Melodic
Big data cloud

Title:

Metadata Schema Management

Abstract:

This document presents two design-time tools of the Melodic platform: the Metadata Schema editor used to create and manage the Metadata Schema, and the Weights Calculation tool that can be used to estimate and/or fine-tune the weights of polynomial utility functions used by Melodic solvers. Apart from tool descriptions, theoretical information needed to comprehend their functioning is also given. In addition, a short walkthrough (accompanied by screenshots) is provided for each tool, in order to help the reader getting an idea of how these tools look and operate.

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

Deliverable reference:
D3.1

Date:
26 Jan 2018

Responsible partner:
ICCS

Editor(s):
Ioannis Patiniotakis

Author(s)
Yiannis Verginadis,
Ioannis Patiniotakis,
Christos Chalaris,
Gregoris Mentzas

Approved by:
Ernst Gunnar Gran

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/D3.1 Metadata Schema Management.pdf

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

Document	
Period Covered	M4-12
Deliverable No.	D3.1
Deliverable Title	Metadata Schema Management
Editor(s)	Ioannis Patiniotakis
Author(s)	Yiannis Verginadis, Ioannis Patiniotakis, Christos Chalaris, Gregoris Mentzas
Reviewer(s)	Sebastian Schork, Feroz Zahid
Work Package No.	3
Work Package Title	Upper ware
Lead Beneficiary	ICCS
Distribution	PU
Version	5.0
Draft/Final	Final
Total No. of Pages	33

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

Table of Contents

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

List of Figures

List of Tables

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

1 Introduction

The purpose of this deliverable is to present two design-time tools developed in the Melodic project, related to the Melodic Metadata Schema. The first tool is the Metadata Schema editor, which can be used to create and maintain the metadata schema. The second tool is used during application modeling, in order to help application developers to calculate and fine-tune the weights (priorities) of the polynomial utility functions. This information will be valuable for formulating the utility function to be used by the Melodic Solvers and Utility Generator mechanisms, as described in the Melodic project deliverable D2.2 *Architecture and Initial Feature Definitions* [1], in order to generate deployment plans meeting the constraints of a CAMEL model. This tool also takes advantage of the Metadata Schema.

1.1 Scope of the Document

This document is intended for the general audience interested in learning about two design-time tools of the Melodic project used to manage the Metadata Schema and assist administrators to assign appropriate weights to those Metadata Schema elements used in a CAMEL model utility function. The work reported in this deliverable relies on concepts and ideas of the Melodic Metadata Schema introduced and detailed in the Melodic project deliverable D2.4 *Metadata Schema* [2].

1.2 Structure of the Document

The rest of this document is structured as follows. In Chapter 2, a brief recap of the Melodic Metadata Schema is given, followed by the technical description of the Metadata Schema editor. In the first part of Chapter 3, a weights calculation method based on pairwise comparisons is presented. In the second part of the chapter, the Weights Calculation tool is analysed. The document concludes in Chapter 4.

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

2 Metadata Schema Editor

In this chapter, the Melodic Metadata Schema editor is presented. It is a graphical web-based tool that has been developed in the context of the Melodic project, in order to enable the creation, modification and management of the Melodic Metadata Schema. We note that we have also developed an eclipse EMF-based version of this editor for cases where the potential Melodic adopter would prefer to use it. Before delving into the details of the Metadata Schema editor, it is necessary to briefly recap the purpose and nature of the Melodic Metadata Schema. For more information, the reader may also refer to deliverable D2.4 [2] .

2.1 Metadata Schema – Overview

In the following, the Melodic Metadata Schema and Metadata Schema Management process are briefly presented. The former is a scheme for organising various concepts and metadata information related to cloud and big-data applications into a vocabulary. The latter refers to the process of defining and maintaining the Metadata Schema for a specific application.

As stated in D2.4 [2] *"Melodic Metadata Schema is meant to provide a comprehensive, modular and extensible vocabulary for modelling cloud application aspects, including application placement, application security and big-data aspects."* It comprises three sub-models.

- Application Placement sub-model.
- Big-data sub-model.
- Security Context sub-model.

Each sub-model comprises a hierarchical (tree-like) structure of concepts, where the more generic concepts reside at the higher level of the sub-model, whereas the more specialised ones are placed at lower levels under the generic one. Concepts can be characterized by several properties while they can be instantiated through concept instances. For example, if we need to know the geographical location where processing of a certain big-data type will take place, then a relevant geographical processing location concept should be defined, along with its related instances (e.g., EU, GR, ASIA) and properties (e.g., latitude, longitude).

The three sub-models of the Metadata Schema are thoroughly detailed in the corresponding deliverable D2.4 [2] , while a bird's eye view of the schema can be found in Figure 1, where the most significant top-level concepts of each sub-model are depicted.

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

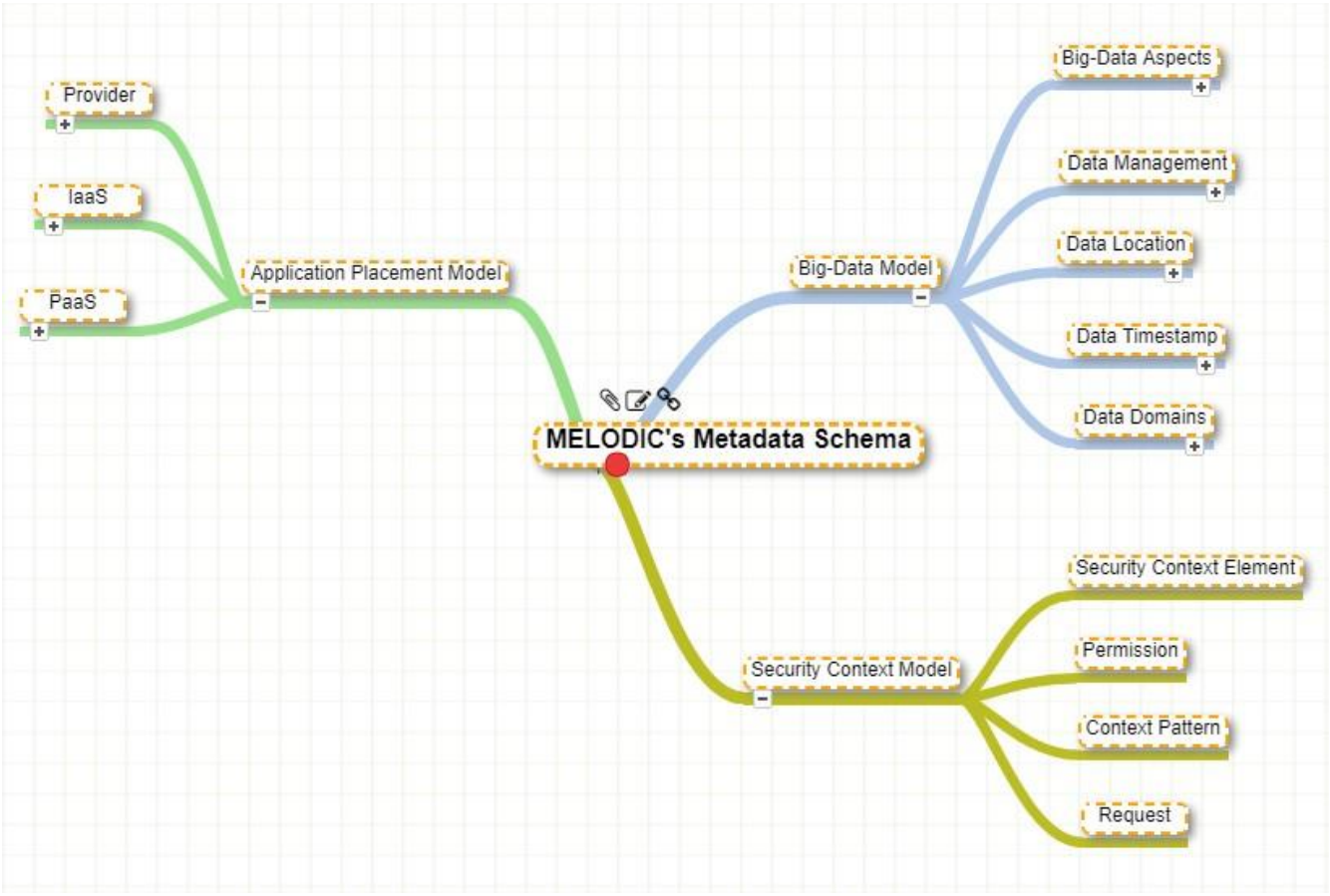

Figure 1: Melodic Metadata Schema overview

The first sub-model of the Metadata Schema is the *Application Placement sub-model*, which encompasses a number of concepts and properties that can be used for two main purposes: (a) for describing the requirements, constraints and preferences of a cloud application placement, and (b) secondly for describing the available cloud offerings, mainly at IaaS and PaaS levels. IaaS related concepts, like *processing* (e.g. CPU), *storage* (e.g. Capacity), *network capabilities* (e.g. Bandwidth) or *virtualization*, as well as PaaS related concepts, like *platform type* and *security controls*, are part of the Application Placement sub-model.

The second sub-model is the *Big-data sub-model*, which is meant to describe cross-cloud application and data management. It includes concepts used to describe data properties and aspects that must be taken under consideration during application placement and application reconfiguration planning. Big-data related concepts like *Volume, Velocity,* and *Quality;* data management concepts like *Acquisition, Persistency, Processing*, and *Data Location;* and temporal context as well as data domains, for instance *Finance, Social Networking*, have been captured in this sub-model.

The *Security Context sub-model* is the last part of the Melodic Metadata Schema, which encompasses concepts useful when describing and enforcing context-aware access control policies. The two central concepts are *Security Context Elements* and *Context Patterns*, which

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

are used to define context-aware access control policies for the Melodic platform's authorisation service (see deliverable D2.2 [1]), which will be detailed in the related deliverable D5.3 *Security Requirements and Design*.

It is expected that the Metadata Schema will be adapted to the specific business and technical requirements of each application of a single organisation.

2.1.1 Metadata Schema Management

Metadata Schema Management can be defined as the process of creating and maintaining the Metadata Schema for a specific Melodic-enabled application. This may involve the refinement or grounding of the metadata schema proposed in D2.4 [2] for the needs of a specific cloud application. Typically, this task is undertaken by the administrator.
Metadata Schema Management is required in cases where new concepts need to be taken into consideration, already modelled concepts need to be redefined, or even when obsolete concepts are no longer used and thus need to be removed from the schema. The following are just a few exemplary cases where the Metadata Schema is affected and it needs to be validated and possibly updated.

