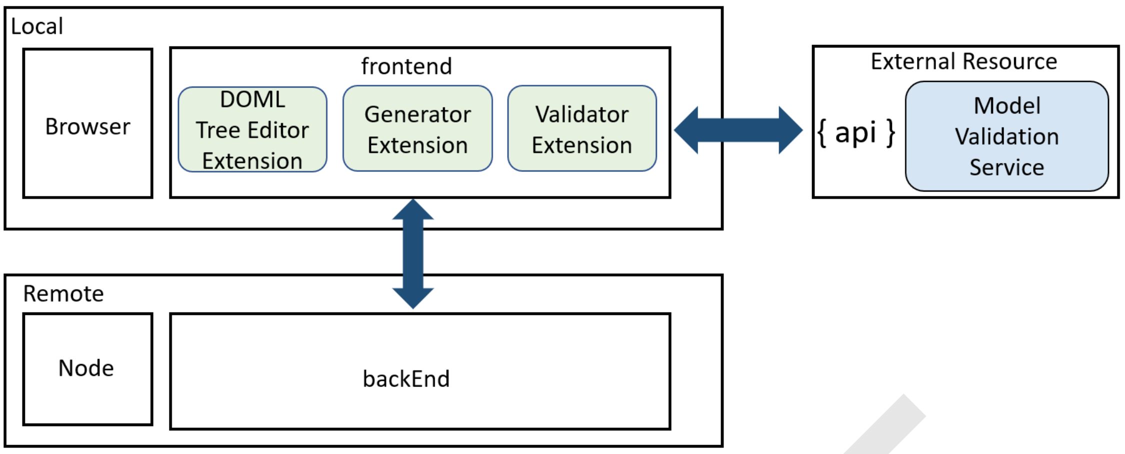
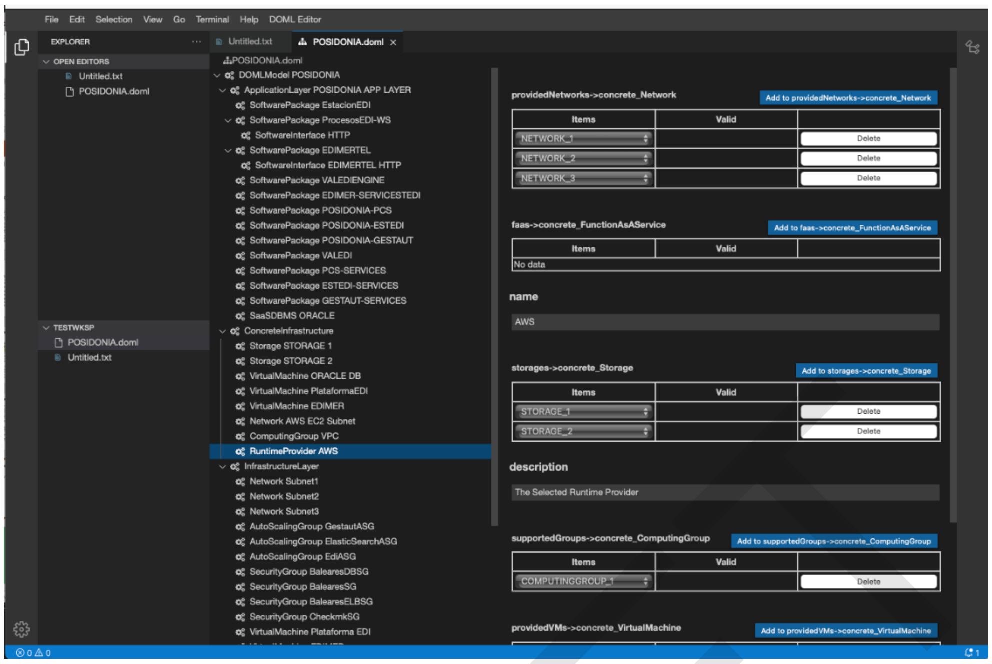


Figure 15: PIACERE IDE – EXTENSIONS (source: PIACERE's own contribution)

- **DOML Tree Editor**. This extension allows users to create infrastructure instances based on the DOML. Internally, it uses a tree editor combined with forms to create the instances. Currently, it supports 100% of the elements of the current version of DOML. Some grammar validation checks are still pending, and it will be implemented as soon as the DOML v1 is closed. In Figure 17, a screenshot of the DOML tree editor with an instance of the Posidonia UC provided by Prodevelop has been included. At any moment, it is possible to change from the tree view editor to a textual JSON representation (see Figure 16), and vice versa.



```
POSIDONIA.doml
∨ DOMLModel POSIDONIA
  ∨ ApplicationLayer POSIDONIA APP LAYER
      SoftwarePackage EstacionEDI
    ∨ SoftwarePackage ProcesosEDI-WS
        SoftwareInterface HTTP
    ∨ SoftwarePackage EDIMERTEL
        SoftwareInterface EDIMERTEL HTTP
      SoftwarePackage VALEDIENGINE
      SoftwarePackage EDIMER-SERVICESTEDI
      SoftwarePackage POSIDONIA-PCS
      SoftwarePackage POSIDONIA-ESTEDI
      SoftwarePackage POSIDONIA-GESTAUT
      SoftwarePackage VALEDI
      SoftwarePackage PCS-SERVICES
      SoftwarePackage ESTEDI-SERVICES
      SoftwarePackage GESTAUT-SERVICES
      SaaSDBMS ORACLE
  ∨ ConcreteInfrastructure
      Storage STORAGE 1
      Storage STORAGE 2
      VirtualMachine ORACLE DB
      VirtualMachine PlataformaEDI
      VirtualMachine EDIMER
      Network AWS EC2 Subnet
      ComputingGroup VPC
      RuntimeProvider AWS
  ∨ InfrastructureLayer
      Network Subnet1
      Network Subnet2
      Network Subnet3
      AutoScalingGroup GestautASG
      AutoScalingGroup ElasticSearchASG
      AutoScalingGroup EdiASG
      SecurityGroup BalearesDBSG
      SecurityGroup BalearesSG
      SecurityGroup BalearesELBSG
      SecurityGroup CheckmkSG
      VirtualMachine Plataforma EDI
```

providedNetworks->concrete_Network — Add to providedNetworks->concrete_Network

Items	Valid	
NETWORK_1		Delete
NETWORK_2		Delete
NETWORK_3		Delete

faas->concrete_FunctionAsAService — Add to faas->concrete_FunctionAsAService

Items	Valid	
No data		

name

AWS

storages->concrete_Storage — Add to storages->concrete_Storage

Items	Valid	
STORAGE_1		Delete
STORAGE_2		Delete

description

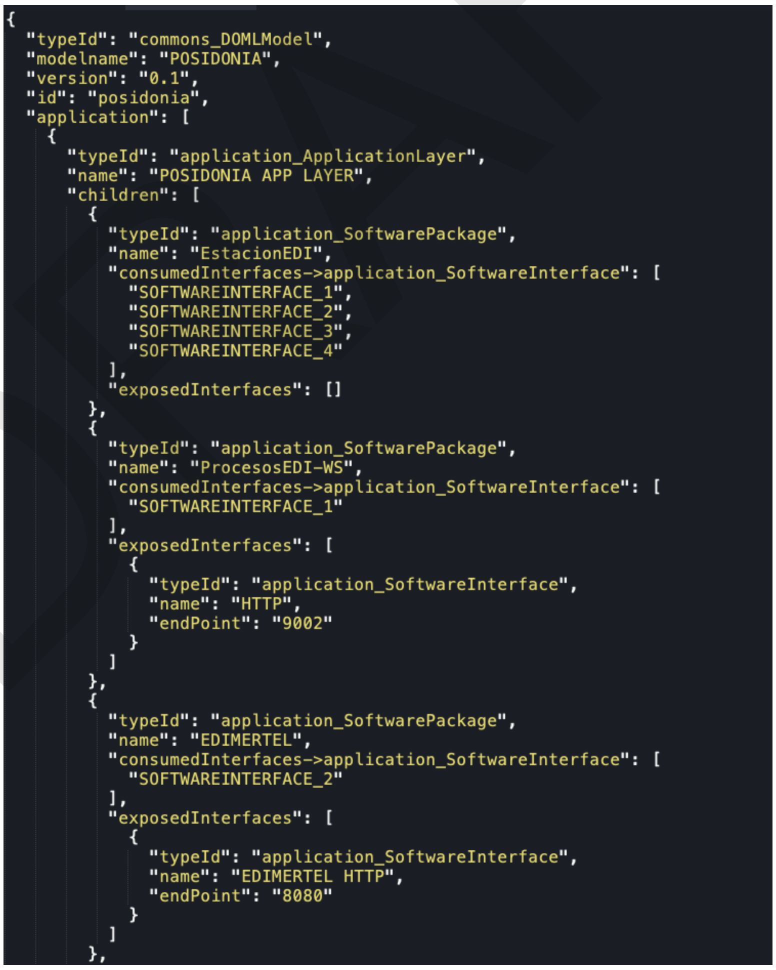
The Selected Runtime Provider

supportedGroups->concrete_ComputingGroup — Add to supportedGroups->concrete_ComputingGroup

