Miquel Buxons Vives

Design and implementation of data aggregation modules for the European platform Diviz

Programmers and User guides for developed modules

Directed by Aïda Valls Mateu

Computer science degree

Universitat Rovira i Virgili

Tarragona

February 2021

Contents

List of Tables

List of Figures

Chapter 1

Developed modules

In this chapter, I will do a complete documentation of the 5 implemented modules. First the theoretical definition of the operations done in the modules will be given. Next, second section is devoted to the general framework of Diviz and all the data structures and requirements to follow to implement a module in this platform. In this section, I explain the first contribution of the work, which is the definition of the format of 3 data structures in XMCDA for version v3. Finally, the third section details the main contributions of the work, that is, the design, implementation, examples, and test cases of each of these 5 modules:

- OWA operator
- OWA weights descriptors
- ULOWA operator
- Defuzzification
- Fuzzy labels descriptors

The source code, the test cases and the documentation generated for each module, are all uploaded on my personal Github. Buxons [2020] The design and development has been made according the requirements of the ITAKA research group, under the revision of Dr. Aïda Valls. However, it also has to be claimed the role of Sébastien Bigaret, researcher and development engineer in IMT Atlantique Bretagne Pays de la Loire, who has helped to resolve doubts and upload all the code to the web service used by Diviz, thus establishing with me a very close collaboration

In 2011, Jordi Canals made the first implementation of these methods in Java. I recommend you to have a look his documentation about the first implementation if you want to see the past updates of these modules Canals [2011].

Both implementations are based on the same theory about *Ordered Weights Average* (OWA) family modules, but they differ in their programming language and internal structure. For this new implementation some features have been improved, the internal behaviour of some modules has been changed, new functionalities have been included, and some I/O parameters use different XMCDA structures. Due to all these structural changes and the new design, the source code of the old implementation has not been used for the new version, being the code generated in this project original.

1.1 Theoretical framework behind modules

1.1.1 OWA (Ordered weighted Average)

Ordered Weights Averaging (OWA) operators are defined by R. R. Yager in Yager [1988], these operators are used to aggregate the crisp values in decision-making schemes and they obtain a single representative value from a set of data. This result is achieved by averaging the criteria product with a weights vector. The main particularity of OWA family operators is that the order in which the user introduces the values does not have vital importance as before the operations, there are a sort step. Into OWA model, the central behavioural policy of the aggregation procedure is that a set of weights are attached to the values according to their degree of performance. The values are first sorted from the best one to the worst one, and a weight is assigned to the positions of the values, ending with the aggregation of all the obtained values.

Since Prof. Yager introduced the order weighted averaging (OWA), they have been used in many different applications, such as:

- fuzzy control
- linguistic summaries
- market analysis
- image compression

Usually OWA input sets of data are values, but this methodology can accept linguistic data in its variants, LOWA and ULOWA, with the introduction of the fuzzy set element. They will be explained in further detail in the ULOWA subsection. Here we present the numerical version and its properties Isern et al. [2017]:

Definition: we define OWA as an aggregation operator, with these components:

Firstly, OWA is defined with elements commonly found in aggregation operators as:

- Set of alternatives, $A = \{a_1, a_2, \ldots, a_p\}$
- Family of criteria, $C = \{c_1, c_2, \ldots, c_n\}$
- Criteria evaluate the degree of satisfaction for an alternative concerning to some property: $c_i(a_j) = u_{ji} \epsilon S$
- Next element necessary to OWA calculation is a weights vector with dimension n, this operator permits to define aggregation policies. This component satisfies:

  $$\sum_{j=1}^{n} w_j = 1 \;\; w_j \, \epsilon \, [0, 1]$$

  Finally, a weighted average is calculated after sorting the values decreasingly ($u_{\sigma(i)} > u_{\sigma(i+1)}$)

  $$OWA(u_1, u_2, \ldots, u_n) = \sum_{i=1}^{n} \; w_i u_{\sigma(i)}$$

Notice that there is not an association between w_i and $u_{((i))}$. Currently, this weight element is associated with a particular position. As mentioned previously, OWA is an average operator, accordingly, it has the following properties:

- Idempotency: $H(u, u, u, \ldots, u) = u$
- Monotonicity: $u_j > u_j \rightarrow H(u_1, \ldots, u_j, \ldots, u_p) \geq H(u_1, \ldots, u_j, \ldots, u_p)$
- Neutrality: $H(u_1, u_2, u_3, \ldots, u_p) = H(u_{(\sigma(1))}, u_{(\sigma(2))}, u_{(\sigma(3))}, \ldots, u_{(\sigma(p))})$
- Boundaries: $\bigwedge_{j=1}^{p} u_j \leq H(u_1, u_2, u_3, \ldots, u_p) \leq \bigvee_{j=1}^{p} u_j$
- Associativity: $H(u_1, u_2, u_3) = H(H(u_1, u_2), u_3)$
- Decomposability:$H(u_1, u_2) = u^{'} \rightarrow H(u_1, u_2, u_3, \ldots, u_p) = H(u^{'}, u^{'}, u_3, .., u_p)$

An OWA calculation example can be:

1. Define an alternative set
$$A = (a)$$
2. Define a criterion set
$$C = (u_1, u_2, u_3, u_4)$$
3. Create the value associated vector to each criterion for the alternative:
$$C(a) = (0.5, 0.3, 0.2, 0.8)$$
4. Define weights vector with the same dimension of criteria:
$$W = (0.3, 0.1, 0.2, 0.4)$$
5. Sort $C(a)$ in descending order:
$$C(a) = (0.8, 0.5, 0.3, 0.2)$$
6. Finally, use OWA weights to aggregate these re-ordered arguments:
$$OWA(C) = 0.8 * 0.3 + 0.5 * 0.1 + 0.3 * 0.2 + 0.2 * 0.4 = 0.43$$

Weight vector properties

Given the set of weights W that the decision making specifies for the OWA operator, the character of the aggregation policy can be formalised using different measures. The most used ones are presented below Yager [1996]:

- Orness measures the degree of replaceability between the criteria. High orness means that the decision making is satisfied if only a few of the criteria are giving high utility values. On the contrary, a low orness (or high andness) means that the decision making will be only satisfied if all of the criteria are simultaneously fulfilled with high scores.
$$\alpha(W) = \sum_{i=1}^{n} w_i(\frac{n-j}{n-1}), with\ \alpha\ \epsilon\ [0, 1]$$
Outstanding cases:

  - When $\alpha \cong 1$, high orness (disjunctive).
  - When $\alpha \cong 0$, high andness (conjunctive).
  - When $\alpha \cong 0.5$, neutral averaging.

- The balance is another measure to evaluate the same "behaviour". When calculating the minimum, we are performing a conjunctive policy (simultaneity, andness) and when calculating the maximum, we are performing a disjunctive policy (repleaceability or orness).

$$Bal(W) = \sum_{i=1}^{n} w_i(\frac{n+1-2j}{n-1})$$

  Outstanding cases:

  - $Bal(W) = -1$, minimum operator.
  - $Bal(W) = 1$, maximum operator.
  - $Bal(W) = 0$, arithmetic averaging.

- The entropy measures how many criteria are used to obtain the final result.

$$H(W) = \sum_{i=1}^{n} w_i \; ln(w_i)$$

  Outstanding cases:

  - When $w_j = \frac{1}{n}$, high entropy (all values used).
  - When $w_q = 1$ and $w_j = 0$ for all $j \neq q$,then $H(W) = 0$, minimal information used.

- The divergence is evaluating whether the aggregation is concentrated on the extreme values or it is smoothly considering all the utility values.

$$Div(W) = \sum_{i=1}^{n} w_i(\frac{n-j}{n-1} - \alpha(W))^2$$

  Outstanding cases:

  - For the optimistic and pessimistic case, $Div(W) = 0$.
  - If $w_j = 1$ for some j,then $Div(W) = 0$.

Weights calculation based on Orness value

The OWA weight vector can be calculated from its quantifiers. The quantifier is used to generate an OWA weighting vector W of dimension n. This weighting vector is then used in an OWA aggregation to determine the overall evaluation for each alternative Yager [1996].
Thus the process used in quantifier guided aggregation is as follows:

- Use Q to generate a set of OWA weights, $w_1, w_2, \ldots, w_n$.
- For each alternative x in X calculate the overall evaluation

We can consider a quantifier function that depends on the Orness measure to calculate the weights. In this case the weights are generated as:

$$w_i = Q(\frac{i}{n}) - Q(\frac{i-1}{n}) \quad for\ i = 1...n$$

If we consider the quantifier

$$Q(r) = r^{\alpha} \quad \alpha \geq 0$$

Where α is

$$\alpha = \frac{1}{orness - 1}$$

The non-decreasing nature of Q it follows that $w_i > 0$. Furthermore, from the regularity of Q, $Q(1) = 1$ and $Q(0) = 0$, it follows that $\sum_i w_i = 1$. Thus we see that the weights generated are an acceptable class of OWA weights.

1.1.2 ULOWA (Unbalanced Linguistic Ordered Weighted Average)

In this section, we will concentrate on the aggregation of linguistic terms, in an extension of the OWA operator for a set of different linguistic labels Isern et al. [2017].
In this operator, the values that are aggregated are linguistic terms, each of them associated to a certain meaning (i.e. semantics). The meaning of the term is represented by a mapping into a numerical scale, known as Domain of Reference. The linguistic terms are defined in terms of Fuzzy Sets. A fuzzy set is a generalization of a Boolean Set. It allows a graduated degree of membership between 0 and 1.
The semantics of each linguistic label is given by a trapezoidal or triangular membership function $\mu : X \to [0, 1]$, that is represented with a tuple $P = (p1, p2, p3, p4)$, where $p1p2p3p4$ are the points in the reference domain X which define the trapezoid (Figure 1.1).

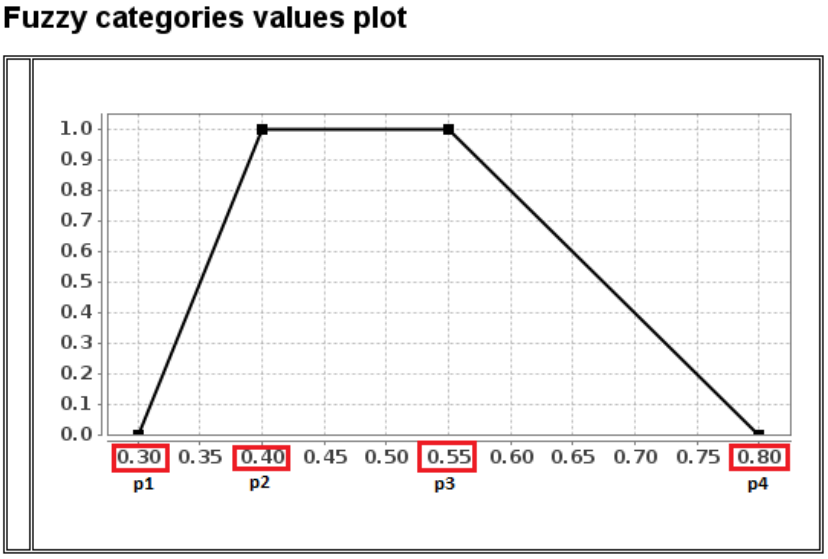

Figure 1.1: Fuzzy number trapezoid

Some special cases can be defined:

- $p1 = p2$ and $p3 = p4$ then P corresponds to a crisp interval.
- $p2 = p3$ the fuzzy set P is triangular, otherwise it is a trapezoid.
- $p1 = p2 = p3 = p4$, then P is called a crisp real number

The set of possible linguistic labels must be defined in advance. All the labels use the same domain of reference and they satisfy the following property:

$$\sum_{j=1}^{T} \mu_j(x_i) = 1 \ \ \forall \, x_i$$

ULOWA operator is an extension of LOWA operator that is designed to deal with unbalanced linguistic terms. These two approximations are adaptations of the OWA operator for dealing with linguistic labels. They operate directly on the labels that are aggregated in pairs by using convex combination operation. The following structure corresponds to ULOWA operator:

If $A = a_1, \ldots, a_m$ is the set of labels to aggregate, the ULOWA operator is defined as:

$$ULOWA(a_1, ..., a_m) = W * B^T = C^m\{\beta_h, b_h, h = 2, ..., m\} =$$
$$b_1 \otimes b_1 \oplus (1 - w_1) \otimes C^{m-1}\{\beta_h, b_h, h = 2, ..., m\}$$

Where $W = (w_1, \ldots, w_m)$ is an m-dimensional weighting vector so that $w_i \, \epsilon \, [0, 1]$ and $\sum w_i = 1$; $h = 2, \ldots, m$ and B is the associated ordered label vector (each element $b_i \, \epsilon \, B$ is the i-th largest label in A).We have to take into account that ULOWA operates on fuzzy sets; that is a membership functions of the terms. So, when $m = 2$, the convex combination of the two terms $b_1 = s_j$ and $b_2 = s_i$ is calculating using membership functions:

$$C^2\{w_i, b_i, i = 1, 2\} = w_1 \otimes s_j \oplus (1 - w_1) \otimes s_i = s_k; \ \textit{such that}$$
$$k = \underset{i \leq p \leq j}{\operatorname{argmax}}\{Sim(S_p, \delta)\}$$

In this expression, δ is a crisp number defined as $\delta = (x_k, x_k, x_k, x_k)$ with $x_k = x^*_{s_i} + w_1(x^*_{s_j} - x^*_{s_i})$, where $x^*_{s_i}$is the x-component of the centre of gravity (COG) of the fuzzy set associated to the label $s_i = (p_1, p_2, p_3, p_4)$.

We can define the centre of gravity (COG) of a trapezoidal fuzzy set A as $COG(A) = (x^*_A, y^*_A)$, which can be calculated as follows:

$$COG(s_A) = \begin{cases} y^*_A = \begin{cases} \frac{1}{6}\,(\frac{p_3 - p_2}{p_4 - p_1} + 2) & if \ p_1 \neq p_4 \\ \frac{1}{2} & if \ p_1 = p_4 \end{cases} \\ x^*_A = \frac{y^*_A(p_3 + p_2) + (p_4 + p_1)(1 - y^*_A)}{2} \end{cases}$$

In this way, the aggregation result s_k is the linguistic term comprised between s_j and s_i with the greatest similarity with the intermediate point δ. The similarity between two fuzzy sets is as follows:

$$Sim(P,Q) = \sqrt[4]{\prod_{i=1}^{4}(2 - |p_i - q_i|)} - 1$$

After explaining ULOWA methodology is important to understand its properties, there are explained below:

- Increasing monotonous: with respect to the argument values. Let $A = [a_1, \ldots, a_m]$ and $B = [b_1, \ldots, b_m]$ be two ordered argument vector such that $\forall j, a_j \geq b_j$ then: $ULOWA(A) \geq ULOWA(B)$.
- Commutativity: $ULOWA(a_1, \ldots, a_m) = ULOWA(a_1^{'}, \ldots, a_m^{'})$ where $(a_1^{'}, \ldots, a_m^{'})$ is any permutation of the elements in $(a_1, \ldots, a_m)$.
- Idempotency: in the sense that if $\forall j, a_j = a$, then $ULOWA(a_1, \ldots, a_m) = a$.
- Boundered: for any weighting vector W and ordered labels $A = [a_1, \ldots, a_m]$ $(a_1 \geq a_2 \geq ... \geq a_m)$, then: $min(a_1, \ldots, a_m) \leq ULOWA(a_1, \ldots, a_m) \leq max(a_1, \ldots, a_m)$.

Characterization of fuzzy sets

Unbalanced terms in ULOWA operator permit the definition of linguistic variables with different granularity and distribution.

The difference on the certainty of the terms should be taken into account during the aggregation process, as each label is providing a different amount of information about the evaluated alternative. In fact, if we consider a triangular and trapezoidal fuzzy set, they can be associated to the labels, then the uncertainty of the labels is not only related to their support intervals in the reference domain but also to their kernel.

In Isern et al. [2017], authors propose using a measure of the uncertainty of the linguistic labels as the order-inducing criterion for the aggregation. Thus, the arguments will be ordered by decreasing uncertainty. In this way, the contribution of precise labels is prioritized while the effect of uncertain labels is reduced.

two types of uncertainty in fuzzy sets are recognized:

- Specificity: measures the degree of truth of the sentence "containing just one element". This uncertainty type is related to the measurement of imprecision, which is based on the cardinality of the set, which can be calculated as follows:

$$Sp(A) = 1 - \frac{\text{area under } A}{b - a}$$

- Fuzziness: measures the difference from a crisp set, and the vagueness of the set as a result of having imprecise boundaries, which can be calculated as follows:

$$Fz(A) = 1 - \frac{1}{b-a}\int_a^b |2A(x) - 1|$$

1.2 Diviz framework in detail

Before explaining how the new modules for the previous operators have been designed and implemented, we need to know the Diviz framework. In particular, the way of working with Diviz modules, the general structure for implementation, the requirements of documentation and licensing.

