



**Multi-cloud Execution-ware  
for Large-scale Optimised  
Data-Intensive Computing**

H2020-ICT-2016-2017  
Leadership in Enabling and  
Industrial Technologies;  
Information and  
Communication Technologies

Grant Agreement No.:  
731664

Duration:  
1 December 2016 -  
30 November 2019

[www.melodic.cloud](http://www.melodic.cloud)

Deliverable reference:  
D3.3

Date:  
31 May 2018

Responsible partner:  
7bulls

Editor(s):  
Paweł Skrzypek

Author(s):  
Kyriakos Kritikos

Approved by:  
Ernst Gunnar Gran

ISBN number:  
N/A

Document URL:  
[http://www.melodic.cloud/deliverables/D3.3\\_IDE-plugin  
for\\_data-aware\\_design\\_and\\_development\\_of\\_multi-cloud  
data-intensive\\_applications.pdf](http://www.melodic.cloud/deliverables/D3.3_IDE-plugin_for_data-aware_design_and_development_of_multi-cloud_data-intensive_applications.pdf)

Title:

## IDE-plugin for data-aware design and development of multi-cloud data- intensive applications

Executive summary:

This deliverable presents and analyses a web-based CAMEL editor which enables the modelling of Cross-Cloud data intensive applications. Our analysis focuses first on providing the main requirements that drove the design of this editor as well as its respective use cases. Then it proceeds with the presentation of the editor architecture and relevant implementation details. Next, the main features of the editor are supplied. Finally, a detailed walkthrough of the editor is provided according to a reduced form of a specific use case from the Melodic project. The deliverable concludes with the main directions for future work on the editor which will be followed during the Melodic project lifetime.



This project has received funding from  
the European Union's Horizon 2020 research  
and innovation programme under grant agreement No 731664

Document	
Period Covered	M12-16
Deliverable No.	D3.3
Deliverable Title	IDE-plugin for data-aware design and development of multi-cloud data-intensive applications
Editor(s)	Paweł Skrzypek
Author(s)	Kyriakos Kritikos
Reviewer(s)	Feroz Zahid, Vasilis Stefanidis
Work Package No.	3
Work Package Title	Upper ware
Lead Beneficiary	7bulls
Distribution	PU
Version	1.0
Draft/Final	Final
Total No. of Pages	53

## Table of Contents

1	Introduction.....	6
2	CAMEL Editor.....	7
2.1	Introduction.....	7
2.2	Use Case Analysis and Requirements.....	8
2.2.1	Requirements.....	8
2.2.2	Use Case Analysis.....	10
2.3	Architecture.....	14
2.4	Implementation Details & Usage Instructions.....	15
2.5	Features and Requirement Satisfaction.....	17
2.6	Editor Walkthrough.....	18
2.6.1	Admin Login.....	22
2.6.2	Organisation Model Editing.....	23
2.6.3	Devops Login.....	27
2.6.4	CAMEL Model Editing via Devops.....	27
3	Future Work.....	44
3.1	CAMEL Meta-Model Extensions / Modifications.....	45
3.2	Coverage of Unsatisfied Requirements.....	45
3.2.1	Requirement R5 Coverage.....	46
3.2.2	Requirement R9 Coverage.....	46
3.2.3	Requirement R10 Coverage.....	47
3.3	New Features.....	48
3.3.1	Advanced Model Specification Guidance.....	48
3.3.2	Textual CAMEL Model Editing.....	49
4	Conclusions.....	51
5	References.....	53

## Index of Figures

Figure 1: The overall editor exploitation use case .....	11
Figure 2: The actual CAMEL model editing use case .....	13
Figure 3: The CAMEL organisation model editing use case .....	14
Figure 4: The architecture of the web-based CAMEL editor .....	15
Figure 5: Initially launched form of the CAMEL editor .....	19
Figure 6: Login popup window shown to admin .....	23
Figure 7: Organisation menu enabled content .....	24
Figure 8: New organisation model popup window .....	24
Figure 9: Organisation perspective launched, populated with the new organisation model information .....	25
Figure 10: New user creation .....	26
Figure 11: The new role assignment created .....	27
Figure 12: New application with unfilled information .....	28
Figure 13: New application with completed information .....	28
Figure 14: The new deployment model pop up window .....	29
Figure 15: Addition of SimulationManager component .....	30
Figure 16: The addition of the communication node .....	31
Figure 17: The creation of the SimulationManagerVM virtual machine .....	32
Figure 18: The creation of the SimulationManager Hosting .....	32
Figure 19: The enabled content of the Requirement menu .....	33
Figure 20: The creation of the ubuntu OS requirement .....	33
Figure 21: The quantitative hardware requirements specified for the VM of the SimulationManager .....	34
Figure 22: Filled in information before the SLO is added .....	35
Figure 23: Visualisation of the added SLO .....	35
Figure 24: The metric schedule added .....	36
Figure 25: The measurement window added .....	37
Figure 26: Update of the composite metric context .....	38
Figure 27: The requirement set associated to the VM of the SimulationManager .....	39
Figure 28: The specification of the name for the new scalability model .....	40
Figure 29: The SLO violation non-functional event created .....	41
Figure 30: The horizontal scaling action created for the SimulationWorker component .....	42
Figure 31: The scalability rule created for the SimulationWorker component .....	43

## Index of Tables

Table 1: The aspects covered by the editor according to which CAMEL release and the respective role responsible for manipulating them .....	8
Table 2: The current satisfaction level of the editor design requirements .....	18
Table 3: Timeline for the delivery of forthcoming updates and new features of the web-based CAMEL editor .....	44

# 1 Introduction

Any kind of software tool that offers a certain functionality needs to become usable through the offering of a nice, ergonomic and easy-to-use user interface (UI) which can enable the user to better exploit this tool in the most suitable and user-intuitive way. Many tools have not had the appropriate recognition due to the wrong or improper design and functioning of their UI.

In this respect, the UI of the Melodic platform should enable users to better exploit its underlying functionality in order to optimally cover their needs. Such needs come with automatic and optimal big data application deployment and reconfiguration across different clouds. To cover these needs at the UI level, the Melodic platform offers different UI components: (a) the Metadata Schema Editor; (b) the Weights Calculator; and (c) the CAMEL Editor. The first two editors have been the subject of analysis for the deliverable "D3.1 Metadata schema management" [1].

It should be highlighted here that the CAMEL web-based editor reflects the current release of CAMEL, R1.5, which is in accordance to the 1.5 Release of the Melodic platform. In this respect, it does not cover the (big) data aspect as well as other extensions or modifications to CAMEL that are planned to be performed for the next release of Melodic. Such extensions and modifications will be reflected in the forthcoming releases of the CAMEL editor in the context of the Melodic project. The way these updates as well as other related work directions will be conducted along with their respective planning will be analysed in Chapter 3 of this deliverable.

This deliverable aims at being a companion to the Melodic platform users, i.e., application owning organisations, by introducing to them the web-based CAMEL editor of the Melodic platform, as well as explicating how it can be exploited. It is written in such a manner that it can be comprehended by any kind of reader, provided that he/she has some small background on model-driven engineering and cloud computing.

The rest of this document is structured as follows. The next chapter analyses the web-based CAMEL editor. Chapter 3 explicates how this editor will evolve in the context of the Melodic project. Finally, the last chapter summarises this deliverable.

## 2 CAMEL Editor

The goal of this chapter is to analyse the web-based CAMEL editor. To this end, this chapter is structured according to the following sections, each having a different analysis sub-goal to satisfy:

- Introduction: this section supplies a short overview of the editor
- Use Case Analysis and Requirements: this section focuses on explaining what were the requirements that drove the editor design. It also analyses the main use cases pertaining to the actual usage/exploitation of this UI component by the different kinds of users of the Melodic platform
- Architecture: this section analyses the main editor parts and supplies some implementation details related to them.
- Features and Requirement Satisfaction: this section highlights the main features of the CAMEL editor and explicates which from its original (design) requirements are fully or partially satisfied.
- Editor Walkthrough: this section provides a walkthrough over the usage of the editor based on the modelling of a cloud-based application that pertains to a specific use case of the Melodic project.

### 2.1 Introduction

The CAMEL web-based editor is a UI component which enables users to edit CAMEL models of big data cloud applications over the web. It was originally developed in the context of the PaaSage<sup>1</sup> project to cover the editing of CAMEL models for two kinds of users: (a) (platform) admins; (b) devops users. This editor does exploit Eclipse technologies and has been also developed as an Eclipse plug-in, but, in contrast to the other editors of CAMEL (see “D2.1 System specification document” [2] for an analysis of them), it does not necessarily rely on any Eclipse environment in order to be actually launched. This is due to the fact that it can be compiled into a standard WAR file which can be launched by any kind of servlet container like Tomcat<sup>2</sup>. Furthermore, it is an on-line editor which means that the CAMEL models edited are immediately stored in the underlying Model Repository. This web-based editor originally covered only particular aspects of CAMEL, namely the requirements and organisation ones. However, as it was decided (3rd Melodic Plenary Meeting (PM) in Oslo) to use this editor as the main facility via which CAMEL models can be managed in the Melodic platform, it has been extended with the capability to cover all aspects that are relevant in terms of user input. Further, this editor will follow soon the extensions

---

<sup>1</sup> [www.paasage.eu](http://www.paasage.eu)

<sup>2</sup> <http://tomcat.apache.org/>

performed in CAMEL to cover the big data aspect, mapping to the new CAMEL release (R2.0), according to its implementation plan supplied in Chapter 3. These extensions have been already analysed in “D2.2 Architecture and initial feature definitions” [3]. In the following table, we highlight the respective aspects of CAMEL that are currently, or will be soon, covered by this editor (see Camel Release column) as well as the corresponding (user) role that is going to manipulate them. Please note that there is now also space for the business (manager) role in terms of allowing this role to specify some high-level requirements (e.g., overall application cost constraints and optimisation objectives). The asterisk in the values/cells of the CAMEL Release column indicates that the respective aspects are covered in the R1.5 release of CAMEL, but they are planned to be extended/modified towards delivering the CAMEL R2.0 release.

*Table 1: The aspects covered by the editor according to which CAMEL release and the respective role responsible for manipulating them*

CAMEL Aspect	CAMEL Release	User Role
Deployment	R1.5*	Devops
Requirement	R1.5*	Devops, Business
Metric	R1.5*	Devops
Scalability	R1.5*	Devops
Data	R2.0	Devops
Unit	R1.5*	Devops
Type	R1.5*	Devops
Organisation	R1.5*	Admin

## 2.2 Use Case Analysis and Requirements

### 2.2.1 Requirements

The original web-based CAMEL editor version was first presented to the project use case partners both in a teleconference as well as at the 3rd partner meeting in Oslo. There, the use case partners expressed their preference of the web-based editor over the textual editor of CAMEL as the web-based editor seemed to better suit their requirements. In particular, the web-based editor seemed to better follow the current working practices of their users while it was also more user-intuitive, especially with respect to better guiding the users in providing the right information in a valid manner.



In this respect, once this web-based editor was selected to become part of the Melodic platform and especially its UI, particular further requirements were posed to it in order to extend its usage, both in terms of covering additional aspects as well as covering all possible user roles. Such requirements (both new and old; old in terms of the original design of the editor), which were also expressed via the Melodic project's JIRA service, were the following:

- **R1 – Application Deployment Coverage:** Enable a valid specification of the deployment structure and requirements for big data applications
- **R2 – Requirement Coverage:** Enable a valid specification of the non-functional requirements of big data applications
- **R3 – Scalability Coverage:** Enable a valid specification of scalability rules for these applications
- **R4 – Metric Coverage:** Enable a valid specification of metrics and their computation formulas. The latter should be specified via mathematical expressions by exploiting a certain mathematical language.
- **R5 – Big Data Coverage:** Enable a valid description of the (big) data that are manipulated by big data applications
- **R6 – Organisation Coverage:** Enable a valid specification of organisational information, including users and their roles
- **R7 – Online CAMEL Model Manipulation:** Allow the manipulation of models, including both the addition, updating and deletion of models in an online manner by applying the respective changes immediately on the underlying Model Repository. This enables users not to perform an extra step after the editing of CAMEL models for storing them in the Model Repository before their exploitation by the Melodic platform can be initiated.
- **R8 – Controlled Model Access:** Allow a controlled access on the different model aspects based on the respective roles of an organisation and their rights.
- **R9 – Application Deployment Launching:** Launch the deployment of a big data application once its CAMEL model has been specified and is complete.
- **R10 – Cooperation with Metadata Schema Related Editors:** Cooperate (directly or indirectly) with the metadata-schema-related editors to allow the user to: (a) incorporate the weights of specific optimisation objectives to the respective optimisation requirements in the CAMEL model; and (b) to annotate the various information pieces that are specified in the CAMEL model via respective metadata elements (i.e., concrete elements from the instance of the metadata schema exploited by the Melodic platform).

Requirements R1-R6 cover the extent of the kinds of models to be manipulated via the web-based editor while highlighting the need for imposing the semantics of the CAMEL language. Requirements R7-R8 explicate what kind of manipulation over the models should be allowed to be performed, but only in a controlled manner by corresponding valid users of the platform. Requirement R9 enables users to immediately launch the deployment of a big data application once its CAMEL model has been completely specified, bypassing in this way any kind of

script/command-line based method or interface that might alternatively be supplied by the Melodic platform. Finally, Requirement R10 enables the cooperation between the CAMEL and the metadata-schema-related editors towards enhancing the CAMEL model with user preferences and annotations.

Please note that as a side-effect of Requirements R1-R6, the CAMEL web-based editor would need to also cover the unit and type aspects of CAMEL. This could be regarded as a derived Requirement R11. However, as it will be shown later on in this chapter, the management of these two aspects is included in the management of metrics, thus the realisation of Requirement R4. Furthermore, Requirement R9 could be considered as slightly beyond the scope of the CAMEL editor. However, it has been decided to be satisfied for facilitating the more rapid deployment of big data applications. Moreover, as the CAMEL editor is the main responsible component for checking the validity of CAMEL models, it is more natural to employ it in order to also assess their completeness before they can be used for launching the deployment of the corresponding big data applications.

### 2.2.2 Use Case Analysis

Based on the above requirements as well as the expected functionality to be delivered by the CAMEL editor, a set of use cases were developed so as to guide the design of the extension of this editor. These use cases are now analysed in detail in the following paragraphs.

### 2.2.2.1 Overall Editor Exploitation Use Case

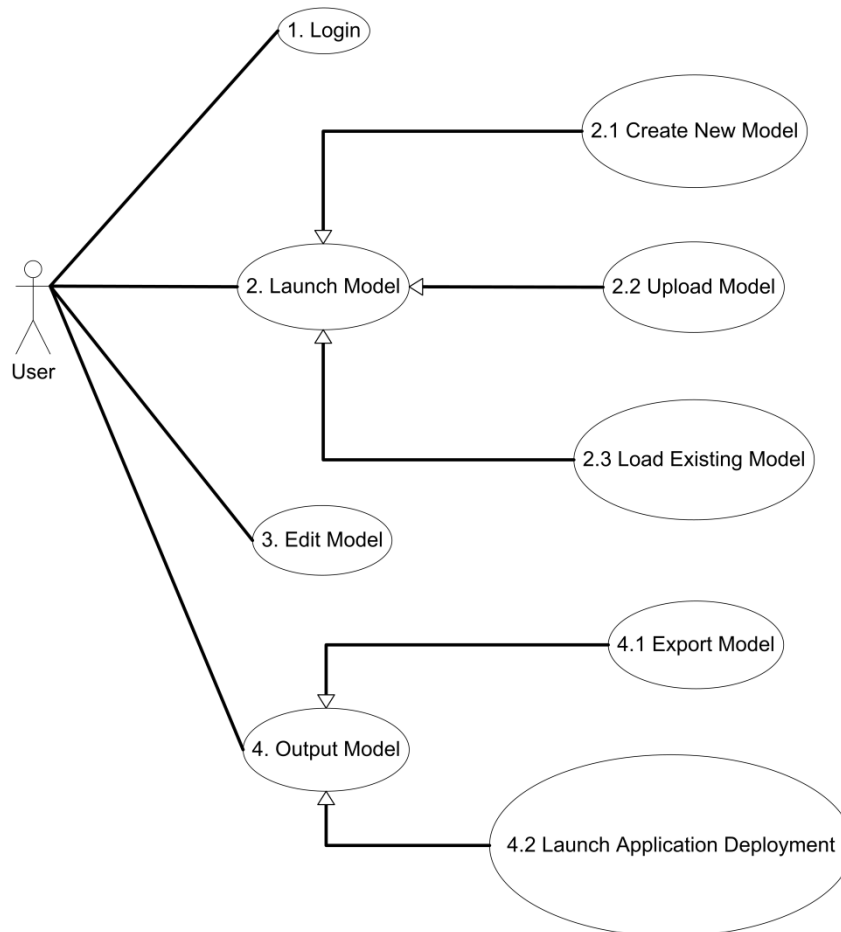


Figure 1: The overall editor exploitation use case

In this use case, which is depicted in Figure 1, the user first logs in using the editor. Once the user authentication is successful, the user should launch the respective model in order to start editing it. Such a launching can be performed in three different ways: (a) a new model is created; (b) an existing model is loaded; (c) a model is uploaded/imported (which could map to the case where this model has been specified by a different editor, like the textual one). Once this launching is done, the editing of the model can be conducted. This editing is elaborated more in a separate use case. Finally, the user can exploit the final model, after its editing in two alternative ways: (a) export the model (e.g., in order to be exploited via a different CAMEL editor, like the textual one); or (b) use the model for initiating application deployment.

#### 2.2.2.2 CAMEL Editing Use Case

The editing of a CAMEL model is covered by the following use case which is depicted in Figure 2. As can be seen, the editing of a CAMEL model can be broken down into the editing of the respective CAMEL sub-models (e.g., requirement, deployment, etc.). Such an editing can be activated or de-activated depending on the rights of the user that performs it. In particular, an admin user can only edit organisation models while a business user can edit only requirement models. On the other hand, the most privileged role is that of the devops which can edit all kinds of CAMEL models apart from the organisation model.

In the editing of any kind of CAMEL sub-model, the respective elements can be annotated through the usage of an instance of the Metadata Schema. In this respect, a corresponding sub-case (seen at Figure 2 – 3.7 *Annotate Model*) has been generated which involves the retrieval of this metadata schema instance and its exploitation for the annotation of the CAMEL elements edited. In a more restricted form of exploitation, user preferences are retrieved from the Connected Data Objects (CDO) *Model Repository* and exploited for updating the weights of the respective optimisation objectives given by the user within the requirement model that is currently edited. This is depicted in Use Case 3.2.1 *Exploit Weights* (seen Figure 2). Both the instance of the metadata schema as well as the user preferences are the output of two editors, the Metadata Schema Editor and the Weights Calculator, respectively. Their indirect exploitation by the CAMEL editor is also in line with the Use Case “Metadata Schema usage in App Modelling” in deliverable D3.1 [1].

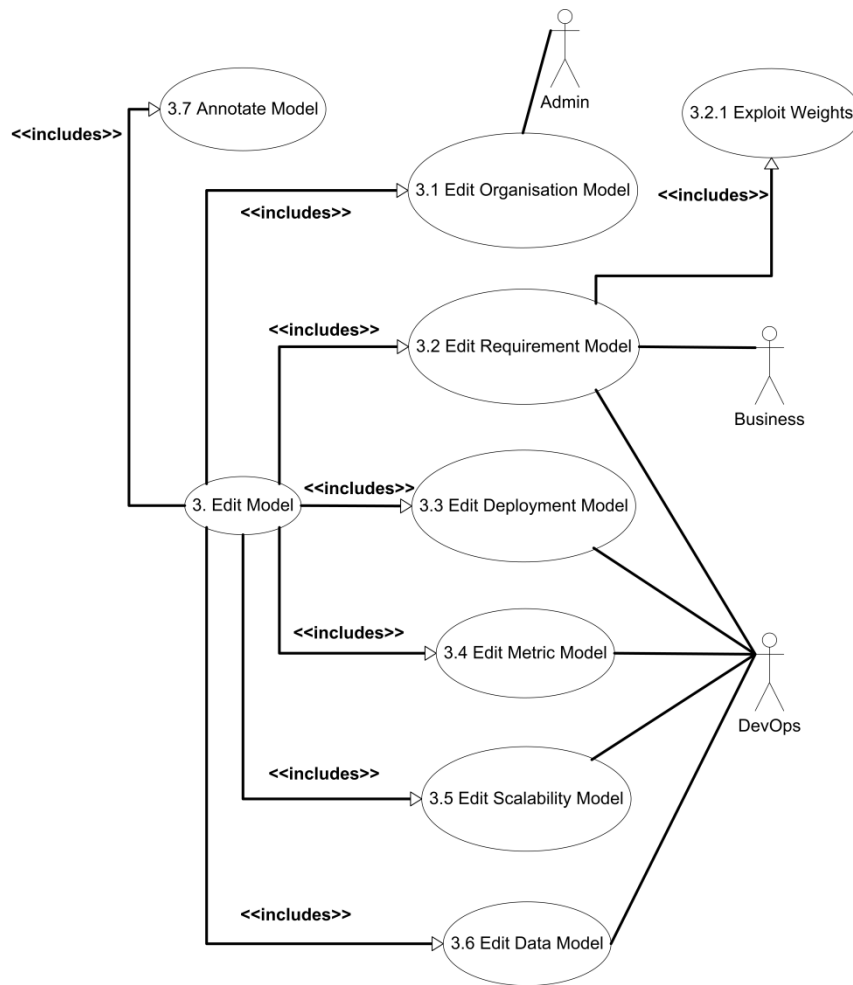


Figure 2: The actual CAMEL model editing use case

### 2.2.2.3 User Management Use Case

This use case, depicted in Figure 3, attempts to elaborate more on the way an organisation model can be edited by the admin user. Such an elaboration is performed also to highlight that this editing should precede the actual editing of a CAMEL model of a user application. First the devops users of the organisation need to be modelled, before they can start editing any application-specific CAMEL model.