Items	Valid	
COMPUTINGGROUP_1		Delete

providedVMs->concrete_VirtualMachine — Add to providedVMs->concrete_VirtualMachine

Figure 17: Posidonia UC Tree Editor Representation

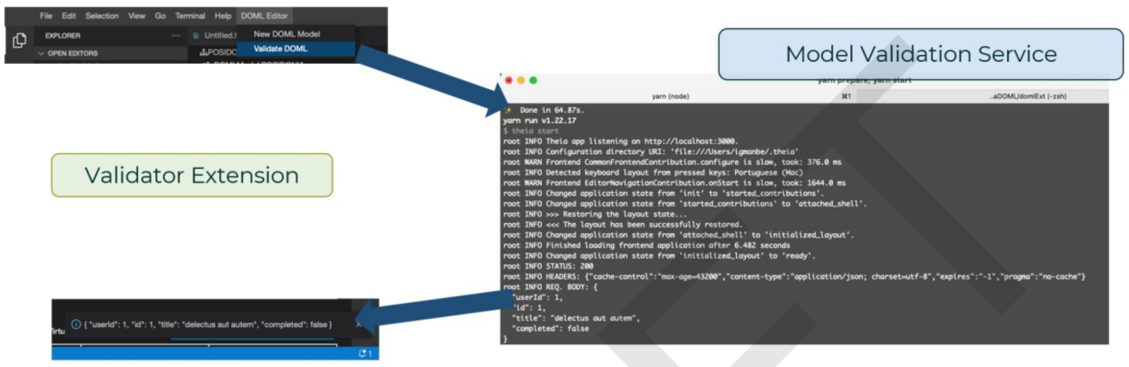


```
{
  "typeId": "commons_DOMLModel",
  "modelname": "POSIDONIA",
  "version": "0.1",
  "id": "posidonia",
  "application": [
   {
     "typeId": "application_ApplicationLayer",
     "name": "POSIDONIA APP LAYER",
     "children": [
       {
         "typeId": "application_SoftwarePackage",
         "name": "EstacionEDI",
         "consumedInterfaces->application_SoftwareInterface": [
           "SOFTWAREINTERFACE_1",
           "SOFTWAREINTERFACE_2",
           "SOFTWAREINTERFACE_3",
           "SOFTWAREINTERFACE_4"
         ],
         "exposedInterfaces": []
       },
       {
         "typeId": "application_SoftwarePackage",
         "name": "ProcesosEDI-WS",
         "consumedInterfaces->application_SoftwareInterface": [
           "SOFTWAREINTERFACE_1"
         ],
         "exposedInterfaces": [
           {
             "typeId": "application_SoftwareInterface",
             "name": "HTTP",
             "endPoint": "9002"
           }
         ]
       },
       {
         "typeId": "application_SoftwarePackage",
         "name": "EDIMERTEL",
         "consumedInterfaces->application_SoftwareInterface": [
           "SOFTWAREINTERFACE_2"
         ],
         "exposedInterfaces": [
           {
             "typeId": "application_SoftwareInterface",
             "name": "EDIMERTEL HTTP",
             "endPoint": "8080"
           }
         ]
       },
```

Figure 16: Posidonia UC JSON representation

- **Generator Extension.** This extension is only a proof of concept to demonstrate how to integrate a KR inside the IDE. It could be used as an example or guidance for integrating some KR.

- **Validator Extension**. This extension is only a proof of concept to show how to integrate a KR inside the IDE. The particularity of this extension is that it uses external resources. In this case, part of the functionality to be integrated is located in an external component which is invoked through a REST Service (see Figure 18). It could be used as an example or guidance for integrating some KR through REST APIs.



```
✨  Done in 64.87s.
yarn run v1.22.17
$ theia start
root INFO Theia app listening on http://localhost:3000.
root INFO Configuration directory URI: 'file:///Users/igmanbe/.theia'
root WARN Frontend CommonFrontendContribution.configure is slow, took: 376.0 ms
root INFO Detected keyboard layout from pressed keys: Portuguese (Mac)
root WARN Frontend EditorNavigationContribution.onStart is slow, took: 1644.0 ms
root INFO Changed application state from 'init' to 'started_contributions'.
root INFO Changed application state from 'started_contributions' to 'attached_shell'.
root INFO >>> Restoring the layout state...
root INFO <<< The layout has been successfully restored.
root INFO Changed application state from 'attached_shell' to 'initialized_layout'.
root INFO Finished loading frontend application after 6.482 seconds
root INFO Changed application state from 'initialized_layout' to 'ready'.
root INFO STATUS: 200
root INFO HEADERS: {"cache-control":"max-age=43200","content-type":"application/json; charset=utf-8","expires":"-1","pragma":"no-cache"}
root INFO REQ. BODY: {
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
```

```
{ "userId": 1, "id": 1, "title": "delectus aut autem", "completed": false }
```

Figure 18: Posidonia *UC Validation service*

CHALLENGES ENCOUNTERED IN THE IMPLEMENTATION OF THE IDE

This was the project partners' first approach in using Theia + EMF.cloud. Previous knowledge in IDE development using classic Eclipse could not be reused because the development paradigm of Theia is completely different, as well as the utilities available. The lack of experience in this technology, together with the lack of information/documentation, for critical Theia aspects, has meant that things have been slower than expected and that on many occasions a “trial and error method” has been used.

Some functionalities required to develop the IDE are not yet supported by Theia or do not work as expected. Alternative tools/extensions and existing workarounds had to be used. Fortunately, the community is very active and willing to help.

3.2.4. PIACERE IDE Next Steps

The functionalities/ features to be added or improved in the next iteration of the IDE are:

- Include a GIT control version extension in the IDE.

- Package the IDE as a container using Docker.

- Develop extensions for each of the PIACERE KRs to be integrated with the IDE. The integration of each KR will be done with the collaboration of the partners involved in its development.

- Add a visual extension based on GLSP to represent main DOML components (OPTIONAL).

- Adding Support to new versions of the DOML and DOML-E.

- Integrate the PIACERE IDE into Eclipse Che to allow collaborative work.

4. Delivery and usage

4.1. Package information

The code of the project is available in the PIACERE Gitlab repository

In the repository, there is a README file which explains how to install the tool. Moreover, there is an "Extensions" folder with the following already created extensions:

- *DOMLExt*. This extension allows users to create infrastructure instances based on the DOML. Internally, it uses a tree editor combined with forms to create the instances.
- *DIMLGenerator Extension.* This extension is only a proof of concept to show how to integrate the Infrastructure Code Generator inside the IDE. It could be as an example or guidance for integrating some KR.
- *DOMLValidator*. This extension is only a proof of concept to show how to integrate the Validator Tool inside the IDE.

The Structure of these extensions follows the structure of a Theia extensions. The main software artifacts are:

- *Lib*. Contains the required libraries needed to run the extension.
- *Node_modules*. Node.js components needed
- *src*: code of the extension
- *README.md*: information about the extension, purpose of the extension, how to install and use.
- *package.json*: script for building the extension package.

4.2. Installation instructions

DOML IDE Tool Extensions

- Prerequisites

  To Install the extensions, you need a clean ubuntu (physical, virtual machine or docker) and to install the required packages

- Installation

  Update the packages manager

  ```
  apt update
  ```

  Install some required packages

  ```
  apt install -y nano curl wget pkg-config libsecret-1-dev g++ gcc
  make python2.7 pkg-config libx11-dev libxkbfile-dev
  ```

  Install NODE VERSION MANAGER

```
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s
"${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads
nvm
```

Install NODE Version 12.18.4 (Theia ensures working with version >12 and <13)

```
nvm install 12.18.4
```

Install NODE PACKAGE MANAGER

```
apt install -y npm
```

Add some additional Node PM configuration parameters:

```
npm config set unsafe-perm true
```

Install some additional Node Packages:

```
npm install -g uuid yarn try-thread-sleep keytar node-gyp

npm install -g serverless --ignore-scripts spawn-sync
```

Note: latest versions of these node Packages are valid for this requierement.