In Diviz, the main functionality is based on modules that receive mandatory and optional inputs, then apply the calculations associated with this decision-making method and finally generating output files. As we analysed previously in Diviz related part in chapter 2, it is possible to concatenate modules if they share the same structure as output parameter for the first module, and as input parameter for the second one, thus strategy allow the creation of more complex algorithms.

The inputs and outputs are defined in single, or composed, structures defined in different files. This data structure follows the XMCDA standardizations. That input can be defined by two or more structures, it is possible, but not very frequent.

The first step for a person who wants to start to develop a new MCDA algorithm must do is to download and install the latest version of Diviz program, which can be found into the Diviz website Consortium [2009a]. In this website, the user can find many different installers with some basic information to complete the installation successfully. Besides, in some operating systems it is necessary to install the Java 8 version to run Diviz environment, but in the most common operating systems this installation will be done automatically when a user installs the program.

To start the execution of any module or set of modules, it is necessary to create a new workflow. To open the workflow menu and click on new. Choose a name for your workflow, as it is represented below in (Figure 1.2):

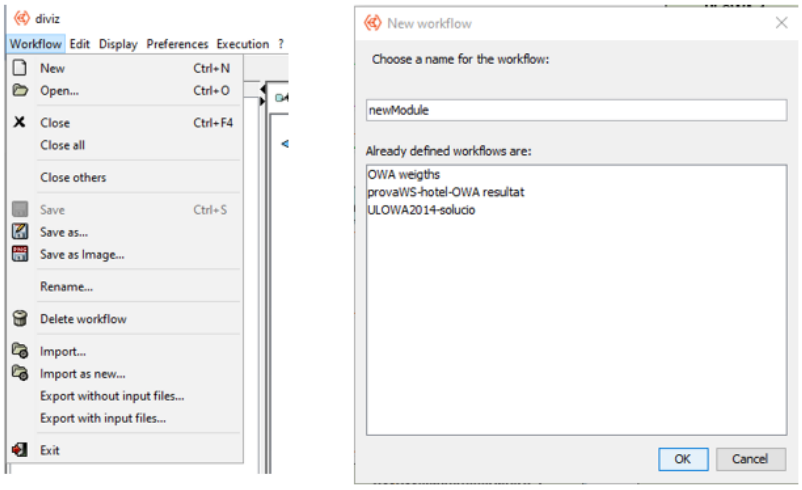

Figure 1.2: Diviz creation workflow menus

With the new workflow created, the central panel is a blank canvas where the user can drag the modules and the input files necessary to complete the execution.

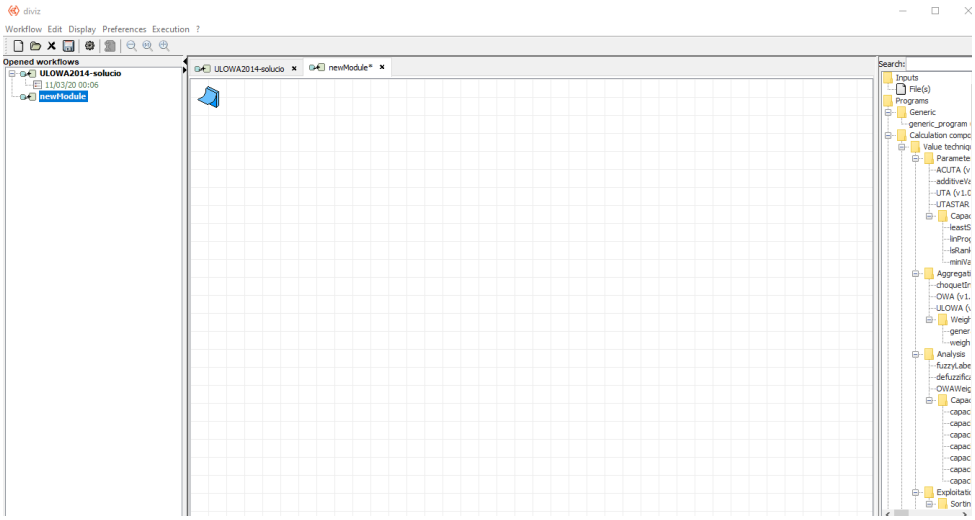

Figure 1.3: Blank canvas

Now it is time to define our example, in the new workflow I will drag and drop two different modules that will be useful to see the different outputs that Diviz modules can generate. These modules are plotFuzzyCategoriesValues OWAWeightsDescriptors (under the category Data Visualisation and calculation components respectively). Simply select them, drag and drop them to the central panel.

To run these two modules, three input files are required. These inputs must be data files which respect the XMCDA standard:

- criteria.xml (contains a list of criterion).
- wheights.xml (contains a list of alternatives values).
- FuzzyNumbers.xml (contains a list of fuzzy numbers).

Important observation: some of these input files structures are deprecated in the new version implemented in this project, but they are good examples to illustrate different cases.

To incorporate the necessary module inputs the user needs to drag and drop a File(s) element from the right panel into the central panel. A dialog window will be open and will ask you to choose an input file. This procedure is required as many times as the input files are required.

Coming up next, connect the input files to the corresponding pin inside the modules in the Canvas, as it is represented below in (Figure 1.4):

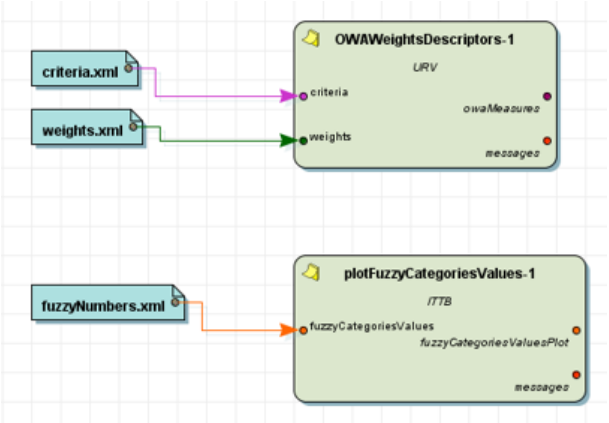

Figure 1.4: Module graphical connections

The last step before ran our example are not applicable to all the modules, but some of them have different options or settings that can be changed before the execution. In our example we can find that case in OWAWeightsDescriptors, in this module, we, as users, can choose between the four different parameters. To make changes to these options inside the module, we right-click on the module and check the "Settings" option, then a menu will open with different changeable parameters, in our example we will change the OWA descriptor to, for example, Orness (Figure 1.5).

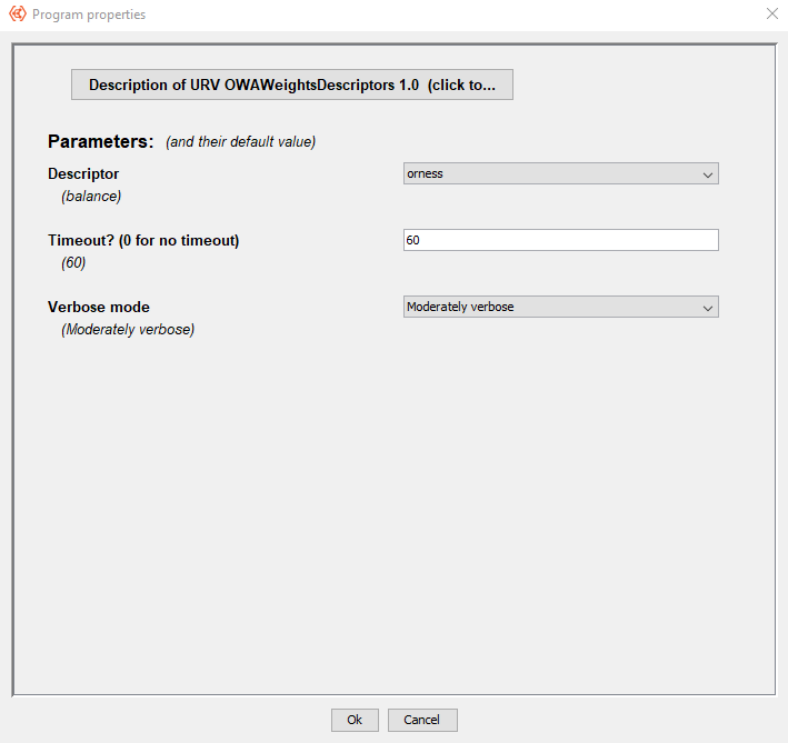

Figure 1.5: Options menu of Diviz standard module

Finally, we can run the example with the engine icon in the menu, then an execution workflow will be generated below and linked with our editable workflow. In this new element, we will find the same modules and input files with the result files generated by the execution, but it is not

editable.

These output files are XMCDA structures and can be used as inputs of other modules if we concatenate the existing modules with the new ones plugging the necessary inputs and output pins of these modules. Diviz has a viewer of xml files that recognizes these XMCDA structures. If we go to our example, one module has graphical output (Figure 1.6), otherwise, the second module has a data table output.

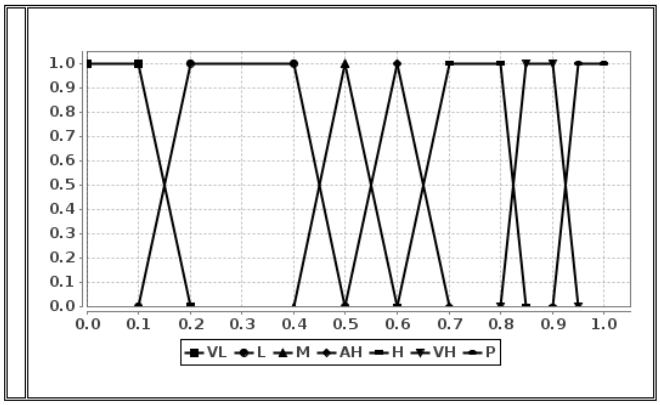

Figure 1.6: Graphical representation of a fuzzy Numbers example

And the second one (Figure 1.7),

Criterion value

-0.6666667

Figure 1.7: numerical representation of a criterion example

1.3 Framework required to develop a module in XMCDA3

To design and develop the new version of the different OWA and ULOWA modules, first I had to know the structure of the existing modules, as well as the way of implementing the new ones in the new version. In the diviz website, I found that currently the modules are implemented basically in Python and R. I have chosen R because is a language focused on mathematical calculations, which can be done with great precision.

In order to be compatible with the existing work in Diviz, and also to easily exchange information with the Diviz managers, I followed the guidelines they provide in the project.

To develop the new implementation in R, I use a Linux operating system machine, in fact, I used a completely new Ubuntu image of Virtual Machine created with the program Virtual Box. The Linux option meant a manual installation of XMCDA-R and rJava. To do that installation, I used the instructions and packages available on XMCDA-R Github Meyer [2019] authored by Patrick Mayer in 2019. In this website you can find the following Linux commands (Figure 1.8)

required to carry out the installation of all the necessary resources to run the new modules' code before put it on the Diviz.

```
In order to build the package, you have to download the jar file of the latest
version of the XMCDA java implementation. For example here :

https://gitlab.com/XMCDA-library/XMCDA-java/builds/5024404/artifacts/file/dist/XMCDA-java-2016.10-1.jar

Create a java directory in the inst directory, and copy the jar there :

mkdir -p /path/to/XMCDA-R/inst/java
cp /path/to/XMCDA-java-2016.10-1.jar /path/to/XMCDA-R/inst/java/

Then build the package with the classical R command :

R CMD build /path/to/XMCDA-R/

To install the package, run the following command (on the generated .tar.gz):

cd ..
R CMD INSTALL XMCDA3_x.x.x.tar.gz

You will have to install the rJava package. On linux machines, start with

R CMD javareconf -e
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_LD_LIBRARY_PATH

Then install the rJava package from R with the

install.packages("rJava")
```

Figure 1.8: Necessary orders to install Linux XMCDA

Finally, to develop all the source code of each module, I based on the documentation of Weighted-Sum Bigaret [2017], this Sébastien Bigaret authored example is used as a reference to new implementations in XMCDA3 and it is written in R language. In the next subsection, I will explain this program's structure with more details. To write this source code I used the RStudio program, an integrated development environment for R and Python, with a console, syntax-highlighting editor that supports direct code execution, and tools for plotting, history, debugging and workspace management, which are available to download freely in the official website.

1.3.1 General structure of a Diviz module in XMCDA3

As we mentioned previously, all new implemented modules are based on the reference program structure of the Weighted-Sum module that you can find on the Github portal Bigaret [2017], authored by Sébastien Bigaret. We can decompound all modules in:

- Source code folder: into this folder there are all source code files, these elements are the core of all module, the followed reference structure distributes all functions in five different R files well differentiated between them:
  - utils.R: this file is responsible of loading all XMCDA structures from the input files in the input directory with the **loadXMCDAv3()** function.
  - inputHandler.R: it is the file containing all functions related to transforming the XMCDA standard structures to R data structures and the input error control, the

main function to do it is **checkAndExtractInputs()** function, but can contain local called sub-functions if it is necessary to develop a well-structured code.

- methodCalculation.R: into this file takes place the module real purpose calculations. This file's main function returns the is resulting in R data structures. It can contain local called sub-functions if it is necessary to develop a well-structured code.
- outputHandler.R: it is the file containing all functions related to transform the R data structures to XMCDA standard structures. The main function to do it is **convert()** function, but it can contain local called sub-functions if it is necessary to develop a well-structured code.
- ModuleNameXMCDAv3.R (the *"main"* file): this is the responsible for guiding all the execution of the particular functions. When the module is running the main script (explained below) execute XMCDAv3 file passing by parameter the input and output directories, at the same time this module loads all the others files that are used to make subtasks. Between each subtask, there is an error control that stops the execution if an error is detected. After that, this function loads the XMCDA structures from the files in the input directory with the function **loadXMCDAv3()**.

```
loadXMCDAv3(xmcdaData, inDirectory, weightsFile, mandatory = TRUE, xmcdaMessages, "criteria")
loadXMCDAv3(xmcdaData, inDirectory, weightsFile, mandatory = TRUE, xmcdaMessages, "criteriaValues")
```

Figure 1.9: Example of the call to the load data function

Then, if the input is loaded correctly, it is time to translate these input XMCDA structures to R data structures to make the calculation and data error control, this task is made by **checkAndExtractInputs()** function.

```
checkAndExtractInputs(xmcdaData, programExecutionResult)
```

Figure 1.10: Example of the call to the input checker function

In this point, the module knows that all input which is necessary to make the calculation is correct and it is time to make it with the function **methodcalculation()**. Now, with the calculation done it is time to transform the data obtained in R structure to XMCDA formal structure, this work corresponds to **convert()** function.

```
xResults = convert(results, xmcdaMessages)
```

Figure 1.11: Example of the call to the output preparation function

Finally, the last remaining job is to write this output XMCDA structure to a file and write the process execution message obtained to a file. **writeXMCDA()** and **putProgramExecutionResult()** helps us to do that job.

```
writeXMCDA(xResults[[i]], outputFilename, xmcda_v3_tag(names(xResults)[i]))
putProgramExecutionResult(xmcdaMessages, infos="")
```

Figure 1.12: Example of the call to the write data functions

This schema (Figure 1.13) developed in MagicDraw19 pretends to make more understandable this source code structure.