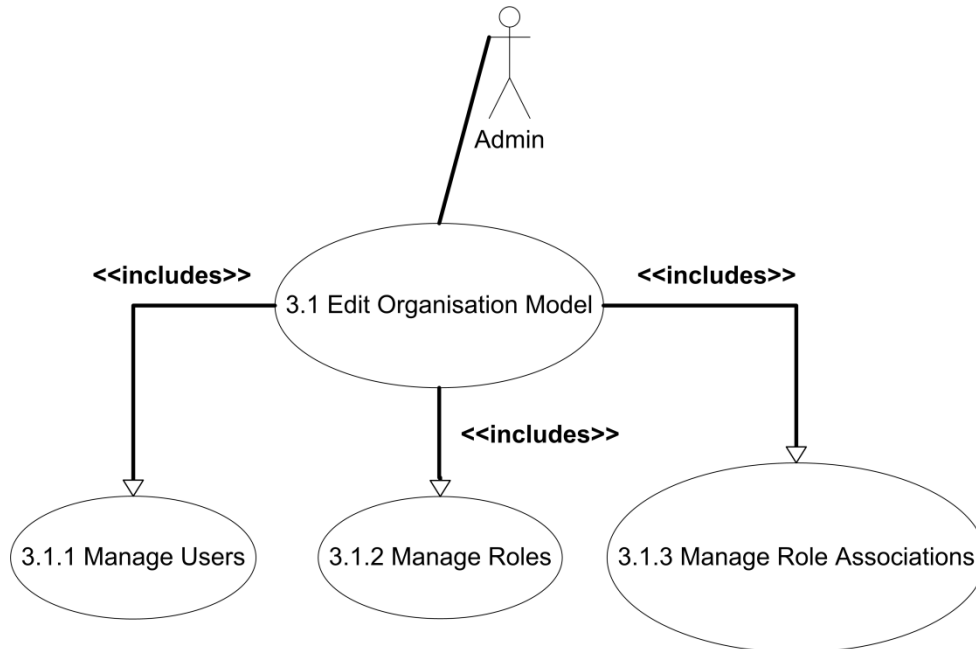


Figure 3: The CAMEL organisation model editing use case

As it can be observed from Figure 3, the editing of an organisation model includes: (a) the management (creation, deletion, updating) of the users, or user groups, of the organisation operating the Melodic platform; (b) the management of the roles that can be assigned to users; and (c) the management of the role assignments, i.e., the assignments of roles to respective organisation users or user groups. Please note that any organisation user should be assigned a respective role (i.e., admin, business or devops) in order to be able to exploit the web-based CAMEL editor.

## 2.3 Architecture

The architecture of the web-based CAMEL editor, which is depicted in Figure 4, is quite simple, as it includes mainly three components: (a) the Admin which is responsible for the manipulation of the CAMEL organisation model of an organisation; (b) the Importer which is responsible for the importing of base CAMEL models (e.g., metric models which include basic metrics like average execution time that can be used for the specification of SLOs or non-functional events) that can be exploited/re-used for creating new ones; and (c) the Editor itself which is a servlet, realising the core web-based editor by offering the CAMEL model editing and storage functionality. The latter component embeds the CDO Client component in order to enable the management of the models edited within the Model Repository. This management includes the storage and loading of CAMEL models as well as their validation.

The Editor also interacts with other components of the Melodic platform in order to deliver its functionality. In particular, it indirectly interacts with the Weights Calculator in order to obtain the weights of non-functional metrics (to be used to annotate respective optimisation objectives), which have been derived from the user preferences and stored in the Model Repository. Furthermore, the Editor indirectly interacts with the Metadata Schema Editor in order to obtain an instance of the metadata schema, stored in the Model Repository, which can be exploited for annotating the CAMEL model elements. On the other hand, the Editor interacts with the platform Control Plane in order to launch the deployment of a big data application specified by the currently manipulated CAMEL model.

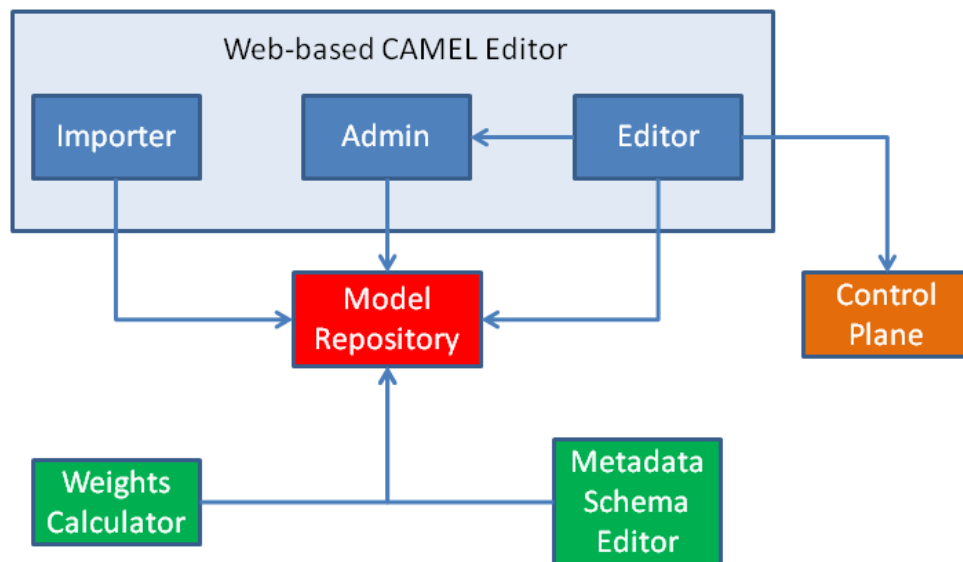


Figure 4: The architecture of the web-based CAMEL editor

## 2.4 Implementation Details & Usage Instructions

Implementation-wise, all of the components of the web-based CAMEL editor have been realised. However, some of their intended interactions with other Melodic components have not yet been implemented. In particular, the editor is not yet able to exploit output produced by the *Metadata Schema Editor* and the *Weights Calculator*. This is due to the fact that the focus of implementation was more on covering all aspects of the R1.5 release of CAMEL.

All the web-based CAMEL editor components have been developed in Java as Maven projects. The *Editor* development exploited the Eclipse's Remote Application Platform (RAP<sup>3</sup>) technology. This technology enables the production of modern web-based applications using a rich in features widget toolkit with a Standard Widget Toolkit<sup>4</sup> (SWT) API. The latter includes interesting

<sup>3</sup> [www.eclipse.org/rap](http://www.eclipse.org/rap)

<sup>4</sup> <https://www.eclipse.org/swt/>

capabilities like drag'n drop, on-demand data loading, and inline editing and drawing. RAP enables to build web applications through pure servlet technology, which can be run on any J2EE container and integrated with Open Service Gateway Initiative<sup>5</sup> (OGSi), if needed. Most importantly, RAP enables to re-use code from different target platforms. This is an essential feature that enabled us to integrate important Eclipse plugins like EMF<sup>6</sup> and CDO.

The source code is available in the project gitlab<sup>7</sup> and can be compiled via Maven. However, for manageability reasons, the code has been split into three projects, mapping to the three main components of the editor architecture. The *Admin* and *Importer* components do not depend on each other, so they can be compiled as a first step (with the usual command "mvn clean install"). However, as the *Editor* depends on the *Admin* component, it needs to be compiled subsequently.

There is a need for a specific execution order via which the editor can be actually run. As a pre-requisite, the CDO Server<sup>8</sup> component should already be running in secure mode. The latter indicates that the security feature<sup>9</sup> of the CDO technology is exploited to secure access to the models managed by the CDO-based *Model Repository*. Once this is done, the order of execution is as follows:

- First, the *Admin* component needs to be executed as it needs to give proper rights to the admin user on the underlying *Model Repository*. These rights indicate that the user can edit the organisation model of the organisation that operates the Melodic platform (i.e., the *platform operator*). This component can be executed by supplying the following command:  
`"java -jar administration-2.0.0-SNAPSHOT-jar-with-dependencies.jar`

`-Deu.paasage.configdir=<path>",`

where <path> is the file directory path where the eu.paasage.mddb.cdo.client.properties file resides (required for properly configuring the encompassed *CDO Client* for connecting to the *CDO Server* that manages the underlying CDO *Model Repository*).

- Second, the *Importer* component needs to be executed in order to import all CAMEL base models into the *Model Repository*. Such models, as already indicated, enable the re-use of their elements for building CAMEL application models. For instance, they include metric & unit models, which enable their re-use for specifying metric conditions and corresponding SLOs as well as the units of new metrics, respectively. The command to execute this component is the following:

`"java -Deu.paasage.configdir=<path> -jar`

`importer-2015.9.1-SNAPSHOT-jar-with-dependencies.jar",` where again <path> is the file directory path where the eu.paasage.mddb.cdo.client.properties file resides.

---

<sup>5</sup> <https://www.osgi.org/>

<sup>6</sup> <http://www.eclipse.org/modeling/emf>

<sup>7</sup> [https://bitbucket.7bulls.eu/projects/MEL/repos/camel/browse/web\\_editor?at=refs%2Fheads%2Foxyen](https://bitbucket.7bulls.eu/projects/MEL/repos/camel/browse/web_editor?at=refs%2Fheads%2Foxyen)

<sup>8</sup> [https://bitbucket.7bulls.eu/projects/MEL/repos/upperware/browse/cdo\\_server](https://bitbucket.7bulls.eu/projects/MEL/repos/upperware/browse/cdo_server)

<sup>9</sup> [https://wiki.eclipse.org/CDO/Security\\_Manager](https://wiki.eclipse.org/CDO/Security_Manager)



- Finally, the *Editor* component needs first to be installed within a servlet container, like Tomcat. Once this is done, then the *Editor* will be launched as a servlet, which can then be immediately exploited by the respective users of the *platform operator*. An example of how this would be done in a Windows execution environment with Tomcat as the servlet container is as follows: (a) configure Tomcat to point to the directory where the `eu.paasage.mddb.cdo.client.properties` file resides by modifying the `catalina.bat` file with the following entry: `'set "JAVA_OPTS=%JAVA_OPTS% -Deu.paasage.configdir=<path>"`; (b) copy the compiled war file of the *Editor* to the `webapps` directory of Tomcat, followed by a start or restart of Tomcat, depending on its current status (*down* or *running*, respectively).

The Editor can be utilised by devops or business users once the admin user has updated the platform operator's organisation model to both model them as users as well as to associate them with respective roles. This actually maps to the following usage scenario, which should be followed in the very first moment that the editor has started to run:

1. Login as super administrator with credentials ("Administrator", "0000").
2. Create the Organisation Model
  - a. Create a User in that model
  - b. Via a role assignment, assign that user to the devops (or business) role (please note that the dates for this assignment should be in correct order, i.e., start date  $\leq$  assignment date  $\leq$  end date)
3. Logout
4. Login as the devops (or business) user by using your already specified credentials
5. Start playing around by creating an application and its deployment/requirement models (or requirement model in case of a business user)

## 2.5 Features and Requirement Satisfaction

By considering the design requirements of the editor, which have been specified in section 2.2.1, Table 2 explicates the level of satisfaction. As it can be seen, most of the requirements are already satisfied while a few of them will be addressed in the future evolution of this editor in the context of the Melodic project. This is in line with the implementation status that has been explicated in section 2.4.

Table 2: The current satisfaction level of the editor design requirements

Requirement	Satisfaction Level
R1 – Application Deployment Coverage	Full
R2 – Requirement Coverage	Full
R3 – Scalability Coverage	Full
R4 – Metric Coverage	Full
R5 – Big Data Coverage	-
R6 – Organisation Coverage	Full
R7 – Online CAMEL Model Manipulation	Full
R8 – Controlled Model Access	Full
R9 – Application Deployment Launching	-
R10 – Cooperation with Metadata Schema Related Editors	-

The editor is a modern web-based application that is available to be used by the platform users in a remote manner, no matter where they reside. Apart from its coverage of the above requirements, which map to certain basic features, additional features have been implemented which are analysed as follows:

- Use of perspectives to allow focused editing of different CAMEL model kinds
- The ability to switch from one perspective to another dynamically and in a controlled manner
- Immediate availability of edited information from one perspective to another
- User-intuitive error handling – errors with respect to user input and CAMEL model validity are appropriately highlighted in a suitable form with up-to-the-point error messages communicated to the modeller while the respective “problematic” input UI element grabs the focus to rapidly correct the current error
- User-intuitive tree-based navigation of different CAMEL model elements pertaining to the current model kind of focus
- Exporting of the whole CAMEL model in XMI form

## 2.6 Editor Walkthrough

The editor has been implemented as a modern web application which is available via a certain URL and is deployed on the VM in which the respective Melodic platform instance resides. In case that the user needs to remotely manage the respective CAMEL models of his/her applications, the

VM should be configured to have the corresponding port open on which the servlet container hosting the editor listens (e.g., 80 or 8080). Obviously, the IP address of the VM should also be known to the user.

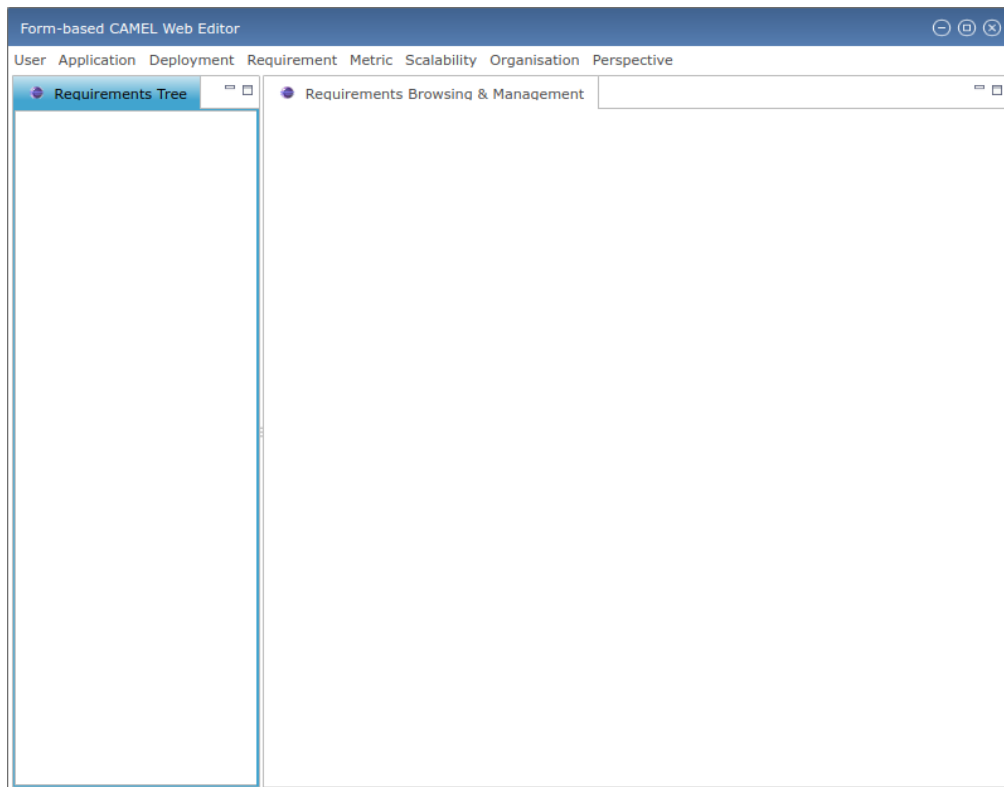


Figure 5: Initially launched form of the CAMEL editor

The UI of the editor in its initially launched form is depicted in Figure 5. As can be seen, the UI comprises three main parts: (a) the menu bar on the top of the UI; (b) the tree viewer on the left; and (c) the form-based editor on the right. The menu bar comprises different menus which map to the different aspects that need to be dealt with by the editor. These menus include:

- **User**: this menu allows the user to login or logout from the editor.
- **Application**: this menu enables the user to manage applications as well as launch their deployment.
- **Deployment**: this menu enables the user to manage the deployment structure of his/her big data applications.
- **Requirement**: this menu enables the user to manage the requirements of his/her application.
- **Metric**: this menu enables the user to manage metrics as well as other metric-related elements, like metric contexts and units.
- **Scalability**: this menu enables the user to manage the scalability rules that will drive the local adaptation behaviour of his/her applications.

- **Organisation:** this menu enables the admin to manage organisational information including users and roles.
- **Perspective:** this menu enables the user to switch from one perspective to the other according to his/her rights.

Apart from the first menu, the next 6 menus enable the actual manipulation of respective CAMEL models. Each menu is dedicated to one kind of CAMEL model with the sole exception of the *Metric* menu via which also other CAMEL model kinds apart from the metric one can be manipulated (unit and type model kinds, in particular). Please note that in this case the manipulation of the other model kinds comes through the actual interface and not via certain items from the menu. Each menu actually maps to a different perspective, which when launched leads to the population of the two other UI parts, i.e., the tree and form-based editor. While the last menu, as already explained, enables switching to another perspective, depending also on both the current user rights as well as the existence of the respective model kind (e.g., metric), inside the current overall CAMEL model that is edited. In particular, the user is presented with perspectives which can be already activated only if both the user has the right to manage the corresponding model kind and this model kind is already included in the overall CAMEL model.

The population of the tree and form editing parts of the UIs depend on the content of the respective CAMEL model (kind), if such a content exists. In any case, the tree-based UI part shows a tree hierarchy with the whole model as the respective root tree node and respective children which map to containers of different elements. For example, in the case of a metric model, the top element of the tree would be a node named after the metric model, while the children of this node would map to containers for metrics, properties and other metric-related elements which are named in plural form after the respective name of the element kind to which they map (e.g., "metrics", "properties", etc.). Indeed, if there exist already elements for a specific element kind, like a metric, then the respective children of the container node (e.g., "metrics") would have been already created (e.g., "average response time" in case of a specific metric).

On the other hand, the form-based UI part of the editor is structured in the form of tabs, where each tab is dedicated to the definition or updating of a respective element kind. These tabs are also named in plural form after the name of the corresponding element kind (e.g., "metrics"). Upon the first launch of the corresponding perspective, the first tab in order is always selected and its actual content is shown. It is always represented as a form, which includes different parts organised into respective named UI element groups/containers. At the bottom of this form, the buttons that enable to manipulate the element to be created or updated are situated. Such an element should be selected from the tree part of the editor. For this selection, the following two cases hold.

First, a container node has been selected. In this case, the form is rather empty as it is prepared to host the editing of information for a new element of the current element kind. For instance, if the "metrics" container is selected, this means that a new metric element will need to be created in

the current CAMEL metric model. Upon this selection, both the respective tab (“metrics”) and the corresponding form will be initialised, while only the “Add” button will be enabled.

Second, a particular element has been selected, e.g., a specific metric. In this case, we have again the selection of the correct tab in the form-based UI part with the exception that now: (a) the form is populated with the information of the current element (e.g., its name and unit, in case of a *Metric* element); and (b) the “Add” button is now disabled and the “Update” and “Delete” buttons are enabled. The latter buttons allow the user either to update the information of the metric, if it has been modified, or to delete this metric from the current (CAMEL metric) model.

An important feature of the editor is that the editor does not require users to supply all the information needed by a normal, valid CAMEL model. The editor actually takes care of the completion of some information automatically by considering its correlation with information already supplied. For instance, in the context of VMs in deployment models, the user does not have to provide the provided host port for this VM. This is internally created by the editor as soon as the VM element is graphically created by the user. This nice feature, along with proper error handling for model validation purposes, makes the editor not only user-intuitive and friendly, but also an important companion to the modeller which guides and assists him/her in faster production of only valid CAMEL models.