Install Yeoman and Theia Extension Generator

```
npm install -g yo generator-theia-extension --unsafe-perm=true --
allow-root
```

CREATING A THEIA ENVIRONMENT

Create a directory for Theia

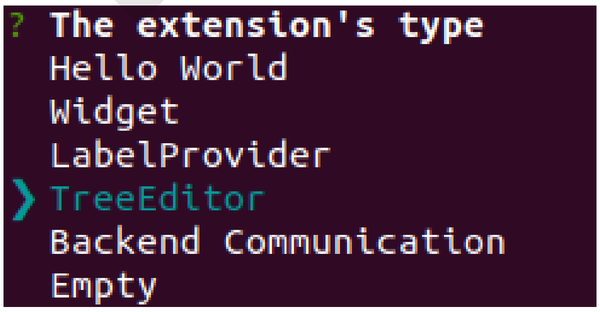
```
cd $HOME

mkdir TheiaExample; cd TheiaExample
```

Create the Theia Environment and an example Theia Extension (select TreeEditor as the extension type to generate)

```
yo theia-extension
```

After this instruction, you have to choose “TreeEditor” option in the list of extensions



```
? The extension's type
  Hello World
  Widget
  LabelProvider
❯ TreeEditor
  Backend Communication
  Empty
```

Figure 19: Theia extension's type

INSTALLING DOML EXTENSIONS

Download the Extensions folder from this repository

Copy DOMLValidator, domlExt and domlgenerator folders to root TheiaExample folder

Open package.json file and add these lines to the workspaces field:

```
"DOMLValidator",

"domlExt",

"domlgenerator"
```

Open browser-app/package.json file and add these lines to the dependencies field:

```
"domlext": "0.0.0",

"domlgenerator": "0.0.0",

"domlvalidator": "0.0.0",
```

COMPILE AND LAUNCH

Compile the extensions:

```
cd $HOME/TheiaExample/DOMLValidator ; yarn prepare

cd $HOME/TheiaExample/domlgenerator ; yarn prepare

cd $HOME/TheiaExample/domlExt ; yarn prepare
```

Compile the Theia browser app

```
cd $HOME/TheiaExample/browser-app ; yarn prepare
```

Launch the Theia browser app

```
cd $HOME/TheiaExample/browser-app ; yarn start
```

4.3. User Manual

Figure 20 is the main window (page) after starting PIACERE IDE. It contains several sections: menu (1), open editors' box (2), current workspace (3) and main working area (4).

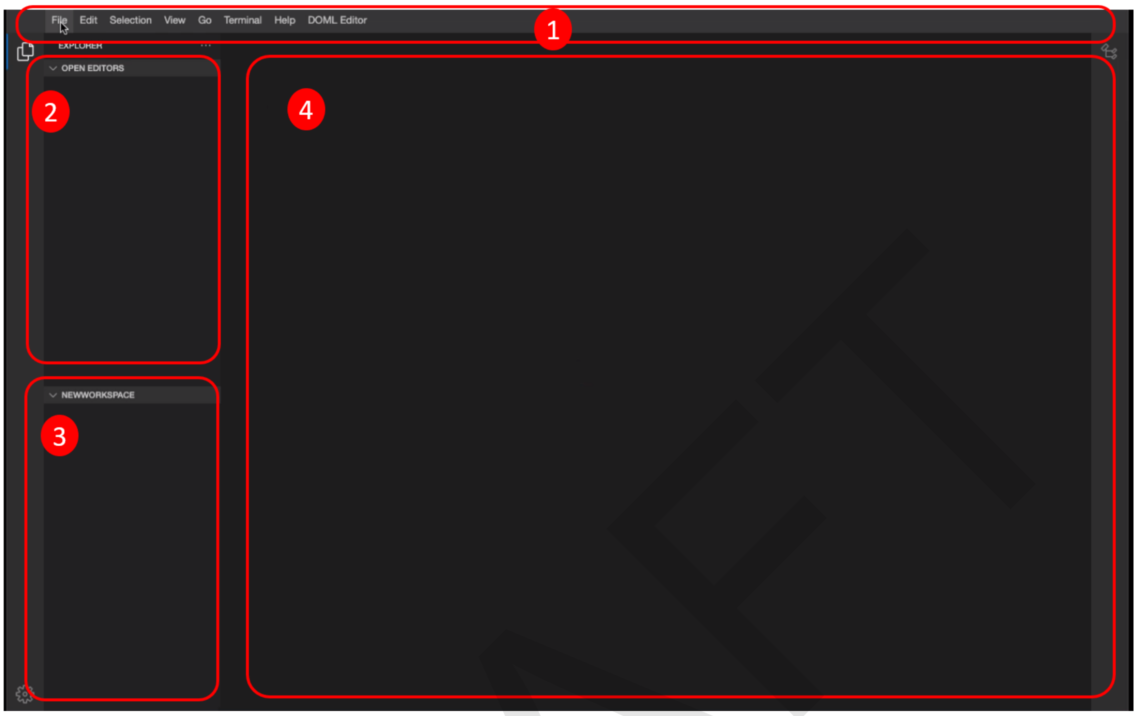


Figure 20: PIACERE IDE - main window

The first task before start working is to create a new workspace or choose an existing one (see Figure 21).

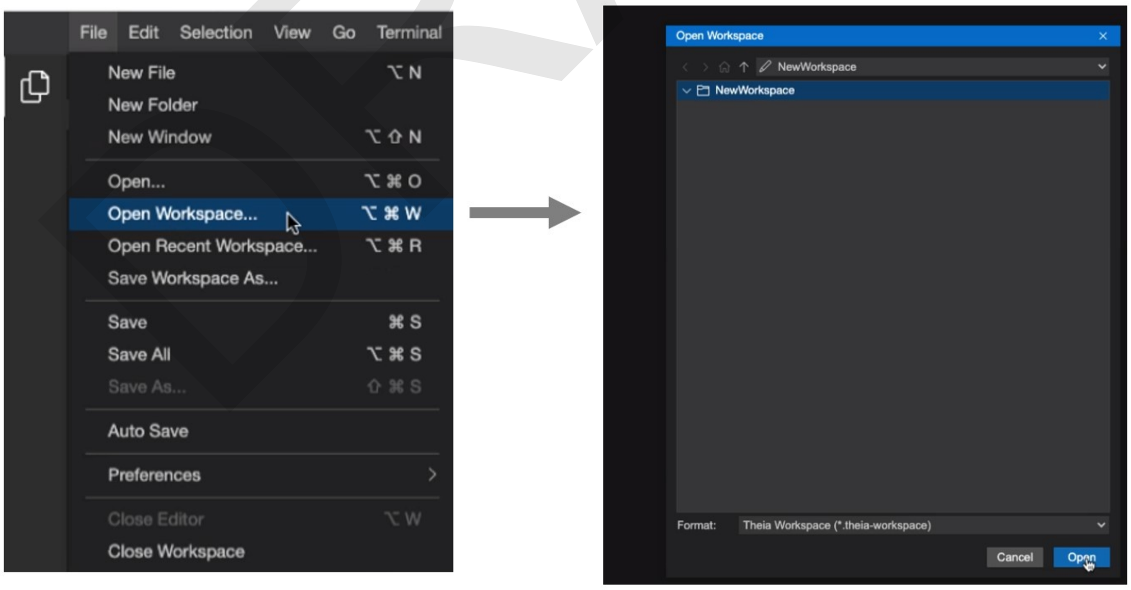


Figure 21: Open Workspace

After creating or selecting a workspace, it is possible to create a new project, that basically is an instance of the DOML. To do that, the user should select the menu option "New DOML Model" that is included in the DOML Editor main menu. This action opens a dialog to specify the name of the project (see Figure 22).