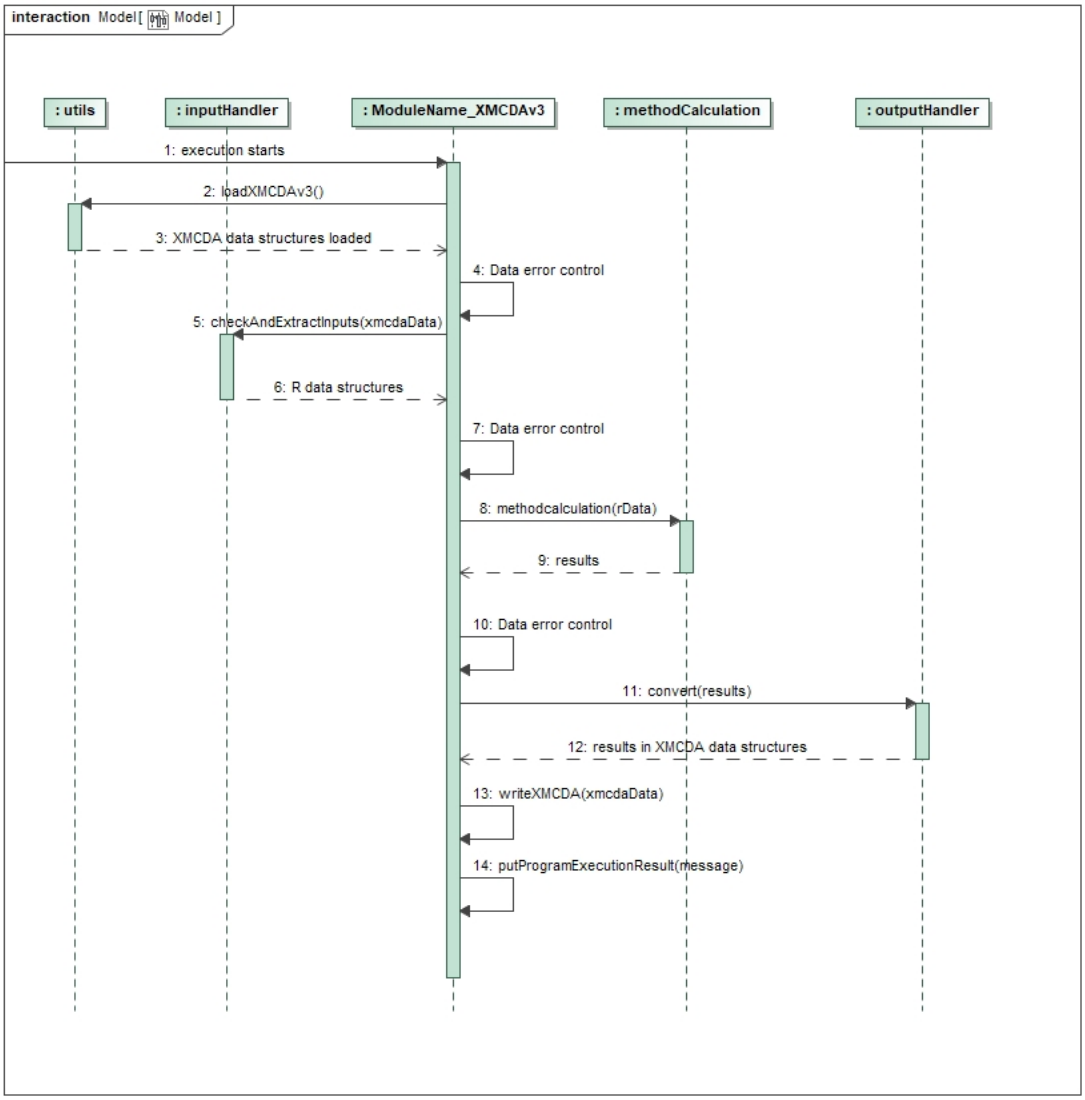

Figure 1.13: Sequence diagram of standard Diviz module functionality

- Test folder: it is the location where all tests are done to validate the module correctness. For each test case, it will be two different folders, one of them for input files and a Readme to define the purpose of the test case. This first folder acts as input directory and is named as **inX.v3**. The second folder is the responsible for storing the output files generated in the test cases, for the module, acts as output directory, the name for this folder is **outX.v3**, where X is the test case number, as it is illustrated in the following capture (Figure 1.15). That naming structures are a convention to make easier the execution of all test cases and

recognition to a tester.
In the analysis of each module individually we will see the complete description of the particular test cases for all modules.

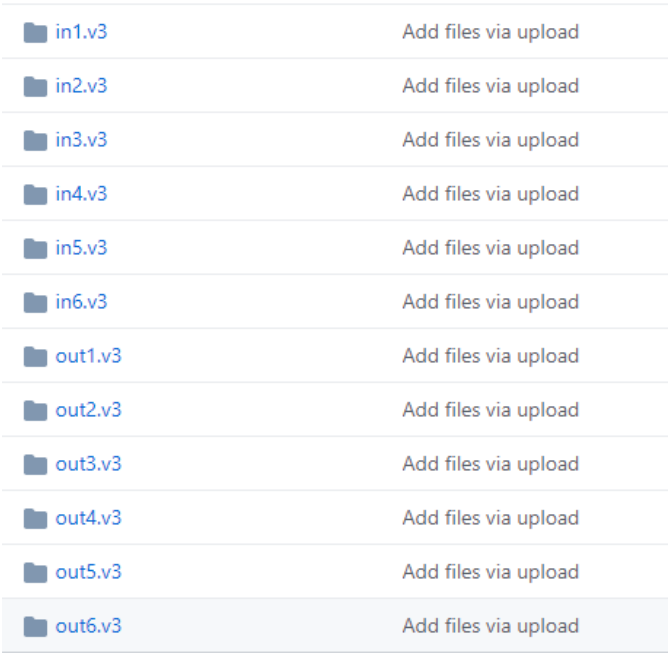

Figure 1.14: Example of test folder structure

- description-wsDD.xml: this file is an xml containg all the relevant data about the module, input and output files. This information is used to build the list of available modules website with information of each module implemented, you can find this available modules list into this reference link Consortium [2009b].
- README file: it is a short guide on how to run all test tests.
- LICENCE file: it is an informative file with the current license under which the code has been published, MIT License. If you want more information, there is a short subsection explaining this issue below.
- Module.sh: this is the main script to execute the module. It has the same name of module and needs the input and output directory as a parameter to realise its task.
- tests.sh: this is the responsible for calling the previous script as many times as tests we have into the tests folders, using the corresponding convention for names. This tests script does not write into any output directory. This script is the public one and guarantees the correctness of all tests done. Its purpose is to compare the current files in output directory with the output extracted by the execution, if there are equals, the test is passed successfully.
- tests2.sh: this is the responsible of calling the main module script as many times as tests we have into the tests folders, using the corresponding convention for names. This tests script is private to the author of the modules, and I used to generate the output files to the output directory to validate the implementation correctness.

```
#! /bin/bash

CMD="./OWA.sh"

if [ $# != 0 ]; then
  echo "Usage: ${0}" >&2
  exit 1
fi

version=3

NB_TESTS=$(find tests -maxdepth 1 -type d -regex '.*/in[0-9]*\.v'"$
{version}"'$' | wc -l)

for i in $(seq 1 ${NB_TESTS}); do
    IN="tests/in${i}.v${version}"
    OUT="tests/out${i}.v${version}"
    ${CMD} "${IN}" "${OUT}"
done
```

Figure 1.15: Script used by execute locally all test cases

1.3.2 Documentation file

This is an important part of the implementation; this xml file has to be published on the website of Diviz, into the available modules section Consortium [2009b], the purpose of this file is to allow a complete and quick comprehension about the module functionality for all community user, explaining the current version, authors, possible contacts, description about the module functionality as well as all relevant aspects of the input and output files for example:

- id: the identifier of the parameter number, the standard identifier structure used are "inputX" or "outputX" where X was the parameter number.
- name: the name of the stablished for the internal parameter file name.
- displayName: the name used into the Diviz workbench. This is the name the final user will see.
- isOptional: only required for input files, boolean attribute which indicates if this input parameter is mandatory (0), or is optional (1).
- description: brief description of the parameter, where its purpose as well as useful information for the user are usually described.
- xmcda Tag: parameter used to identify the XMCDA structure used to store data.
- xmcda structure: Schematic representation of the XMCDA structure used by the parameter, it is the same as that tagged in the previous attribute. Non-necessary for output files.

This documentation will be available on the official website, but currently you can only read this xml files in my personal Github Buxons [2020], where there is one documentation file named as **description-wsDD.xml** into each module folder. Anyway, I have attached a copy of all the documentation files, in the same format so that you could find it on my GitHub in Appendix A.

1.3.3 License file

Although all code generated in this project is open-source and it is free to download, it is necessary to distinguish the available actions that an external person is allowed to do, and responsibilities assumed or not by the author. We will use the MIT license.

MIT license is a permissive Free Software License Free Software License which means that it imposes very few limitations on reuse and, therefore, it has an excellent License Compatibility. The MIT license allows reuse of software within proprietary Software. On the other hand, this license is compatible with many copy-left licenses, such as the GNU General Public License.

As an additional and interesting issue, in 2015, MIT license, according to Black Duck Software and data from GitHub, became the most popular license above the GPL license variants.

```
MIT License

Copyright (c) 2020 Aida Valls, Miquel Buxons, Universitat Rovira i Virgili

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the
Software without restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

Figure 1.16: MIT License file example

1.4 Changes in data input structures from v2 to v4

As we have seen previously, each input and output data complies with the XMCDA standard. Ergo, each one is stored, generated and recognized because it fulfils this standard. More specifically, XMCDA provides a set of structures that allow the standardization of all the procedures and data typologies that are needed to develop and share MCDA modules.

When someone wants to contribute in including new modules to Diviz, he/she must follow the XMCDA language in the inputs and outputs. However, in this case, as our modules are the only ones that use weights for positions, linguistic terms and fuzzy sets, the managers of Diviz asked to the ITAKA group to collaborate in the definition of the format of the input and output files that use linguistic data.

In this project, I took the files from the previous version and I updated all XMCDA data structures of inputs and outputs files to version 4.0.0. XMCDA format. These are not substantial changes in all cases, but more or less all of the data files changed the format since the XMCDAv2

is the current version at the time of the first implementation. We can classify these adjustments in three categories.

- File header format changes: this is the most frequent change, in fact, it is necessary updating to all input and output files. The header is an important part of an XMCDA file and it allows to all elements in Diviz to recognise the version of the standard is used. As example, the current header format to XMCDAv4.0.0. is (Figure 1.17):

```
<?xml version="1.0" ?>
<xmcda xmlns="http://www.decision-deck.org/2020/XMCDA-4.0.0" xmlns:xsi="http://www.w3.org/2001/-
XMLSchema-instance" xsi:schemaLocation="http://www.decision-deck.org/2020/XMCDA-4.0.0 http://-
www.decision-deck.org/xmcda/_downloads/XMCDA-4.0.0.xsd">
```

Figure 1.17: Current header for files that run in XMCDAv4.0.0. structure

- Minor data structure changes: in this case, they are motivated by the change in the structure format used during the time elapsed until the new version. Usually, these changes the previous structure, so it is easy to interpret it.
  An example of this could be showed in the case of performance, illustrated below in (Figure 1.18), a subcategory of performance table, which previously could contain a set of tagged "value", without a label that included all of them in a subcategory, as if it happens in the new version in which said structure needs a label "values" that gathers all the "value" tags.

```
- <performanceTable>
  - <alternativePerformances>
      <alternativeID>Poznanzki</alternativeID>
    - <performance>
        <criterionID>location</criterionID>
      - <values>
        - <value>
            <real>0.0</real>
          </value>
        </values>
      </performance>
```