We should highlight that error handling actually enables to conform to the semantics of CAMEL which have been supplied by both the Ecore<sup>10</sup> CAMEL model and its OCL specification. Such a conformance is achieved by forbidding the creation or updating of elements which are not valid (either internally or externally as they introduce inconsistencies with respect to other CAMEL models). This denying of user CAMEL model manipulation actions is accompanied by proper error messages as well as a change of UI element focus to exactly pinpoint for the user the right UI part in which the respective resolution needs to be performed. For example, in the case of VM creation, if the user does not provide any name or an empty one for the VM, then an error will be prompted indicating that a non-empty VM name should be supplied. Further, the respective text field in the form-based UI part of the editor will grab the focus to allow the user to supply the needed (VM) name.

In order to start the walkthrough on the different aspects of CAMEL that can be manipulated via the editor, we consider the scenario which has been partly explained in section 2.2.2 (user registration – also mapping to Use Case 3.1 seen at Figure 3). The only difference lies in the very last step of this scenario, which will be elaborated further. In order for the presentation to be as real and up-to-the-point as possible, we consider the situation that a registered devops user desires to create a CAMEL model for the Melodic’s Traffic Management use case<sup>11</sup>.

---

<sup>10</sup> <https://wiki.eclipse.org/Ecore>

<sup>11</sup> <http://melodic.cloud/use-cases.html>

In order not to make the presentation verbose and lengthy, we rely on a restricted form of this use case for editor illustration purposes. In this restricted form, we will focus mainly on four main CAMEL aspects, i.e., deployment, requirement, metric and scalability. These are actually the aspects that are usually edited in the context of a traditional (i.e., non-data-based) CAMEL model.

Concerning the deployment aspect, the devops user requires to model two internal application components, the *Simulation Manager* and the *Simulation Worker*. The first component is responsible for managing the whole traffic simulation experiment, while the worker is a piece of functional unit that takes care of conducting a single simulation. Obviously, both of these components will communicate with each other and will be associated with certain configuration commands. In addition, they will be hosted on VMs with different quantitative hardware characteristics. However, both VMs will also share some features, like the same operating system (OS) (Ubuntu, in this particular case).

The quantitative hardware requirements that each VM will need to satisfy are the following:

- *SimulationManagerVM*: cores in [1,4], CPU in [1.0,3.0], RAM in [2,4] GB and storage size in [100, 1000] GB.
- *SimulationWorkerVM*: cores in [72,144], RAM in [144,288] GB and storage size in [100, 500] GB.

The devops user will require to scale the *Simulation Worker* component with 5 more instances where its *average execution time* goes beyond 5 minutes. The *average execution time* of such a component will be measured every 7 minutes with a measurement window of 10 minutes. Such a metric is computed from the *raw execution time* metric, which is computed on demand at the component (instance) side every time a simulation is ended.

### 2.6.1 Admin Login

The admin of the platform operator needs to select the *User* menu and the *Login* option. Once this is done, a pop-up window (see Figure 6) is launched which enables the admin to provide his/her credentials. The admin can then press the "OK" button in order to be authenticated.

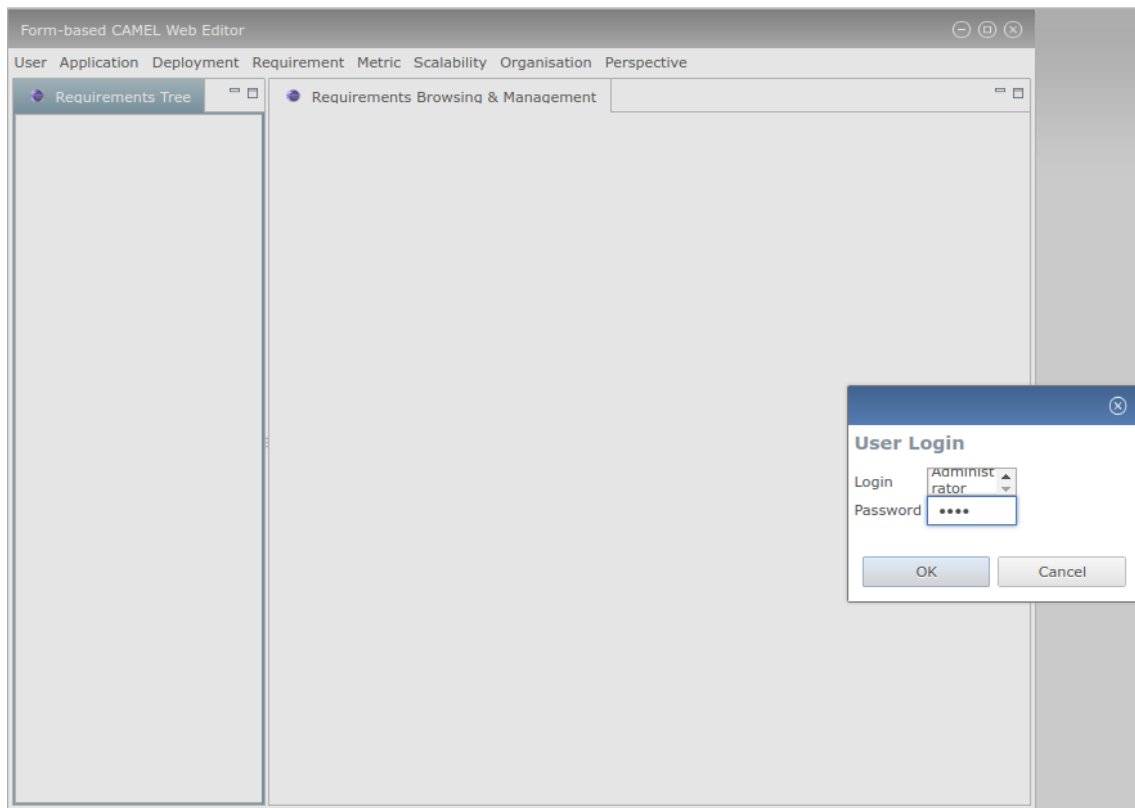


Figure 6: Login popup window shown to admin

## 2.6.2 Organisation Model Editing

Upon successful admin authentication, the respective perspectives/menus that the admin user can manipulate can be selected. As we consider that the Melodic platform has been just created, the platform operator's organisation model is not yet generated. In this respect, the "New Organisation Model" option in the Organisation menu will be activated by the editor (see Figure 7). The admin would need to press this option in order to start the editing of the organisation model to be created. Please note that in the case that the organisation model already existed, it would have been directly launched, but only for users which undertake the admin role.

As the organisation model will be newly edited, the admin is first presented with a pop up window that prompts him/her to supply the name of that model, as well as to determine whether this model should belong to a simple or cloud provider organisation. This is depicted in Figure 8.

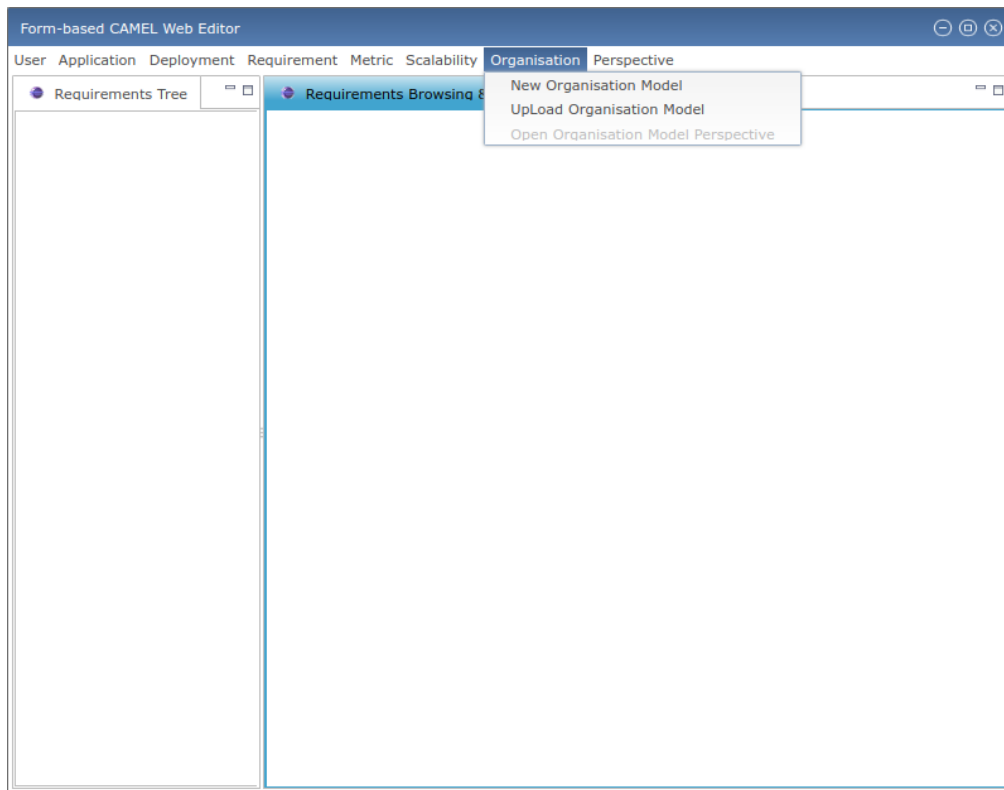


Figure 7: Organisation menu enabled content

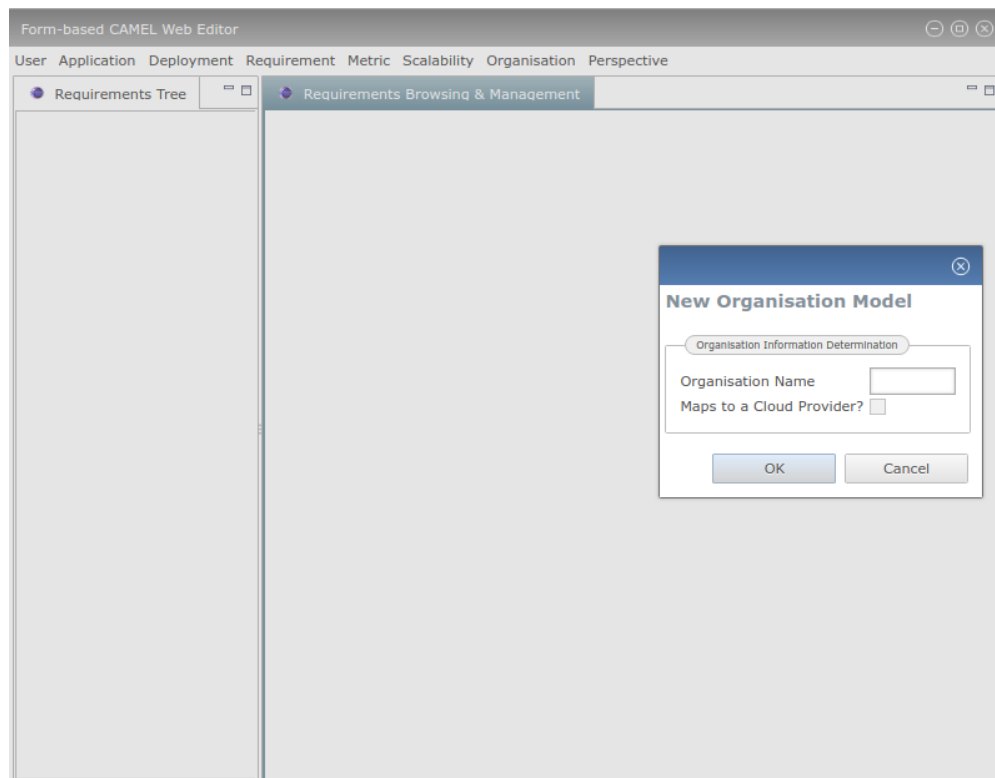


Figure 8: New organisation model popup window



Once the right input is given by the admin, the two central UI components of the editor start to become populated.

Initially, the model, as shown in the tree-part of the UI (see Figure 9), involves the creation of an organisation plus three role elements (admin, devops, business). Once the admin selects the organisation element (named as "cet"), the form-based UI part will focus on the respective Organisation tab, which includes forms enabling the admin to supply particular information for his/her organisation - like the email and contact address. If the organisation is a cloud provider, additional information can be enabled and thus supplied.

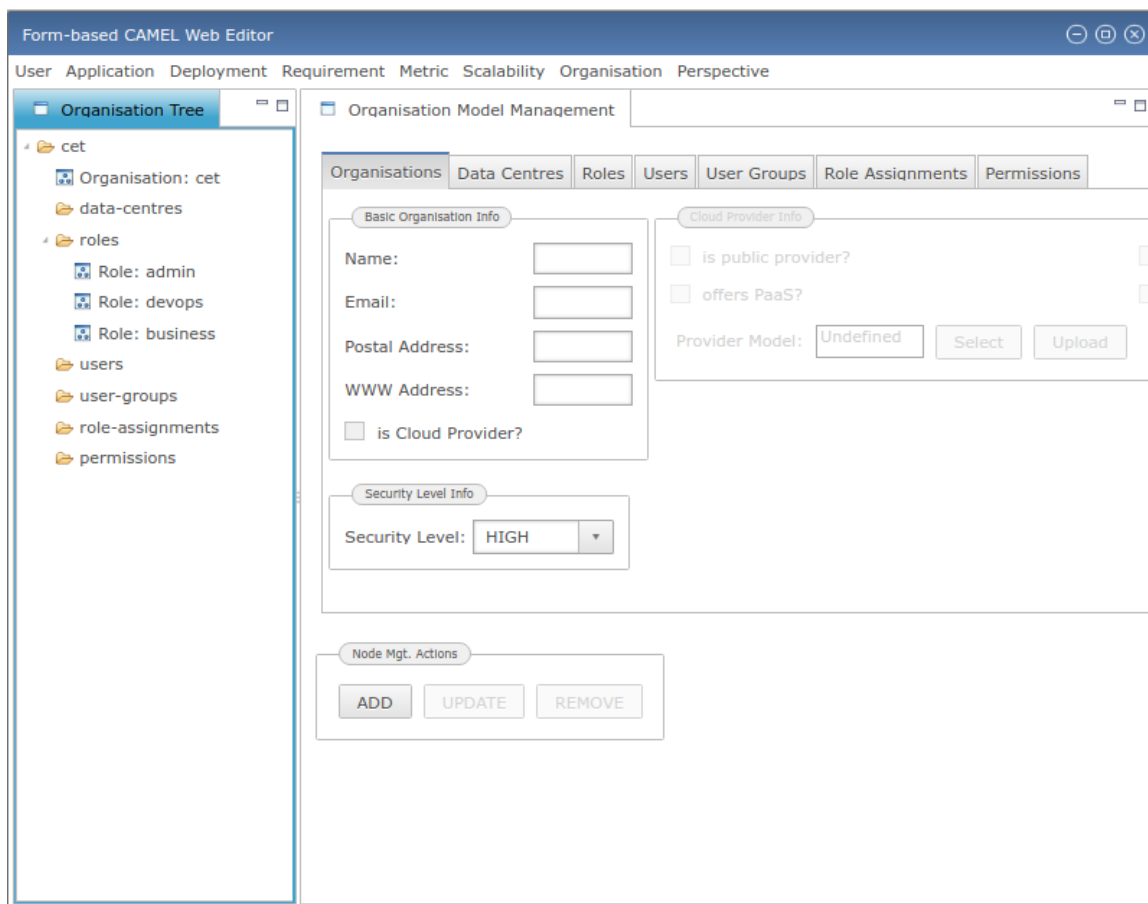
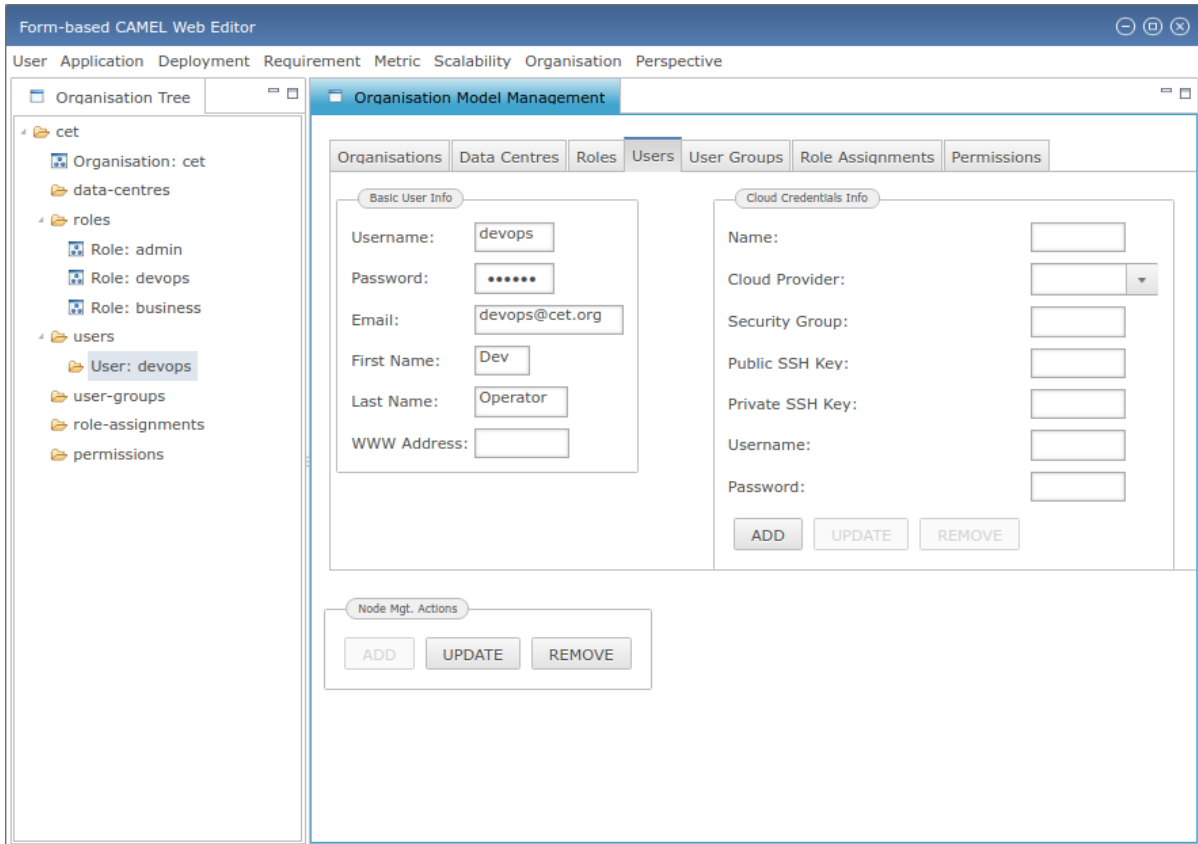


Figure 9: Organisation perspective launched, populated with the new organisation model information

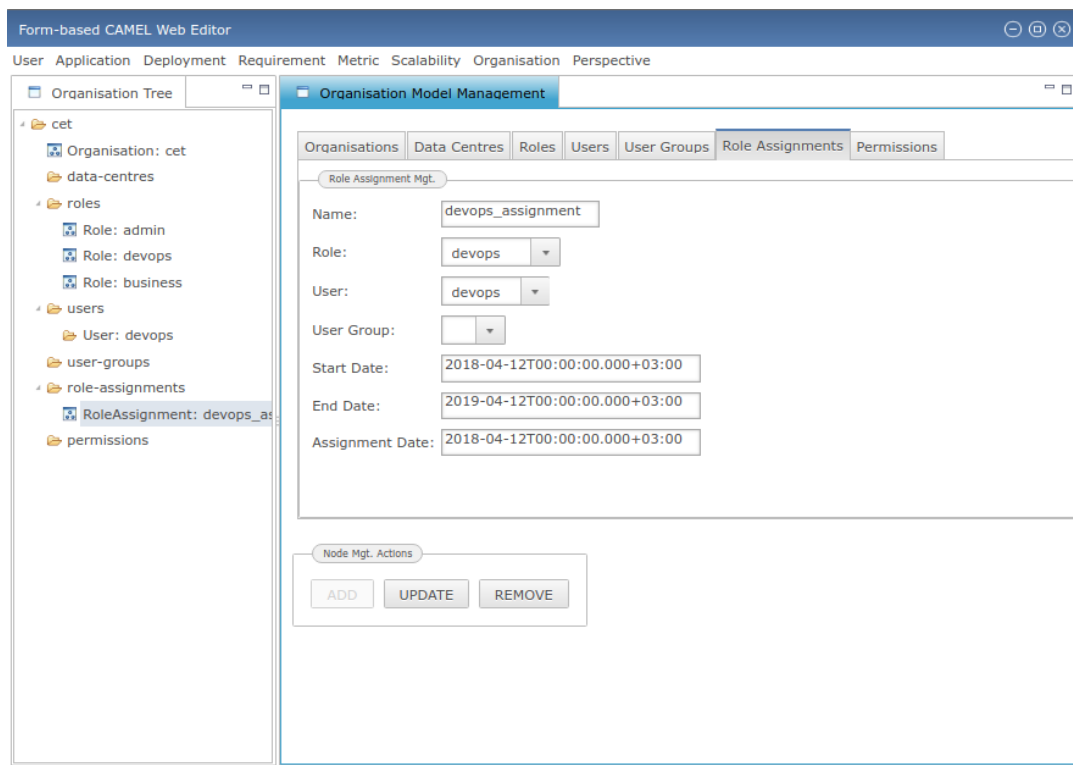
As the admin requires to create a new user, he/she selects the "users" node in the tree part of the UI. This selection then enables to focus on the "Users" tab in the form-based UI part. There, the admin can provide all necessary information about the user (e.g., first and last name, email, and name/login & password) and press the "Add" button. This can then lead to: (a) the creation of a new element named after the name/login of the user created in the tree under the "users" container node; as well as (b) the change in status of the management buttons in the form-based UI part to now allow the modification of information of the newly created user or its deletion (see Figure 10).