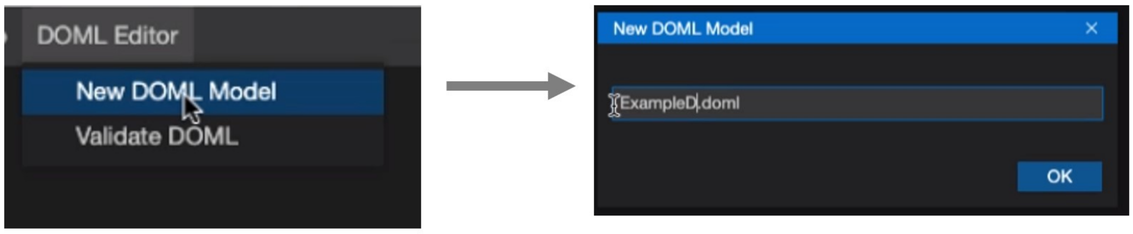


Figure 22: New DOML Model

Once the project name is specified, the project will be created with the main grouping elements of the DOML created by default. The DOML instance is represented in a tree view, which will be the default editor DOML instances. This tree editor allows the user to easily navigate between the hierarchy of DOML elements/components (see Figure 23).

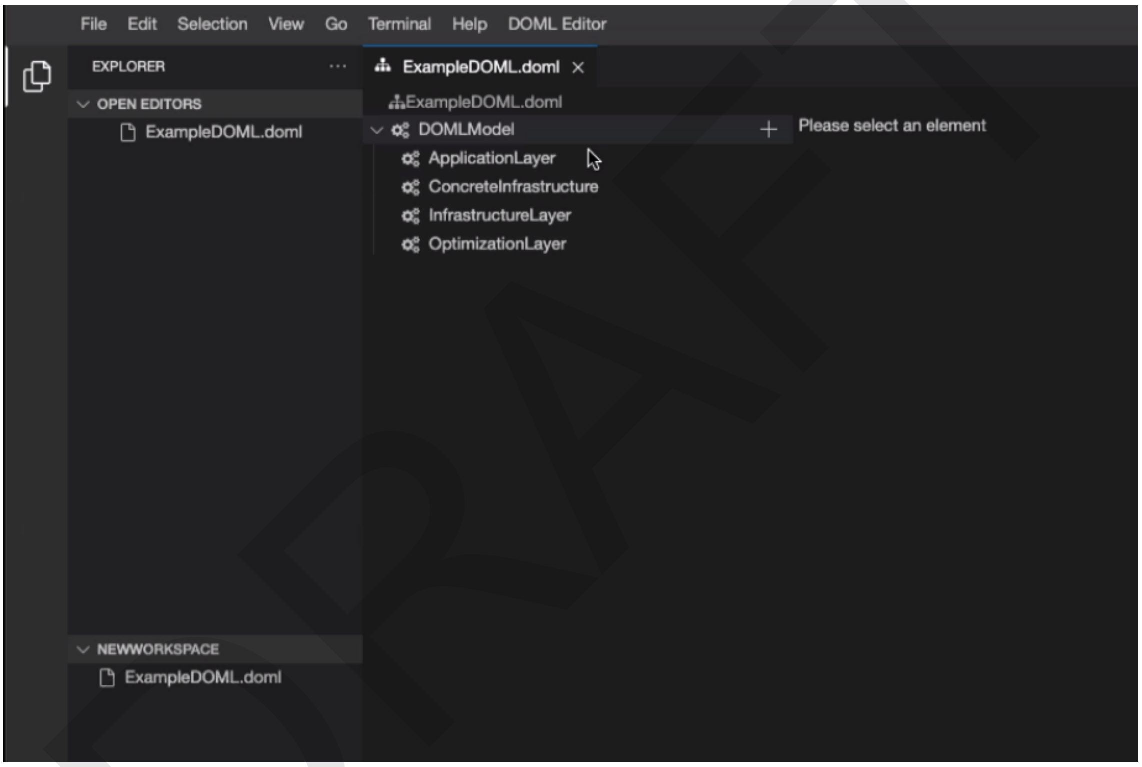


Figure 23: Empty model specification

From the tree view, it is possible to create new elements by clicking the “plus icon” in the tree item emergent menu. As soon as you click the button, the element is created and appears in the tree editor (Figure 24). At any moment, it is possible to remove a tree item from the “trash can icon”.

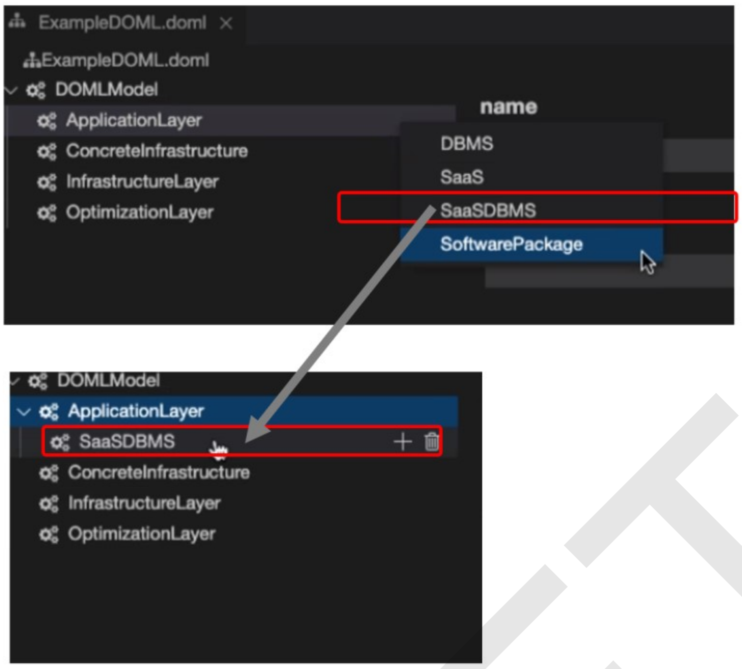
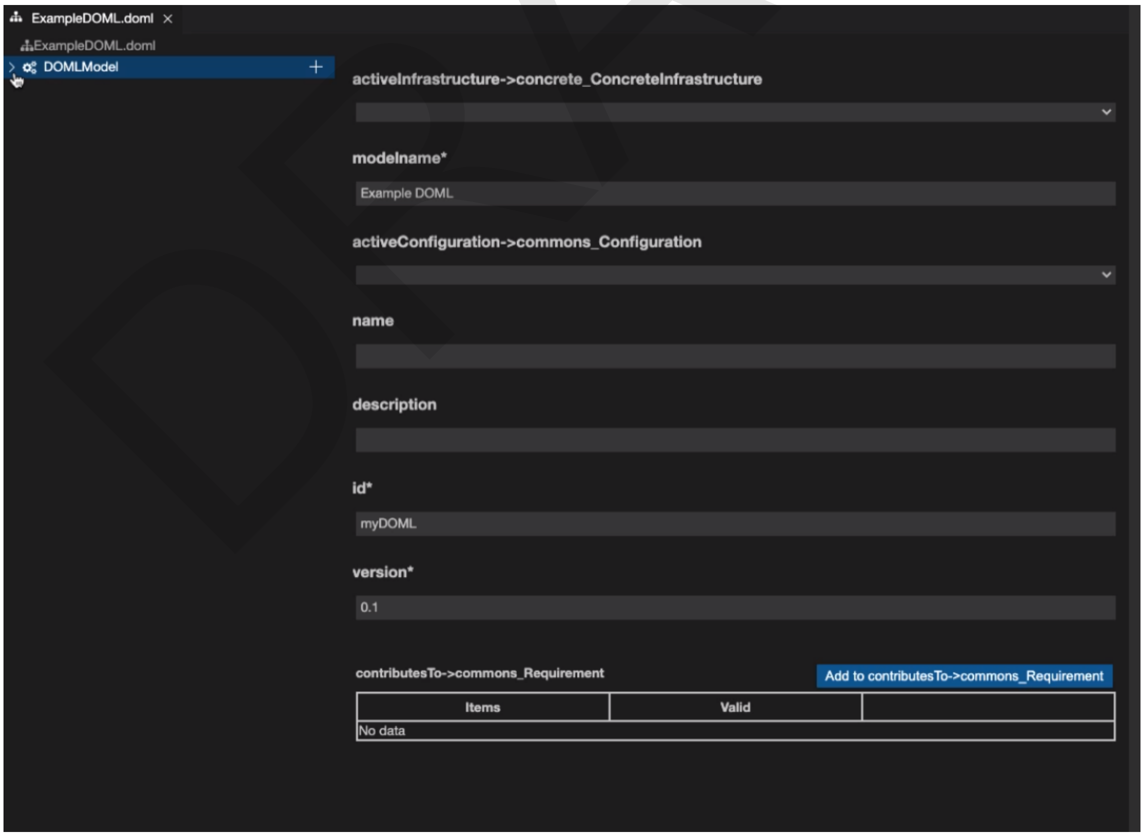