- At application design-time, i.e. when designing an application and modelling the application data domain. Alternatively, when modeling a pre-existing application that is ported to the Melodic platform.
- When new application placement or big-data processing conditions arise (including new requirements). For instance, when an application needs to process or manage new data types, or when the application data processing and/or storage specifications evolve.
- When new access control policies need to be enforced using new (additional) environmental conditions that must be taken into consideration emerge.

Specialised tools are required in order to efficiently manage the Metadata Schema. For this purpose, a Metadata Schema editor has been developed and will be presented in the next section. Moreover, the CAMEL editor (from the PaaSage project[1]) will be enhanced with the capability to encompass Metadata Schema elements into its models, catering for a loose interaction with it and also providing a nice way for extending current semantics of the CAMEL language (further details will be provided in terms of the deliverable D3.3 *IDE-plugin for data-aware design and development of multi-cloud data-intensive applications* [3]).
For more information on the Melodic Metadata Schema and the related CAMEL extensions, please refer to deliverable D2.4 [2].

[1] https://paasage.ercim.eu/

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

2.2 Metadata Schema Editor

In this section, the Metadata Schema editor is presented. First, we supply its use cases and design requirements. Next, the design decisions are explained, followed by the description of the editor architecture and its components. An example of Metadata Schema serialization is also provided and a short editor walkthrough is eventually presented.

2.2.1 Use Cases and Requirements

The main goal of the Metadata Schema editor is to manage the Metadata Schema elements and structure. This goal can be broken down to a series of specific capabilities that must be offered to the application administrator for creating, updating, deleting and retrieving the Metadata Schema concepts and properties, as well as importing/exporting Metadata Schema to the Models Repository or to a file. The Metadata Schema retrieval and editing capabilities are depicted in Figure 2 whereas the import/export capabilities are depicted in Figure 3.

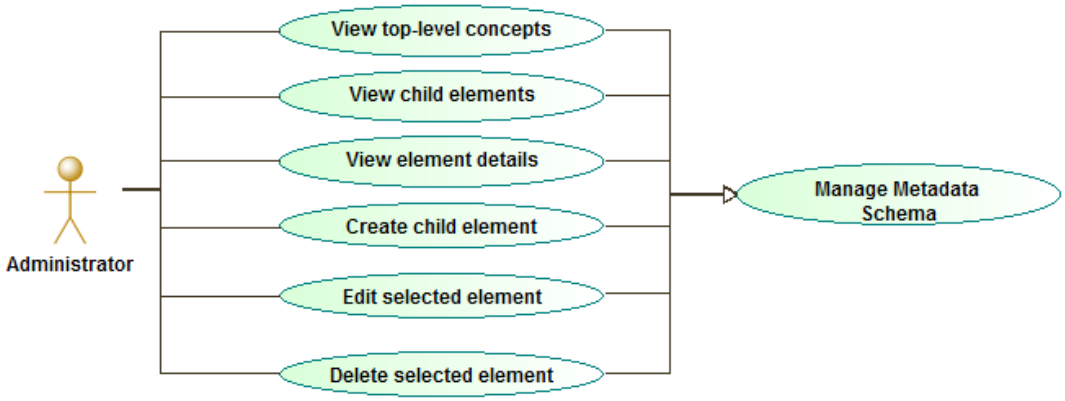

Figure 2: "Metadata Schema Management" use case

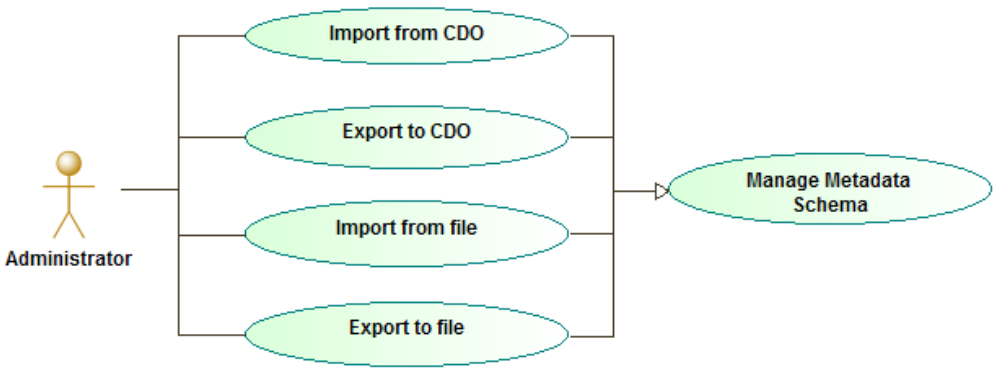

Figure 3: "Metadata Schema Import/Export" UML use case

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

A special requirement for the Metadata Schema editor is the capability to connect and interact with the Melodic Models Repository, used by several Melodic components, in order to store the Metadata Schema as it happens with the application's CAMEL model. When stored in the Models Repository, the Metadata Schema becomes "visible" to the rest of the Melodic components, thus achieving metadata-level integration between components. The Models Repository is implemented using the Eclipse Connected Data Objects (CDO) persistence and distribution framework[2]. The Metadata Schema editor is able to store the Metadata Schema into and retrieve it from the Models Repository.
Figure 4 depicts the use case of the storage of the Metadata Schema into the Models Repository of the Melodic platform, and its subsequent usage in application modeling via the CAMEL editor. This diagram implies the metadata-level integration between the two editors.

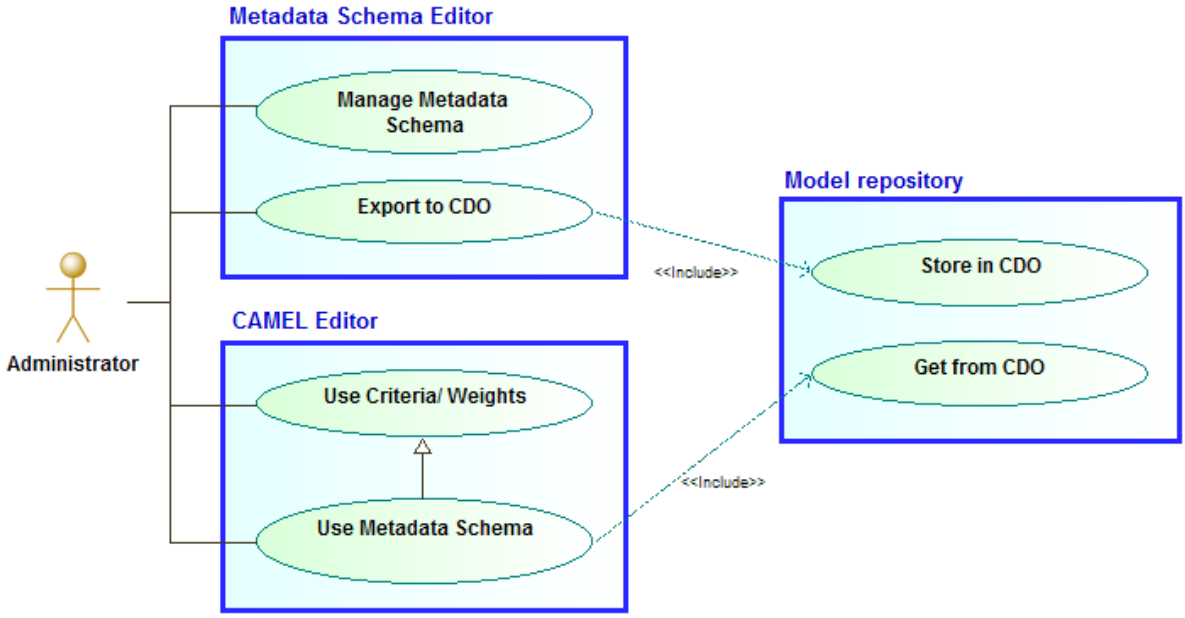

Figure 4: "Metadata Schema usage in App Modeling" UML use case

2.2.2 Architecture

In this section, the architecture of the Melodic Metadata Schema editor is discussed. Figure 5 depicts the components of the architecture as well as the interacting "external" components (i.e., the Models Repository of the Melodic platform) and user roles.

[2] http://www.eclipse.org/cdo/

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

According to consortium decisions (based on requirement analysis), the Metadata Schema editor is delivered as a web-based application that takes advantage of several modern technologies available (AJAX[3], CSS[4], DOM[5], and REST[6] to name a few). This decision aspires the project to deliver tools that are easy to learn and intuitive to use, and at the same time are relatively simple to integrate with the rest of the relevant Melodic framework components.
Another important decision taken was to separate the editor capabilities, responsible for the actual processing, storage and management of the Metadata Schema, from other capabilities responsible for interacting with the end user and other Melodic platform components. The Metadata Schema Management (MSM) capability acts as the middleware responsible for the Metadata Schema CRUD operations[7], schema serialization and deserialization, and applies the necessary validation checks[8]. The MSM middleware is implemented as a web service that provides a RESTful API for use by the other editor components (MSM middleware in Figure 5). This design results in a multi-tier architecture, which is typical to several modern web-based systems. Following this pattern helps keeping the Metadata Schema Management capability neutral to whom or through which channels Metadata Schema Management and control will take place. In release 1.5 of the Melodic platform, the Metadata Schema editor includes a web-based user interface (Admin-facing component in Figure 5), and an additional component for connecting to Melodic Models Repository for retrieving and storing metadata schema (CDO client in Figure 5). Eventually, an internal storage capability has also been included in the architecture for temporarily caching schema changes and for faster viewing, thus reducing interactions with the Models Repository. In this way we motivate the storage of complete and coherent versions of the vocabulary in the Models Repository. This is quite beneficial since changes to the vocabulary persisted in the Models Repository directly affect the CAMEL editor (see more details in section 2.2.4). The Models Repository depicted in Figure 5 is not part of the editor architecture, but part of the overall Melodic platform (see deliverable D2.2 [1]).

A feature of this design is that the Metadata Schema editor can be embedded into or integrated with other platforms, by adapting or replacing the admin-facing and CDO client components with other platform-specific services and components.