```
- <performanceTable>
  - <alternativePerformances>
      <alternativeID>Poznanzki</alternativeID>
    - <performance>
        <criterionID>location</criterionID>
      - <value>
          <real>0.0</real>
        </value>
      </performance>
```

Figure 1.18: Illustration of a minor change in parameter files

- Proposal of new file format: this is the case in which we were required to define a new structure for the files of OWA and ULOWA operators. These changes have been made in accordance with experience after the usage of the previous structures during these years. In addition, we have made a proposal that is compatible with the new simplified version of the XML language and looking for a new implementation that would be more useful and understandable. The new structures were proposed by me, agreed with my project tutor, Dra. Aïda Valls, and with the managers of Diviz, Dr. Bigaret and Dr. Meyer. We have defined the structure of 3 data structures:
  - Weights: previously, this input was structured in the XMCDA structure of Alternatives Values. This structure allows storing all weights values as a values list. It is a very easy to work with and understandable structure, but if we focus on the syntax of this solution, it said that the weights vector is a list of different alternatives and weights in OWA family are a vector of values that permits to evaluate all criteria of an alternative. To replace this structure and, at the same time, improve the performance, the new implementation work with weights vector structured in one structure

with an auxiliary function written in the software.
The weights values vector is represented using a criteriaSetsValues structure defined in XMCDA. In this structure, we are able to set a identifier and a mcdaConcept, all values are set using a values list where user only have to worry about the order that she/he wants, any identifier is required for any weights.

A representative scheme of that structure would be (Figure 1.19):

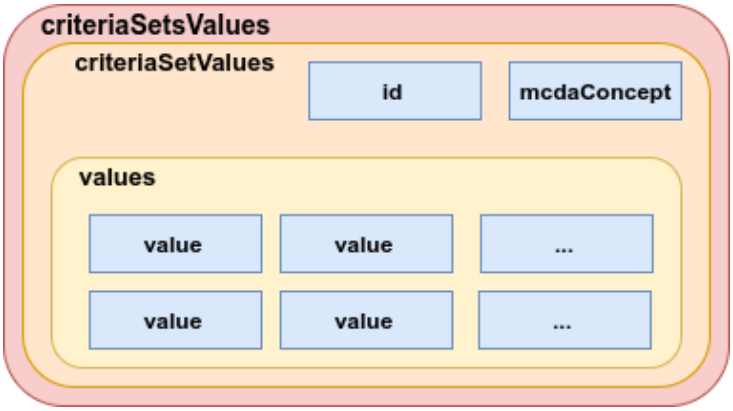

Figure 1.19: Weights file current structure

– Orness: this input structure was not necessary for the first version of OWA and ULOWA modules. It is used in this project because there is a new functionality into these modules that allows the Diviz user to choose which type of input weights can use. In Orness option, the user gives a program parameter XMCDA structure, in which there are two fixed parameters; an activation attribute which is a Boolean and a Orness real value. To obtain that weights vector based on Orness value using the methodology explained in subsection (1.1.1).These formulas depends on n, that is the number of weights that we want to generate $w = [w_1, w_2, \ldots, w_n]$ A representative scheme for store this Orness value would be (Figure 1.20):

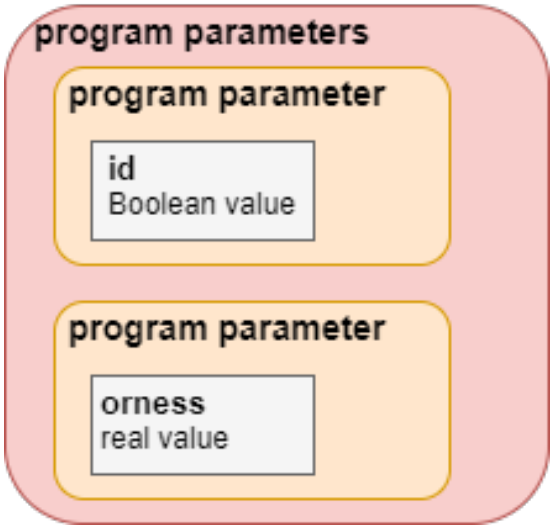

Figure 1.20: Orness file current structure

– Fuzzy Numbers: this is the structure that has changed the most, in fact, it changed 3 times during the implementation, the final version consists on a criteriaScales structure. In the past, all fuzzy numbers were structured as only one categoryValue in which there is a list of fuzzy numbers. This structure is a bit difficult to work with but it requires that complexity to correctly store all the possible information. The following schema represents the past fuzzy numbers old structure would be (Figure 1.21):

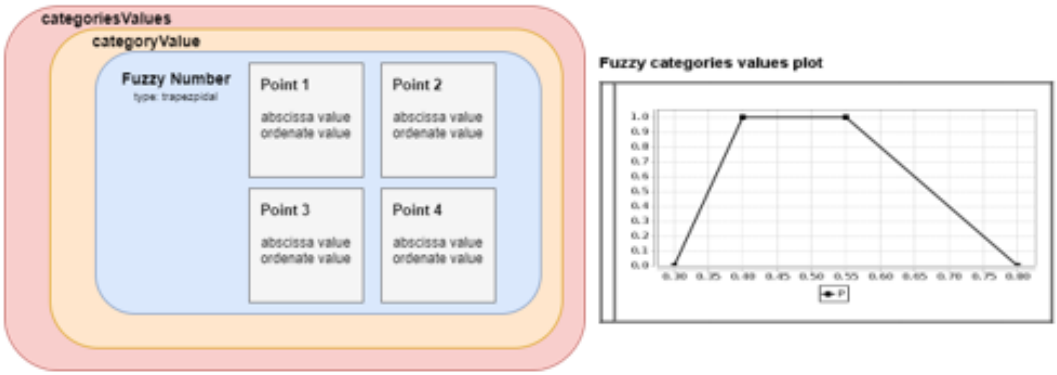

Figure 1.21: Fuzzy numbers file old structure

We can see that there is a big amount of information in a fuzzy number, but this structure, during the years, demonstrate not to be understandable sometimes and it is difficult to work with. After some meetings, we decide to standardize the fuzzy numbers structure, making it a little bit more complex but more accurate and precise. With this new structures users can develop fuzzy memberships using a qualitative scale domain, that allows to add a preference direction inside (not used in this modules). Inside this structure user can create a valued labels list where each valued label will be a fuzzy label, in which, user will be able to propose the fuzzy number function defined by segments that construct the label (e.g. 3 segments define a trapezoidal fuzzy label). The new schema for fuzzy numbers XMCDA representation (Figure 1.22):

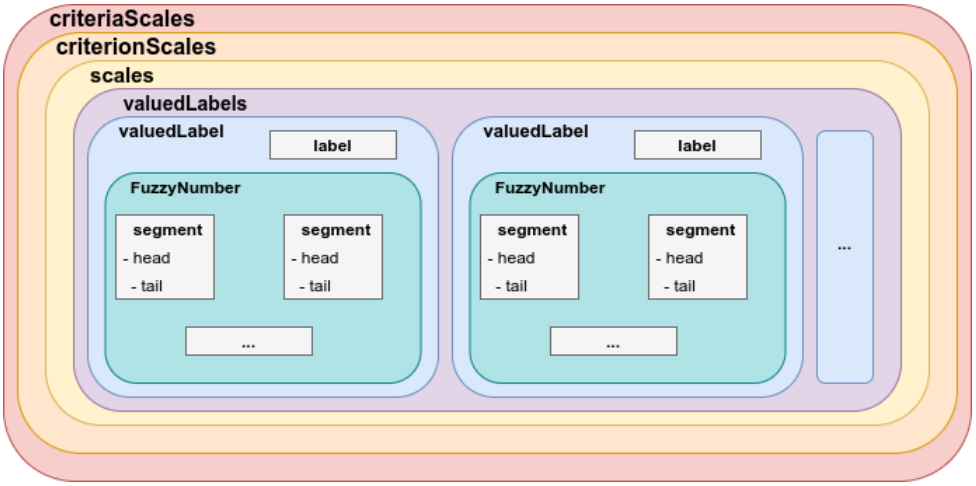

Figure 1.22: Fuzzy numbers file current structure

In the XMCDA version 4.0.0 a function is implemented to transform a csv file to a fuzzy membership, *csvToXMCA_criteriaScales()*. To sum up this subsection, (Table 1.1) indicates the changes and design decisions made into input XMCDA data structures for this project.

Design decisions table	**Header changes**	**Minor changes**	**Complete changes**
Alternatives	Yes	Yes	No
Criteria	Yes	No	No
Performance table	Yes	Yes	No
Weights	Yes	No	Yes
Fuzzy Numbers	Yes	No	Yes
Orness value	Yes	No	Yes

Table 1.1: Changes done in input files

1.5 XMCDA data structure used for this implementation

For all developed modules to Diviz, they receive parameters and generate outputs in XMCDA data structures format, the modules developed in this project there are not an exception. It is not necessary to use a structure for each type of input or output parameter, often, these parameters are structured on the same XMCDA options. This option will never cause errors if it is taken into account when developing the module. For this development the used XMCDA structures are nine in total, represented in the following (Table 1.2):

XMCDA structure	**Parameters using this structure**
alternatives	Alternatives.
alternativesValues	OWA & ULOWA results, Fuzziness, Specificity, COG, COM & Ordinal.
criteria	Criteria.
criteriaValues	Balance, Divergence, Entropy & Orness
performanceTable	Performance Table.
criteriaSetsValues	Weights.
criteriaScales	Fuzzy numbers.
programParameters	Orness as input.
programExecutionResults	Output messages.

Table 1.2: XMCDA3 structures used for each parameter

For all these XMCDA structures I have prepared the class diagram that explains their structural tree. The following class diagrams, internally, are organized so that the main structure is placed at the top and all its subclass are below it, and thus, concurrently.

1.5.1 alternatives

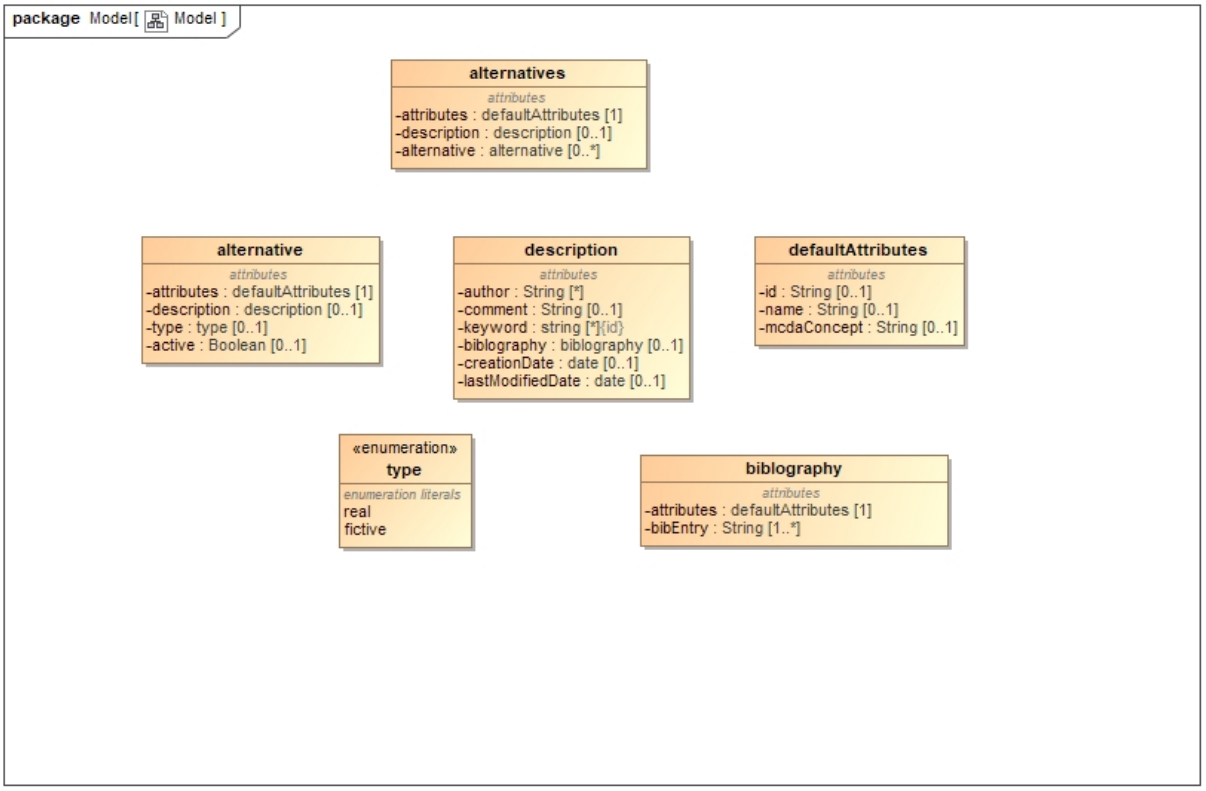

Figure 1.23: XMCDA3 alternatives structure

1.5.2 alternativesValues

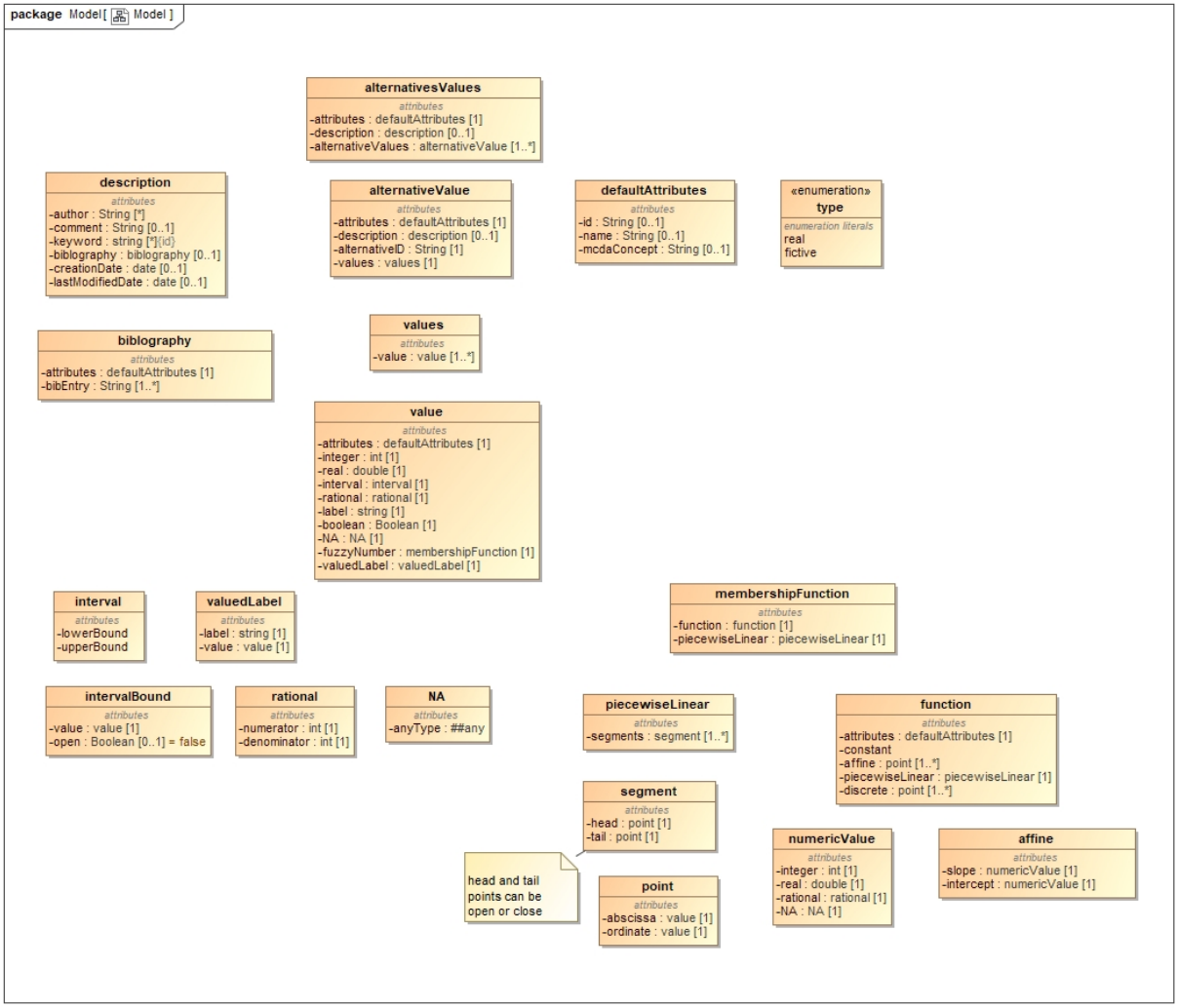

Figure 1.24: XMCDA3 alternativesValues structure

1.5.3 criteria

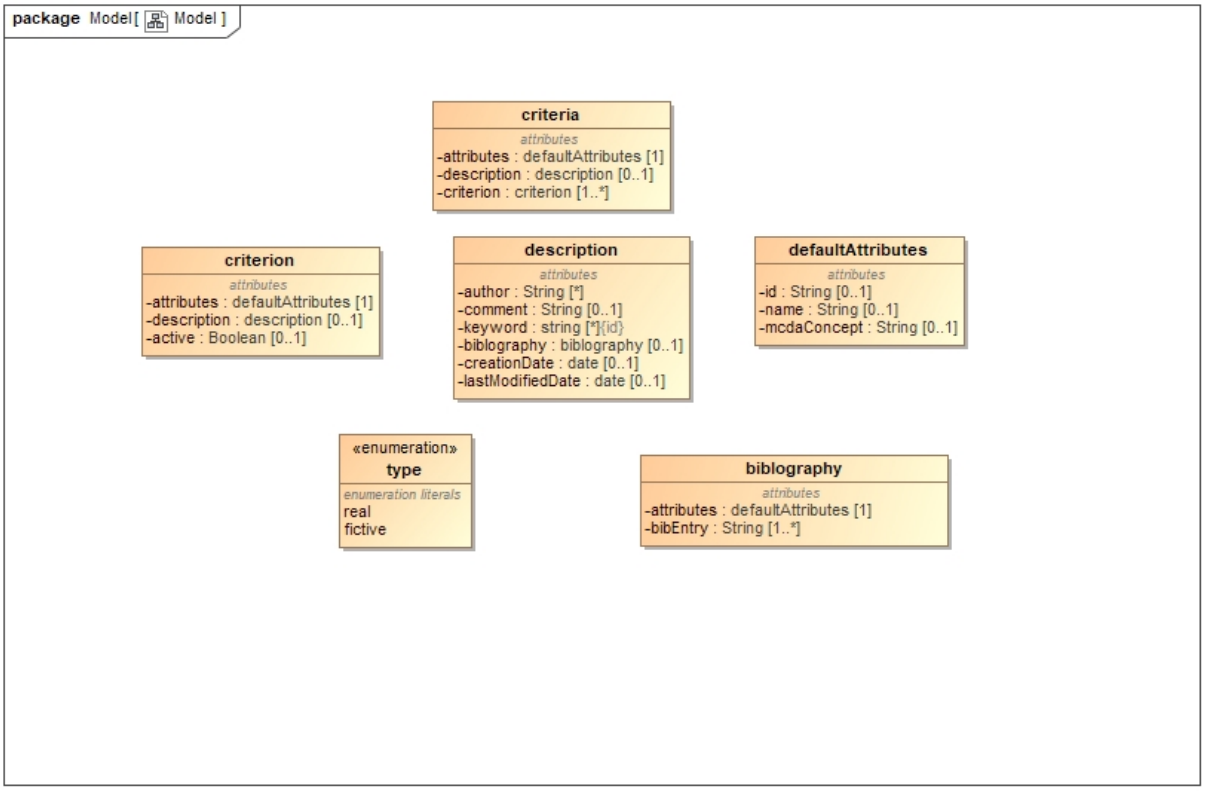

Figure 1.25: XMCDA3 criteria structure

1.5.4 criteriaValues

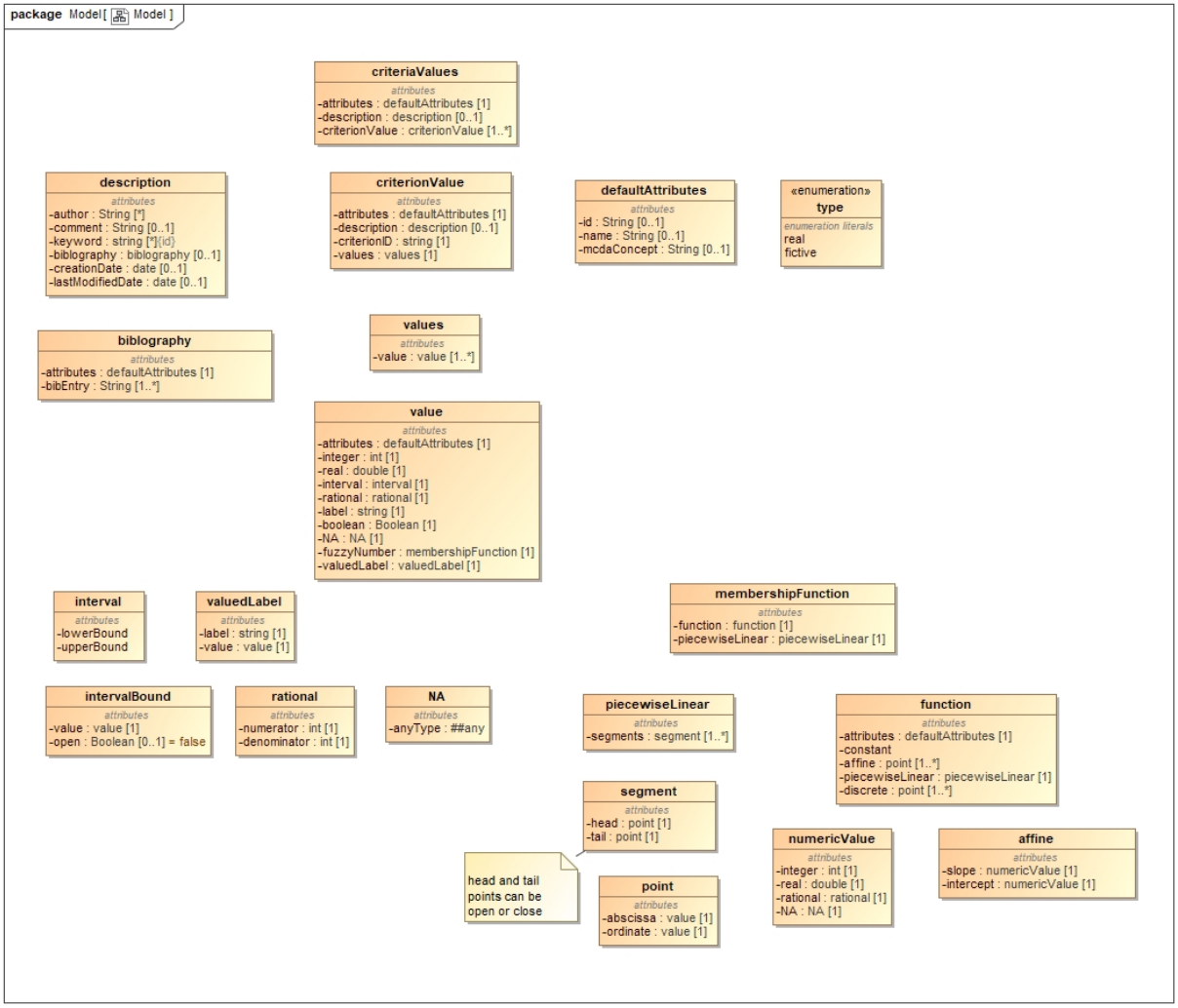

Figure 1.26: XMCDA3 criteriaValues structure

1.5.5 categories

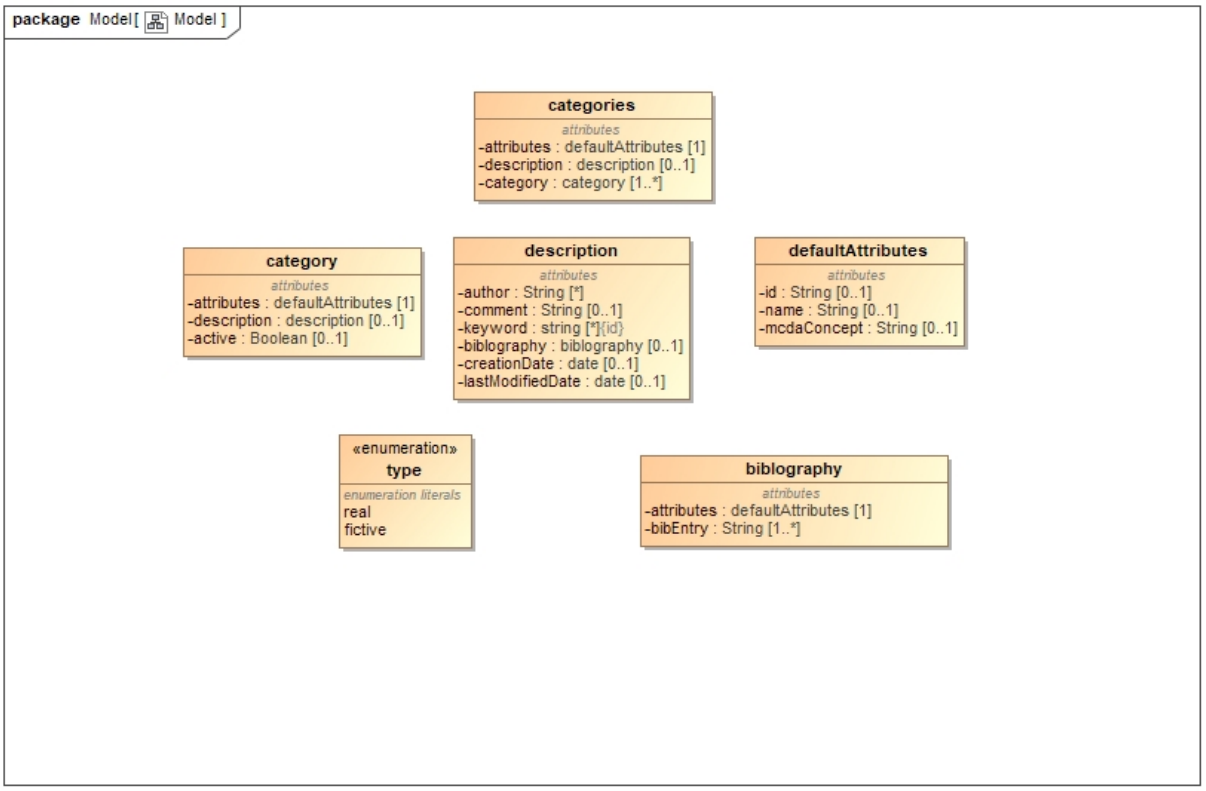

Figure 1.27: XMCDA3 categories structure

1.5.6 categoriesValues

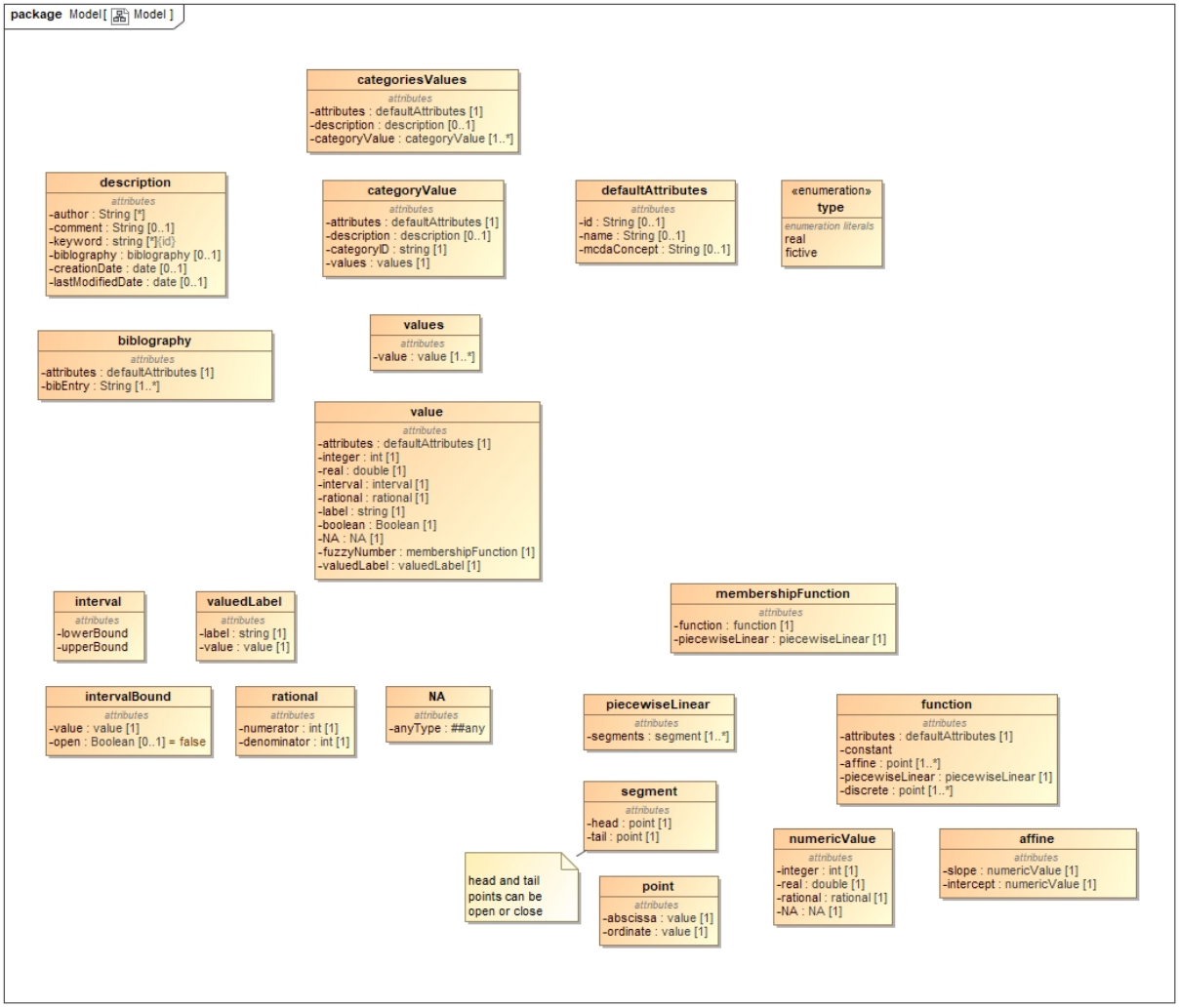

Figure 1.28: XMCDA3 categoriesValues structure

1.5.7 performanceTable

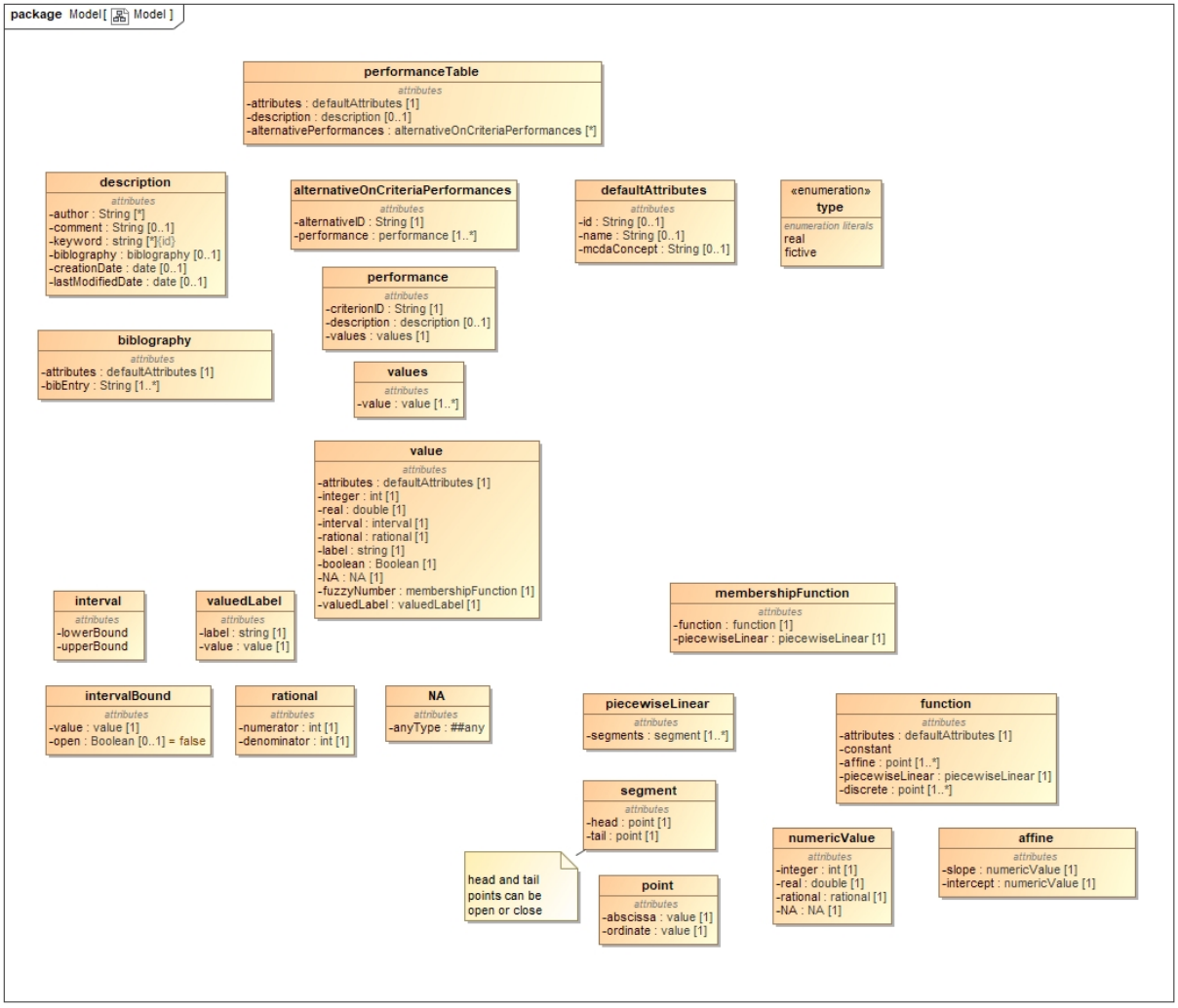

Figure 1.29: XMCDA3 performanceTable structure

1.5.8 programParameters

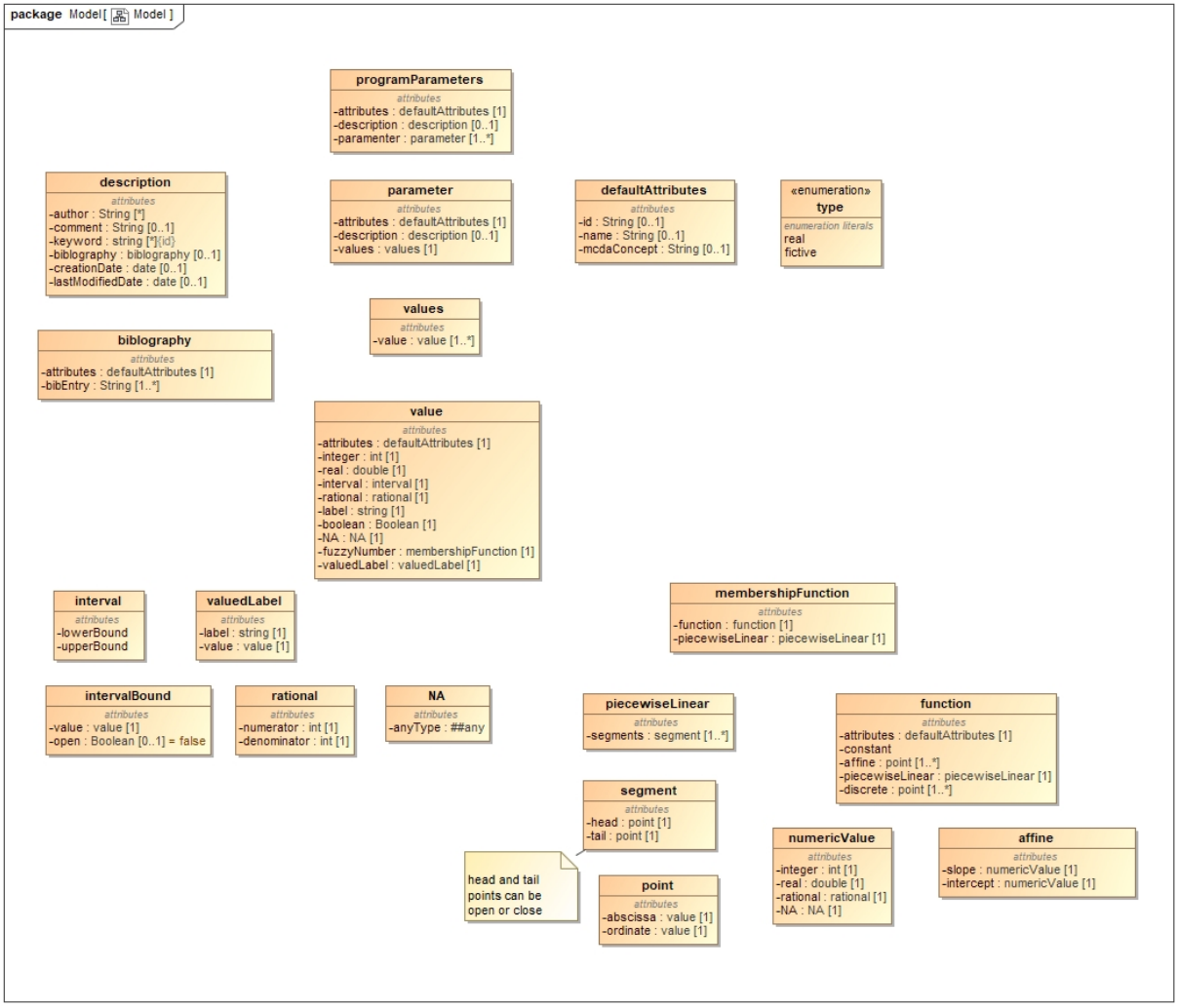

Figure 1.30: XMCDA3 programParameters structure

1.5.9 programExecutionResult

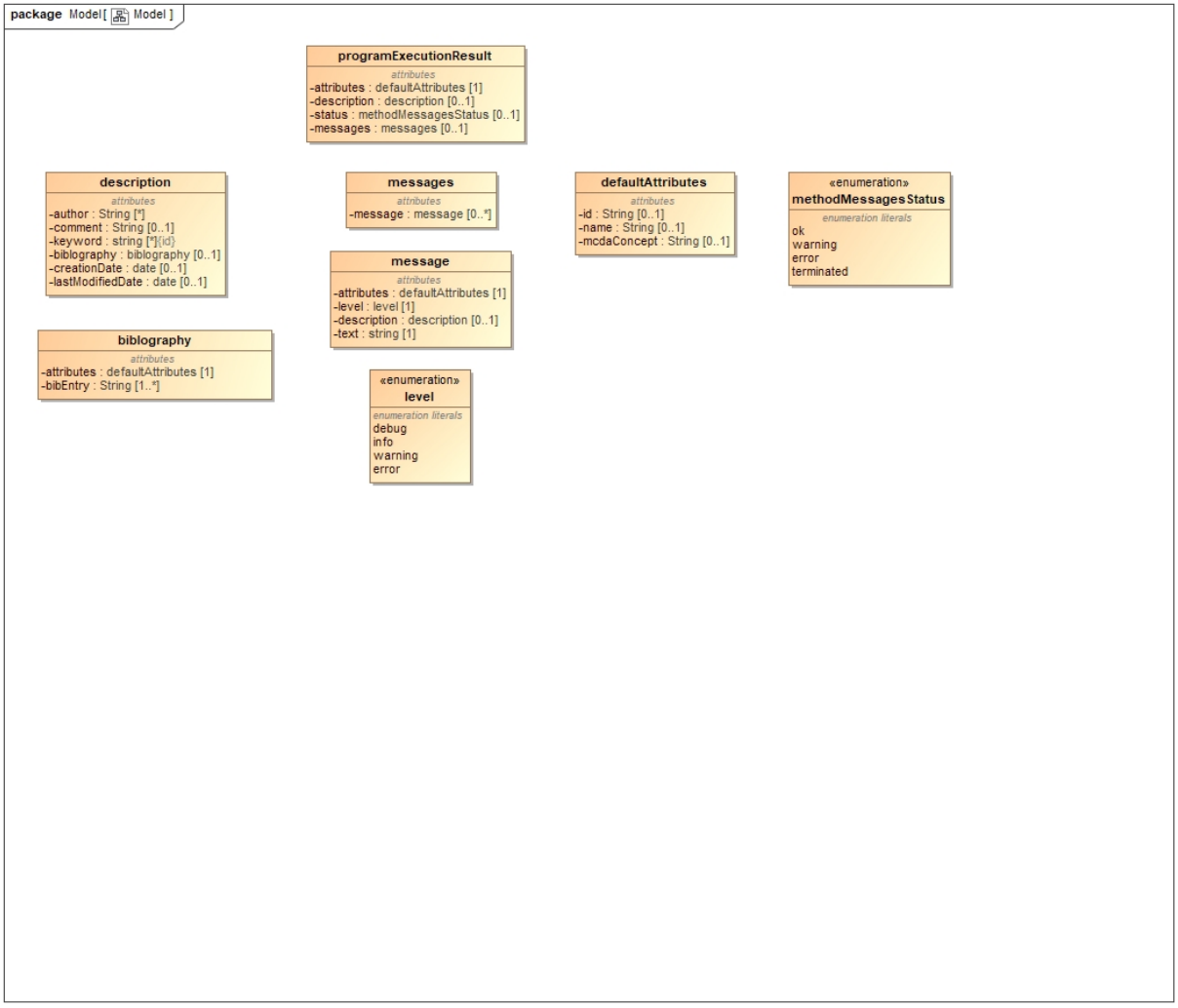

Figure 1.31: XMCDA3 programExecutionResult structure

1.6 Design and implementation of the new modules

In this section the 5 modules designed, developed and deployed are explained in detail. Each of the subsections follows the same structure, from the input/output data, to the operations, finishing with the use case tests and an example.

1.6.1 OWA Module

This module implements the OWA aggregation operator: The Ordered Weighted Averaging operators, commonly called OWA operators, provide a parameterized class of mean type aggregation operators.

OWA was defined by R.R. Yager in 1988. Here we provide an implementation of OWA operator. The main characteristic of this aggregation operator is the fact that the weights are associated with values instead of criteria. In this way, we can define different aggregation policies according to the importance that we associate with high and low performance scores.

Inputs and Outputs

To represent all input and output of this module I have done the following (Table 1.3) explaining the specific characteristics for each parameter: *The parameter "isOptional" is only applied on

I/O Parameters	id	name	displayName	isOptional	tag	structure
alternatives	input0	alternatives	alternatives	No	alternatives	4.2.12.1
criteria	input1	criteria	criteria	No	criteria	4.2.12.3
performanceTable	input3	performanceTable	performanceTable	No	performanceTable	4.2.12.5
weights	input2	weightsOWA	weights	Yes	criteriaSetsValues	-
orness input	input4	programParameters	orness	Yes	programParameters	4.2.12.8
OWA result	output0	alternativesValues	owaResults	-	alternativesValues	4.2.12.2
message	output1	messages	messages	-	programExecutionResults	4.2.12.9

Table 1.3: OWA module parameters

input parameters and all structure column values are the subsection number into this documentation where this structure is well-specified.

Module design and implementation

In (Figure 1.32), we can see the flow of the program operation in OWA module.

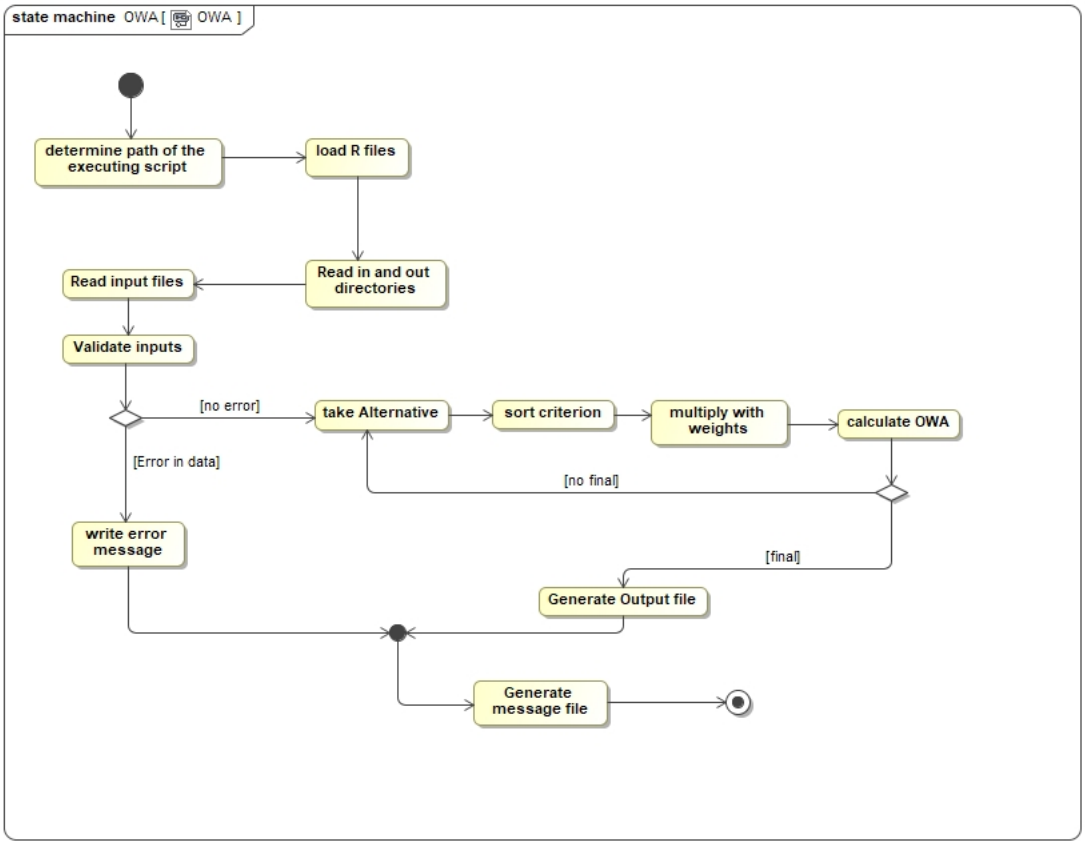

Figure 1.32: OWA state-machine diagram

In this module I want to highlight the new checks applied to the weights file, inherent to the change made in its structure, these controls provide the security that these weights are provided in the correct nomenclature.
Also, characteristics are checked as if the names established in the performance table match with those mentioned in the alternatives and criteria. Among other checks on the format of each file that you can see in the testing section.

The inclusion of the new Orness input parameter stands out, the program provides the possibility of calculating the vector of weights with this Orness value. This change and the possibility to obtain those weights through traditional way, forces that these two files to be optional, both can never be provided together, but one of them is required to perform the calculation.

Tests

To verify the correctness of my new implementation of OWA module, I prepared a complete set of test cases to certify that I contemplate the maximum of errors that can be made, or limitations imposed by the method itself to be correct, for example, the range of limitations in an input parameter or non-correct input data types.

For this task, I have designed (Table 1.4) with the purpose of having all the well-established verified cases, so that you can see the number of test cases, their purpose, the output obtained and whether it is expected in each case.

Test number	Case purpose	Expected output
1	All input files correct (simple case).	correct results.
2	Complex correct case.	correct results.
3	ID in alternatives no coincident with performance table.	Error: there are some different id's in alternatives table and the alternatives in Performance table.
4	Different number of active criteria and OWA weights.	Error: different number of active weights and active criteria.
5	the performance table has some NA values.	correct results, with NaN as a OWA result for each alternative with almost one NaN value in criteria.
6	Some weights have no numeric values.	Error: The weights must be numeric values only.
7	some inactive alternatives	correct result, without the inactive alternative.
8	A criteria element with false value (number of weight coincident with number of active criterion).	correct results. The inactive criteria are unused to do the OWA calculation.
9	OWA weights table have some NA values.	Error: there is a missing value (NA) in OWA weights vector.
10	missing mandatory input.	Error: An error has occurred while loading the input files.
11	All input correct (simple case) with Orness value in program parameter file.	correct results.
12	Program parameters without the correct data type of activation parameter.	Error: Activation parameter in Orness program parameters structure must be Boolean.