Figure 24: DOML element creation

In addition, a form appears to fill in the properties of the newly created item. For each type of DOML element the form is configured accordingly (Figure 25).



activeInfrastructure->concrete_ConcreteInfrastructure

modelname*
Example DOML

activeConfiguration->commons_Configuration

name

description

id*
myDOML

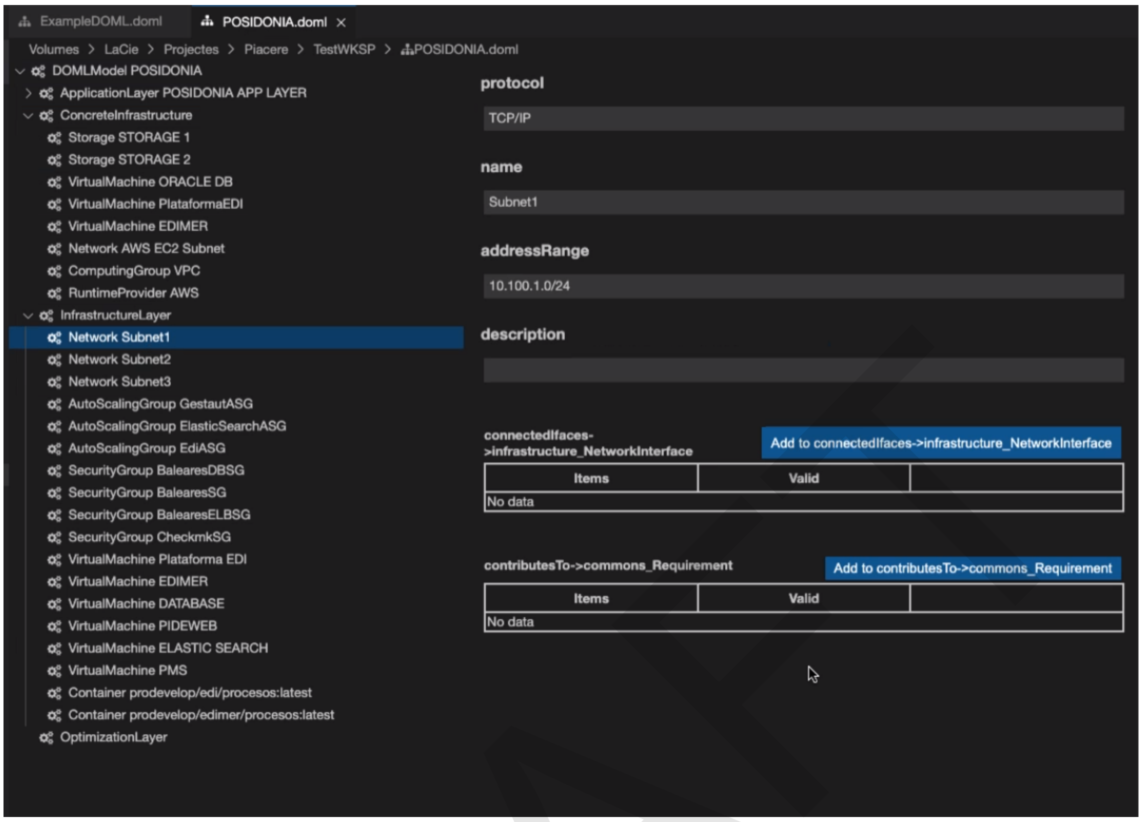
version*
0.1

contributesTo->commons_Requirement

Items	Valid	
No data		

Figure 25: DOML model form

In Figure 26, you can see an example of an instance of the DOML after defining several DOML elements.



ExampleDOML.doml POSIDONIA.doml ×

Volumes > LaCie > Projectes > Piacere > TestWKSP > POSIDONIA.doml

- DOMLModel POSIDONIA
  - ApplicationLayer POSIDONIA APP LAYER
  - ConcreteInfrastructure
    - Storage STORAGE 1
    - Storage STORAGE 2
    - VirtualMachine ORACLE DB
    - VirtualMachine PlataformaEDI
    - VirtualMachine EDIMER
    - Network AWS EC2 Subnet
    - ComputingGroup VPC
    - RuntimeProvider AWS
  - InfrastructureLayer
    - Network Subnet1
    - Network Subnet2
    - Network Subnet3
    - AutoScalingGroup GestautASG
    - AutoScalingGroup ElasticSearchASG
    - AutoScalingGroup EdiASG
    - SecurityGroup BalearesDBSG
    - SecurityGroup BalearesSG
    - SecurityGroup BalearesELBSG
    - SecurityGroup CheckmkSG
    - VirtualMachine Plataforma EDI
    - VirtualMachine EDIMER
    - VirtualMachine DATABASE
    - VirtualMachine PIDEWEB
    - VirtualMachine ELASTIC SEARCH
    - VirtualMachine PMS
    - Container prodevelop/edi/procesos:latest
    - Container prodevelop/edimer/procesos:latest
  - OptimizationLayer

protocol
TCP/IP

name
Subnet1

addressRange
10.100.1.0/24

description

connectedIfaces->infrastructure_NetworkInterface

Items	Valid	
No data		

contributesTo->commons_Requirement

Items	Valid	
No data		

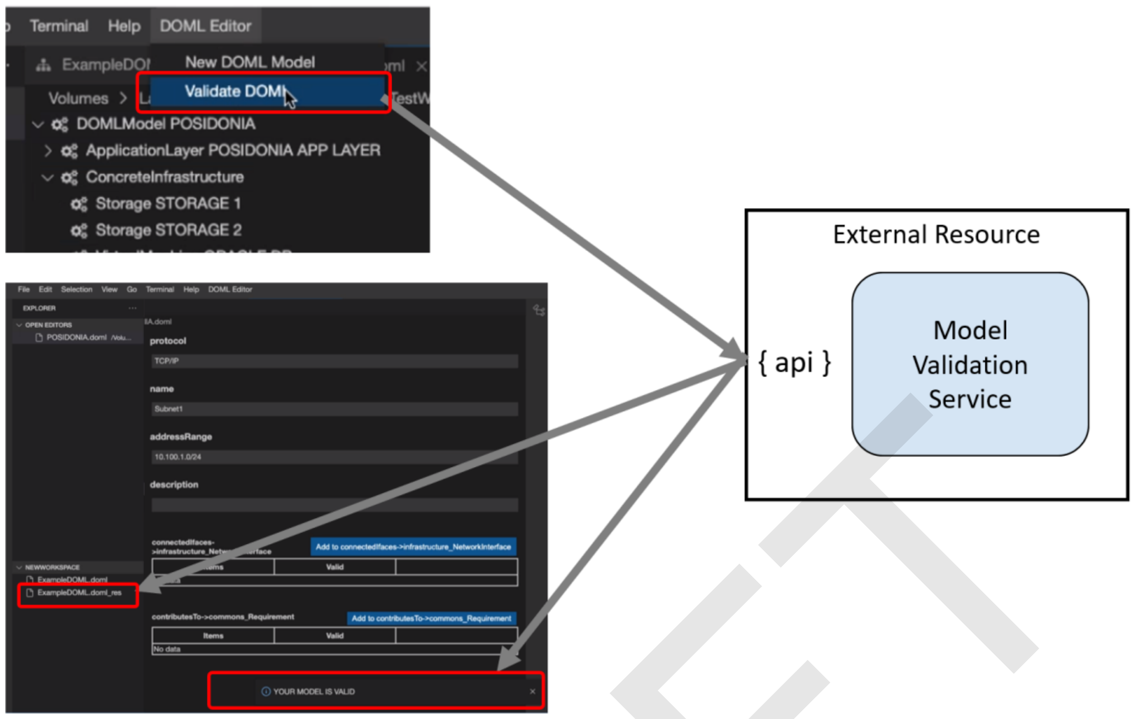
Figure 26: DOML instance of POSIDONIA Use Case

Integration with the validation tool

From PIACERE IDE, it is possible to launch other PIACERE tools. This section shows an example of an integration of an external tool.

From the menu, it is possible to launch the validation tool, that is running as an independent service. This service has a REST API that will be called from IDE when the user selects the "Validate DOML" menu option (Figure 27).