---

[3] http://adaptivepath.org/ideas/ajax-new-approach-web-applications/

[4] https://www.w3.org/Style/CSS/Overview.en.html

[5] https://www.w3.org/DOM/#what

[6] https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest

[7] CRUD operations: create, read, update, and delete

[8] Basic validation checks are currently supported; mainly uniqueness and type checks

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

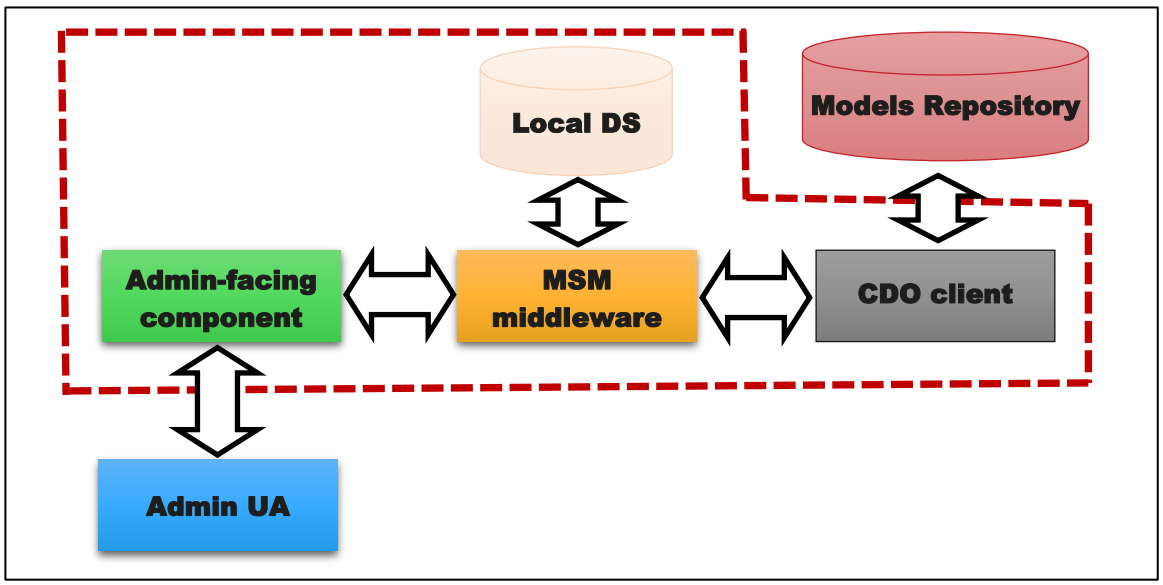

Figure 5: Metadata Schema Editor Architecture

The core of the Metadata Schema editor lies within the area enclosed by the red dotted line in Figure 5. It encompasses four components, which implement the relevant functionalities. Three of them provide the means for humans or external software to interact with the editor, whereas the fourth (in grey box) implements the integration to the Melodic platform's Models Repository. Next, a brief presentation of the Metadata Schema editor components is provided:

- Metadata Schema Management middleware (MSM middleware). This component is the core of the editor. It implements the business logic and is responsible for maintaining and modifying the internal Metadata Schema representation and related data structures. It offers a RESTful API implemented as a dedicated web service, which can be invoked in order to retrieve, modify and import/export the Metadata Schema. The Admin-facing component interacts with the MSM middleware through this API, in order to carry out the administrator commands. Furthermore, the same API can be used by third party software to interact with the MSM middleware, thus allowing editor extensions towards new versions with enhanced or differentiated capabilities. The MSM middleware uses the CDO client component in order to store or update the Metadata Schema model in the Melodic Models Repository. It also uses the Local datastore (internal to the editor), for temporarily storing and caching Metadata Schema.
- Admin-facing component. This component constitutes the user-facing part of the editor. Specifically, it offers a dynamic web page that enables the administrator to retrieve and manage the Melodic Metadata Schema using a graphical user interface. It also serves all accompanying web assets required (JavaScript libraries, CSS style sheets, images, etc). Furthermore, it provides the back-end web endpoint that serves the web page AJAX requests. The web page (it runs in the administrator's browser) captures user actions and forwards them to the back-end endpoint, using AJAX calls. The back-end subsequently

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

translates the actions into appropriate MSM middleware API calls, receives the middleware responses and translates them back to user-presentable information (returned as JSON[9] responses).

- **Local datastore (DS).** The Local DS is an internal repository that the editor uses to temporarily store the Metadata Schema (see chapter 3). This datastore is also used by Weights Calculator to store information about weights (see section 3.2.2). The current implementation uses a triple store as its Local datastore component (namely Fuseki[10]) through a dedicated serialization library that loosely follows JPA[11] conventions.
- **CDO client.** This component is responsible for the communication with the Melodic Models Repository. It encompasses a number of EMF[12]-style classes and interfaces, suitable for creating EMF models of the Metadata Schema, thus making its saving to CDO a relatively simple task. The CDO client also acts as an abstraction layer, allowing a revision of the current Melodic CDO model and upgrades of the Models Repository, without affecting the rest of the editor.

The Melodic platform components that interact with the Metadata Schema editor are described in the following.

- **Models Repository**. This is the Melodic platform component responsible for storing the various application and data models, pertaining both to design-time and runtime phases. The Models Repository has been implemented using the Eclipse Connected Data Objects (CDO) framework. The XML Metadata Interchange (XMI[13]) format is used for importing and exporting models to/from the Models Repository. Furthermore, suitable EMF-style classes have been defined in order to facilitate the programmatic access to the models stored in the Models Repository.
- **Admin User-Agent (UA).** It is the administrator's web browser, which must be a modern web 2.0 one.

Figure 6 describes a "sub-concept creation" action, thus illustrating by example the internal functioning of the editor. Sub-concept creation means the creation of a new concept that specialises a pre-existing (more generic) concept and its addition into the Metadata Schema.

---

[9] https://www.json.org/

[10] https://jena.apache.org/documentation/serving_data/

[11] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[12] http://www.eclipse.org/modeling/emf/

[13] XMI is an OMG standard for exchanging metadata information using XML. http://www.omg.org/spec/XMI/ http://www.omg.org/spec/XMI/

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

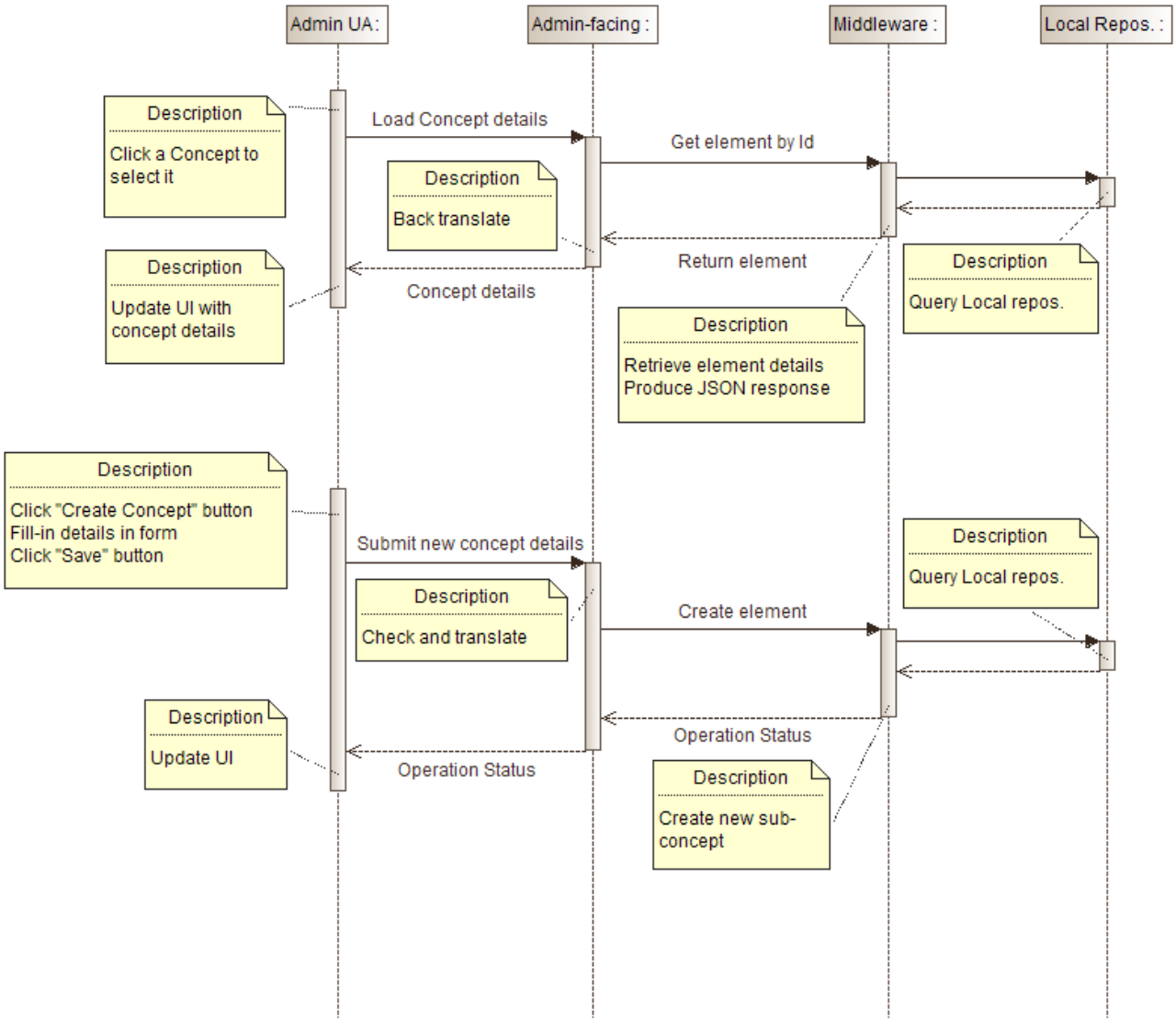

Figure 6: "Create Sub-Concept" sequence diagram

The Metadata Schema editor has been implemented using Java and is based on the architecture discussed above. The source code of the Metadata Schema editor is publically available on the official Melodic code repository[14].

[14] https://bitbucket.7bulls.eu/projects/MEL/repos/metadata-schema/browse/muse

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

2.2.3 Serialization

In this section a small but easily comprehensible example of Metadata Schema serialization is given (see Listing 1 below), involving the following concepts: Application Placement model, IaaS, Processing, GPU and some of their properties. The serialization format complies with the well-known XMI standard for metadata exchange and storing. The editor is capable of importing and exporting the metadata schema into/from the Models Repository using this format.