13	Program parameters with wrong id, for example "divergence".	Error: Structural error in program parameters file.
14	Orness parameter with value out of the range [0:1].	Error: Orness value must be a real number inside the [0:1] range.
15	Weights table and active Orness are provided.	Error: too many weights options supplied (Weights table and Orness, only one is needed).
16	Weights table and Orness are provided, but Orness is inactive.	correct results using Weights table values.

Table 1.4: OWA tests cases

Example

As a practical case, I will propose you the following example with these input parameters:

- Four active alternatives (a1, a2, a3, a4)
- Five active criterion (c1, c2, c3, c4, c5)
- The weights table are unused for this example. The weights will be generate using the Orness value (0.5) that will generate the neutral averaging weights table (0.2, 0.2, 0.2, 0.2, 0.2).
- The performance table used will be as the following (Table 1.5):

	c1	c2	c3	c4	c5
a1	0.2	2	4	3	9
a2	3	6	3.7	0	1
a3	6.5	5	6	3	9.8
a4	1	7	14	10	0.01

Table 1.5: Performance table for this example

After running the module, we obtain the following solution represented in (Table 1.6) and graphically in (Figure 1.33):

Alternatives	Values
a1	3.64
a2	2.74
a3	6.06000000005
a4	6.402

Table 1.6: Results obtained applying Orness value

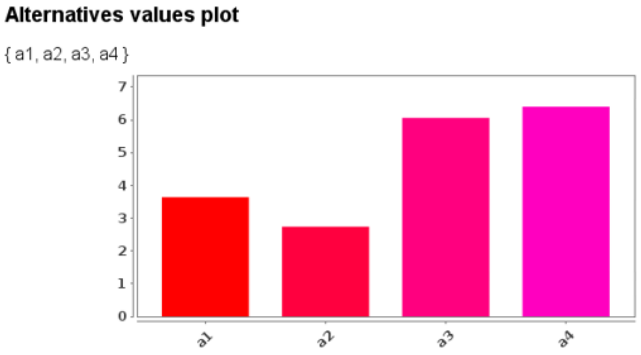

Figure 1.33: Graphical results

Or we can also generate an example with a vector of manually generated weights like: W=(0.2, 0.1, 0.0, 0.0, 0.7). obtaining the following solution represented in (Table 1.7) and graphically in (Figure 1.34):

Alternatives	Values
a1	2.34
a2	1.57
a3	4.71
a4	3.807

Table 1.7: Results obtained applying weights vector

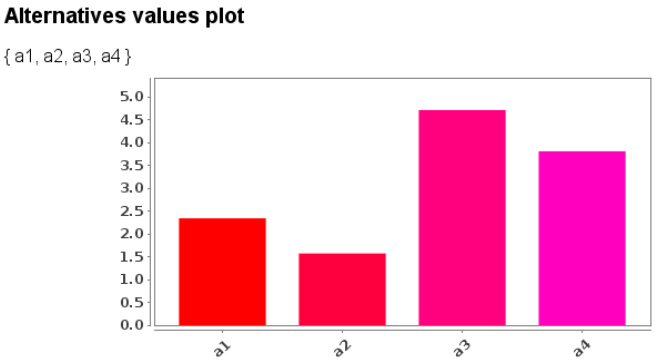

Figure 1.34: Graphical results

This practical case has been run on the new implementation modules, and graphics were generated using the results that have been previously used and exporting them to the latest available version of Diviz which can be used as input for the module **plotAlternativesValues** developed by ITTB.

1.6.2 ULOWA Module

This module implements the ULOWA aggregation operator: Unbalanced Linguistic Ordered Weighted Average. Aggregation operators for linguistic variables usually assume uniform and symmetrical distribution of the linguistic terms that define the variable.

However, there are some problems where an unbalanced set of linguistic terms is more appropriate to describe the objects. ULOWA accepts a set of linguistic labels defined with unbalanced fuzzy sets. The fuzzy sets must define a fuzzy partition on the set of reference values. They can be defined by trapezoidal or triangular membership functions.

Inputs and Outputs

To represent all input and output of this module I have done the following (Table 1.8) explaining the specific characteristics for each parameter:

I/O Parameters	id	name	displayName	isOptional	tag	structure
alternatives	input0	alternatives	alternatives	No	alternatives	4.2.12.1
criteria	input1	criteria	criteria	No	criteria	4.2.12.3
performanceTable	input3	performanceTable	performanceTable	No	performanceTable	4.2.12.5
weights	input2	weightsOWA	weights	Yes	criteriaSetsValues	-
orness input	input5	programParameters	orness	Yes	programParameters	4.2.12.8
fuzzyNumbers	input4	fuzzyNumbers	fuzzyNumbers	No	criteriaScales	-
ULOWA result	output0	alternativesValues	ulowaResults	-	alternativesValues	4.2.12.2
message	output1	messages	messages	-	programExecutionResults	4.2.12.9

Table 1.8: ULOWA module parameters

*The parameter "isOptional" is only applied on input parameters and all structure column values are the subsection number into this documentation where this structure is well-specified.

Module design and implementation

In (Figure 1.35), we can see the flow of the program operation in ULOWA module.

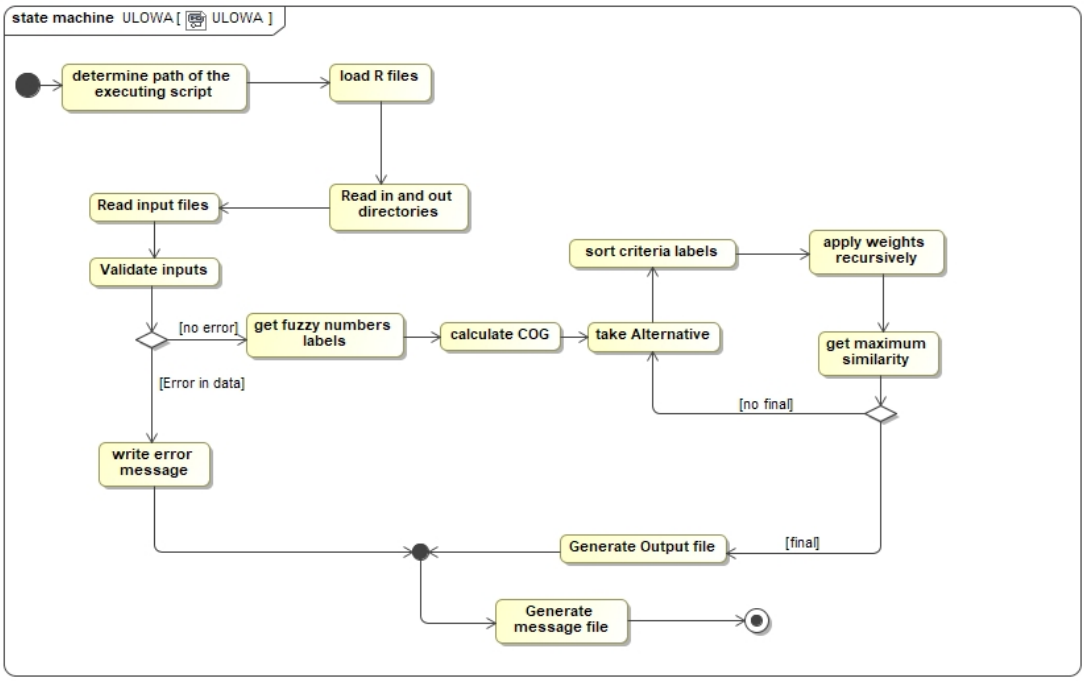

Figure 1.35: ULOWA state-machine diagram

Apart from the same change made in the module like in OWA module, due to the addition of the optional Orness parameter this module stands out for the use of recursion to perform a calculation. This is because it was more efficient and understandable. This is the only functionality of the project programmed in this way.

$$ULOWA(a_1, ..., a_m) = W * B^T = C^m\{\beta_h, b_h, h = 2, ..., m\} =$$
$$b_1 \otimes b_1 \oplus (1 - w_1) \otimes C^{m-1}\{\beta_h, b_h, h = 2, ..., m\}$$

Tests

To verify the correctness of my new implementation of ULOWA module, I prepared a complete set of test cases to certify that I contemplate the maximum of errors that can be made, or limitations imposed by the method itself to be correct, for example, the range of limitations in an input parameter or non-correct input data types.
For this task, I have designed (Table 1.9) with the purpose of having all the well-established verified cases, so that you can see the number of test cases, their purpose, the output obtained and whether it is expected in each case.

Test number	Case purpose	Expected output
1	All input files correct.	correct results.
2	Complex correct case.	correct results.
3	ID in alternatives no coincident with performance table.	Error: there are some different id's in alternatives table and the alternatives in Performance table.
4	Different number of active criteria and ULOWA weights.	Error: different number of active weights and active criteria.
5	the performance table has some NA values.	Error: number of items to replace is not a multiple of replacement length.
6	Some weights have no numeric values.	Error: The weights must be only numeric values.
7	some inactive alternatives	correct results.
8	A criteria element with false value (number of weight no coincident with number of active criterion).	Error: different number of active weights and active criteria.
9	ULOWA weights table have some NA values.	Error: there is a missing value (NA) in ULOWA weights vector.
10	NA in the fuzzy Numbers values.	Error: the continuity in (Very-Low) fuzzy label is not guaranteed, check the segments of the labels.
11	missing mandatory input.	Error: An error has occurred while loading the input files.
12	All input correct (simple case) with Orness value in program parameter file.	correct results.
13	Program parameters without the correct data type of activation parameter.	Error: Activation parameter in Orness program parameters structure must be Boolean.
14	Program parameters with wrong id, for example "divergence".	Error: Structural error in program parameters file.
15	Orness parameter with value out of the range [0:1].	Error: Orness value must be a real number inside the [0:1] range.
16	Weights table and active Orness are provided.	Error: too many weights options supplied (Weights table and Orness, only one is needed).
17	Weights table and Orness are provided, but Orness is inactive.	correct results using Weights table values.

Table 1.9: ULOWA tests cases

Example

As a practical case, I will propose you the following example with these input parameters:

- Four active alternatives (a1, a2, a3, a4)
- Five active criterion (c1, c2, c3, c4, c5)
- The weights table are unused for this example. The weights will be generate using the Orness value (0.5) that will generate the neutral averaging weights table (0.2, 0.2, 0.2, 0.2, 0.2).
- The fuzzy numbers used are the following seventh label table (Figure 1.36):

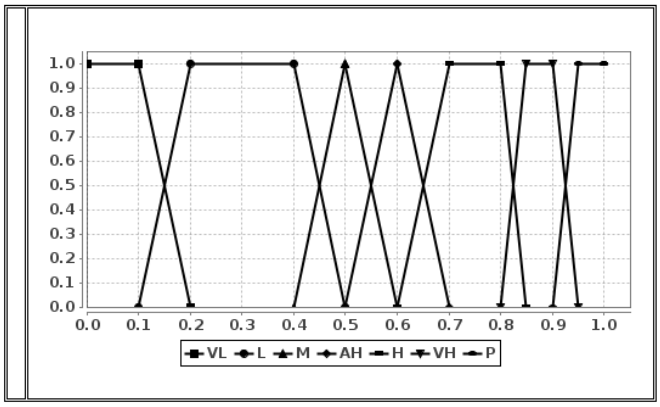

Figure 1.36: Fuzzy numbers for the example

- The performance table used will be as the following (Table 1.10):

	c1	**c2**	**c3**	**c4**	**c5**
a1	VL	P	AH	AH	M
a2	VL	M	AH	H	VL
a3	L	AH	P	P	VH
a4	M	VH	VL	VH	VL

Table 1.10: Performance table for this example

After running the module, we obtain the following solution represented in (Table 1.11) and graphically in (Figure 1.37):

Alternatives	**Labels**
a1	Medium
a2	Low
a3	Almost-High
a4	Medium

Table 1.11: Results obtained applying Orness value

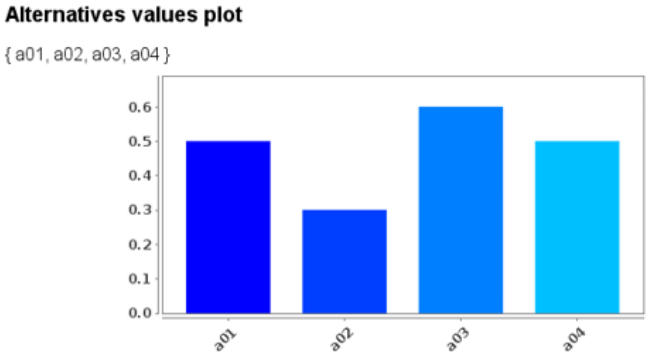

Figure 1.37: Graphical results

Or we can also generate an example with a vector of manually generated weights like: W=(0.2, 0.1, 0.0, 0.0, 0.7). obtaining the following solution represented in (Table 1.12) and graphically in (Figure 1.38):

Alternatives	Labels
a1	Low
a2	Low
a3	Medium
a4	Low

Table 1.12: Results obtained applying weights vector

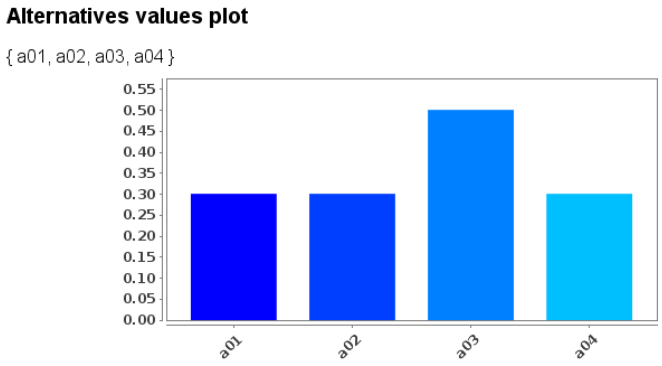

Figure 1.38: Graphical results

This practical case has been run on the new implementation modules, and graphics were generated using the results that have been previously used and exporting them to the latest available version of Diviz which can be used as input for the module **plotAlternativesValues** developed by ITTB.

1.6.3 OWADescriptors Module

Compute measures of weights given to the Ordered Weighted Average. The Ordered Weighted Averaging operators, commonly called OWA operators, provide a parameterized class of mean type aggregation operators.
For OWA weights exists different measures to characterise a set of weights associated to an OWA operator. In this module we implement the measures: balance, divergence, entropy and Orness.

Inputs and Outputs

To represent all input and output of this module I have done the following (Table 1.13) explaining the specific characteristics for each parameter:

I/O Parameters	id	name	displayName	isOptional	tag	structure
Weights	input0	weights	weightsOWA	No	criteriaSetsValues	-
balance	output0	balance	balance	-	criteriaValues	4.2.12.4
divergence	output1	divergence	divergence	-	criteriaValues	4.2.12.4
entropy	output2	entropy	entropy	-	criteriaValues	4.2.12.4
orness	output3	orness	orness	-	criteriaValues	4.2.12.4
message	output4	messages	messages	-	programExecution Results	4.2.12.9

Table 1.13: OWADescriptors module parameters

*The parameter "isOptional" is only applied on input parameters and all structure column values are the subsection number into this documentation where this structure is well-specified.

Module design and implementation

In (Figure 1.39), we can see the flow of the program operation in OWADescriptors module.

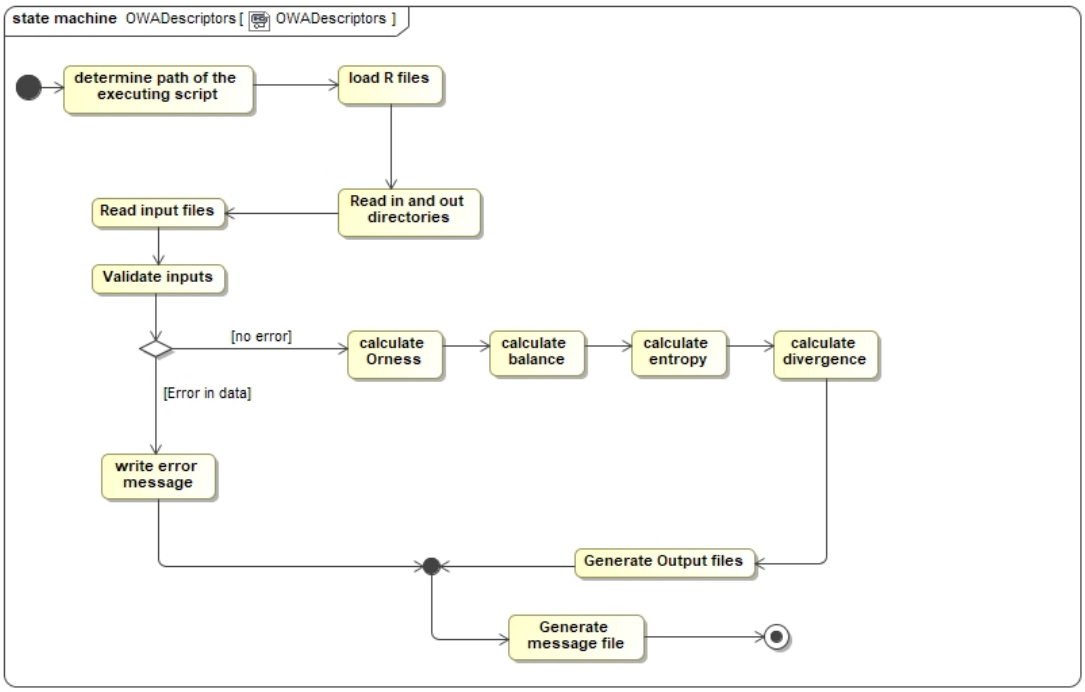

Figure 1.39: OWADescriptors state-machine diagram

The main difference of this module respect to its previous implementation is that, in this new design, it is only necessary to receive the weight vector as a parameter, it is not necessary to receive the criteria to execute because it do not have any functionality.

On the other hand, instead of the user have to choose which weight vector descriptor he wants to receive, the module calculates the four output parameters in the same execution and provides them to the user.

Tests

To verify the correctness of my new implementation of OWADescriptors module, I prepared a complete set of test cases to certify that I contemplate the maximum of errors that can be made, or limitations imposed by the method itself to be correct, for example, the range of limitations in an input parameter or non-correct input data types.

this task, I have designed (Table 1.14) with the purpose of having all the well-established verified cases, so that you can see the number of test cases, their purpose, the output obtained and whether it is expected in each case.

Test number	Case purpose	Expected output
1	All input files correct.	correct results.
2	Complex correct case.	correct results.
3	Weights with no numeric in inputs.	Error: The weights must be numeric values only.
4	Weight summation result >1.	Error: the summation of all active weights must be exactly 1.0.
5	More than 1 weights table supplied.	Error: More than one weights table supplied.
6	special case result in Orness.	correct results. Orness = 0.5.
7	special case result in balance.	correct results. Balance = 0.
8	special case result in entropy.	correct results. Entropy = 0.
9	Special case result in divergence	correct results. Divergence = 0
10	Weights with Na values.	Error: there is a missing value (NA) in OWA weights vector.