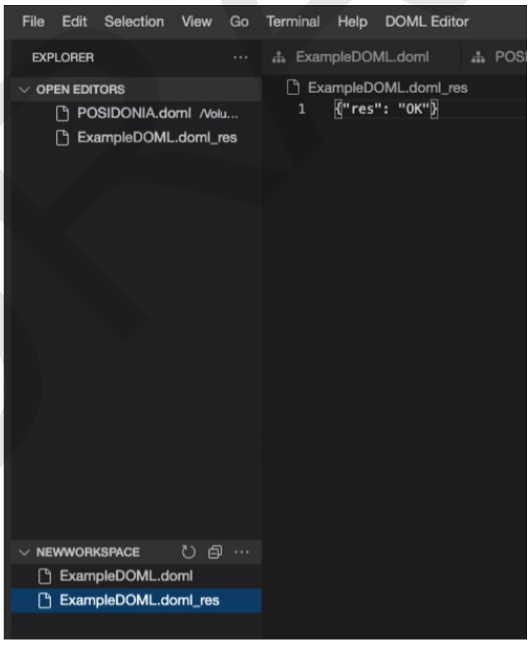
After calling the external service, it will process the petition and will provide a response to the IDE. The IDE will process the results of the invocation of the remote service.



protocol
TCP/IP
name
Subnet1
addressRange
10.100.1.0/24
description

Figure 27: Invocation of an external service

The validation service generates a file with the result of the invocation. This file (*_res) is listed in the open editor area. In this example, the model was correct, and the validation did not find any problem in the DOML instance (Figure 28).



```
{"res": "OK"}
```

Figure 28: Validation response file

Save options

From the File menu option, the user can save the project at any time by selecting the menu options “Save” or “Save All”. Another interesting option would be to activate the Auto Save feature to periodically save the project. With this option, the user does not need to worry about saving the project manually.

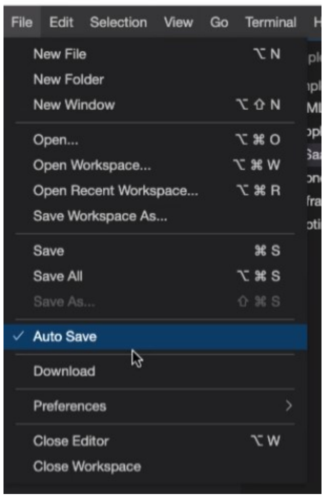
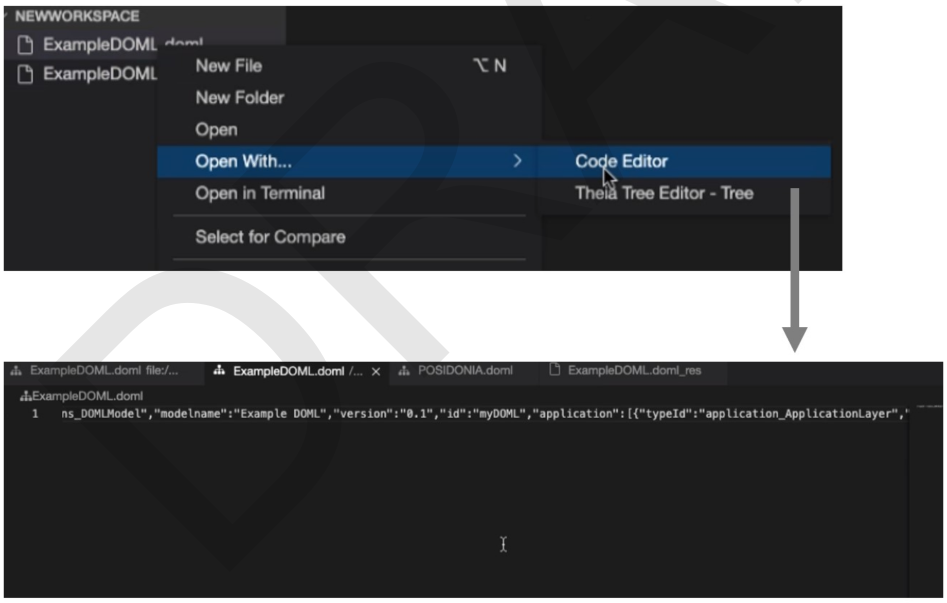


Figure 29: Auto Save option

Code Editor

If you want to see the instance of the model in plain text, you can change the view to the code editor. This editor shows the JSON representation of the DOML instance of the project (Figure 30).



```
ns_DOMLModel","modelname":"Example DOML","version":"0.1","id":"myDOML","application":[{"typeId":"application_ApplicationLayer","
```

Figure 30: Code editor

4.4. Licensing information

The license is still to be defined, but being Piacere IDE a tool that uses Eclipse Theia, PIACERE IDE license should be conditioned by the licenses permitted for products derived from Theia.

4.5. Download

The first version of the PIACERE IDE is hosted in the git repository of PIACERE under the project name “T35 IDE”. The URL of the project is:

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

In the repository, there is a README file which explains how to install the tool. Moreover, you have an “Extensions” folder with the “domlExt” extension created for modeling HW infrastructures using DOML and the initial Extension prototypes “DOMLValidator” and “domlgenator” used to validate how other tools could be integrated into the PIACERE IDE.

5. Conclusions and future work

This deliverable has described the initial version of the PIACERE IDE, which includes a DOML extension that allows users to define the infrastructure of an application/services based on the initial version of the DOML. Moreover, two extra extensions have been created as an integration example with the Infrastructural Code Generator (KR3) and Verification Tool (KR5). The initial version of the IDE is available in the PIACERE repository [35]

In addition to explaining the current status of the IDE, the document presented the analysis of the different candidate technologies to develop the IDE. The selection of the technology has been mainly based on the ability to be an online tool and the support of models/metamodels.

In the following months, the current implementation of the IDE will be used by the three PIACERE use cases in order to model their infrastructure. This will be a good proof of concept, as it will allow us to get feedback from real potential users and to consider such comments in future versions.

In the following versions of the deliverable, new versions of the IDE will be presented. The new IDE will provide support for the new versions of the DOML and DOML-E and will contain extensions for integrating other KRs into the IDE. It is also likely that the versions of Theia and some of the plugins used will be upgraded, as they are constantly evolving.

6. References

[1] «Visual Studio Code» [Online]. Available: https://code.visualstudio.com/
[2] «Monaco Editor» [Online]. Available: https://microsoft.github.io/monaco-editor/
[3] «Visual Studio Codium» [Online]. Available: https://vscodium.com/
[4] «Eclipse Theia» [Online]. Available: https://theia-ide.org/
[5] «GitHub Codespaces» [Online]. Available: https://visualstudio.microsoft.com/es/services/github-codespaces/
[6] «Eclipse IDE» [Online]. Available: https://www.eclipse.org/eclipseide/
[7] «Eclipse Modelling Framework» [Online]. Available: https://www.eclipse.org/modeling/emf/
[8] «Eclipse Che» [Online]. Available: https://www.eclipse.org/che/
[9] «Kubernetes» [Online]. Available: https://kubernetes.io/
[10] «Docker» [Online]. Available: https://www.docker.com/
[11] «Eclipsecon Europe 2019: Eclipse Theia and Che, explained and explored» [Online]. Available: https://www.youtube.com/watch?v=oVjAknEtlLQ
[12] «DevConf Czech Republic 2020: Eclipse Che & the future of Cloud Development Tool» [Online]. Available: https://www.youtube.com/watch?v=zTuVpkl5FZU
[13] «ECT Tools: Eclipse Theia vs. Eclipse Che vs. VS Code» [Online]. Available: https://www.youtube.com/watch?v=XWDArhNOXRo
[14] «Visual Studio Code Extensions API» [Online]. Available: https://code.visualstudio.com/api
[15] «Eclipse Theia Extensions» [Online]. Available: https://theia-ide.org/docs/authoring_extensions
[16] «Open VS Code Extension Registry» [Online]. Available: https://open-vsx.org/
[17] «Eclipsecon 2021. Virtual Event. October 25-28. » [Online]. Available : https://www.eclipsecon.org/2021/
[18] «Theia: Product Vision. Delft Students on Software Architecture Blog. [Online]. Available : https://2021.desosa.nl/projects/theia/posts/essayone-productvision/
[19] «Theia: From Vision to Architecture». Delft Students on Software Architecture Blog. [Online]. Available: https://2021.desosa.nl/projects/theia/posts/essay2/
[20] «Eclipse Theia extensions and plugins. [Online]. Available: https://theia-ide.org/docs/extensions
[21] «EMF.cloud [Online]. Available: https://www.eclipse.org/emfcloud/
[22] «Eclipse Graphical Language Server Platform . [Online]. Available: https://www.eclipse.org/glsp/
[23] «Eclipse IDE Base plugins» [Online]. Available: https://download.eclipse.org/eclipse/downloads/
[24] «Eclipse Graphical Modelling Framework» [Online]. Available: https://www.eclipse.org/gmf-runtime/
[25] «Eclipse Graphiti» [Online]. Available: https://www.eclipse.org/graphiti/
[26] «Eclipse Sirius» [Online]. Available: https://www.eclipse.org/sirius/
[27] «Eclipse Acceleo» [Online]. Available: https://www.eclipse.org/acceleo/