Listing 1: Sample XMI serialization

```
<?xml version="1.0" encoding="UTF-8"?>
<mms:MmsConcept xmi:version="2.0"
 xmlns:xmi="http://www.omg.org/XMI"
 xmlns:mms="http://www.melodic.eu/metadata/0.0.1"
 name="Melodic Metadata Schema" id="attr-mms" uri="mms:attr-mms"
 description="Root Concept" topLevel="true">
  <concept name="Application Placement model"
   id="attr-mms--app-placement"
   uri="mms:attr-mms--app-placement"
   description="Application Placement model">
    <concept name="IaaS"
     id="attr-mms--app-placement--iaas"
     uri="mms:attr-mms--app-placement--iaas"
     description="IaaS">
      <concept name="Processing"
       id="attr-mms--app-placement—iaas--processing"
       uri="mms:attr-mms--app-placement--iaas--processing"
       description="Processing">
        …
        <concept name="GPU"
         id="attr-mms--app-placement—iaas—processing--gpu"
         uri="mms:attr-mms--app-placement--iaas—processing--gpu"
         description="GPU">
          <property name="hasMinNumberofCores"
           id="attr-mms--app-placement--iaas—processing--gpu--hasMinNumberofCores"
           uri="mms:attr-mms--app-placement--iaas—processing--gpu--
hasMinNumberofCores"
           isDataProperty="true" rangeUri="xsd:positiveInteger"
           description="hasMinNumberofCores"/>
          …
        </concept>
      </concept>
    </concept>
    …
  </concept>
  …
</mms:MmsConcept>
```

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

2.2.4 Metadata Schema and CAMEL model

As already mentioned, the Metadata Schema is meant to be used as a common vocabulary between Melodic components. Since CAMEL is used to describe the deployment of a Melodic-enabled multi-cloud application and its requirements, it encompasses references to the Metadata Schema. Therefore, CAMEL has been extended to support such references (see deliverable D2.2 [1]).

Listing 2 gives an excerpt of a CAMEL model referencing two Metadata Schema elements (the *GPU* concept and the *hasMinNumberofCores* concept property). The highlighted parts are the references and they are introduced with the *annotation* field.

In this example, the CAMEL model requires that the VM's GPU should have at least 2 cores. The introduced requirement has taken the form of a sub-feature of CAMEL's VM element which references the GPU concept, from Metadata Schema, for which an attribute mapping to the hasMinNumberofCores property has also been defined, taking the value of 2.

Listing 2: CAMEL model excerpt referencing Metadata Schema property

```
<?xml version="1.0" encoding="ASCII"?>
<camel:CamelModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:camel="http://www.camel-dsl.org/camel"
 xmlns:constraint="http://www.camel-dsl.org/camel/constraint"
 xmlns:type="http://www.camel-dsl.org/camel/type" xsi:schemaLocation="…"
 name="MyApplication">
  <deploymentModels name="MyApplicationDepModel">
    <internalComponents name="MyComponent">
      <requiredHost name="MyComponentRequiredHost">
    </internalComponents>
    <vms name="MediumProcessingVM">
      <subFeatures name="GPU" annotation="mms:GPU">
        <attributes name="minCoreNumber" annotation="mms:hasMinNumberofCores"
         unitType="CORES">
          <value xsi:type="type:IntegerValue" value="2"/>
        </attributes>
      </subFeatures>
      <providedHosts name="VMHost"/>
    </vms>
    <hostings name="ComponentToVM" providedHost="…" requiredHosts="…">
  </deploymentModels>
  …
</camel:CamelModel>
```

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

2.3 Metadata Schema Editor Walkthrough

In this section, we present a short walkthrough of the Metadata Schema editor usage, which involves the modification of the Metadata Schema. Based on the editor's GUI, the administrator is able to update the Metadata Schema concept hierarchy by adding, removing or modifying concept details. In the following section, we will present (a) the initialization of the editor's Local Repository from Models Repository, and (b) a simple use case, where a new sub-concept is added under a parent concept.

2.3.1 Initialization

Before using the editor, it is necessary to fetch the Metadata Schema from the Models Repository into the Local datastore. This is achieved by clicking on the "Models repos. → Local repos." menu item. The menu appears by clicking the hamburger style glyph ☰ at the upper left corner of the page. Figure 7 illustrates the editor menu.

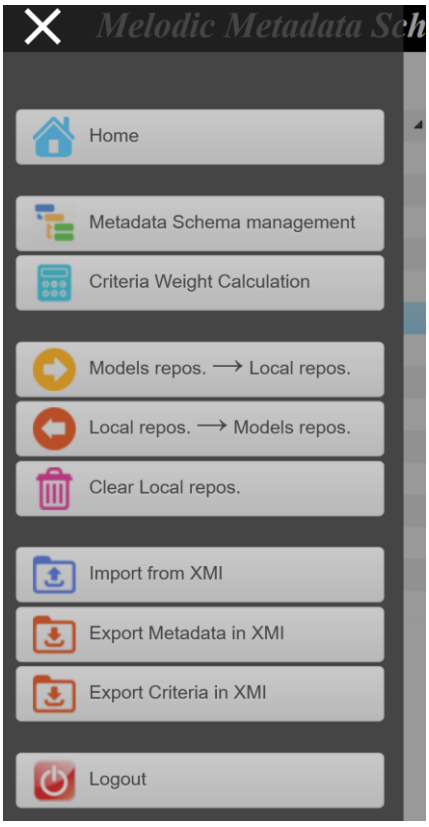

Figure 7: Metadata Schema Editor Menu

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

From the menu, the user can also transfer any changes made in the Metadata Schema, from the Local Repository into the Models Repository (menu item "Local repos. → Models repos.") thus making them visible to the rest of the Melodic components. Other options include emptying the Local Repository (menu item "Clear Local repos."), exporting Metadata Schema from Models repository (in XMI format), and reversely importing XMI files into the Models Repository. Eventually, the menu provides links to the Metadata Schema editor's home page as well as to the Weights Calculation tool web page (presented in the following chapter 3).

2.3.2 Editor usage

By selecting "Metadata Schema management" from the menu, the Metadata Schema editor web page is loaded (see Figure 8). Using the tree-view in the left-hand side of the page, the user can navigate through the Metadata Schema. The tree is lazy-loaded from the web server, as the user expands its nodes. Concepts are represented with yellow folder icons in the tree view, whereas other artifacts (e.g. concept properties) are represented with coloured balls. By selecting a tree node, its details are loaded into the "Node Properties" form, found in the main area of the page. No web page reloading takes place, but the page dynamically refreshes its data when they become available from server. Furthermore, the form fields change according to the type of the tree node selected (i.e. concept, concept property or concept instance). For example, the Range form field is only available for the concept property nodes. The white text fields can be edited while the grey ones are read-only. At the bottom of the page, there is a row of buttons for creating new nodes (concepts, properties, and instances), deleting the selected node or saving changes in the form. Buttons are disabled and re-enabled as the selected node in the tree-view changes. For instance, when selecting a property, the buttons for creating new child concepts, instances or properties are disabled.
In the remaining section, a simple use case, where a new sub-concept will be created, is detailed. First, the user selects the "Data management" node in the tree view, as shown in Figure 8. This node will be the (immediate) parent concept of the concept to be added. Upon selection, the node's data are fetched from the Local Repository and are displayed in the "Node Properties" form. Next, the "Create Concept" button must be pressed to start creating the new sub-concept.

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

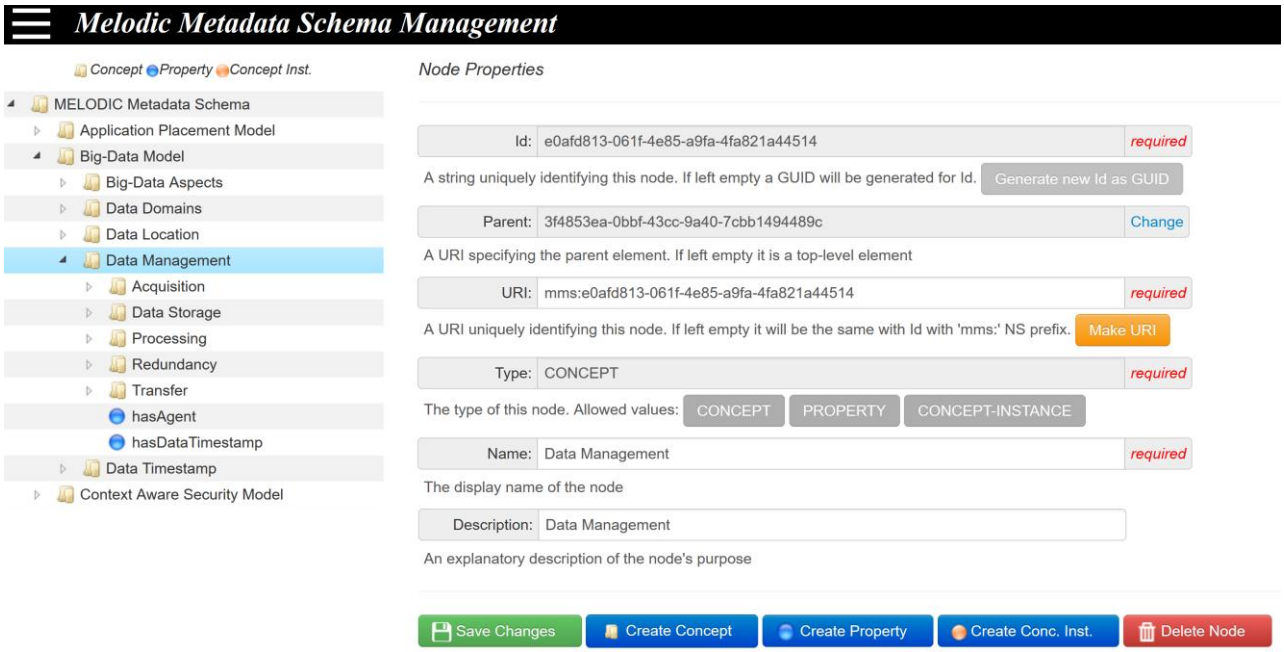

Figure 8: Selection of the parent concept

The same functionality is also available through a context popup menu, which is available by right clicking on the parent node, and then clicking the "New Concept" item (see Figure 9).