The screenshot shows the 'Form-based CAMEL Web Editor' interface. The top navigation bar includes tabs: User, Application, Deployment, Requirement, Metric, Scalability, Organisation, and Perspective. The 'Organisation' tab is active, and within it, the 'Organisation Model Management' sub-tab is selected. On the left, the 'Organisation Tree' shows a hierarchy: cet (Organisation), data-centres, roles (admin, devops, business), users (devops), user-groups, role-assignments, and permissions. The 'Users' sub-tab is active in the main form area. It contains two sections: 'Basic User Info' and 'Cloud Credentials Info'. The 'Basic User Info' section has fields for Username (devops), Password (masked), Email (devops@cet.org), First Name (Dev), Last Name (Operator), and WWW Address. The 'Cloud Credentials Info' section has fields for Name, Cloud Provider (dropdown), Security Group, Public SSH Key, Private SSH Key, Username, and Password. At the bottom of the form, there are 'ADD', 'UPDATE', and 'REMOVE' buttons. Below the form, there is a 'Node Mgt. Actions' section with 'ADD', 'UPDATE', and 'REMOVE' buttons.

Figure 10: New user creation

However, the duty of the admin is not yet over. The created user also needs to be assigned to a certain role. This creates the need for selecting the “role assignments” container in the tree. This action will then lead the focus on the “Role Assignments” tab of the form-based UI editor part, which is now ready to host information related to the current user. In particular, the admin can select both the user from a drop-down user list (containing all users that have been created so far) as well as the actual role to be assigned to him/her (again obtained from a drop-down list, which initially contains the already generated 3 aforementioned roles). The admin should also supply the starting and final validity date for the assignment, as well as the date in which the assignment was introduced to the system. Once all this information is entered, the admin can press the “Add” button and the role assignment will be created, shown then as an element/child of the “role assignments” container node in the tree (see Figure 11).



The screenshot shows the 'Form-based CAMEL Web Editor' interface. On the left is an 'Organisation Tree' with a hierarchy: cet (Organisation) -> data-centres -> roles -> Role: devops -> RoleAssignment: devops\_assignment. The main panel is titled 'Organisation Model Management' and has tabs for Organisations, Data Centres, Roles, Users, User Groups, Role Assignments, and Permissions. The 'Role Assignments' tab is active, showing a 'Role Assignment Mgt.' form. The form contains the following fields: Name (devops\_assignment), Role (devops), User (devops), User Group (empty), Start Date (2018-04-12T00:00:00.000+03:00), End Date (2019-04-12T00:00:00.000+03:00), and Assignment Date (2018-04-12T00:00:00.000+03:00). At the bottom, there is a 'Node Mgt. Actions' section with buttons for ADD, UPDATE, and REMOVE.

Figure 11: The new role assignment created

As the responsibility of the admin hereby ends, the admin can now logout from the editor. Internally, the admin needs to communicate to the user his/her credentials to enable him/her to utilise the editor - as will be shown next.

## 2.6.3 Devops Login

The newly created devops user can now authenticate in order to have access to the editor's main functionality. He/she just have to provide his/her credentials in the pop up window that will be launched when the "Login" option is selected in the "User" menu.

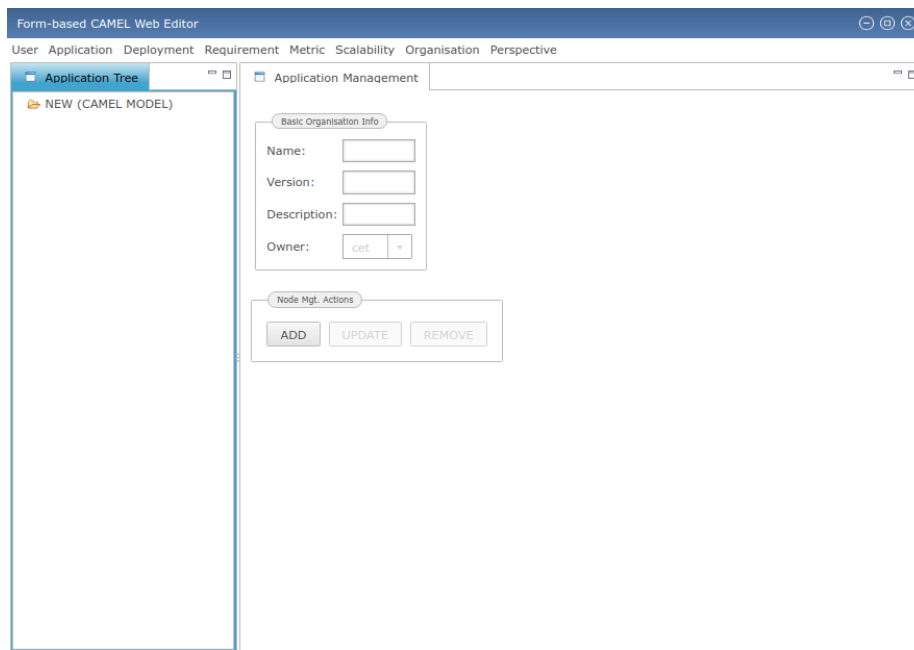
## 2.6.4 CAMEL Model Editing via Devops

### 2.6.4.1 Application Aspect

The devops user, once successfully authenticated, will be allowed to manipulate almost all aspects apart from the organisation model. The first action that he/she has to perform is to create a new application. Please note that such an application is considered to be mapped to a whole CAMEL model which includes one (sub-)model per each kind (e.g., one deployment and requirement sub-model). The devops would then have to select the "New Application" option in the "Application" menu.

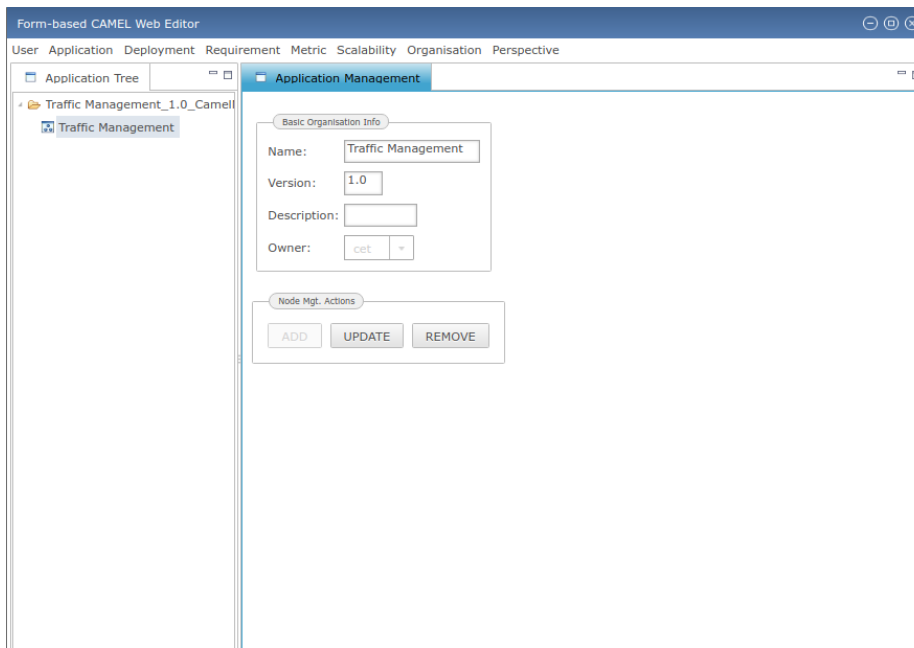
Upon this selection, the two main UI parts of the editor will be initially populated, as shown in Figure 12. The tree part will contain one node temporarily named as "NEW". On the other hand, the form-based part will depict a form that includes text fields, which can be filled in by the user,

related mostly to the actual permanent name of the application to be created and its version. Once the completion of these fields is over and the user presses the “Add” button, the application is created as a child node of the current one in the tree, while at the form-based UI part we have the disabling of the “Add” button and the enablement of the other two. This is depicted in Figure 13. Please note that the application creation also leads to the change of the temporary name of the CAMEL model to be permanently constructed by considering the generated application’s name and version.



The screenshot shows the 'Form-based CAMEL Web Editor' interface. The top navigation bar includes tabs for User, Application, Deployment, Requirement, Metric, Scalability, Organisation, and Perspective. The left sidebar shows the 'Application Tree' with a single node labeled 'NEW (CAMEL MODEL)'. The main panel is titled 'Application Management' and contains two sections: 'Basic Organisation Info' and 'Node Mgt. Actions'. The 'Basic Organisation Info' section has four input fields: 'Name' (empty), 'Version' (empty), 'Description' (empty), and 'Owner' (set to 'cet' with a dropdown arrow). The 'Node Mgt. Actions' section contains three buttons: 'ADD', 'UPDATE', and 'REMOVE'. The 'ADD' button is highlighted with a blue border.

Figure 12: New application with unfilled information



The screenshot shows the 'Form-based CAMEL Web Editor' interface. The top navigation bar is the same. The left sidebar shows the 'Application Tree' with a node labeled 'Traffic Management\_1.0\_Camell' and a sub-node labeled 'Traffic Management'. The main panel is titled 'Application Management' and contains two sections: 'Basic Organisation Info' and 'Node Mgt. Actions'. The 'Basic Organisation Info' section has four input fields: 'Name' (filled with 'Traffic Management'), 'Version' (filled with '1.0'), 'Description' (empty), and 'Owner' (set to 'cet' with a dropdown arrow). The 'Node Mgt. Actions' section contains three buttons: 'ADD', 'UPDATE', and 'REMOVE'. The 'UPDATE' button is highlighted with a blue border.

Figure 13: New application with completed information

#### 2.6.4.2 Deployment Aspect

The devops now has to create the deployment model of his/her application. In this respect, he/she can select the “New Deployment Model” option in the “Deployment” menu. A pop-up window will then be launched which will enable the user to supply the name of the deployment model to be generated (see Figure 14).

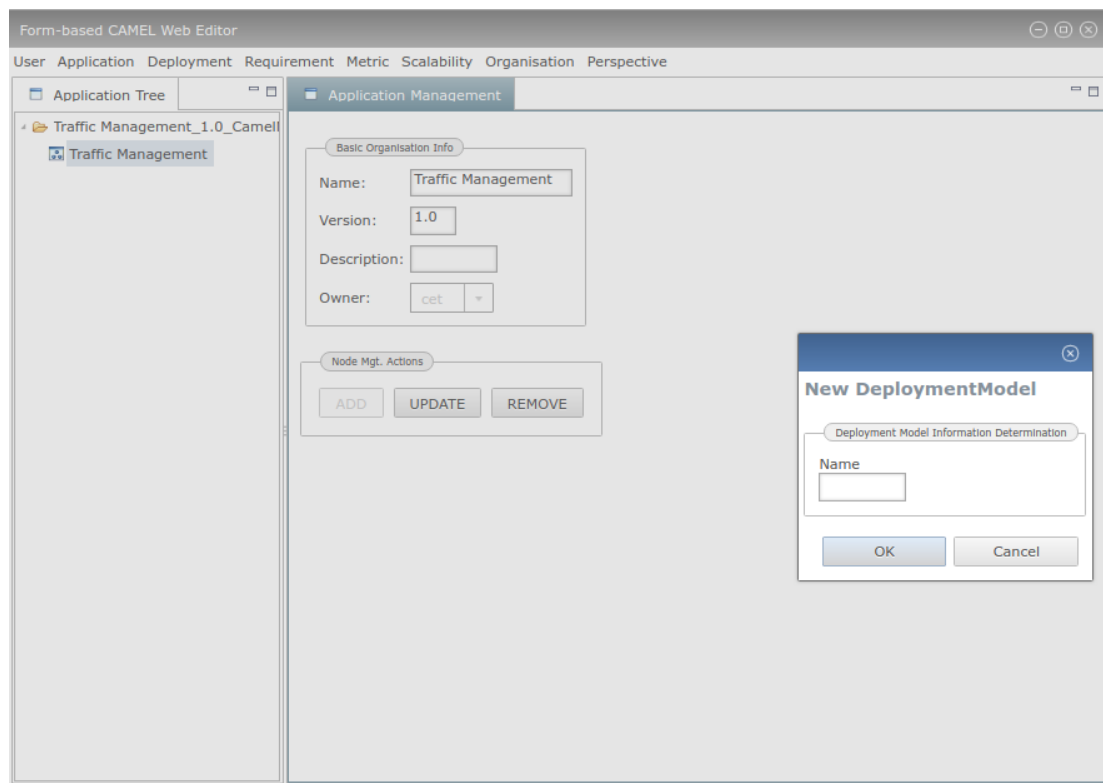


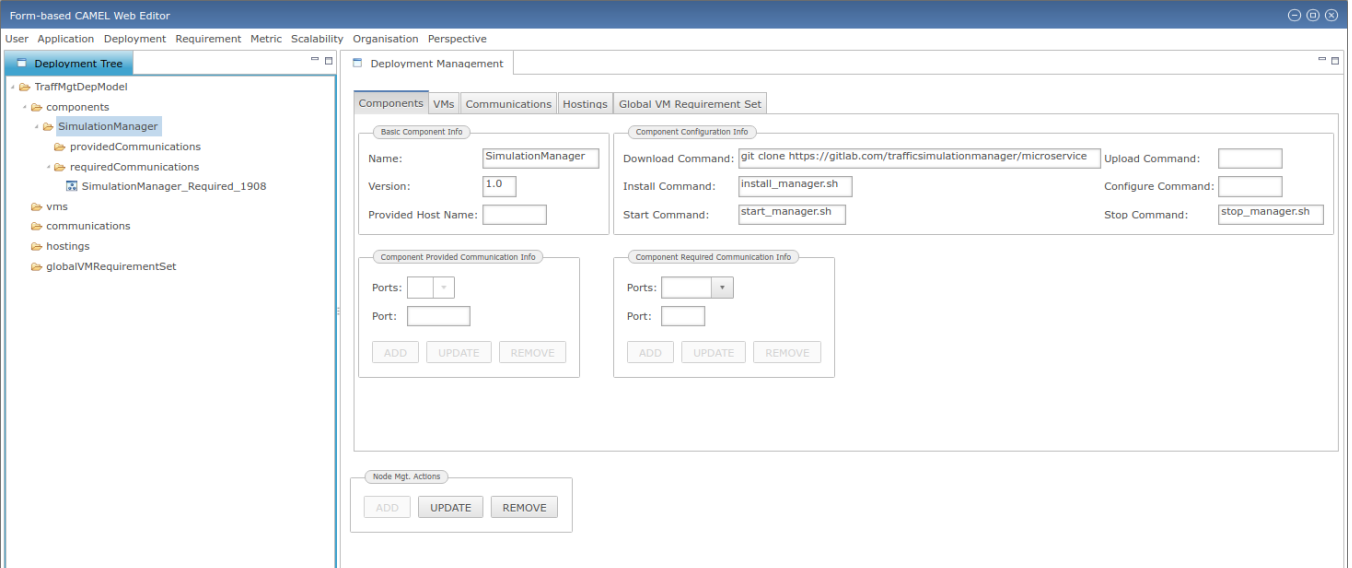
Figure 14: The new deployment model pop up window

Once this is successful, the two main UI parts of the editor are initiated. This then enables the devops user to start the fill-in of the right information. The devops user should first start with the creation of the internal application components. Two main components need to be generated: *SimulationManager* and *SimulationWorker*. These two components need to also communicate with each other such that the first exposes a required communication port and the second a provided communication port. As each application component is always hosted by a specific VM, the required hosting port of both components does not need to be provided. However, this does not hold for their configuration, which needs to be supplied.

Due to space economy reasons, we focus now on what the devops user will perform to specify all the needed information for the *SimulationManager* component. First, he/she would select the “components” container in the tree part of the UI. Then, the devops user would need to supply the name and the configuration of the component. The latter is within an UI element group devoted

to the supply of the different component lifecycle commands that should be utilised for properly configuring an application component. The devops would also need to add the port in the provided communication list in the form-based UI part. Once this is done, the user can press the “Add” button at the bottom part of the form-based UI part. This would then lead to the creation of the *SimulationManager* component as a child of the “components” container in the tree UI part (see Figure 15).

Once both the *SimulationManager* and *SimulationWorker* components are created, the devops user is required to specify the needed communication between the two components. He/she should then select the “communications” container node in the tree UI part and then supply the right information at the form-based UI part: (a) the name of the communication; (b) the two components to be connected by selecting the right items from the respective drop-down lists; and (c) the provided and required communication ports of these two components from two corresponding drop-down lists. We should note here that selecting one component is not enough as we need to know its exact port that is to be used in the respective communication. However, we need to highlight that once a component is selected from the drop-down list, only then will its ports be shown in another drop-down list so that they can be selected. Once this is done, the user can then press the “Add” button and the respective communication node will be created in the tree UI part (see Figure 16).



The screenshot displays the 'Form-based CAMEL Web Editor' interface. On the left, a 'Deployment Tree' sidebar shows a hierarchy: 'TrafMgtDepModel' > 'components' > 'SimulationManager' (highlighted). Below 'SimulationManager' are 'providedCommunications' and 'requiredCommunications'. A specific entry 'SimulationManager\_Required\_1908' is visible under 'requiredCommunications'. The main area, titled 'Deployment Management', contains several tabs: 'Components', 'VMs', 'Communications', 'Hostings', and 'Global VM Requirement Set'. The 'Components' tab is active, showing a form for 'SimulationManager'. The form includes fields for 'Name' (SimulationManager), 'Version' (1.0), and 'Provided Host Name'. It also has sections for 'Component Configuration Info' with fields for 'Download Command' (git clone https://gitlab.com/trafficsimulationmanager/microservice), 'Install Command' (install\_manager.sh), 'Start Command' (start\_manager.sh), 'Upload Command', 'Configure Command', and 'Stop Command' (stop\_manager.sh). Below these are sections for 'Component Provided Communication Info' and 'Component Required Communication Info', each with 'Ports' and 'Port' dropdowns and 'ADD', 'UPDATE', and 'REMOVE' buttons. At the bottom, a 'Node Mgt. Actions' section also has 'ADD', 'UPDATE', and 'REMOVE' buttons.

Figure 15: Addition of *SimulationManager* component

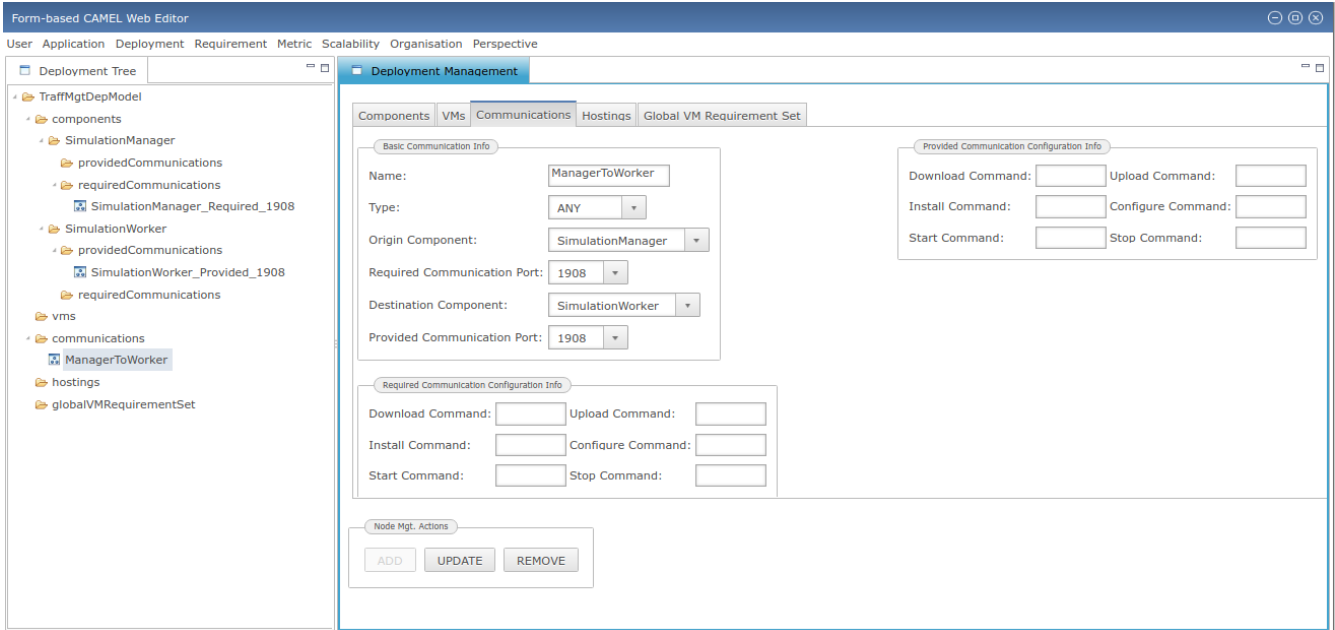


Figure 16: The addition of the communication node

Each application component needs to be deployed on a different VM. The *SimulationManager* needs to be deployed on the *SimulationManagerVM*, while the *SimulationWorker* needs to be deployed on the *SimulationWorkerVM*. Both VMs have some common requirements, but also different ones, thus they will be associated with different VM requirement sets. However, these sets can be completed only when the requirement model of the current application is specified. In this sense, the only thing that the devops user would need to perform now is to specify the VMs as well as their respective hostings, i.e., their connection with the internal application components that they will host.