11	No input supplied.	Error: An error has occurred while loading the input files.

Table 1.14: OWADescriptors tests cases

Example

As a practical case, I will propose the following example with this input parameter:

- The weights table used for this example is (0.2, 0.2, 0.2, 0.2, 0.2).

After running the module, we obtain the following solution (Table 1.15):

Properties	Results
Balance	$2.77555e^{-17}$
Divergence	0.125
Entropy	1.60944
Orness	0.5

Table 1.15: Results

This practical case has been run on the new implementation modules.

1.6.4 Deffuzification Module

This module implements three different methods to defuzzificate the result of a ULOWA operation (Unbalanced Linguistic Ordered Weighted Average). Aggregation operators for linguistic variables usually assume uniform and symmetrical distribution of the linguistic terms that define the variable. However, there are some problems where an unbalanced set of linguistic terms is more appropriate to describe the objects than others. ULOWA accepts a set of linguistic labels defined with unbalanced fuzzy sets. The fuzzy sets must define a fuzzy partition on the set of reference values. They can be defined by trapezoidal or triangular membership functions.
For this method we apply three different operations for every ULOWA alternative result; COG, COM and Ordinal.

Inputs and Outputs

To represent all input and output of this module I have done the following (Table 1.16) explaining the specific characteristics for each parameter:

I/O Parameters	**id**	**name**	**displayName**	**isOptional**	**tag**	**structure**
ULOWA result	input0	alternativesValues	linguistic scores	No	alternativesValues	4.2.12.2
fuzzyNumbers	input1	fuzzyNumbers	fuzzyNumbers	No	criteriaScales	-
COG	output0	defuzzificationCOG	COG	-	alternativesValues	4.2.12.2
COM	output1	defuzzificationCOM	COM	-	alternativesValues	4.2.12.2
ORD	output2	defuzzificationORD	Ordinal	-	alternativesValues	4.2.12.2
message	output3	messages	messages	-	programExecution Result	4.2.12.9

Table 1.16: Deffuzification module parameters

*The parameter “isOptional” is only applied on input parameters and all structure column values are the subsection number into this documentation where this structure is well-specified.

Module design and implementation

In (Figure 1.40), we can see the flow of the program operation in Deffuzification module.

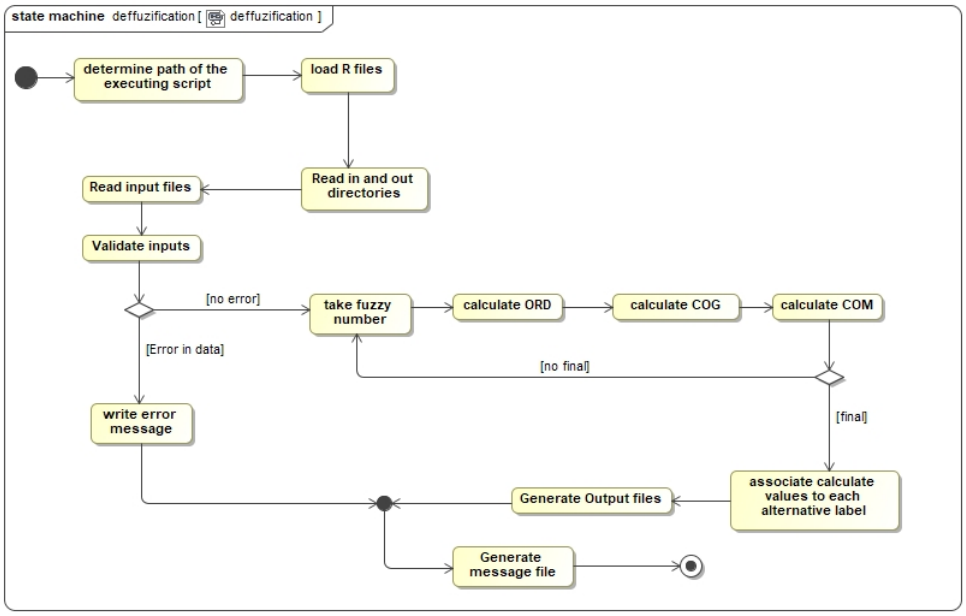

Figure 1.40: Deffuzification state-machine diagram

The main change made in this module focuses on the generation of output files, as in the previous case, instead of that the user has to choose which descriptor he wants to receive (COG, COM or Ordinal), the module calculates the three output parameters in the same execution and provides them to the user.

- COG value consists to apply the formula of center of gravity explained in (1.1.2):

$$COG(s_A) = \begin{cases} y_A^* = \begin{cases} \frac{1}{6}\,(\frac{p_3 - p_2}{p_4 - p_1} + 2) & if\ p_1 \neq p_4 \\ \frac{1}{2} & if\ p_1 = p_4 \end{cases} \\ x_A^* = \frac{y_A^*(p_3 + p_2) + (p_4 + p_1)(1 - y_A^*)}{2} \end{cases}$$

- In COM (center of Maxima), the defuzzified value is taken as the element with the highest membership values. When there are more than one element having maximum membership values, the mean value of the maxima is taken.
  Let A be a fuzzy set of label l_i with membership function $\mu_A(x)$ defined over $x\ \epsilon\ X$, where X is a universe of discourse. The defuzzified value is let say x^* of a fuzzy set and is defined as,

$$X^* = \frac{\sum x_i\ \epsilon\ M\ x_i}{|M|}$$

  Here, $M = \{x_i | \mu_A(x_i)$is equal to the height of the fuzzy set $A\}$ and $|M|$ is the cardinality of the set M.

- Ordinal descriptor consists of: let A be a fuzzy set of the i-th label in the set of labels L, the defuzzified value x^* of A will be $x^* = i$, being i the position of the label in L.

Tests

To verify the correctness of my new implementation of Deffuzification module, I prepared a complete set of test cases to certify that I contemplate the maximum of errors that can be made, or limitations imposed by the method itself to be correct, for example, the range of limitations in an input parameter or non-correct input data types.
For this task, I have designed (Table 1.17) with the purpose of having all the well-established verified cases, so that you can see the number of test cases, their purpose, the output obtained and whether it is expected in each case.

Test number	Case purpose	Expected output
1	All input files correct.	correct results.
2	Complex correct case.	correct complex results.
3	ID in alternatives table, not coincident with fuzzy number ID.	Error: (Ultra-Perfect) alternative value don't corresponds with any fuzzy label.
4	NA in a fuzzy Number.	Error: the continuity in (Very-Low) fuzzy label is not guaranteed, check the segments of the labels.
5	No enough categories supplied.	Error: the fuzzy sets do not define a fuzzy partition, which is a requirement of the operator, problem between (Very-Low) and (Very-High) fuzzy labels.
6	NaN value in the alternatives values.	'match' requires vector.

Table 1.17: Deffuzification tests cases

Example

As a practical case, I will propose the following example with this input parameter:

- The fuzzy numbers used are the following seventh label table (Figure 1.41):

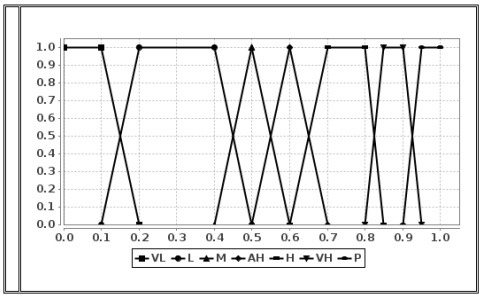

Figure 1.41: Fuzzy numbers for this example

- As alternatives values I used the output generated for ULOWA module represented in the following table (Figure 1.42):

Alternatives	Labels
a1	Medium
a2	Low
a3	Almost-High
a4	Medium

Figure 1.42: Alternatives for this example

After running the module, we obtain the following solution (1.43):

Alternatives	COG	COM	ORD
a1	0.50	0.50	3
a2	0.30	0.30	2
a3	0.60	0.60	4
a4	0.50	0.50	3

Figure 1.43: Results for this example

This practical case has been run on the new implementation modules, and graphics were generated using the results that have been previously used and exporting them to the latest available version of Diviz which can be used as input for the module **plotFuzzyCategoriesValues** developed by ITTB.

1.6.5 Fuzzy Labels Descriptors Module

The purpose of this module is to obtain the two types of uncertainty in fuzzy sets are recognized:

- Specificity, related to the measurement of imprecision, which is based on the cardinality of the set. Specificity and fuzziness refer to two different characteristics of fuzzy sets, measures the degree of truth of the sentence containing just one element.
- Fuzziness, or entropy, which measures the vagueness of the set as a result of having imprecise boundaries. Fuzziness measures the difference from a crisp set.

Inputs and Outputs

To represent all input and output of this module I have done the following (Table 1.18) explaining the specific characteristics for each parameter:

I/O Parameters	id	name	displayName	isOptional	tag	structure
fuzzyNumbers	input0	fuzzyNumbers	fuzzyNumbers	No	criteriaScales	-
specificity	output0	specificity	specificity	-	alternativesValues	4.2.12.2
fuzziness	output1	fuzziness	fuzziness	-	alternativesValues	4.2.12.2
message	output2	messages	messages	-	programExecutionResults	4.2.12.9

Table 1.18: fuzzyNumbersDescriptors module parameters

*The parameter "isOptional" is only applied on input parameters and all structure column values are the subsection number into this documentation where this structure is well-specified.

Module design and implementation

In (Figure 1.44), we can see the flow of the program operation in fuzzyNumbersDescriptors module.

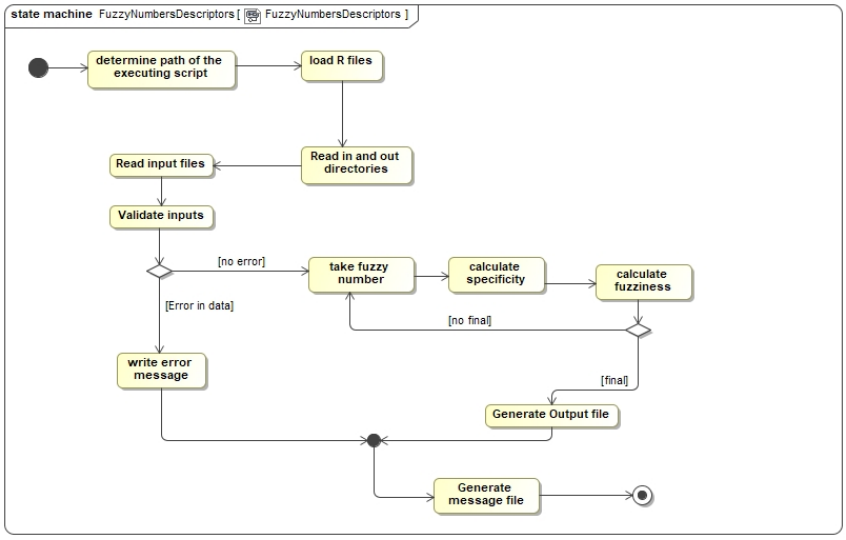

Figure 1.44: fuzzyNumbersDescriptors state-machine diagram

Tests

To verify the correctness of my new implementation of fuzzyLabelsDescriptors module, I prepared a complete set of test cases to certify that I contemplate the maximum of errors that can be made, or limitations imposed by the method itself to be correct, for example, the range of limitations in an input parameter or non-correct input data types.
For this task, I have designed (Table 1.19) with the purpose of having all the well-established verified cases, so that you can see the number of test cases, their purpose, the output obtained and whether it is expected in each case.

Test number	Case purpose	Expected output
1	All input files correct, simple case.	correct results.
2	Complex correct case.	correct results.
3	No ordered point in a fuzzy number values.	Error: There are, almost one, fuzzyNumber not ordered.
4	Out of fuzzy numbers rank (0 <= fuzzy numbers <= 1).	correct results, any problems because you can take a different rank.
5	No correct coverage of fuzzy numbers v1.	Error: the fuzzy sets do not define a fuzzy partition, which is a requirement of the operator, problem between (Low) and (Medium) fuzzy labels.
6	No correct coverage of fuzzy numbers v2.	Error: the fuzzy sets do not define a fuzzy partition, which is a requirement of the operator, problem between (Very-Low) and (Low) fuzzy labels.
7	No numeric value assigned to segment value.	Error: the numericity in (Very-Low) fuzzy label is not guaranteed, check the segments of the labels.
8	More than 1 fuzzy numbers table supplied.	Error: Incorrect number of fuzzy number tables supplied.
9	No fuzzy numbers value table supplied.	Error: An error has occured while loading the input files.
10	NA in a fuzzy number value.	Error: the continuity in (Very-Low) fuzzy label is not guaranteed, check the segments of the labels.

Table 1.19: Fuzzy numbers tests cases

Example

As a practical case, I will propose the following example with this input parameter:

- The fuzzy numbers used are the following seventh label table (Figure 1.45):

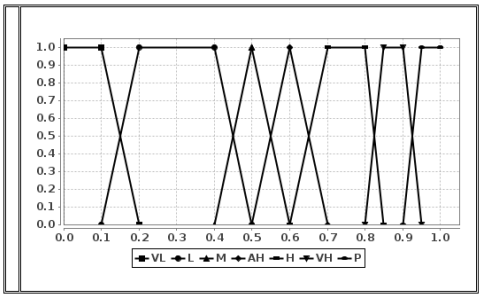

Figure 1.45: Fuzzy numbers for this example

After running the module, we obtain the following solution (Figure 1.46):

Fuzzy Labels	Specificity	Fuzziness
Very-Low (VL)	0.85	0.05
Low (L)	0.7	0.01
Medium (M)	0.9	0.01
Almost-High (AH)	0.9	0.01
High (H)	0.825	0.075
Very-High (VH)	0.9	0.05
Perfect (P)	0.925	0.025

Figure 1.46: Results for this example

This practical case has been run on the new implementation modules, and graphics were generated using the results that have been previously used and exporting them to the latest available version of Diviz which can be used as input for the module **plotFuzzyCategoriesValues** developed by ITTB.

Bibliography

Miquel Buxons. Developed diviz functions. `https://github.com/MikiBuxons/DevelopedDivizFunctions`, May 2020. Accessed on 2020-06-11.

Jordi Canals. Intelligent decision support tools in decision deck. *repositori-urv*, 2011.

Ronald R. Yager. *On ordered weighted averaging aggregation operators in multi-criteria decision making*, pages 183–190. IEEE Transactions on Systems, Man and Cybernetics, 1988.

David Isern, Lucas Marin, Aïda Valls, Antonio Moreno, and José M. Merigó. Induced unbalanced linguistic ordered weighted average and its application in multi-person decision making. 2017.

Ronald R. Yager. Quantifier guided aggregation using owa operators. *International Journal of Intelligent Systems*, 1996.

Decision Deck Consortium. Diviz-download. `https://www.diviz.org/download.html`, 2009a. Accessed on 2020-06-09.

Patrick Meyer. Xmcda-r. `https://gitlab.com/XMCDA-library/XMCDA-R`, 2019. Accessed on 2020-04-05.

Sébastien Bigaret. weightedsum-r. `https://gitlab.com/XMCDA-library/weightedSum-R/`, 2017. Accessed on 2020-04-16.

Decision Deck Consortium. Diviz-list available modules. `https://www.diviz.org/wsindex-1.html`, 2009b. Accessed on 2020-06-10.

Appendices

Appendix A

Description files

Into this Annex there are all module documentations files, documentation written in xml language. You can find these documents into each module folder named as "description-wsDD.xml" on my personal GitHub Buxons [2020] or, soon, it will be published on the official website of the Diviz program, in the section of list of available modules.

A.1 OWA description