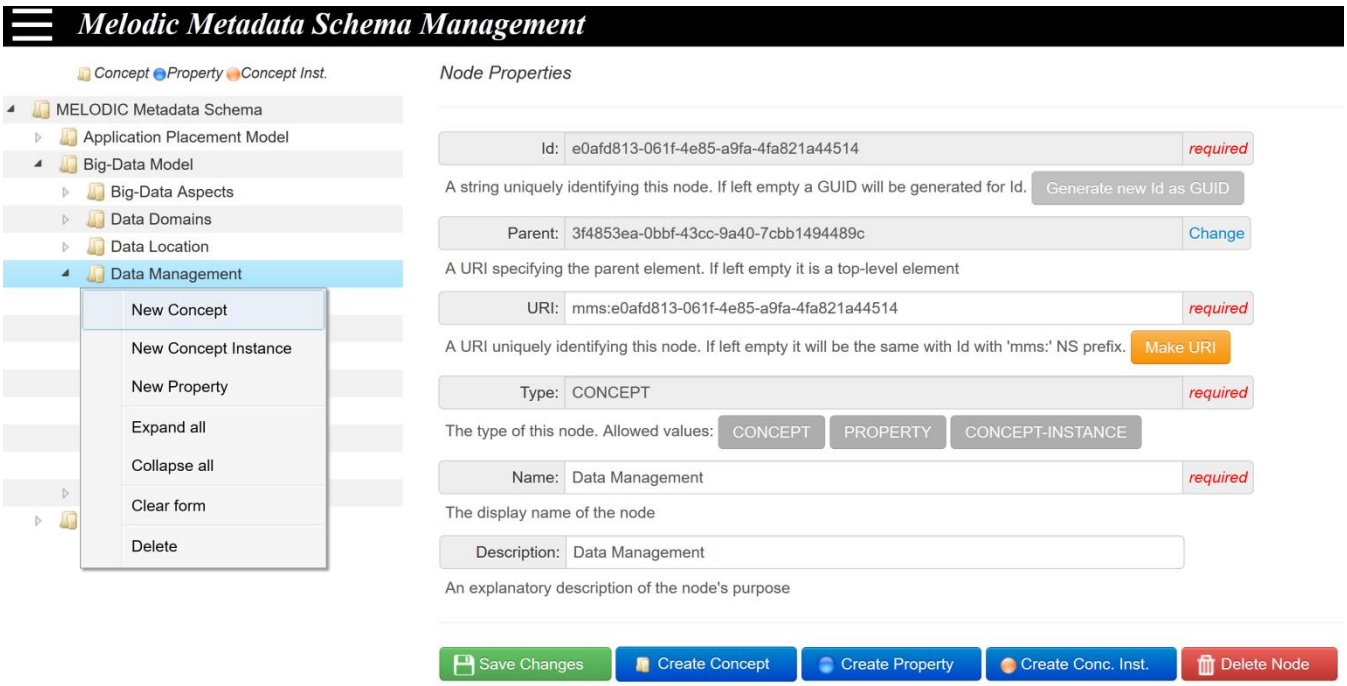

Figure 9: Editor's context popup menu

Both actions (i.e. pressing "Create Concept" or clicking "New Concept" in the context popup menu) will make the "Node Properties" form (in the main page area) to adapt in order to include only those fields that are relevant to Concepts (i.e. Id, Parent Id, URI, Type, Name, and Description). The values of the Id and the URI fields are automatically completed using a random Unique ID and the Parent Id field is populated with the Id of the parent node (which is selected in the tree view), as shown in Figure 10.

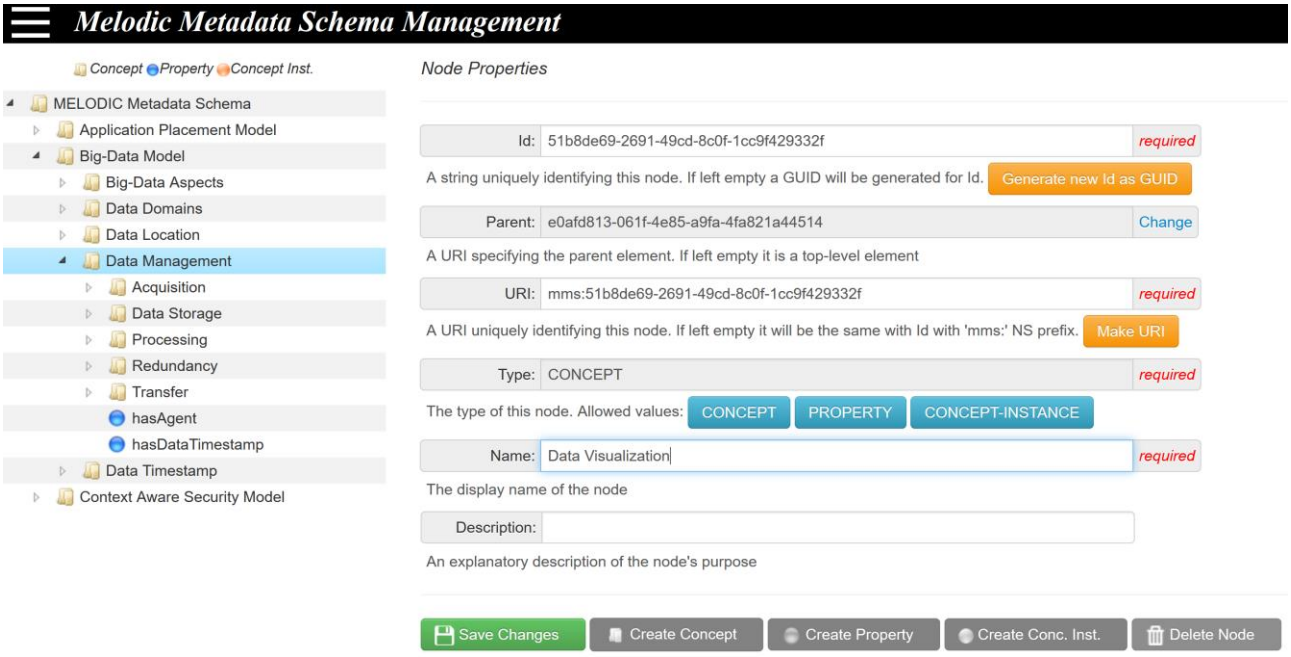

Figure 10: New Concept form

The user can modify the Id and URI values if needed, fill-in the new concept's name, for instance "Data Visualization", and optionally give a description, in the corresponding fields. Eventually, by pressing the "Save Changes" button, the new sub-concept's data are submitted to the Local Repository for storing. Afterwards, the tree view is refreshed in order to include the newly added concept.

2.4 Eclipse EMF-based Metadata Schema editor

Apart from the web-based Metadata Schema editor, a Java-based desktop release of the editor is also available. It was generated using the Eclipse EMF tools and the same Ecore model used to generate the Metadata Schema Java classes and interfaces. This editor, shown in Figure 11, will not be further detailed since it is a by-product of modeling the domain classes using Eclipse EMF.

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

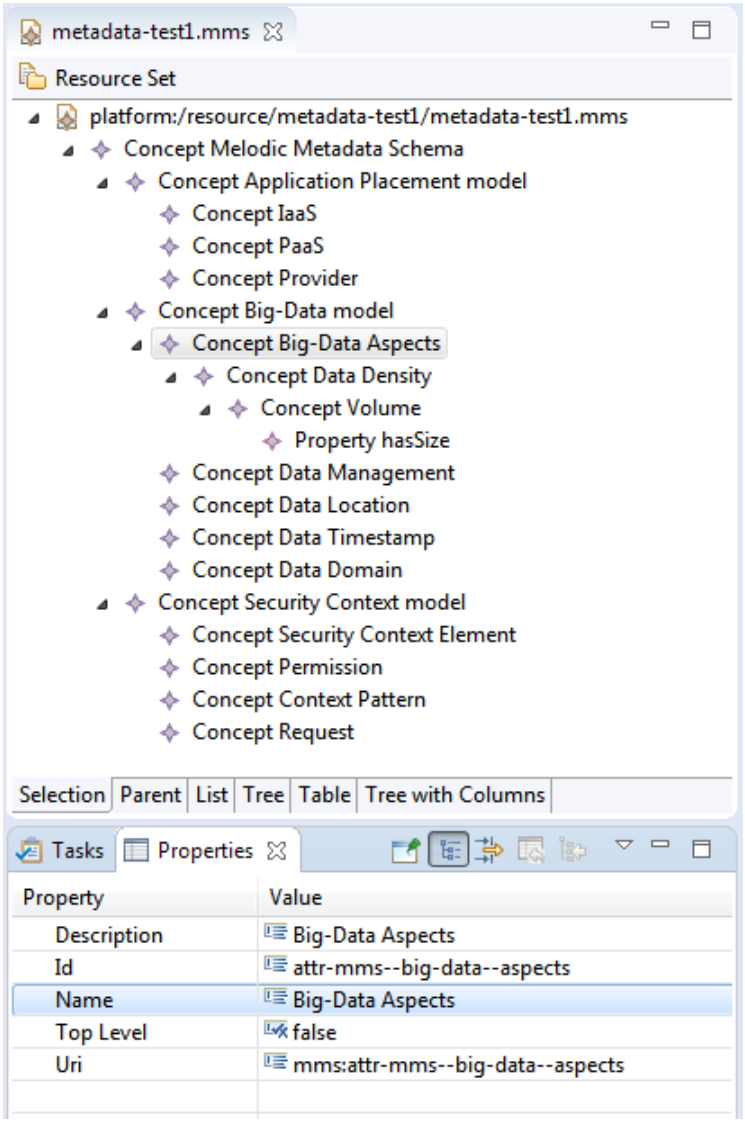

Figure 11: Eclipse EMF-based Metadata Schema editor

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

3 Weights Calculation Tool

In this chapter, the Melodic Weights Calculation tool will be presented. This is a web-based tool developed in the context of the Melodic project, in order to provide application administrators with a systematic method of assigning weights to polynomial utility functions that can be encompassed in CAMEL models. However, this is a generic tool, which can be used in any case where weight or priority calculation is required. Therefore we have preserved the terminology used in the original method and thus referring to the items being compared as "criteria", regardless of if they are utility function terms or something else. A short description of the weight calculation method will be presented in the next section, followed by the detailed description of the tool in the rest of the chapter.