For space economy reasons, Figure 17 depicts what the devops user will specify for the *SimulationManagerVM*. As it can be seen, the hosting port of that VM does not need to be specified as it is internally managed by the editor. Figure 18 depicts the hosting to be specified for connecting this VM with the *SimulationManager* component. As it can be observed, the devops user will need to specify in this case the name of the hosting as well as the two components that are to be connected together in it.

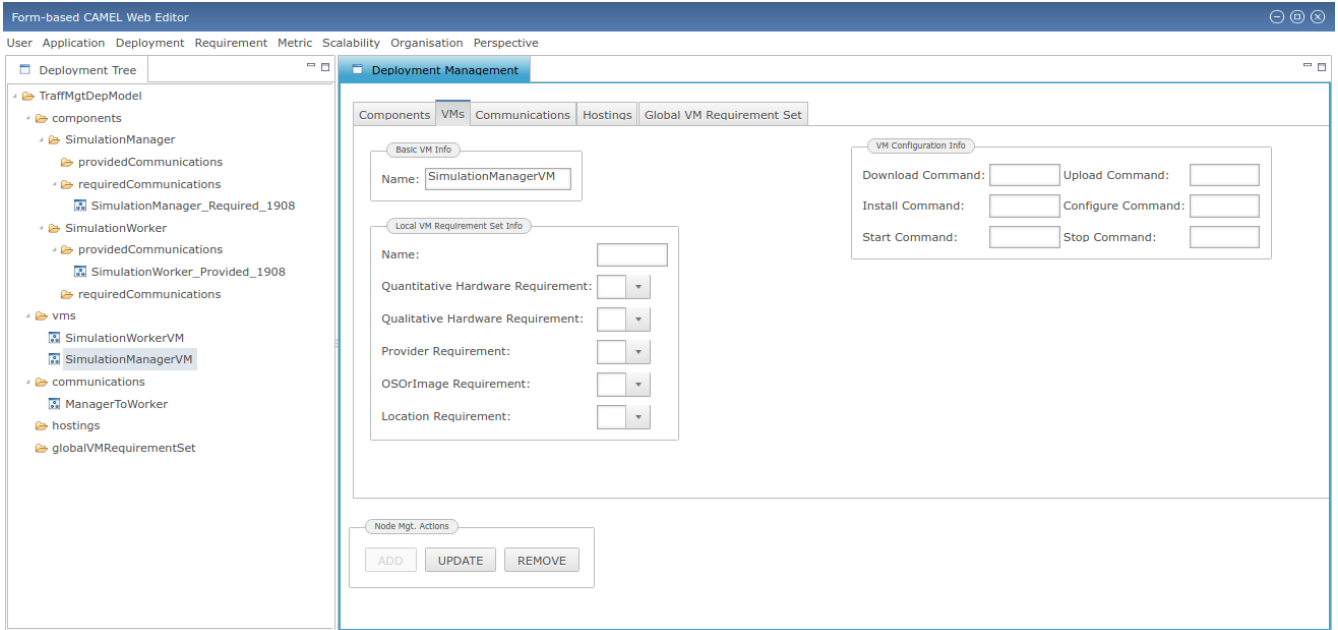


Figure 17: The creation of the *SimulationManagerVM* virtual machine

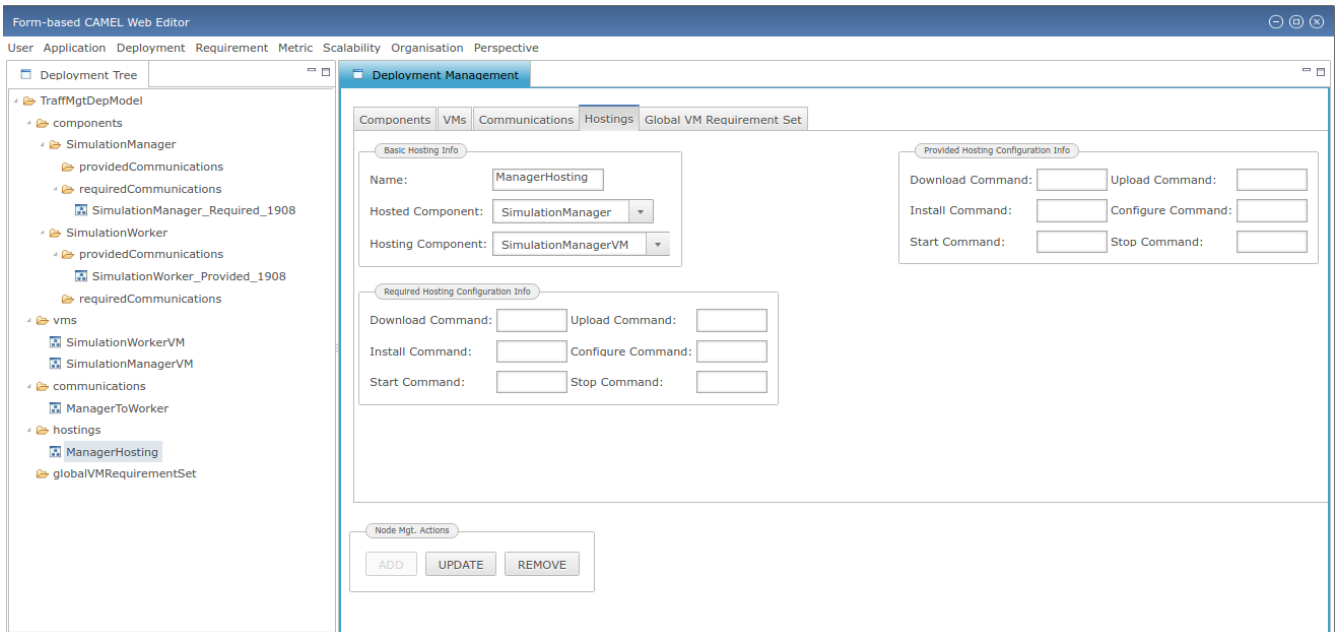


Figure 18: The creation of the *SimulationManager* Hosting

### 2.5.4.3 Requirement Aspect

Once the devops user finishes the first round of deployment model specification, he/she can move on to the requirement aspect. In this case, as there is no existing requirement model yet, he/she will need to specify its name by selecting the *Create New Requirement Model* option from the *Requirement* menu (see Figure 19). Once he/she finishes with specifying the name and presses the *OK* button, the model will be created and the respective two main UI parts of the editor will be launched as initially empty.



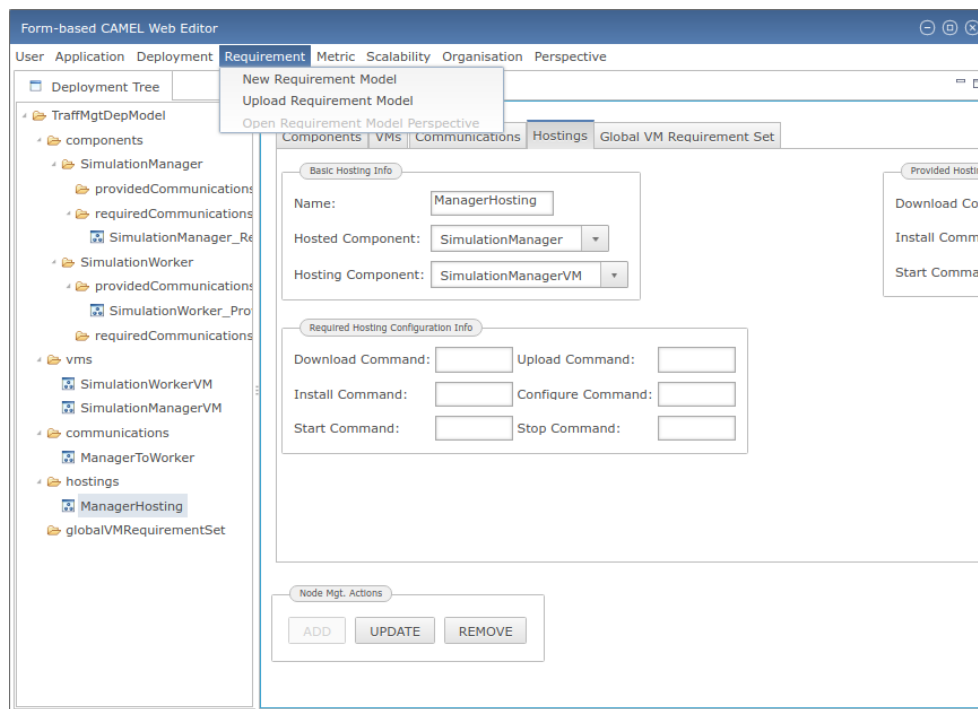


Figure 19: The enabled content of the Requirement menu

The devops user will first concentrate on specifying requirements for the respective VMs that have been already modelled. First, he/she will specify an OS requirement. He/she then needs to select the OS tab and provide three pieces of information: (a) the name of the requirement; (b) the actual OS by selecting it from a drop-down list; and (c) whether this OS should be 64-bit or not. This information, as completed by the devops user, is depicted in Figure 20.

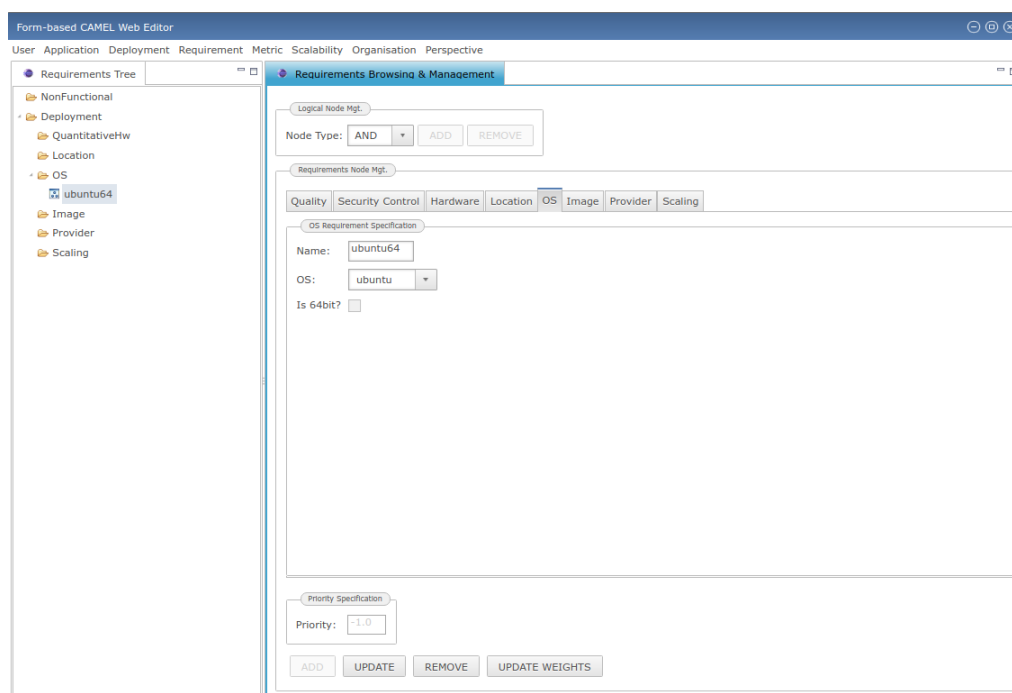
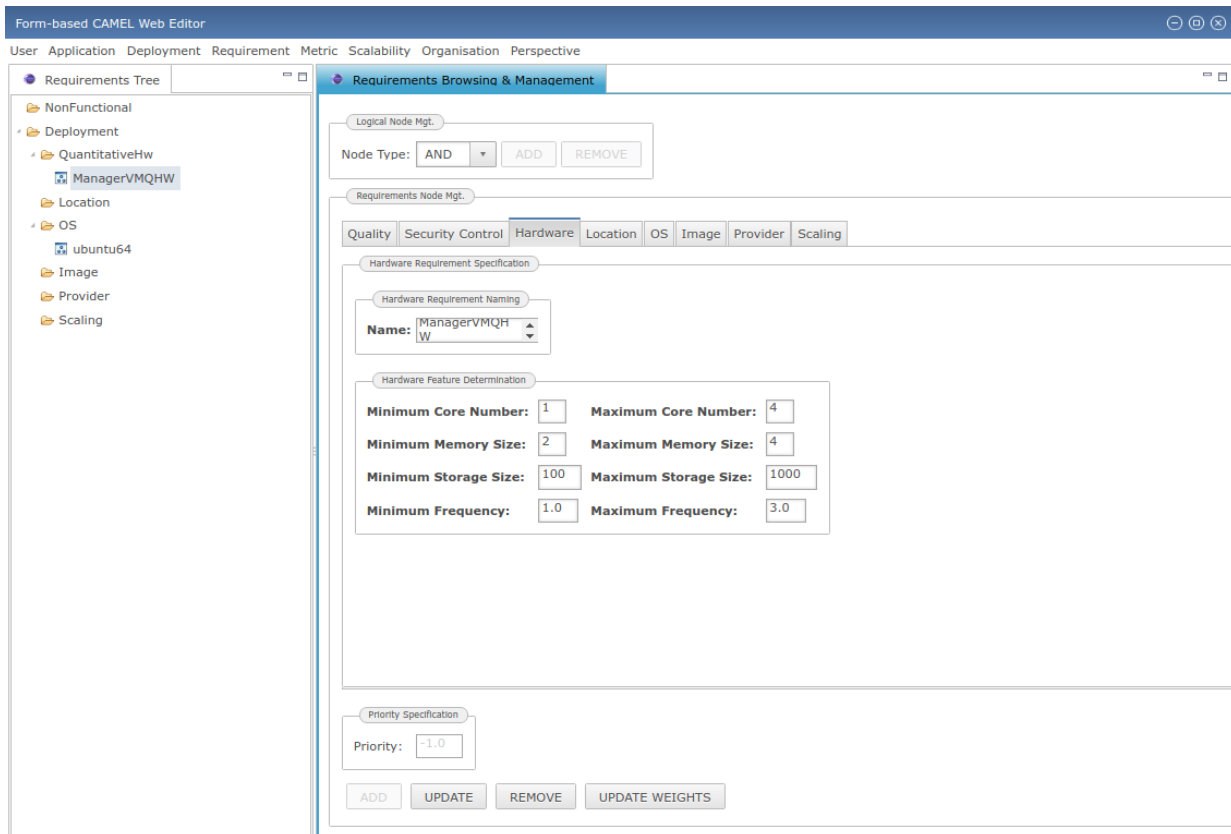


Figure 20: The creation of the ubuntu OS requirement

Next, the devops user will specify the quantitative hardware requirements for the two VMs. For economy of space reasons, we just show Figure 21, which depicts the information to be specified for the *SimulationManagerVM* after the devops user selects the Hardware tab.



The screenshot shows the 'Form-based CAMEL Web Editor' interface. On the left is a 'Requirements Tree' with a hierarchy: NonFunctional > Deployment > QuantitativeHw > ManagerVMQHW. The main area is titled 'Requirements Browsing & Management'. It has tabs for Logical Node Mgt., Requirements Node Mgt., and Hardware Requirement Specification. The 'Hardware' tab is active. Under 'Hardware Requirement Specification', there is a 'Hardware Requirement Naming' section with a 'Name' field set to 'ManagerVMQHW'. Below that is a 'Hardware Feature Determination' section with a grid of input fields for Minimum and Maximum values for Core Number, Memory Size, Storage Size, and Frequency. At the bottom, there is a 'Priority Specification' section with a 'Priority' field set to '-1.0'. Buttons for ADD, UPDATE, REMOVE, and UPDATE WEIGHTS are at the bottom right.

Figure 21: The quantitative hardware requirements specified for the VM of the *SimulationManager*

Now, it could be argued that the user has finished with the modelling of the requirements, as the SLO that he/she needs to specify would require the specification of a new metric, which can only be performed via the metric perspective. Fortunately, the editor can utilise already defined metrics that are incorporated in the (basic) metric model initially imported in the (CDO) *Model Repository* (by the *Importer*). In this sense, the devops user can immediately move to specifying the SLO with the sole exception that details concerning the context of the respective metric will still need to be specified via accessing the metric perspective.

To specify this SLO, the devops user can select the *NonFunctional* node in the tree-based UI part or just the Quality tab. Figure 22 depicts the information that he/she has to enter. In particular, the user needs to define the actual metric to be used, the comparison operator (LESS\_THAN) and the respective threshold (5) for this SLO. He/she also needs to explicate that the metric condition concerns the *SimulationWorker* component and that a new metric context needs to be specified (and not an existing one re-used). Once the "Add" button is pressed, we can observe (see Figure 23) that the metric context has taken a different name, which reflects the fact that this context has been automatically produced by the system and not manually edited by the devops user.