[28] «Eclipse ATL» [Online]. Available: https://www.eclipse.org/atl/
[29] «Eclipse QVTo» [Online]. Available: https://www.eclipse.org/mmt/qvto
[30] «Eclipse UML2» [Online]. Available: https://www.eclipse.org/modeling/mdt/?project=uml2
[31] «Eclipse M2E» [Online]. Available: https://www.eclipse.org/m2e/
[32] «Eclipse Papyrus» [Online]. Available: https://www.eclipse.org/papyrus/
[33] «Eclipse EGit» [Online]. Available: https://www.eclipse.org/egit

[34] «Xtext» [Online]. Available: https://www.eclipse.org/Xtext
[35] «PIACERE REPOSITORY» [Online]. Available: https://git.code.tecnalia.com/piacere
[36] «Xpand» [Online]. Available: https://projects.eclipse.org/projects/modeling.m2t.xpand
[37] «InversifyJS» [Online]. Available: https://inversify.io/
[38] «Language Server Prototol» [Online]. Available: https://github.com/Microsoft/language-server-protocol

[39] GitHub, GitHub pricing [Online]. Available: https://github.com/pricing
[40] PIACERE consortium ; D3.1 - PIACERE Abstractions, DOML, DOML-E – v1, 2021
[41] https://eclipsesource.com/blogs/2021/05/27/migrating-eclipse-plugins-to-eclipse-theia-or-vs-code

7. APPENDIX: PIACERE IDE based on Eclipse desktop

This appendix describes the PIACERE IDE using Eclipse IDE + EMF, with these technologies it is feasible to develop the PIACERE IDE, but finally, the use of these technologies were discarded in favour of Theia and EMF.cloud, which is its natural evolution.

However, an effort was made to design the IDE using the classic eclipse, in order to minimise risks in case of having any impediment with the development of the IDE using Theia. In this way, in the situation of having to redo the IDE with the classic Eclipse, the exercise of knowing which plugins would be necessary would already be done and we could start immediately. Part of the work done in Theia could be reused and it would not imply having to start from scratch.

7.1. Technical Description

The Eclipse IDE 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. Section 2.1.3 describes this technology in detail.

To build the IDE some Eclipse modelling plugins should be added. In section 7.1.2 a list of these plugins is provided along with an explanation of the functionality provided by each one. This is only an initial list that could vary during the development of the IDE.

Thanks to EMF, the IDE can support metamodels. In the case of the PIACERE IDE, DOML and DOML-E will be the supported metamodels used for modeling application infrastructures.

IDE Integration mechanism

The pivotal tool of the project is the IDE and it will give support to the PIACERE Framework. The other Key Results of the project will be integrated into the IDE using some of the integration mechanisms that these technologies provide. The PIACERE IDE will be customized with suitable plug-ins that will integrate the different tools, in order to minimize the learning curve and simplify adoption of IaC approach. Not all KR/tools will be integrated in the same way. Several integration patterns, focusing on the Eclipse plugin architecture, will be defined. They will allow the implementation and incorporation of application features very quickly.

To integrate the other PIACERE KR into the IDE, a set of menus will be enabled to facilitate access to these KRs. Some of these tools could be fully integrated into Eclipse and should be developed as Eclipse plugins. On the other hand, other tools could be run in isolation and through a REST API or any other integration mechanism could be invoked remotely. In this case, from the Eclipse menu the tool would be invoked via a REST service.

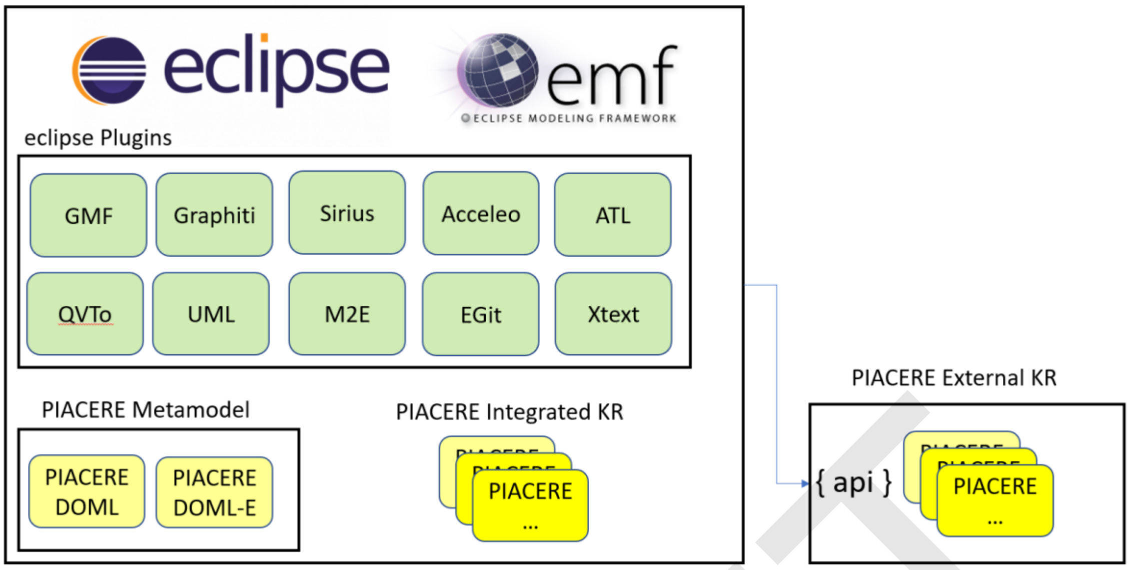


Figure 31: eclipse IDE components (soure : PIACERE's own contribution)

The IDE will use "Cheat sheets", which is a built-in mechanism for displaying mini tutorials "Wizards", in order to guide users on how to use the different tools. Cheat sheets are quick-and-dirty instructions for how to perform multi-step processes in Eclipse, displayed on the side of the workbench where you can quickly and easily step through them.

7.1.2 Components description

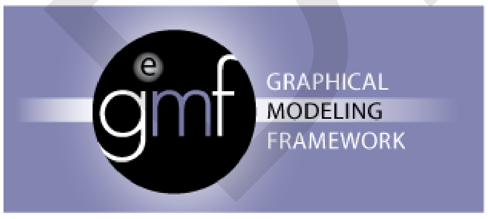
This section lists some modelling plugins that could be interesting for building the PIACERE IDE.

Eclipse base

These plugins are the core of Eclipse [23] and involve the minimum plugin set to build an executable desktop application.

The current version of these plugins is 4.21.

GMF Runtime



Graphical Modelling Framework [24] is the most used library to develop graphical editors on Eclipse. It is mainly based on EMF models, and generates EMF based models too.

The current version of GMF is >= 1.9.0.