3.1 Weight Calculation Method

In release 1.5 of the Melodic platform, the CAMEL editor and the CP generator component (part of the Upperware) will be enhanced in order to be able to model polynomial utility functions. Typically, weights are real numbers ranging between 0 and 1 and can also be represented as percentages (between 0% and 100%). If the sum of all weights equals to 1 or 100%, the weight set is called *normalized*.

Despite being simple and straightforward to assign weights to a small number of criteria, it is, however, a tedious task to find suitable weights for a larger number of criteria, especially when some of them are contradicting (e.g. increase availability and decrease cost). To help application administrators alleviate this problem, the Weights Calculation tool has been developed and is offered.

The method implemented in the Melodic Weights Calculation tool is based on the Analytic Hierarchy Process (AHP) [4]. The most prominent feature of AHP is that it decomposes the weight calculation problem into a series of pairwise comparisons. That means that the application administrator must indicate which criterion is more significant in every pair, and how much more significant, using an empirical scale typically ranging from 1 (indicating "equally important") to 9 (meaning "extremely more important"). Using these partial, pairwise comparisons, the method combines them into a set of weights. Although subjective judgements are involved (i.e. pairwise comparisons made by expert users) the method delivers correct results, and it has been widely used for several years across different application domains [5].

To help the reader better comprehend the method, in the remaining section an example will be given after explaining each step of the method. In this example, the six following Metadata Schema concepts will be used in the (polynomial) utility function:

- *Data Domains*, *Data Location* and *Data Timestamp* (in the Big-data sub-model), and
- *Processing*, *Redundancy* and *Transfer* (in Data Management in the Big-data sub-model)

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

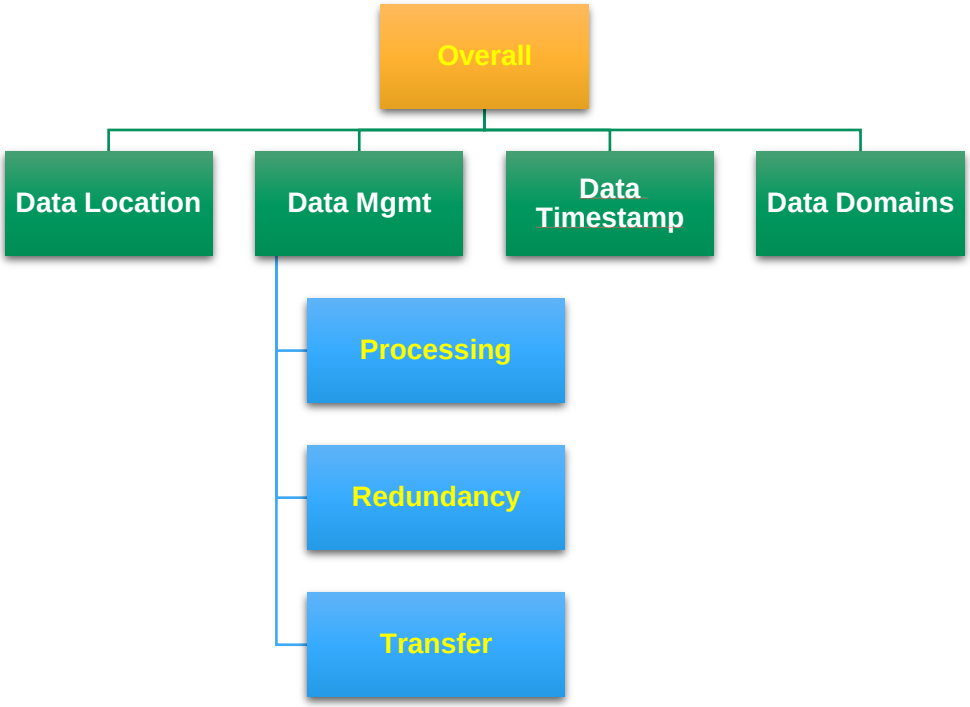

Figure 12: Sample hierarchy derived from Metadata Schema

The method exploits the first two phases of the AHP. The first one requires to group criteria into a hierarchical (tree-like) fashion. Pairwise comparisons can take place only between criteria belonging to the same group and level (i.e. criteria represented by sibling nodes in the hierarchy). This approach reduces the number of required comparisons. For example, if there are 10 criteria, without hierarchical structuring it would require 45 pairwise comparisons. If, however, criteria are organized in two equal-sized groups, under a common root, it would take 21 pairwise comparisons. Moreover, hierarchical structuring enables organizing criteria into relatively homogenous groups and sub-groups, based on their domain, type or degree of detail.
The Metadata Schema presented in deliverable D2.4 [2] will be used as the hierarchical criteria grouping in terms of the first phase of AHP, for comparing the terms of the utility function. Therefore, application administrators are not required to undertake the task of grouping criteria and creating the corresponding hierarchy. Figure 12 illustrates the hierarchical grouping of the six criteria of the example, as it derives from the Metadata Schema.

In the second AHP phase, pairs are derived from the hierarchical structure of the first phase. For the root node and each intermediate node their direct children nodes must be compared, each one with all its siblings. In the example, there are two such nodes, (i) the root node (marked as *Overall*), and (ii) the *Data Management* node. Therefore, two groups of nodes are formulated, where the included nodes must be pairwise compared to all their siblings. Table 2 and Table 3 (the leftmost columns) list all possible combinations of the corresponding comparison pairs. The order of criteria in each pair is not significant.

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

Following, the pairwise comparisons take place, where the application administrator indicates the most important criterion in each pair, as well as its relative importance over the counter criterion. Relative importance is expressed as a positive integer, typically between 1 and 9. Table 1 lists the commonly used meanings attributed to each relative importance value. In the example, the administrator responses for the two groups are included in Table 2 and Table 3. Eventually, the relative importance values from all comparisons are then combined to calculate weights (see next section 3.1.1).

Table 1: Standard meaning of relative importance values in AHP

Importance	Definition
1	Equal importance
2	Equal to moderately importance
3	Moderate importance
4	Moderate to strong importance
5	Strong importance
6	Strong to very strong importance
7	Very strong importance
8	Very to extremely strong importance
9	Extreme importance

Table 2: Example comparison pairs for top-level group

Pairs	Preference
Location vs. Management	2 – Almost equal
Location vs. Timestamp	4 – Moderate to strong
Location vs. Domains	9 – Extreme
Management vs. Timestamp	4 – Moderate to strong
Management vs. Domains	9 – Extreme
Timestamp vs. Domains	5 – Strong

Table 3: Example comparison pairs for second-level group

Pairs	Preference
Processing vs. Redundancy	5 – Strong
Processing vs. Transfer	1/3 – Moderate less pref'd
Redundancy vs. Transfer	3 – Moderate

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

3.1.1 Relative Weight Calculation

For every group, a relative importance comparison matrix is compiled using the responses of administrator in the corresponding pairwise comparisons. Let the criteria of a group be numbered from 1 to K. The outcome of the comparison of criterion *i* and criterion *j* (where *i, j* are between 1..K), will be placed in row *i* and column *j* of the matrix. In its transposed position (i.e. row *j* and column *i*) the inverse value will be set (e.g. if the comparison outcome is 4, in transposed position we will put ¼); see Table 4. On the matrix diagonal, all elements are set to 1, meaning that comparing a criterion to itself always yields equal importance.
Following, eigenvector analysis is applied on the matrix. Eigenvectors including non-real elements are rejected since they would lead to weights that are complex numbers. An eigenvector where all elements are real numbers and at least one is non-zero is a suitable solution. This eigenvector is normalized so that its elements sum to 1, thus representing the relative weights for the criteria of the given group. The above process is applied on all groups at all levels in the hierarchy. Eventually, every node in the criteria hierarchy will be attributed with a relative weight. Table 4 gives the relative importance comparison matrices and the corresponding relative weights (resulting after eigenvector analysis and normalization) for the two groups of the example.

Table 4: Relative weight calculation steps

Relative importance comparison matrix

$$\begin{array}{c} \\ DL \\ DM \\ DT \\ DD \end{array} \begin{array}{c} \begin{array}{cccc} DL & DM & DT & DD \end{array} \\ \begin{bmatrix} 1 & 2 & 4 & 9 \\ 1/2 & 1 & 4 & 9 \\ 1/4 & 1/4 & 1 & 5 \\ 1/9 & 1/9 & 1/5 & 1 \end{bmatrix} \end{array}$$

Relative weights after eigenvector analysis

Criteria	Weights
Data Location	0.49
Data Management	0.35
Data Timestamp	0.12
Data Domains	0.03

Relative importance comparison matrix

$$\begin{array}{c} \\ PR \\ RD \\ TR \end{array} \begin{array}{c} \begin{array}{ccc} PR & RD & TR \end{array} \\ \begin{bmatrix} 1 & 5 & 1/3 \\ 1/5 & 1 & 3 \\ 3 & 1/3 & 1 \end{bmatrix} \end{array}$$

Relative weights after eigenvector analysis

Criteria	Weights
Processing	0.39
Redundancy	0.28
Transfer	0.33

Normalization is achieved by scaling each weight using a normalization factor (Eq. 1) so that the sum of the scaled (normalized) weights sum to 1 (Eq. 3). The normalization factor is the inverse of the sum of the initial weight values (Eq. 2). Let K be the number of criteria:

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

$$w_i^{norm} = f \cdot w_i^{init} \quad \textit{(Eq. 1)}$$

$$f = 1 \Big/ \sum_{i=1}^{K} w_i^{init} \quad \textit{(Eq. 2)}$$

After normalization it holds:

$$\sum_{i=1}^{K} w_i^{norm} = 1 \quad \textit{(Eq. 3)}$$

3.1.2 Overall Weight Calculation

The last step of the method involves a flattening operation where relative weights are combined to get the overall weight of each criterion, which expresses its importance compared to any other criterion across all groups. Without this step, only criteria belonging to the same group would be comparable. Flattening is achieved by multiplying the relative weight of a criterion with the relative weights of all its predecessor nodes in the hierarchical structure (i.e. all nodes in the tree path from criterion node up to the root). Figure 13 depicts the criteria hierarchy of the example, where every node is annotated with its relative and overall weight. For example, the overall weight of *Processing* results by multiplying its relative weight in *Data Management* group, which is *0.39*, with the relative weights of its parent nodes, i.e. $0.39 \times 0.35 \times 1.00 = 0.1365 \approx 0.14$.