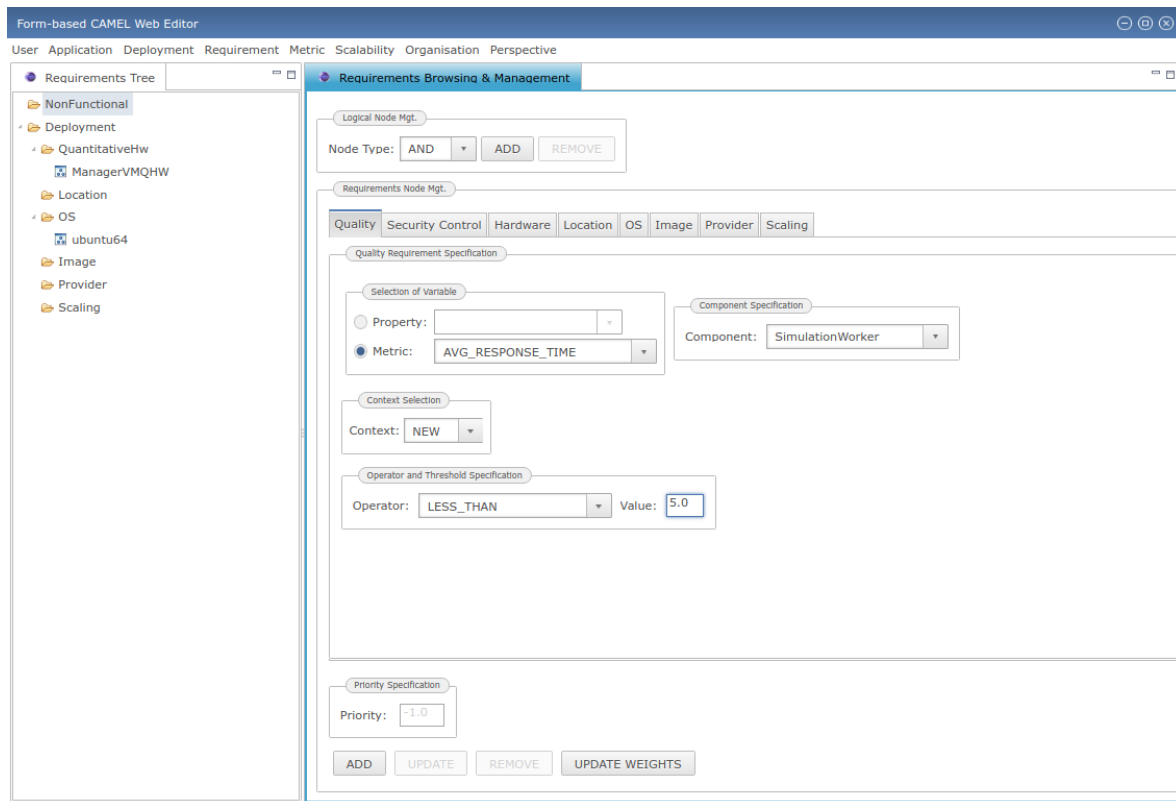


Figure 22: Filled in information before the SLO is added

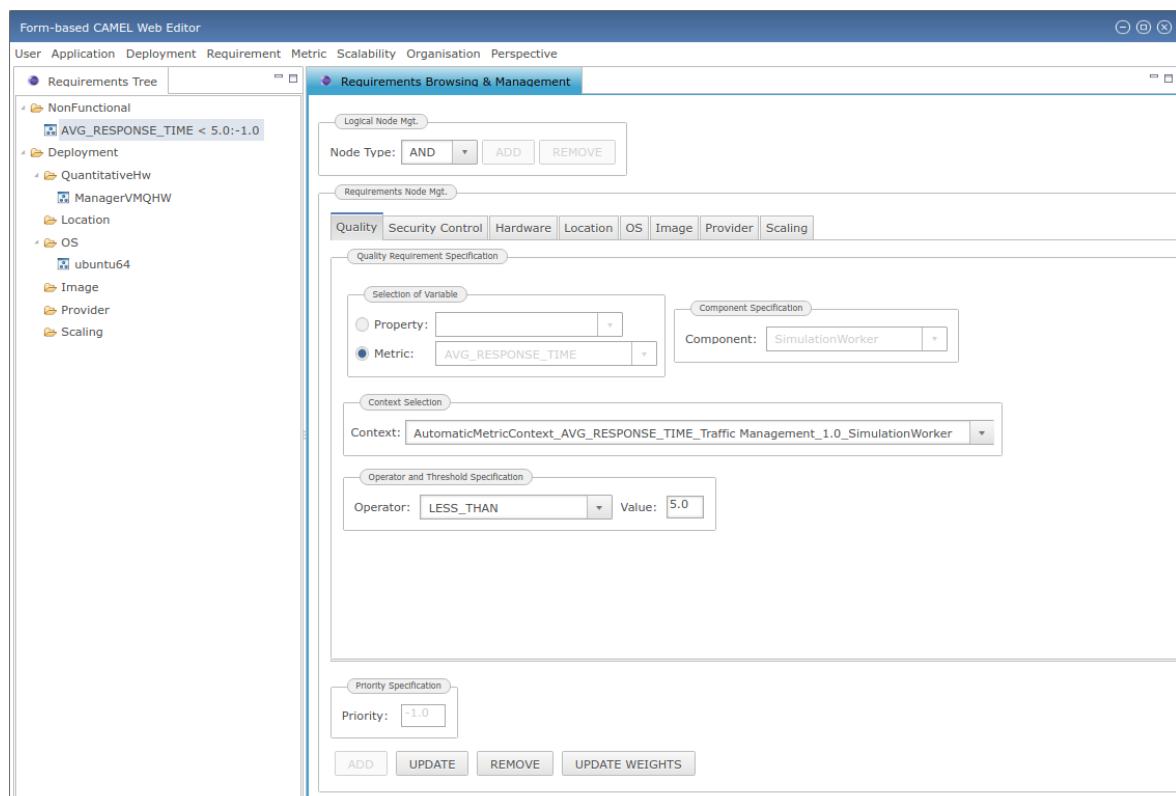


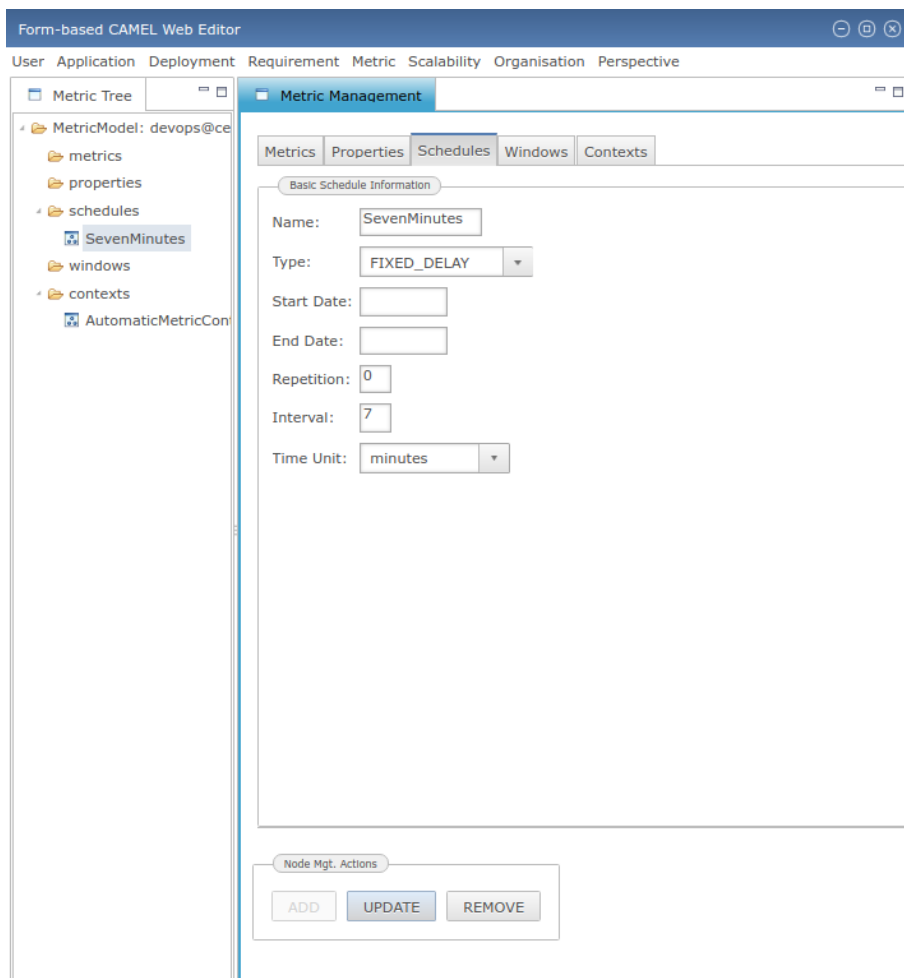
Figure 23: Visualisation of the added SLO

#### 2.6.4.4 Metric Aspect

As indicated previously, the metric context specified for the *average execution time* metric is not complete. In particular, we also need to specify additional details which map to the measurement schedule and window of the metric. These details require from the devops user to revert to the metric perspective.

Astonishingly, the devops user can already observe that a metric model has been already created, as well as an execution context included in it. This is due to the fact that an SLO has been already specified at the requirement model. This means that such an SLO should map to a metric condition, which should be associated in turn with a certain metric context. Both latter elements are automatically inserted (taking also automatically specific names) into the metric sub-model of the current CAMEL model, while this sub-model is newly created, if it does not already exist.

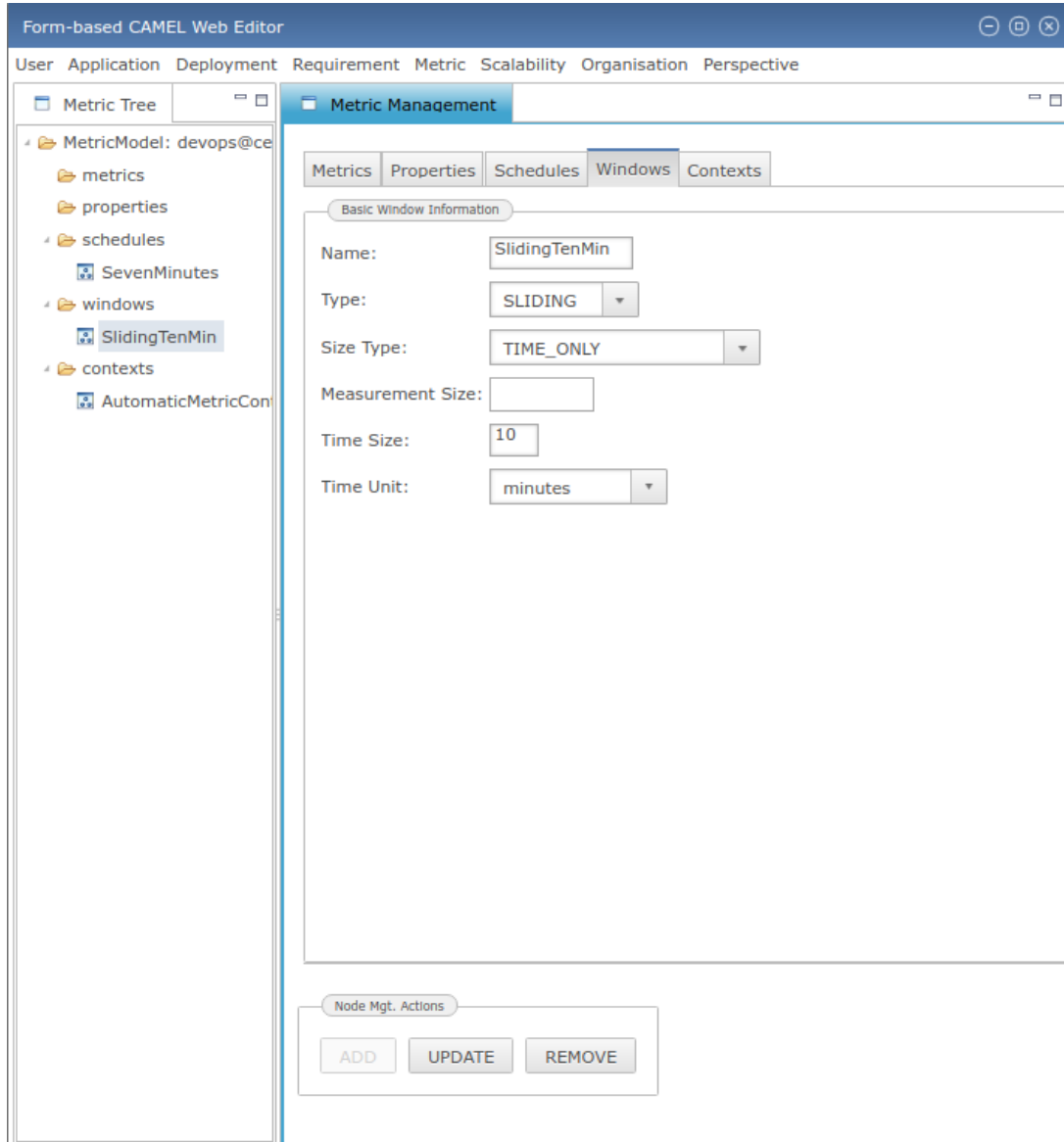
In order to complete the modelling of the composite metric context that is automatically created, the user needs to specify two information elements: a schedule and a window. Concerning the schedule, the devops supplies information about (a) the type of the schedule (FIXED\_DELAY); (b) its (repetitive) interval/time period (i.e., 7); and (c) its time-based unit (*minutes*). This is depicted in Figure 24.



The screenshot displays the 'Form-based CAMEL Web Editor' interface. On the left, a 'Metric Tree' shows a hierarchy: 'MetricModel: devops@ce' containing 'metrics', 'properties', 'schedules', 'windows', and 'contexts'. Under 'schedules', 'SevenMinutes' is selected. The main area is titled 'Metric Management' and has tabs for 'Metrics', 'Properties', 'Schedules', 'Windows', and 'Contexts'. The 'Schedules' tab is active, showing 'Basic Schedule Information' with the following fields: 'Name' (SevenMinutes), 'Type' (FIXED\_DELAY), 'Start Date' (empty), 'End Date' (empty), 'Repetition' (0), 'Interval' (7), and 'Time Unit' (minutes). At the bottom, there is a 'Node Mgt. Actions' section with 'ADD', 'UPDATE', and 'REMOVE' buttons.

Figure 24: The metric schedule added

For the (measurement) window, the devops should specify the following information pieces, which are depicted in Figure 25: (a) its type (SLIDING); (b) its size type (TIME\_ONLY); (c) the time period (10); and (d) the time-based unit (*minutes*). This information indicates that a time-based sliding window is to be created with a size of 10 minutes.



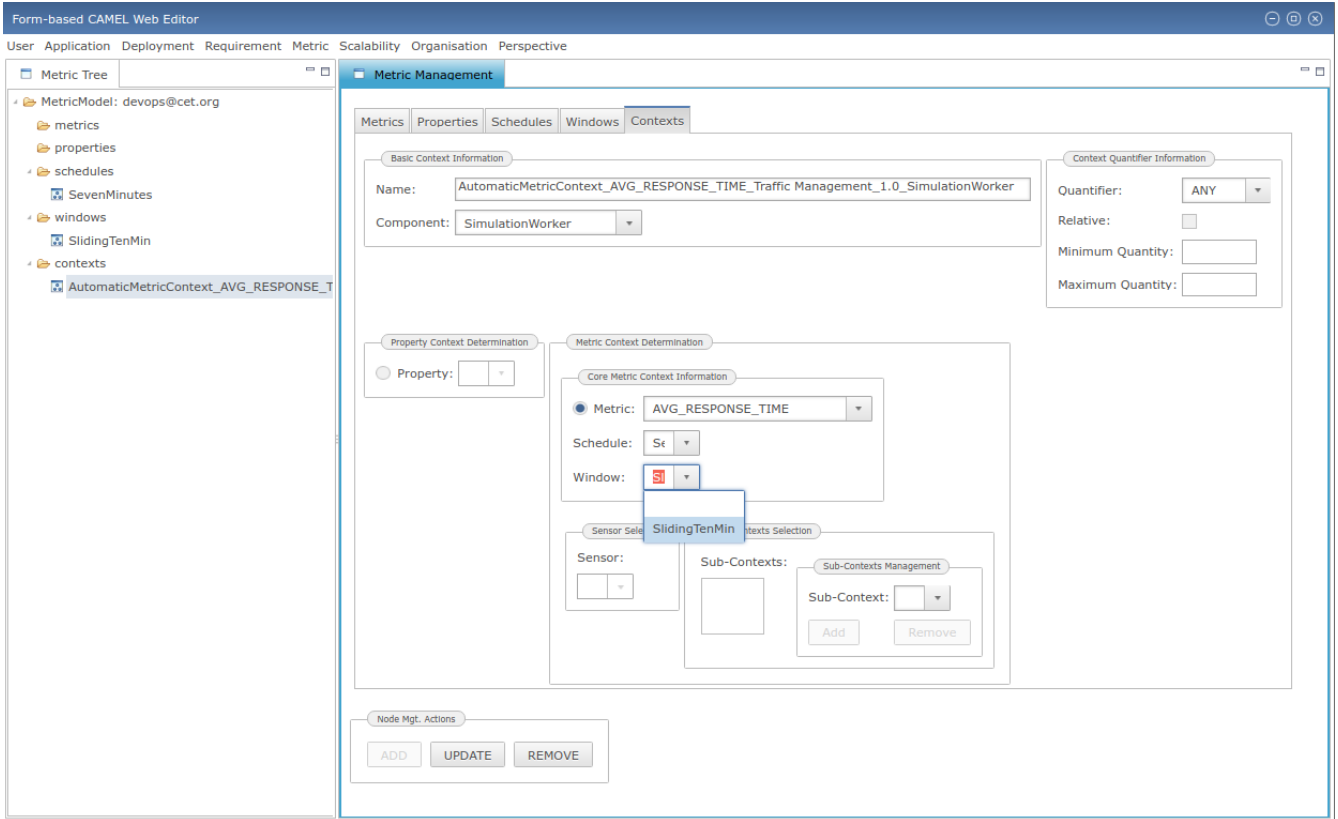
The screenshot shows the 'Form-based CAMEL Web Editor' interface. On the left is a 'Metric Tree' with a hierarchy: MetricModel: devops@ce, metrics, properties, schedules (containing SevenMinutes), windows (containing SlidingTenMin), and contexts (containing AutomaticMetricCon). The 'SlidingTenMin' window is selected. The main area is 'Metric Management' with tabs: Metrics, Properties, Schedules, Windows, and Contexts. The 'Windows' tab is active, showing 'Basic Window Information' for 'SlidingTenMin'. The configuration is as follows:

Field	Value
Name	SlidingTenMin
Type	SLIDING
Size Type	TIME_ONLY
Measurement Size	
Time Size	10
Time Unit	minutes

At the bottom, there are 'Node Mgt. Actions' buttons: ADD, UPDATE, and REMOVE.

Figure 25: The measurement window added

Once the devops creates both metric model elements, he/she can click on the metric context of focus and just select these two elements from the respective drop-down lists/combo. This is depicted in Figure 26. He/she can then press the *Update* button and the context will be updated. This then ends the editing of the CAMEL metric model.



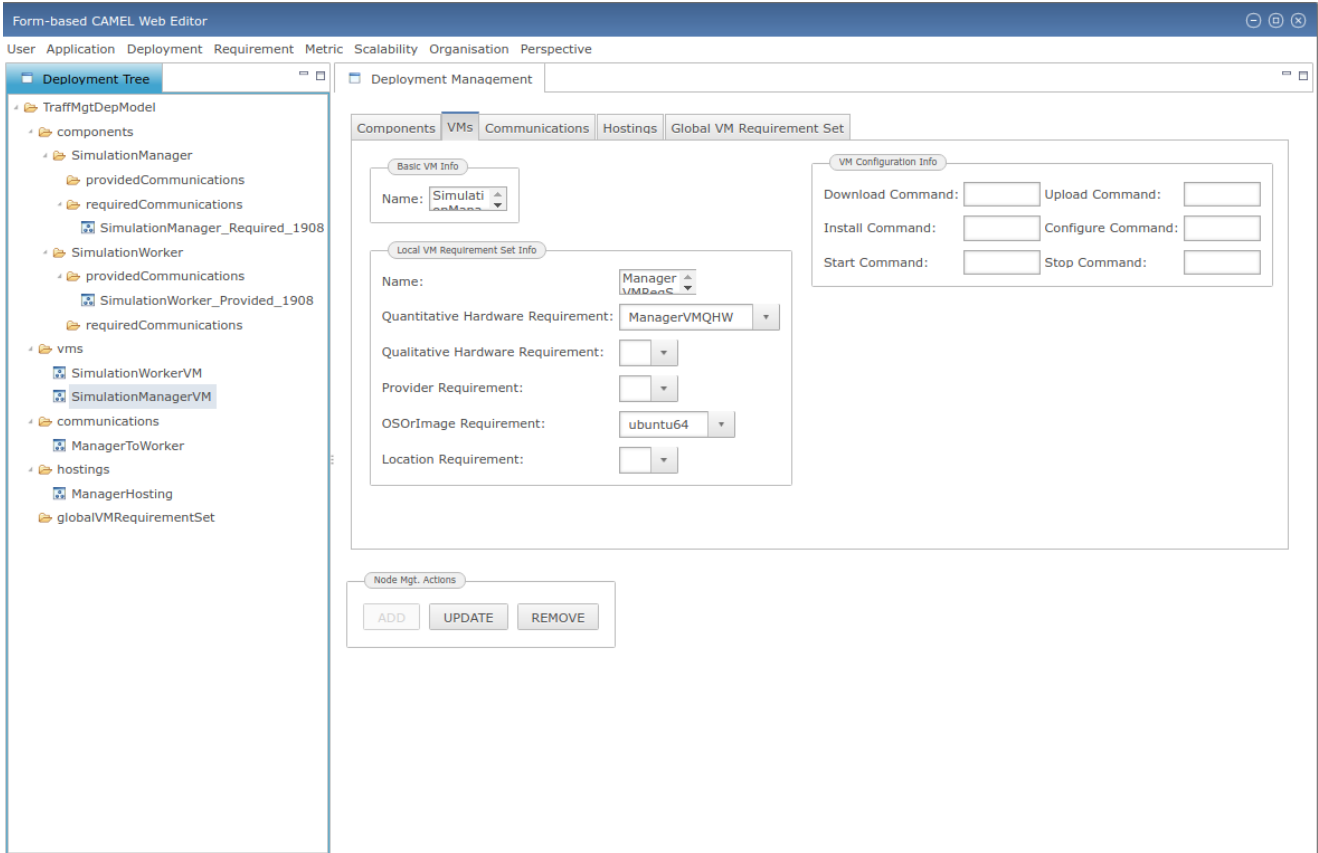
The screenshot displays the 'Form-based CAMEL Web Editor' interface. On the left is a 'Metric Tree' showing a hierarchy: MetricModel: devops@cet.org > metrics > properties > schedules > SevenMinutes > windows > SlidingTenMin > contexts > AutomaticMetricContext\_AVG\_RESPONSE\_T. The main workspace is titled 'Metric Management' and has tabs for Metrics, Properties, Schedules, Windows, and Contexts. The 'Contexts' tab is selected, showing a form for 'AutomaticMetricContext\_AVG\_RESPONSE\_TIME\_Traffic Management\_1.0\_SimulationWorker'. The form includes sections for 'Basic Context Information' (Name, Component), 'Context Quantifier Information' (Quantifier: ANY, Relative, Minimum/Maximum Quantity), 'Property Context Determination' (Property), 'Metric Context Determination' (Metric: AVG\_RESPONSE\_TIME, Schedule: Se, Window: SlidingTenMin), 'Sensor Selection', 'Sub-Contexts', and 'Sub-Contexts Management'. At the bottom is a 'Node Mgt. Actions' panel with ADD, UPDATE, and REMOVE buttons.