Graphiti

Graphiti [25] is another library for developing graphical editors on Eclipse, that enables rapid development of state-of-the-art diagram editors for domain models

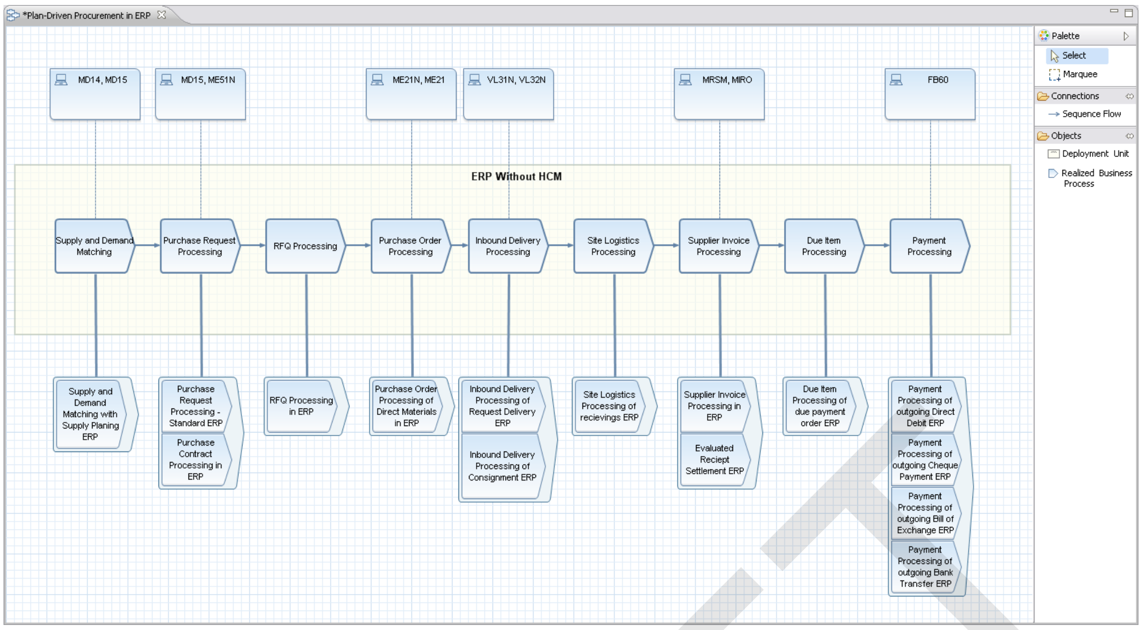

*Plan-Driven Procurement in ERP
MD14, MD15
MD15, ME51N
ME21N, ME21
VL31N, VL32N
MRSM, MIRO
FB60
ERP Without HCM
Supply and Demand Matching
Purchase Request Processing
RFQ Processing
Purchase Order Processing
Inbound Delivery Processing
Site Logistics Processing
Supplier Invoice Processing
Due Item Processing
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
Palette
Select
Marquee
Connections
Sequence Flow
Objects
Deployment Unit
Realized Business Process

Figure 32: Graphiti Editor Example (source: Graphiti)

The current version of Graphiti is 0.12.0.

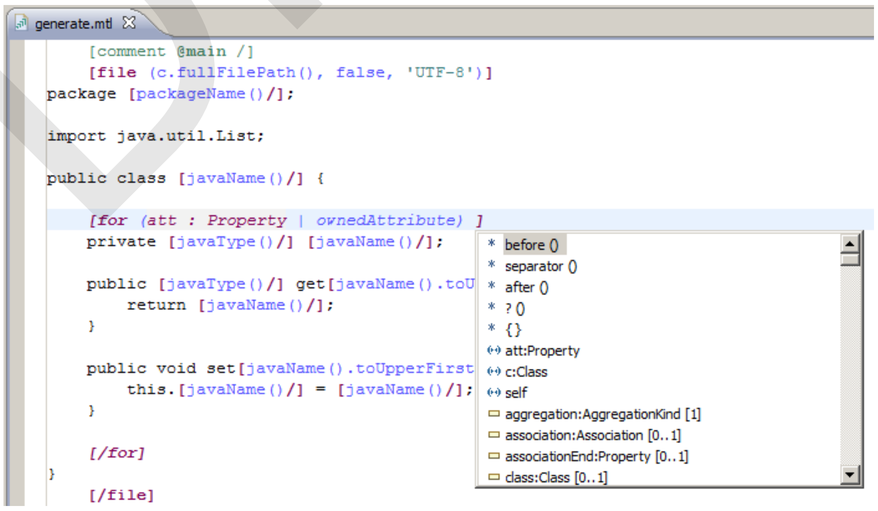
Sirius

Sirius [26] is a tool based on EMF and GMF that allows to easily create graphical modelling workbenches. A modeling 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 [27] is a template-based technology to create code generators from an EMF model, that 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]
```

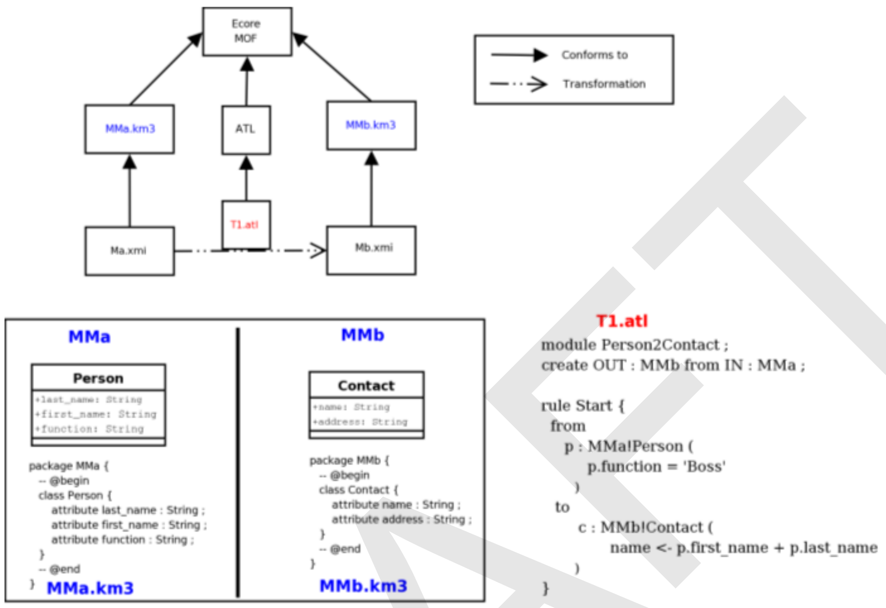
```
before ()
separator ()
after ()
? ()
{}
att:Property
c:Class
self
aggregation:AggregationKind [1]
association:Association [0..1]
associationEnd:Property [0..1]
class:Class [0..1]
```

Figure 33: Acceleo template example (source: Acceleo)

The current version of Acceleo is 3.7.

ATL

ATL [28] 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 34: ATL transformation Example (source: ATL)

The current version of ATL is 4.5.0.

QVTo

QVTO [29] is another language that provides 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 [30] is a set of plugins based on EMF that brings support to UML OMG Metamodel in Eclipse. Although UML2 provides 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 [31] 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 [32] 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.

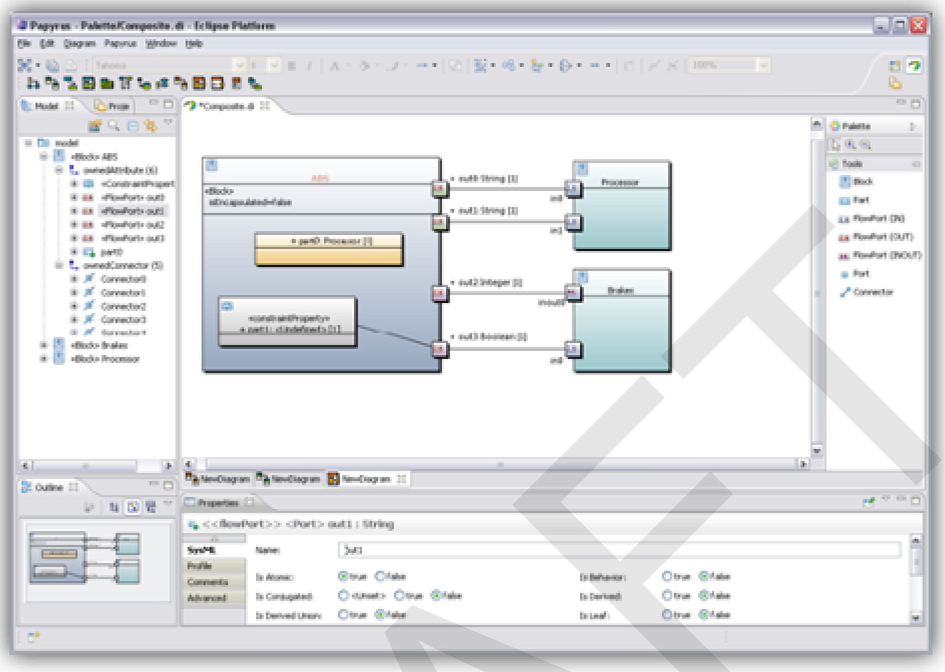


Figure 35: Papyrus instance example (source: Papyrus)

The current version of Papyrus is 5.2.0.

EGit

EGit [33] 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 [34] is a framework for development of programming languages and domain-specific languages. With Xtext the user can define its own language using a grammar language. As a result a full infrastructure can be obtained, including parser, linker, type checker, compiler as well as editing support for Eclipse, any editor that supports the Language Server Protocol and a web browser

The current version of Xtext is 2.25.0.