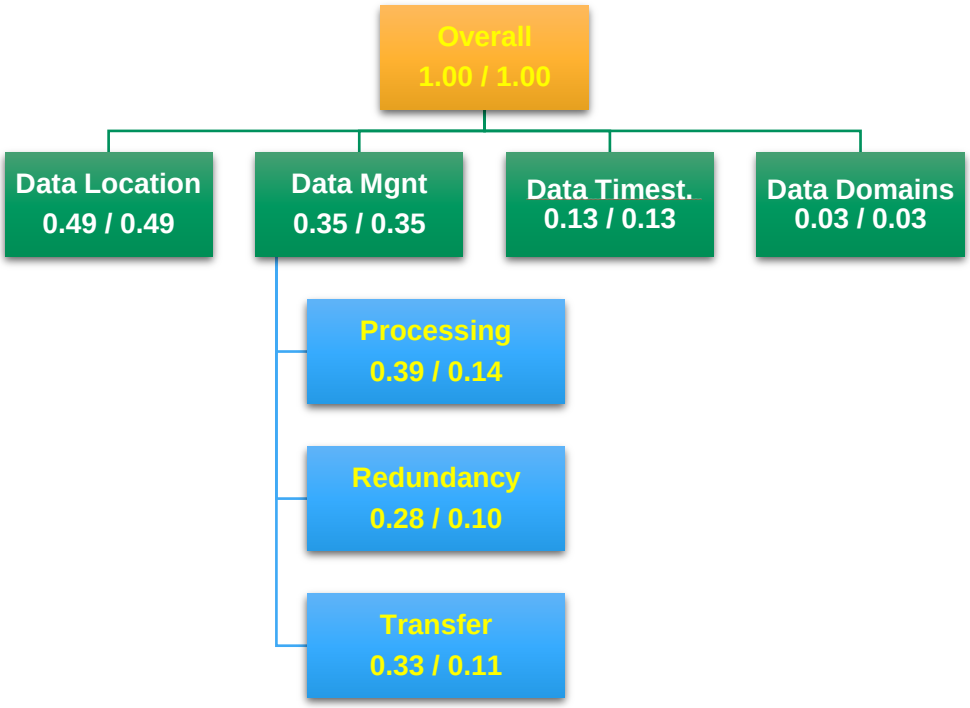

Figure 13: Example hierarchy with relative and overall weights

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

3.2 Weights Calculation Tool

In the remaining chapter, the Weights Calculation tool is analysed. This tool has been developed by following the architecture and design decisions used for Metadata Schema editor. Moreover, the two tools share the same (technical) infrastructure and are included in the same codebase.

3.2.1 Use Cases and Requirements

The main goal of the Weights Calculation tool is to enable selecting concepts, properties or instances of the Metadata Schema that may be used as terms in a polynomial utility function, and also estimate or fine-tune their weights. This goal can be broken down into a series of specific capabilities that must be offered to the application administrator for selecting criteria, assigning weights and answering to a set of pairwise comparisons, from which weights can be calculated. These capabilities are depicted in the following use case diagram (Figure 14).

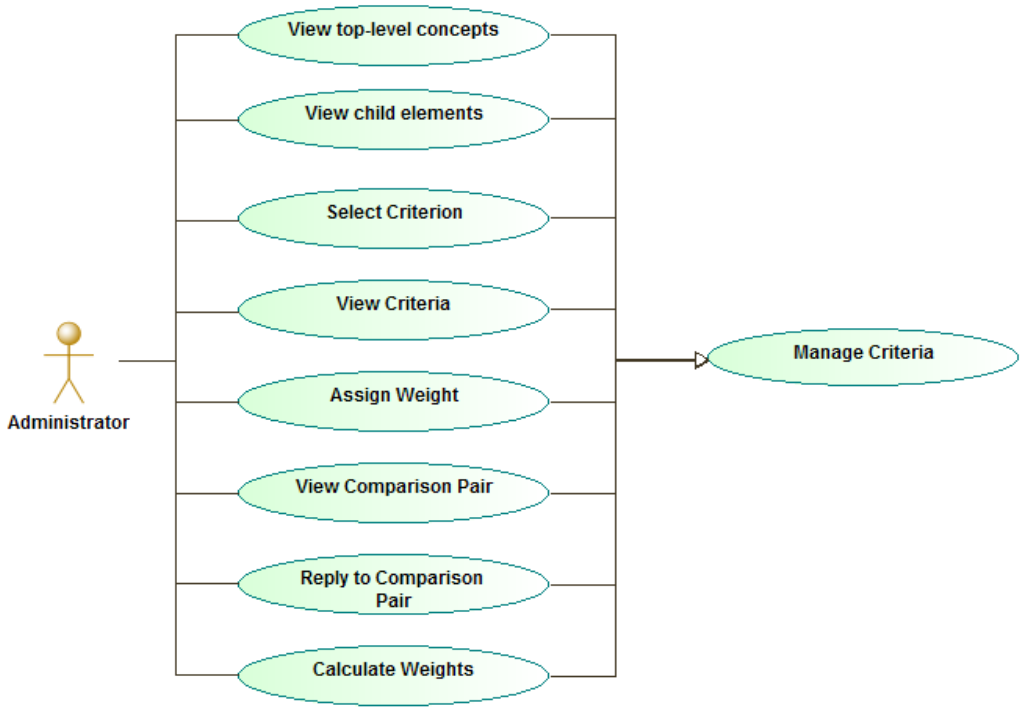

Figure 14: "Criteria Management" use case (includes weights calculation)

A final requirement for the Weights Calculation tool is the capability to connect to the Models Repository, as the current Melodic components do, in order to save the selected criteria along with their weights. Criteria will be saved into the utility function specification (as terms) in the CAMEL model, and will be retrieved and used at runtime by the Melodic solvers and the Utility Generator.

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

Figure 15 depicts the update of a utility function specification in a CAMEL model, in the Models Repository, as well as its usage in other melodic components at runtime. This diagram implies metadata-level integration between the two components.

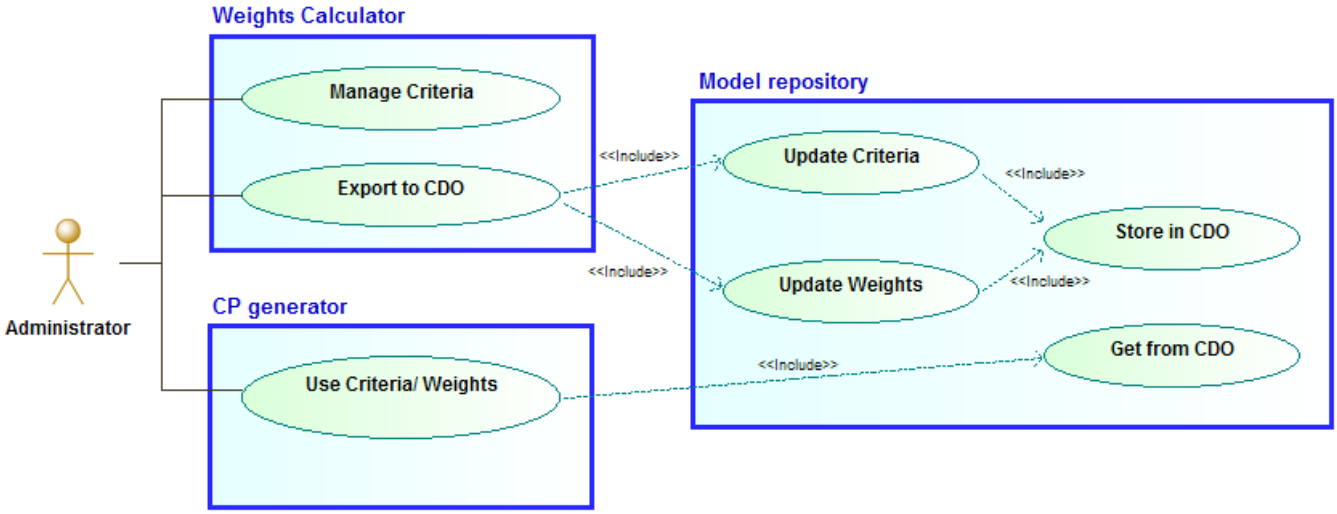

Figure 15: "Weight Calculation tool usage in App Modeling" use case

3.2.2 Architecture

As already mentioned, the Weight Calculation tool has been designed following an architecture similar to the Metadata Schema editor architecture. For this reason, only the components that are different will be discussed in more detail.

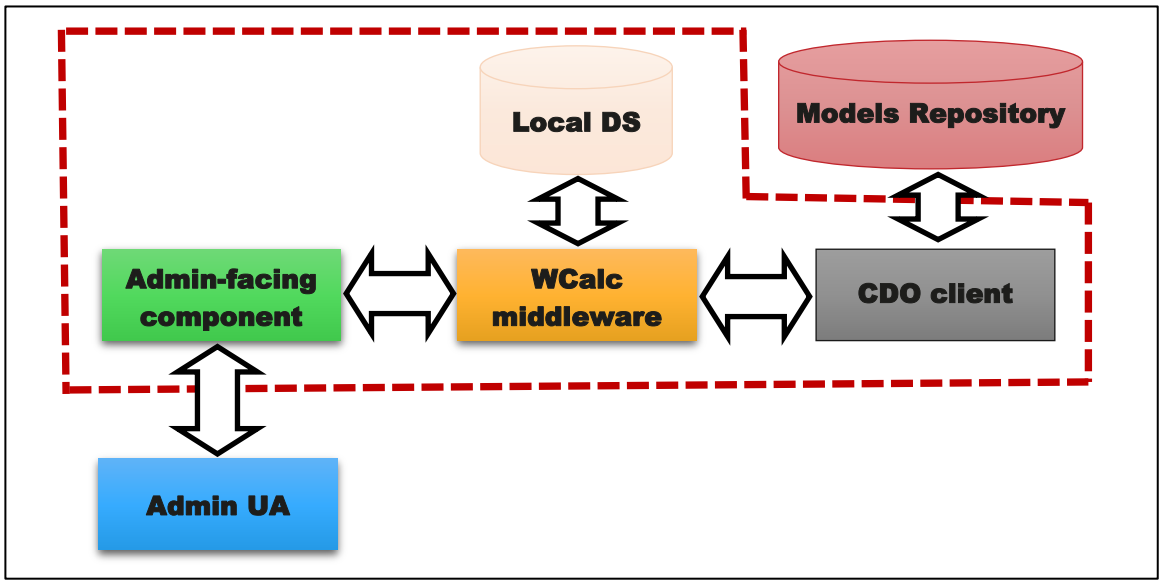

Figure 16: Weights Calculation Tool Architecture

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

In Figure 16, the Weights Calculation (WCalc) middleware component is new to the architecture, compared to the Metadata Schema editor. Common components will not be presented in detail:

- **Weights Calculation middleware (WCalc middleware).** This component is the core of the tool. It implements the business logic and is responsible for storing and modifying the criteria weights and comparison results. It provides a RESTful API that can be used to retrieve, modify and clear criteria, weights and pair comparisons. The Admin-facing component interacts with the WCalc middleware to carry out the administrator commands. The WCalc middleware also uses the CDO client component to update the CAMEL model in the Melodic Models Repository. It also uses the Local (internal) datastore for temporarily storing comparisons and weights, without contacting or affecting the Models Repository at every user action.
- **Admin-facing component.** This component is shared with the Metadata Schema editor and encompasses operations specific to weights calculation, which are distinguishable from the Metadata Schema editor operations.
- **Local datastore.** This component is shared with the Metadata Schema editor. It temporarily stores weights and comparison values.
- **CDO client.** This component is shared with the Metadata Schema editor. It permanently stores weights into the utility function specification. It therefore facilitates the update of the CAMEL model in the Models Repository.