Figure 26: Update of the composite metric context

#### 2.6.4.5 Deployment Aspect Revisited

Please remember that the devops user has created two VMs to host the two components of the Traffic Management application, respectively, for which no requirements have yet been specified. In this respect, as the right set of requirements has been already created, through the use of the editor, the user can now switch to the requirement perspective. He/she can then click on each of the two VMs and map them to the requirements that they need to satisfy.

The latter is shown in Figure 27. For economy of space reasons, we focus only on the *SimulationManagerVM*. For this VM, the devops user will specify the name of the corresponding *VMRequirementSet* as well as select the right kinds of requirements from the respective drop-down lists that have been populated from the actual content of the requirement model. In this case, the user will select the *ubuntu* OS and the *SimulationManagerQHW* quantitative hardware requirements. He/she can then press the *Update* button to persist his/her changes. After updating both VMs, the deployment model is finalised. Thus, the only thing remaining to be modelled is the scalability rule for scaling the Traffic Management application on-demand.



The screenshot displays the 'Form-based CAMEL Web Editor' interface. On the left, a 'Deployment Tree' shows a hierarchy starting with 'TraffMgtDepModel', followed by 'components', 'SimulationManager', and 'SimulationWorker'. Under 'SimulationWorker', there are 'providedCommunications' and 'requiredCommunications' nodes. The 'requiredCommunications' node is expanded, showing 'SimulationManager\_Required\_1908' and 'SimulationWorker\_Provided\_1908'. Below these are 'vms', 'communications', 'hostings', and 'globalVMRequirementSet'. The 'vms' section is expanded, showing 'SimulationWorkerVM' and 'SimulationManagerVM'. The 'SimulationManagerVM' is selected. The main area on the right is titled 'Deployment Management' and contains several tabs: 'Components', 'VMs', 'Communications', 'Hostings', and 'Global VM Requirement Set'. The 'VMs' tab is active, showing 'Basic VM Info' and 'Local VM Requirement Set Info'. The 'Basic VM Info' section has a 'Name' field with a dropdown menu showing 'Simulati' and 'localhost'. The 'Local VM Requirement Set Info' section has fields for 'Name', 'Manager' (dropdown), 'Quantitative Hardware Requirement' (dropdown), 'Qualitative Hardware Requirement' (dropdown), 'Provider Requirement' (dropdown), 'OSOrImage Requirement' (dropdown), and 'Location Requirement' (dropdown). The 'Manager' dropdown is set to 'ManagerVMQHW'. The 'OSOrImage Requirement' dropdown is set to 'ubuntu64'. Below these fields are 'Node Mgt. Actions' buttons: 'ADD', 'UPDATE', and 'REMOVE'. To the right of the 'Local VM Requirement Set Info' section is a 'VM Configuration Info' section with fields for 'Download Command', 'Upload Command', 'Install Command', 'Configure Command', 'Start Command', and 'Stop Command'.

Figure 27: The requirement set associated to the VM of the *SimulationManager*

#### 2.6.4.6 Scalability Aspect

In order to specify the scalability rule for scaling the *SimulationWorker* component, the devops user will have to create a new scalability model from scratch. In this sense, he/she will first select the “New Scalability Model” option from the *Scalability* menu and then supply the name of the model to be created as well as press the “OK” button. This is depicted in Figure 28.

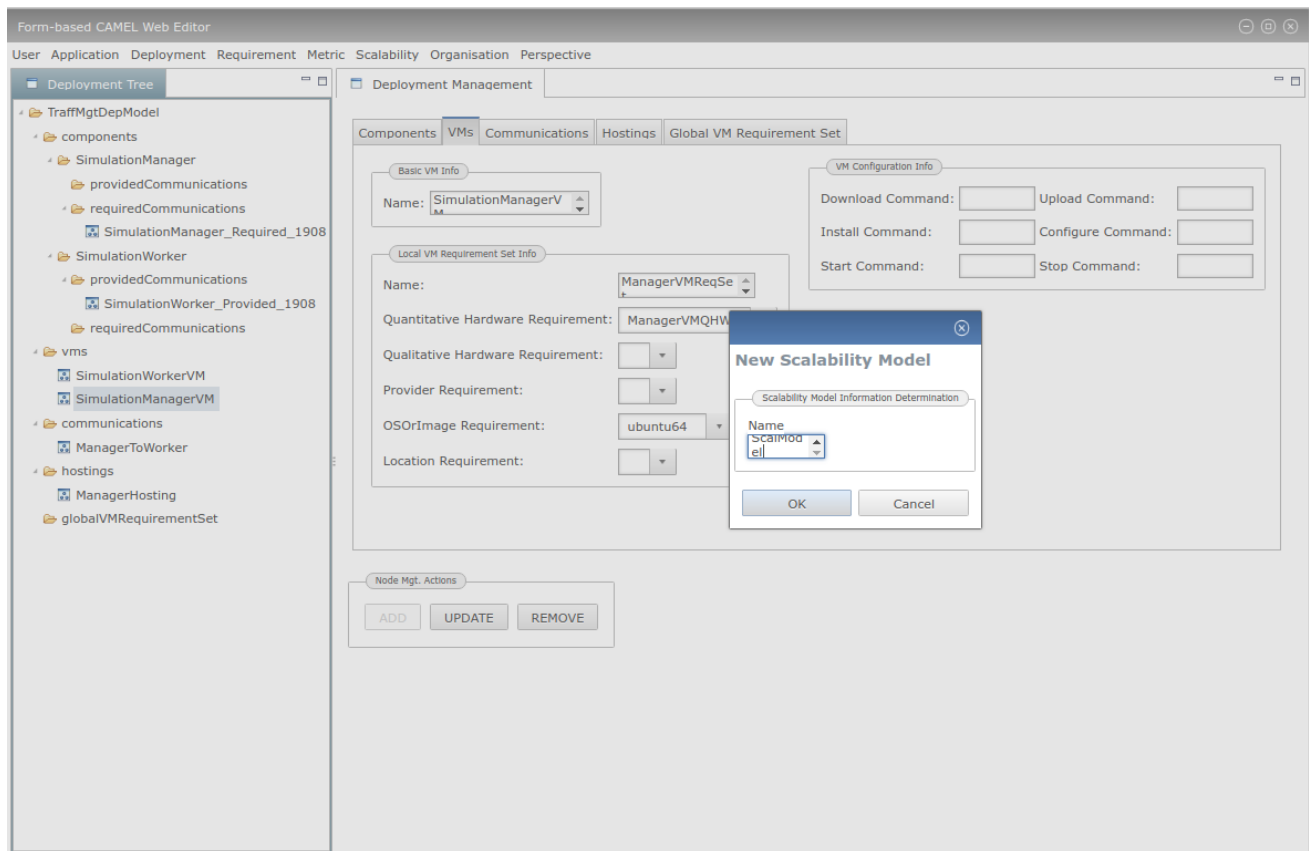
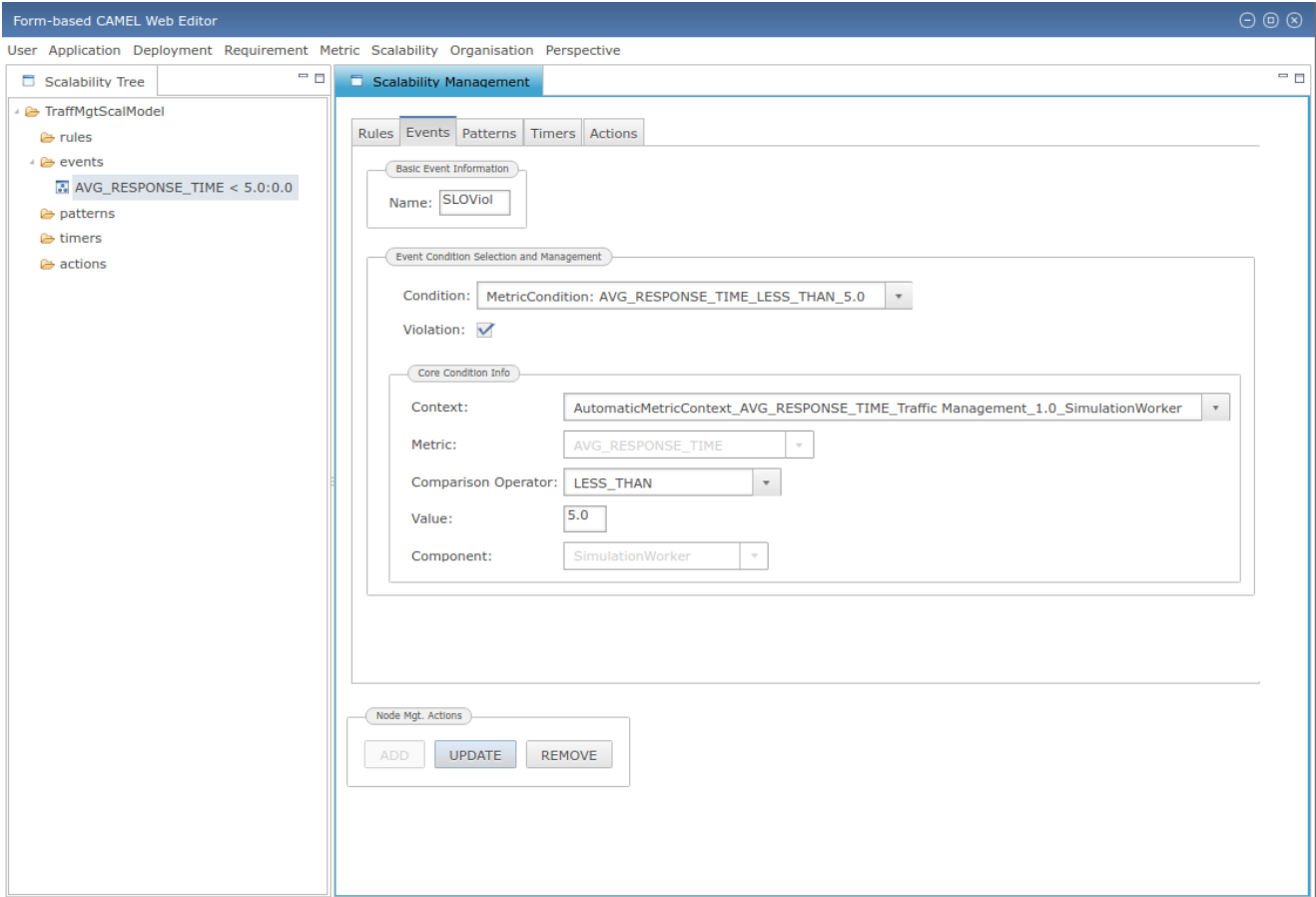


Figure 28: The specification of the name for the new scalability model

A scalability rule is a mapping of events to scaling actions. As such, before such a rule is created, there is a need to specify its two main associated information elements.

An event can be single, non-functional or a composition of other events. In the context of the current application, we just need a non-functional event to be modelled. Fortunately, due to the cross-correlation between the different kinds of CAMEL models, the creation of the SLO has led to the generation of a metric condition. The violation of such condition would then trigger the horizontal scaling action. As such, this violation is the event of focus here. In this case, as the metric condition has been already created, the user can just select it from the respective drop-down list and then indicate that its violation is of interest. This is depicted in Figure 29. By pressing the *Add* button, the respective event will be created in just a matter of 4 moves: the specification of the event name, the selection of the right metric condition, the enabling of the violation check box and the pressing of a button.





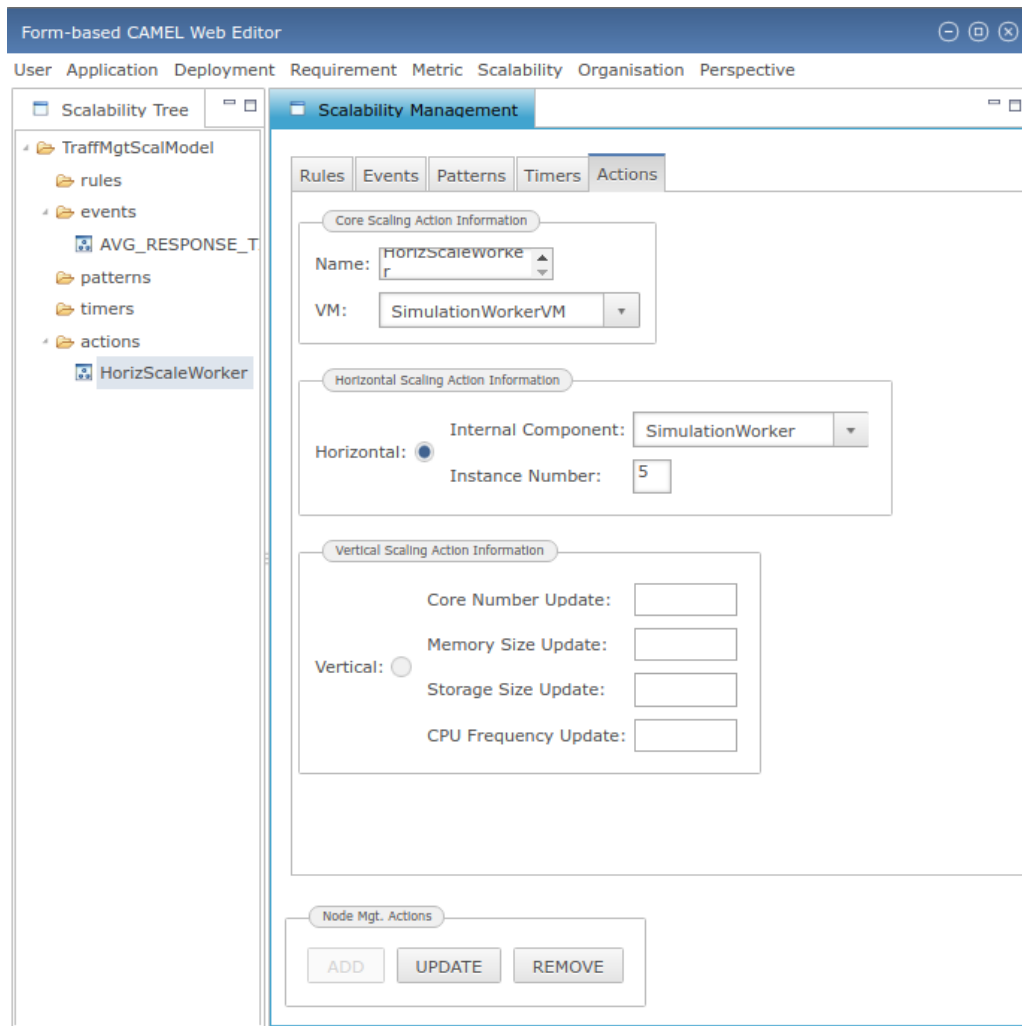
The screenshot shows the 'Form-based CAMEL Web Editor' interface. The top navigation bar includes 'User', 'Application', 'Deployment', 'Requirement', 'Metric', 'Scalability', 'Organisation', and 'Perspective'. The 'Scalability' tab is active, leading to the 'Scalability Management' section. On the left, a 'Scalability Tree' shows a hierarchy: 'TraffMgtScalModel' > 'rules' > 'events' > 'AVG\_RESPONSE\_TIME < 5.0:0.0'. The main area has tabs for 'Rules', 'Events', 'Patterns', 'Timers', and 'Actions', with 'Events' selected. The 'Event Condition Selection and Management' section contains the following fields:

- Basic Event Information:** Name: SLOViol
- Event Condition Selection and Management:**
  - Condition: MetricCondition: AVG\_RESPONSE\_TIME\_LESS\_THAN\_5.0
  - Violation: ☒
- Core Condition Info:**
  - Context: AutomaticMetricContext\_AVG\_RESPONSE\_TIME\_Traffic Management\_1.0\_SimulationWorker
  - Metric: AVG\_RESPONSE\_TIME
  - Comparison Operator: LESS\_THAN
  - Value: 5.0
  - Component: SimulationWorker

At the bottom, the 'Node Mgt. Actions' section includes 'ADD', 'UPDATE', and 'REMOVE' buttons.

Figure 29: The SLO violation non-functional event created

The required scaling action can be easily generated again in a limited number of moves (see Figure 30). In this case, the devops user needs to supply the name of the action; to select the respective VM (*SimulationWorkerVM*) to be scaled from a drop-down list; to select the radio button mapping to an horizontal scaling action; and then to specify both the internal application component to be scaled (*SimulationWorker*) from a drop-down list as well as the number of instances (5) to be created for that component. Once the *Add* button is pressed, the respective scaling action will be created and the sole task remaining to be performed would be to finally specify the corresponding scalability rule.



Form-based CAMEL Web Editor

User Application Deployment Requirement Metric Scalability Organisation Perspective

Scalability Tree

- TraffMgtScalModel
  - rules
  - events
    - AVG\_RESPONSE\_T
  - patterns
  - timers
  - actions
    - HorizScaleWorker

Scalability Management

Rules Events Patterns Timers Actions

Core Scaling Action Information

Name: HorizScaleWorker

VM: SimulationWorkerVM

Horizontal Scaling Action Information

Horizontal: ☒ Internal Component: SimulationWorker

Instance Number: 5

Vertical Scaling Action Information

Vertical: ☐ Core Number Update:

Memory Size Update:

Storage Size Update:

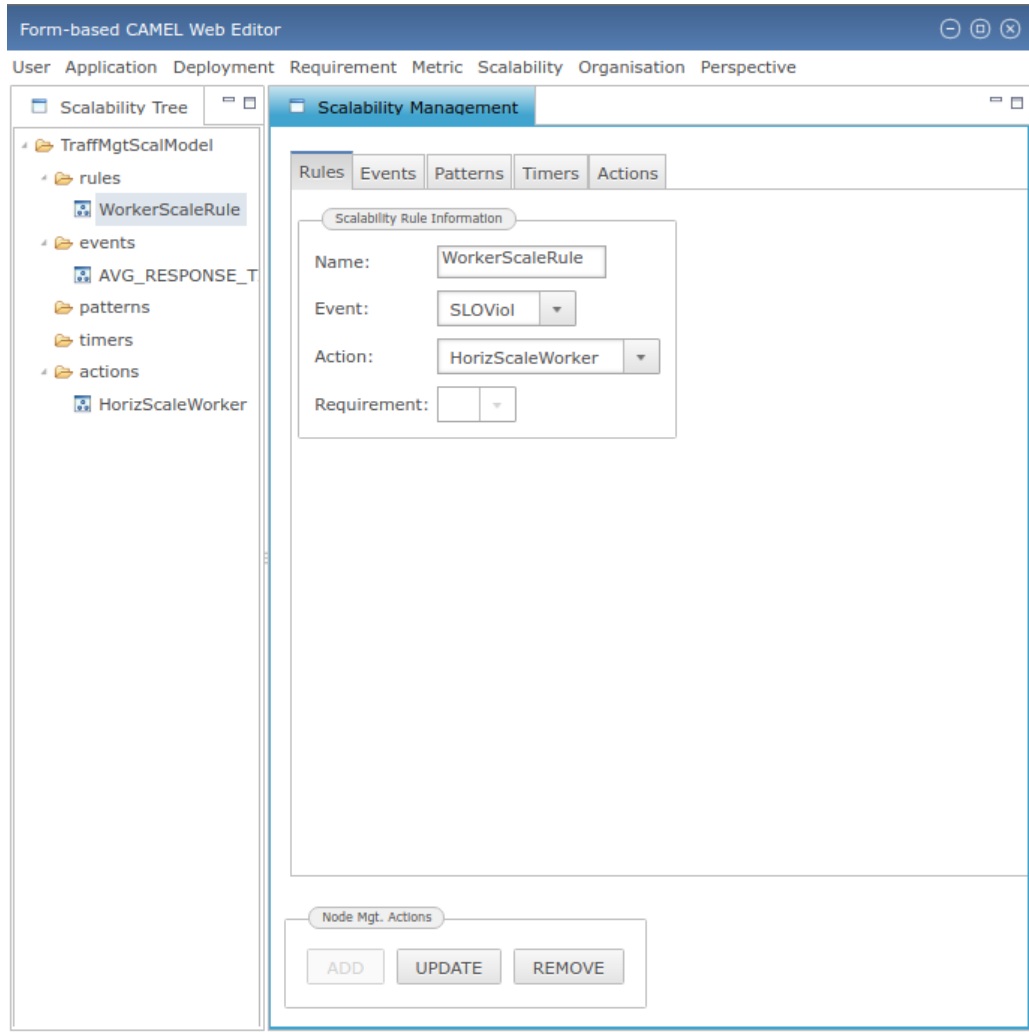
CPU Frequency Update:

Node Mgt. Actions

ADD UPDATE REMOVE

Figure 30: The horizontal scaling action created for the SimulationWorker component

For the latter rule, its specification is conducted in just a matter of 4 moves (see Figure 31): supplying the rule name, selecting the right event and scaling action from the drop-down lists, and finally pressing the *Add* button.