```
<?xml version="1.0" encoding="UTF-8"?>
<program_description xsi:noNamespaceSchemaLocation="http://sma.uni.lu/d2cms/ws/_downloads/description.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <program name="OWA" provider="URV" version="4.0" displayName="OWA"/>
  <documentation>
    <description>This module implements the OWA aggregation operator: The Ordered Weighted Averaging operators, commonly called OWA operators, provide a parameterized class of mean type aggregation operators.
OWA was defined by R.R. Yager 1988. Here we provide an implementation of OWA operator. The main characteristic of this aggregation operator is the fact that the weights are associated to values instead of criteria. In this way, we can define different aggregation policies according to the importance that we associate to high and low performance scores.</description>
    <contact>Aida Valls &lt;aida.valls@urv.cat&gt;</contact>
    <reference>A. Valls, The Ordered Weighted Averaging Operator, In: Proc. IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2010, IEEE Computer Society, Barcelona, Catalonia, 2010, pp. 3063-3070.</reference>
  </documentation>
  <parameters>
    <input displayName="criteria" name="criteria" id="input1" isoptional="0">
      <documentation>
        <description>A list of criteria. Criteria can be activated or desactivated via the &lt;active&gt; tag (true or false).
By default (no &lt;active&gt; tag), criteria are considered as active.</description>
      </documentation>
      <xmcda tag="criteria">
                        <![CDATA[
                  <criteria id="...">
                      <criterion>
                          <active>[...]</active>
                      </criterion>
                      [...]
                  </criteria>
                  ]]>
                   </xmcda>
```