Figure 17 describes the fulfillment of a "Load Weights from Server" action, which fetches previously stored weights along with any comparison pairs from the WCalc server. Thus, it illustrates by example the internal functioning of the tool.

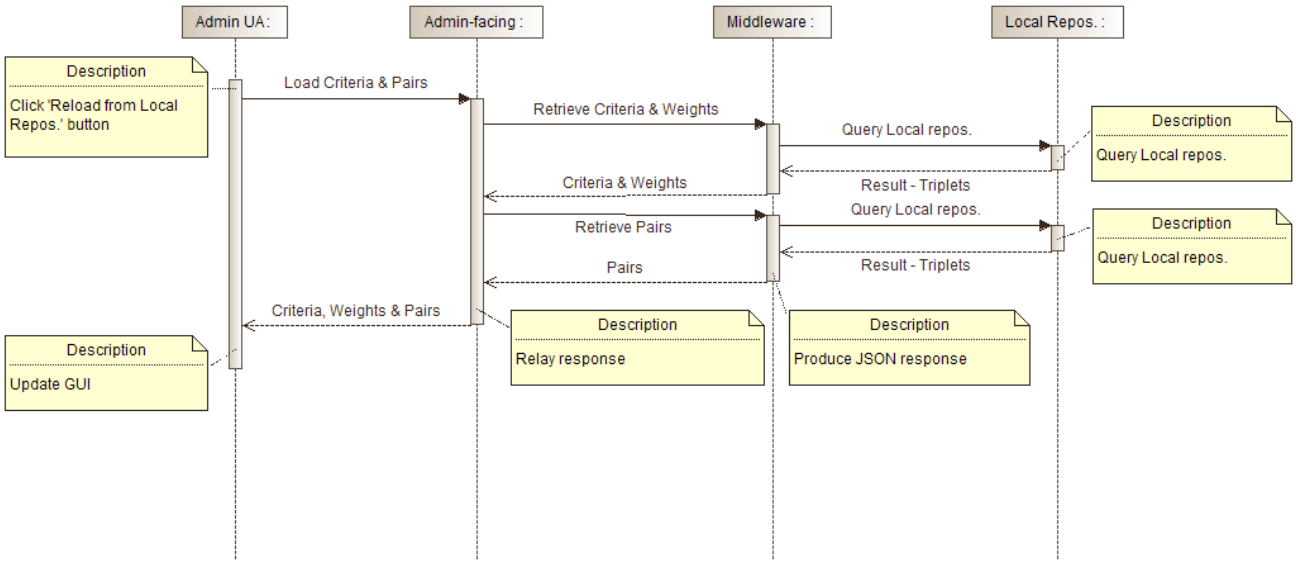

Figure 17: "Load Weights from Server" sequence diagram

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

3.3 Weights Calculation Tool Walkthrough

In this section, we present a short walkthrough of the Weights Calculation tool usage that involves the selection of criteria, the pairwise comparisons of them, weight calculation and eventually the update of the utility function specification in the CAMEL model.

3.3.1 Initialization

Before starting to use the tool, it is necessary to fetch the Metadata Schema from the Models Repository into the Local datastore. This is achieved by clicking on the "Models repos. → Local repos." menu item. The menu appears by clicking the hamburger style glyph ☰ in the upper left corner of the page (see Figure 7).

3.3.2 Tool usage

By selecting "Criteria Weight Calculation" in the menu, the Weights Calculation tool web page is loaded (see Figure 18). Using the tree-view in the left-hand side of the page, the administrator can navigate through the Metadata Schema and select concepts, properties and concept instances, by checking the corresponding checkboxes. For faster loading of the web page, the tree is lazy-loaded from the web server as the user expands its nodes. A selected tree node (concept, instance or property), will be added at the bottom of the criteria list, under the "Criteria Weights" tab, in the main area of the web page. The first time a criterion is added, its weight equals to zero (0). The user is able to change it and assign a value between 0.0 and 1.0. Under the list, the sum of all weights is given, which is updated every time a weight is set. If the sum does not equal to 1.0 (i.e. weights are not normalized), a warning message is displayed and the "Normalize Weights" button (on the right) is enabled. If this button is clicked, all weights will be normalized at once. Using the "Save to Local repos." and "Reload from Local repos." buttons, one can save or reload weights (and any associated pairwise comparisons) to/from the Local datastore. Clicking the "Update Models Repos." button will update the CAMEL model with the new utility function terms and weights.

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

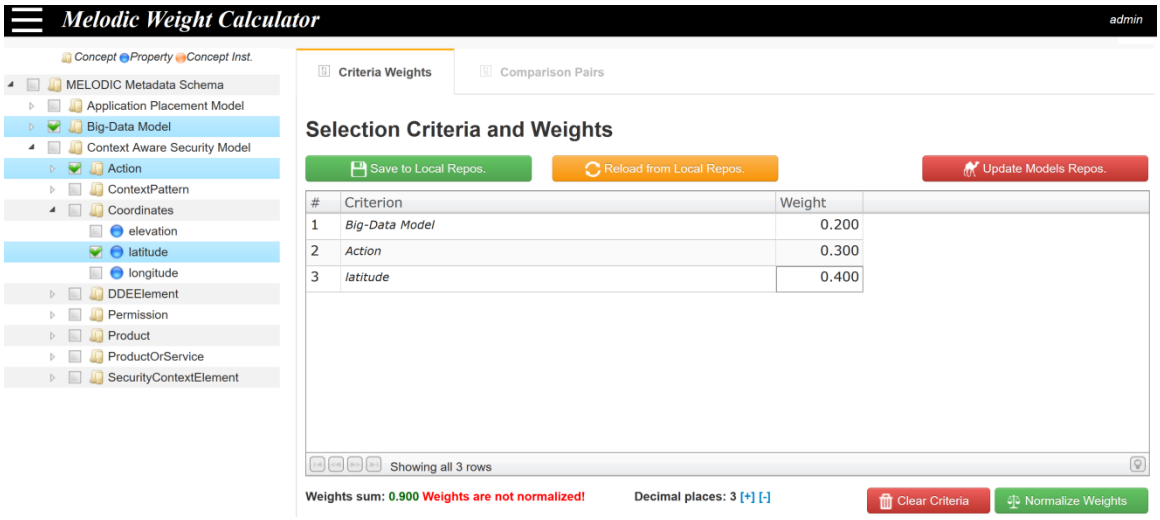

Figure 18: Weights Calculation using the Weights Calculation tool

Apart from adding the selected criterion into the criteria list, ticking a tree node will also cause the creation of new comparison pairs, where the newly selected criterion participates. The new pairs are added in the comparison pairs list under the "Comparison Pairs" tab (see Figure 19). If the pairs list is not visible then clicking the "Comparison Pairs" tab will hide the criteria list and display the comparison pairs list. In this tab, the user can indicate the relative importance of a criterion over its counterpart in a pair. Clicking the "Calculate Weights" button will calculate weights based on pairwise comparison and set them in the criteria list.

Figure 19: Pairwise comparisons in Weights Calculation tool

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

4 Conclusion

In this document, we have presented two design-time tools of the Melodic project. Specifically, we have presented (i) the Metadata Schema editor, used to create and maintain the Melodic Metadata Schema, and (ii) the Weights Calculation tool, used to estimate or fine-tune the weights of utility functions. Furthermore, we have described the method used in weights calculation, which derives from the well-known Analytic Hierarchy Process. This document serves as an introduction to the two aforementioned tools.

In the final release of Melodic platform, we plan to enhance the Metadata Schema editor and Weights Calculation tool with new features and functionalities. The following features are being considered: (i) leveraging of new features that will be implemented in CAMEL, (ii) automated prefill of criteria from the utility function specified in a CAMEL model, and (iii) use of the Weights Calculation tool for calculating weights and priorities of additional Melodic models (apart from utility function), such as the priorities of optimisation goals in Requirements model of CAMEL.

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

5 References

1. Verginadis, Y., Horn, G., Kritikos, K., Zahid, F., Baur, D., Seybold, D., Mazumdar, S., Skrzypek, P., Prusiński, M.: D2.2 Architecture and Initial Feature Definitions. (2017)
2. Verginadis, Y., Patiniotakis, I., Halaris, C., Mentzas, G., Kritikos, K., Jeffery, K.: D2.4 Metadata Schema. (2017)
3. To Be Submitted on M16: D3.3 IDE-plugin for data-aware design and development of multicloud data-intensive applications. (2018)
4. Saaty, T. L.: Decision Making for Leaders: The Analytical Hierarchy Process for Decisions in a Complex World. Wadsworth, Belmont, California (1982)
5. Franek, J., Kresta, A.: Judgment Scales and Consistency Measure in AHP. In Procedia Economics and Finance, Vol. 12, pp. 164-173, ISSN 2212-5671, doi: 10.1016/S2212-5671(14)00332-3 (2014)

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