The screenshot shows the 'Form-based CAMEL Web Editor' window. The top menu bar includes 'User', 'Application', 'Deployment', 'Requirement', 'Metric', 'Scalability', 'Organisation', and 'Perspective'. The 'Scalability' perspective is active. On the left, the 'Scalability Tree' shows a hierarchy: 'TraffMgtScalModel' > 'rules' > 'WorkerScaleRule'. The main area is titled 'Scalability Management' and has tabs for 'Rules', 'Events', 'Patterns', 'Timers', and 'Actions'. The 'Rules' tab is selected, showing a 'Scalability Rule Information' form with the following fields: 'Name' (WorkerScaleRule), 'Event' (SLOViol), 'Action' (HorizScaleWorker), and 'Requirement' (empty). At the bottom, there is a 'Node Mgt. Actions' section with 'ADD', 'UPDATE', and 'REMOVE' buttons.

Figure 31: The scalability rule created for the *SimulationWorker* component

This concludes the specification of the scalability sub-model as well as the whole CAMEL model of the Traffic Management application. As we have seen, the user can very rapidly create a whole CAMEL model via a set of carefully conducted moves which should be performed in the right order and according to the right perspectives/aspects. The user does not need to know fluidly the CAMEL language to perform this. He/she just needs to know what is the correlation between the different kinds of CAMEL models in order to follow the right editing order with respect to these model kinds. This is one of the main advantages of exploiting this editor with respect to the other two ones, which do require a deeper knowledge and a more thorough understanding of CAMEL.

### 3 Future Work

The complete implementation of the web-based CAMEL editor is not yet done. Apart from the design requirements that are still not satisfied, additional features are planned to be incorporated in the editor while existing ones will be updated to conform to forthcoming CAMEL meta-model modifications. In this respect, the editor will evolve over time within the context of the Melodic project. In the following table, we present an overview of the features that will be soon realised, or those that will be updated along with their actual mapping to the respective CAMEL and Melodic platform release that they cover – thus presenting a timeline of the respective future implementation. All these features will be analysed in the next three sections of this chapter which have been split into three groups depending on whether they concern existing features that will be updated, specific editor design requirements which have not been met yet, or new editor features.

*Table 3: Timeline for the delivery of forthcoming updates and new features of the web-based CAMEL editor*

Feature	CAMEL Release	Melodic Release
R5 – Big Data Coverage	R2.0	R2.0
R9 – Application Deployment Launching	R2.0	R2.0
R10 – Cooperation with Metadata Schema related Editors	R2.0	R2.0
CAMEL Meta-Model Extensions/Modifications	R2.0-R2.5	R2.0-R3.0
Advanced Model Specification Guidance	R2.0	R2.0
Textual Camel Editing	R2.5	R3.0

As can be seen from the above table, there is a plan to produce two new CAMEL releases:

- R2.0: this release will conform to the R2.0 release of the Melodic platform and will cover all the big data related CAMEL extensions.
- R2.5: this release will conform to the final release of the Melodic platform and will incorporate additional extensions or modifications to the CAMEL meta-model, which will be possibly derived from both use case and technology partner feedback, or will be related to new features of the platform.

Please note that the notation of the releases conforms to the lifetime of the Melodic project, in accordance to the way the releases of the Melodic platform are identified. In this respect, R2.0 and R2.5 reflect releases that map to the 24 first months (2.0 years) and 30 first months (2.5 years) of the project. Please also mark that stopping at a CAMEL R2.5 release means that the final CAMEL

web-based editor version will be mainly delivered at the 2.5 year of the project, with a possible sole evolution delivered in the last month of the project, incorporating any kind of bug fixing and the possible integration (see analysis and justification below) with the textual editor.

### 3.1 CAMEL Meta-Model Extensions / Modifications

CAMEL will certainly evolve and this has been already indicated with the explication of its forthcoming releases. This means that its meta-model evolutions will lead to a required updating of the web-based editor (if it is decided to migrate to a given release, of course, which will be the case in the context of the Melodic project). However, such an updating can be quite substantial. In particular, it can move into two main directions:

- *updating of the UI*: some CAMEL elements have a direct reflection on the editor UI. So, if they are removed or modified, this also needs to be reflected in the editor implementation. Furthermore, new aspects, like the (big) data one, will need to be handled with new editor perspectives, while their information updating should be reflected on the rest of the perspectives, in case cross-references are concerned.
- *updating of the backend implementation*: the UI is backed up with a backend, which has as the main duty to reflect the user model manipulation actions online in the (CDO) *Model Repository*. This reflection comes mainly with some (model) manipulation and validation logic, as well as some querying abstraction. In both cases, the respective code would need to be updated as meta-model modifications will lead to changing both the main business/manipulation logic as well as the queries that are issued over the *Model Repository*.

With respect to the first direction, the extend of the reengineering work to be conducted concerning the second direction will be substantial. Fortunately, by also following the CAMEL release plan, we believe that the CAMEL meta-model modifications will be performed in a gradual manner, while some of the new/modified CAMEL features might be usable only in the last Melodic platform release. In this sense, we will have the opportunity to distribute the work to be conducted for accommodating this feature until month 30 of the Melodic project.

### 3.2 Coverage of Unsatisfied Requirements

Three main design requirements are not yet covered by the CAMEL web-based editor. Thus, their forthcoming coverage is analysed in the following three sub-sections.

### 3.2.1 Requirement R5 Coverage

Requirement R5 concerns the coverage of the big data aspect. This aspect has an impact also on the CAMEL meta-model and will lead to the modification of some aspects already covered by this meta-model. In this respect, the CAMEL editor will not be just extended, but also updated to cover these modifications. In particular, as the deployment aspect will be mainly impacted, this will then lead to modifying the deployment perspective in the CAMEL editor. The incorporation of a new aspect will also lead to the production of a new perspective, the data one. All these extensions/updates will require some extensive work to be conducted in the editor implementation, which will fortunately be realised in time, especially as the CAMEL meta-model R2.0 release is more or less finalised, with only some details missing to be handled.

### 3.2.2 Requirement R9 Coverage

Requirement R9 is the easiest to handle in the sense that interacting with the *Control Plane* of the platform will just require re-using (client) code that already exists. However, we should note that before deploying a cloud-based application, we need to ensure that the respective CAMEL model is complete and thus “deployable”. This then relates to the realisation of a *CAMEL model completeness* checking functionality, which is coupled to the satisfaction of this requirement. However, as the realisation of this functionality is also quite easy to handle and perform, the overall implementation requirements for this piece of work will be quite lightweight and less time-consuming. In the following, we provide some details on how the model completeness functionality could be realised.

A complete and thus deployable CAMEL model requires to have at least a complete deployment (sub-)model. Such a deployment model should contain internal application components that are not only hosted on a certain VM, but also associated with configuration scripts as well as with communication connections between each other. In this respect, validation of the model completeness would need to assess whether all this information is present. In addition, the editor, in the context of this functionality, could also issue some warning messages in case some optional information is missing, which could include:

- *communication connections between components*: it might be possible that a deployment model would be accepted without having such connections. While this is valid in terms of completeness, it might disclose a possibly erroneous situation where such connections were forgotten to be modelled by the user. In fact, logically speaking, an application which comprises multiple components most of the times, if not always, includes some communication connections between them.
- *quantitative hardware requirements*: without such requirements, any kind of VM might be selected which is not what the user would desire. Usually, application components come with unique hardware requirements that need to be respected. If not, then either the

components will not function properly or there will be a waste of resources, e.g., in the case that the components cannot have their performance improved above a certain resource limit.

- *OS requirements*: some times, components require to be hosted on certain environments that include a specific OS. As such, it can be probable that the non-existence of this kind of requirements can lead to misconfiguration of the respective components and then impact their actual execution.
- *optimisation requirements*: the non-existence of such requirements can lead to random selection of VMs which satisfy the local constraints supplied for each component (or VM type that hosts it). It is thus advocated that such requirements are posed in order to connect this selection with some overall optimisation objectives which are required by the respective user, possibly going from the infrastructure to the application level.

The warning message will indicate the actual cause of the warning as well as ways to remedy it. It could also signify some other missing modelling opportunities, mainly in the context of additional requirements that could be supplied by the user.

We should highlight that the validation of model completeness will be restricted over a structural or syntactic checking of the respective information that might be missing. It is rather hard to raise this at the semantic level as this would then require checking with respect to for instance whether the respective configuration commands given by the user were correct or that components of a certain type should be connected to each other.

In this respect, we expect that this kind of structural or syntactic checking, along with the CAMEL model issuing functionality, will be easy to implement and will be surely delivered at month 24 of the Melodic project.

### 3.2.3 Requirement R10 Coverage

Requirement R10 at first sight seems easy to implement. However, this is not actually the case. Only one part of its implementation can be considered as easy, i.e., the indirect interaction with the Weights Calculator. Indeed, fetching just some weights and incorporating them in a specific part of a CAMEL model is not very hard to realise. Furthermore, any update to the weights can be lazily reflected once the editor is launched and the user desires to modify the current set of weights incorporated in the respective CAMEL model. We highlight here the desire of the user to perform this reflection/updating due to the following reason: the user might have modified the weights in order to incorporate them in future evolutions of his/her application CAMEL model. In this respect, it is not correct to automatically update the current application CAMEL model with the new weights that have been produced.

On the other hand, the synchronisation with the instance of the meta-data schema in terms of the cooperation with the Metadata Schema Editor is of a different nature and cannot be just lazily

performed. Indeed, the modification of the metadata schema instance can immediately lead to the invalidation of some annotations already provided in a CAMEL model. Such modifications might require to take different actions in an automatic manner: (a) in the case of the name of an element in the metadata schema instance is modified, this will require updating the respective annotations in the CAMEL model; (b) in the case of a specific element from this instance is removed, then the respective annotations in the CAMEL model should be reset/removed.

Fortunately, we have incorporated a certain mechanism (see D2.2 [3] for more details) in the CAMEL meta-model in order to cover both kinds of modifications. In particular, the metadata schema has become part of the CAMEL meta-model. In this sense, the annotations in a CAMEL model directly refer to elements of the metadata schema instance and are thus not supplied in the form of a (indirect) String. In this way, both kinds of modifications are immediately reflected in a CAMEL model.

While this automatic synchronisation is well handled through the incorporated mechanism in the CAMEL meta-model, still the work to be performed in the CAMEL web-based editor is substantial as: (a) there is a need to supply annotations for any kind of CAMEL model element that is edited; and (b) in some cases, e.g., in the deployment aspect/perspective, there will be a need to introduce some dynamic CAMEL parts (in form of feature sub-models – see D2.2 [3]) that are produced through referring to the metadata schema instance elements (e.g., supply of a feature of a VM which is not covered in the CAMEL meta-model).

Overall, the coverage of the R10 requirement will map to a certain amount of work. While this work is considerable, we still believe that it can be delivered by the end of the 2nd year of the project.

## 3.3 New Features

### 3.3.1 Advanced Model Specification Guidance

This is the easiest feature to implement apart from the one related to the R9 Requirement. In particular, this feature relates to properly guiding the user/modeller in providing valid CAMEL models. The obligatory fields should be always marked within a UI. This enables the user to always supply information for them without risking in retrieving error messages when the respective information submission functionality is invoked. As such, we believe that this feature is essential to have a better user experience when using the web-based CAMEL editor. To realise it, we will follow some, if not all, of the directions below:

- label in a certain way all obligatory fields within a certain perspective. For instance, we could introduce a red asterisk for all these fields.
- label in a certain way all semi-obligatory fields/elements within a certain perspective. A semi-obligatory field is a field that can become obligatory in certain situations. For example, in case of a unary event pattern, if its type is *WHEN*, this means that a timer



should be given for it. As such, the timer combo in the respective UI should be marked down as semi-obligatory. In this way, by also covering the behaviour of the UI accordingly to enable or disable this combo depending on the type of the unary event pattern, the user is visually presented with hints that will enable him/her to conditionally provide this element. In particular, when the timer combo is enabled, as it is marked down as semi-obligatory, it will need to be supplied.

- supply links to each element or enable to hover over it in order to supply additional information for a certain element, including the fact of whether and when it is obligatory and what are its semantics.
- link the editor to its documentation (e.g., through a help menu) in order for the user to be able to inspect the ways a particular information aspect or piece should be supplied. In this respect, it is also good to produce a very nice and detailed documentation of the editor to be considered as a companion to its actual usage, further guiding the user in the specification of CAMEL models.

### 3.3.2 Textual CAMEL Model Editing

The web-based CAMEL editor has been developed for those users which are accustomed to use form-based UIs to perform their tasks and which do not require to learn the actual CAMEL syntax. In case particular novice users are accustomed towards using textual-based editors, there is a need to support this via the CAMEL editor by considering that the exploitation of the Melodic platform and its UI components should not be confined only in the context of the current use cases and the respective working modes of the corresponding organisations.

In order to cover this possible need, we foresee an incremental strategy to realise it. This strategy includes the following three steps:

- Acquire the ability to (semi-)automatically produce a standalone textual editor for CAMEL. Fortunately, based on the current experience from the PaaSage project, this is easy to perform by exploiting respective Eclipse technologies, and XText<sup>12</sup> in particular. In fact, XText gives the possibility to automatically produce a default textual syntax for a DSL, based on its abstract syntax (i.e., its Ecore model). This signifies that the only issue would be how to modify this default syntax to make it more precise and laconic in order to enable users more rapidly to specify textual CAMEL models. We should also highlight here that such a semi-automatic way of producing the textual syntax of a DSL is really convenient with respect to introducing modifications to it that are reflected according to corresponding DSL updates (on its abstract syntax). In other words, once modifications are performed on certain parts of a DSL, one just has to modify the respective parts of the textual syntax in order to reflect them (and there can be different ways to achieve that).

---

<sup>12</sup> <https://www.eclipse.org/Xtext>

- Transform the standalone textual editor to a web-based one: Fortunately, Eclipse again supplies the capability to produce this editor from the textual syntax of a DSL. However, based on our experience in PaaSage, this capability is not fully automatic, as it is advertised, and some effort is required to support this transformation/transition. However, as Eclipse technologies evolve, we might also foresee that this will be automatically performed in the near future. Nevertheless, we do have the know-how to support this transformation, even if certain manual interventions need to be performed.
- Integration of the textual editor to the web-based one: this is the outmost goal of this strategy. In this way, both the form- and textual-based editor will be supplied in a web-based manner. While this will enable CAMEL to cover different needs and different kinds of users, it is not so easy to perform based on certain restrictions which we outline below:
  - the textual editor works purely on models which are not stored in CDO. In other words, it functions in an offline manner. In order to make it function over CDO, there will be a need to extend it. This will certainly require performing an extensive amount of work, which might be overwhelming and might also be considered outside the scope of the Melodic project.
  - to remedy for the above, we could imagine the case that the CDO-based models that are manipulated by the web-based editor are transformed into a different form which is CDO-free. In this sense, the textual editor could still work on non-CDO models. The issues with this approach are that: (a) it is not always possible to cover the textual editing of a certain model if it cross-references information from other models. In other words, if there is a CAMEL model which cross-references other CAMEL models that have been already stored in CDO, this then means that we cannot transform it completely into a CDO-free model. So, the usage of the textual editor will be restricted in the context of CAMEL models which do not cross-reference other CDO models. Unless we find a certain way to remedy this; (b) there is a need to revert back the textual model into its CDO-based form. This could be tricky as there will be a need to recognise which elements have remained untouched and which ones have been modified. One remedy for this would be to just remove the previous CAMEL model and replace it with a new one. After all, the previous CAMEL model would still be free from cross-references, thus there will be no harm in deleting it from CDO and re-inserting it back.

As the strategy involves multiple steps with a possible increased complexity and effort, we have put the delivery of this feature to year 2.5 of the project, i.e., the final delivery date of the last version of the web-based editor. However, as this feature does not forbid editing CAMEL models via different ways, i.e., the web-based CAMEL form editor, we could still have the possibility to delay its delivery until the end of the Melodic project - which would be beneficial in order to also cover bug fixing and last minute changes. Overall, there will also be a need to make a decision about the resources to be devoted for achieving this integration. If the resources are not enough,

we could still live with the accomplishment of the first step of the above strategy. After all, what matters is to have a way to support the textual editing of CAMEL models. Whether this will be performed in an off-line or a web-based manner is not very critical. An off-line textual editing can still exploit the other forms of interfaces of the Melodic platform in order to initiate application deployments based on CAMEL models. So, we are actually safe even from this side.

Furthermore, we should not forget that there are import and export mechanisms in the web-based editor. This can enable different ways via which the editors can cooperate:

- a user starts with the textual editor and then imports the respective model into the web-based editor. This has the advantage that the model is stored in the CDO model repository and can then be exploited for triggering application deployments.
- while a CAMEL model is edited in the form-based editor, the user might feel more comfortable in specifying some details for it via the textual editor. He/she can then export the model in textual form, edit it via the textual editor and then re-import it back to the web-based editor.

Thus, based on the above analysis, a full integration between the editors is not actually needed, and should only be realised subject to resource availability and the overpassing of corresponding technical obstacles.

## 4 Conclusions

This deliverable focused on presenting the web & form-based CAMEL editor, which can be exploited for on-line editing of CAMEL models and their immediate issuing for launching the application deployment process. The presentation started by supplying the main requirements that drove the design of this editor. Then it continued with the presentation of the respective cases devoted to the usage of this editor, next the architecture and main features of the editor were analysed, and finally a detailed walkthrough of the editor with respect to the specification of the CAMEL model for a reduced form of a specific Melodic use case, i.e., the Traffic Management one, was elaborated.

The editor has been developed to fully comply with the current release of CAMEL, which is identical to the one that was finally produced for the PaaSage project. This release has been deemed the most appropriate for the delivery of the 1.5 release of the Melodic platform. However, moving on to support big data application deployment and adaptive provisioning, both the CAMEL and Melodic platform releases will evolve. In this respect, the editor will be also updated to keep up with the changes in the CAMEL language.

Furthermore, the editor will evolve according to four main directions: (1) metadata-schema-related editor cooperation: realisation of the ability to exploit the output from the metadata-schema-related editors for annotating CAMEL models, and completing the weights/priorities of

optimisation objectives in the CAMEL requirement sub-models; (2) application deployment launching (i.e., Requirement R9): realisation of CAMEL model completeness checking functionality and the consequent launching of an application deployment via this CAMEL model; (3) advanced model specification guidance: the UI of the editor will be updated to better guide the user in specifying valid CAMEL models; and (4) textual CAMEL modelling: to cater for particular kinds of users, that might be involved in organisations that intend to exploit the Melodic platform, which might be accustomed to the textual way of specifying models, it would be a nice and added-value feature to integrate the form-based with the textual editor. This would enable to switch between the different model specification modes on-demand and according to the current situation and user preferences. For instance, the form-based mode could be enabled for specifying most of the CAMEL model while some particular technical details, like the specification of metric formulas, could be performed in a textual manner.

The overall planning of how the web-based CAMEL editor will evolve has also been supplied by explicating which new features will be incorporated and which existing ones will be updated according to specific time points. We should highlight that the last direction can be considered as quite difficult to realise due to a plethora of technical issues. In this respect, we have designated that its possible realisation will be moved to the end of the project, provided that there will be enough resources for achieving it. Even if this is not accomplished, the textual CAMEL editor, working in an offline mode, will also be delivered. Furthermore, we have indicated different ways the two CAMEL editors can interoperate, even if exploited in an individual manner. As such, we signify that: (a) we will supply different kinds of editors to suit different user needs and preferences; and (b) such editors can not only interoperate, but also lead to circumstances where different users collaborate towards specifying the full CAMEL model of an application. In our opinion, this is a major offering to the respective community while it can also cater for achieving a better sustainability of CAMEL, essentially having an effect also after the end of the Melodic project.

## 5 References

- [1] Y. Verginadis, I. Patiniotakis, C. Chalaris and G. Mentzas, "D3.1 Metadata Schema Management", The Melodic H2020 Project Deliverable D3.1, 2018.
- [2] Y. Verginadis, W. Żołnierowicz, P. Skrzypek, D. Seybold, K. Kritikos, S. Mazumdar, A. Schwichtenberg, F. Zahid, J. Domaschka, G. Horn, E. G. Gran, D. Baur, H. Masata and P. Góra, "D2.1 System Specification Document", The Melodic H2020 Project Deliverable D2.1, 2017.
- [3] Y. Verginadis, G. Horn, K. Kyriakos, F. Zahid, D. Baur, P. Skrzypek, D. Seybold, M. Prusiński and S. Mazumdar, "D2.2 Architecture and Initial Feature Definitions", The Melodic H2020 Project Deliverable D2.2, 2018.