```
    </input>

    <input displayName="alternatives" name="alternatives" id="input0" isoptional="0">
      <documentation>
        <description>A list of alternatives. Alternatives can be activated or desactivated via
            the &lt;active&gt; tag (true or false).
By default (no &lt;active&gt; tag), alternatives are considered as active.</description>
      </documentation>
      <xmcda tag="alternatives">
        <![CDATA[
                  <alternatives id="...">
                         <alternative>
                               <active>[...]</active>
                         </alternative>
          [...]
            </alternatives>
        ]]>
      </xmcda>
    </input>

    <input displayName="performanceTable" name="performanceTable" id="input3" isoptional="0">
      <documentation>
      <description>The performance table will contain the all criteria values, por each
          alternative. </description>
      </documentation>
      <xmcda tag="performanceTable">

      <![CDATA[
      <performanceTable>
        <alternativePerformances>
          <alternativeID>[...]</alternativeID>
          <performance>
            <criterionID>[...]</criterionID>
            <value>
              <float>[...]</float>
            </value>
          </performance>
          [...]
        </alternativePerformances>
        [...]
      </performanceTable>
      ]]>
      </xmcda>
    </input>

    <input displayName="weights" name="weightsOWA" id="input2" isoptional="1">
      <documentation>
      <description>The weights are associated to the values of the alternatives rather than to
          the criteria. In this way they can define different aggregation policies. Assuming
          that the values on the alternative will be sorted from the best to the worst, the list
           of weights must be ordered according to the importance that is given to the values,
          from the highest to the lowest.
For example a list of weights as (0.5, 0.5, 0, 0, 0) is ignoring the 3 lowest values, and making
       an average of the two highest ones. A list like (0, 0, 1, 0 ,0 ) is calculating the median,
       while (0, 0, 0, 0, 1) is taking the minimum.
Notice that the sum of weights is required to be 1. In version 4.0 the weights have only one
      structure, where the user can specify her/his weights without using any identifier for each
      weight.
      </description>
    </documentation>
      <xmcda tag="criteriaSetsValues">
```

```
      <![CDATA[
      <criteriaSetsValues>
        <criteriaSetValues id="owa-weights" mcdaConcept="OWA weights">
          <criteriaSetID>...</criteriaSetID>
          <values>
            <value>
              <real>...</real>
            </value>
            <value>
              [...]
            </value>
            [....]
          </values>
        </criteriaSetValues>
      </criteriaSetsValues>
        ]]>
      </xmcda>
    </input>

    <input displayName="orness" name="programParameters" id="input4" isoptional="1">
      <documentation>
      <description>The weights are associated to the values of the alternatives rather than to
          the criteria. In this way they can define different aggregation policies. In this
          option, the system computes the OWA weights vector from the degree of orness indicated
           as input. The orness values must be in the continuous range from 0 to 1. Notice that
          0.0 means andness, 0.5 neutrality, 1.0 corresponds to maximum orness.
</description>
      </documentation>
      <xmcda tag="programParameters">
      <![CDATA[
           <programParameters>
                <parameter !-- REQUIRED --> id="active">
                   <values>
           <value>
             <boolean>...</boolean>
           </value>
         </values>
                </parameter>
                <parameter !-- REQUIRED --> id="orness">
                   <values>
           <value>
             <real>...</real>
           </value>
         </values>
                </parameter>
           </programParameters>
      ]]>
      </xmcda>
    </input>

    <output displayName="owaResults" name="alternativesValues" id="output0">
      <documentation>
      <description>Result obtained from the OWA aggregation on each alternative. It will be a
          numeric value.</description>
      </documentation>
      <xmcda tag="alternativesValues"/>
    </output>

    <output displayName="messages" name="messages" id="output1">
      <documentation>
      <description>A status message.</description>
```

```
133       </documentation>
134       <xmcda tag="methodMessages"/>
135     </output>
136
137   </parameters>
138 </program_description>
```

A.2 ULOWA description

```
<?xml version="1.0" encoding="UTF-8"?>
<program_description xsi:noNamespaceSchemaLocation="http://sma.uni.lu/d2cms/ws/_downloads/description.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <program name="ULOWA" provider="URV" version="4.0" displayName="ULOWA"/>
  <documentation>
    <description>This module implements the ULOWA aggregation operator: Unbalanced Linguistic Ordered Weighted Average. Aggregation operators for linguistic variables usually assume uniform and symmetrical distribution of the linguistic terms that define the variable. However, there are some problems where an unbalanced set of linguistic terms is more appropriate to describe the objects. ULOWA accepts a set of linguistic labels defined with unbalanced fuzzy sets. The fuzzy sets must define a fuzzy partition on the set of reference values. They can be defined by trapezoidal or triangular membership functions.
    </description>
    <contact>Aida Valls &lt;aida.valls@urv.cat&gt;</contact>
    <reference>A. Valls, The Unbalanced Linguistic Ordered Weighted Averaging Operator, In: Proc. IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2010, IEEE Computer Society, Barcelona, Catalonia, 2010, pp. 3063-3070.</reference>
  </documentation>
  <parameters>
    <input displayName="criteria" name="criteria" id="input1" isoptional="0">
      <documentation>
        <description>A list of criteria. Criteria can be activated or desactivated via the &lt;active&gt; tag (true or false).
By default (no &lt;active&gt; tag), criteria are considered as active.</description>
      </documentation>
      <xmcda tag="criteria">
                          <![CDATA[
                  <criteria id="...">
                      <criterion>
                          <active>[...]</active>
                      </criterion>
                      [...]
                  </criteria>
                  ]]>
                   </xmcda>
    </input>

    <input displayName="alternatives" name="alternatives" id="input0" isoptional="0">
      <documentation>
        <description>A list of alternatives. Alternatives can be activated or desactivated via the &lt;active&gt; tag (true or false).
By default (no &lt;active&gt; tag), alternatives are considered as active.</description>
      </documentation>
      <xmcda tag="alternatives">
        <![CDATA[
                <alternatives id="...">
                       <alternative>
                            <active>[...]</active>
                       </alternative>
          [...]
            </alternatives>
        ]]>
      </xmcda>
    </input>

    <input displayName="performanceTable" name="performanceTable" id="input3" isoptional="0">
      <documentation>
      <description>The performance table will contain only string values, which correspond to the identifiers of the labels defined in the domain of the linguistic variable.</
```

```
          description>
47      </documentation>
48      <xmcda tag="performanceTable">
49
50      <![CDATA[
51      <performanceTable>
52        <alternativePerformances>
53          <alternativeID>[...]</alternativeID>
54          <performance>
55            <criterionID>[...]</criterionID>
56            <value>
57              <label>[...]</label>
58            </value>
59          </performance>
60          [...]
61        </alternativePerformances>
62        [...]
63      </performanceTable>
64      ]]>
65      </xmcda>
66    </input>
67
68    <input displayName="weights" name="weights" id="input2" isoptional="1">
69      <documentation>
70      <description>The weights are associated to the values of the alternatives rather than to
            the criteria. In this way they can define different aggregation policies. Assuming
            that the values on the alternative will be sorted from the best to the worst, the list
             of weights must be ordered according to the importance that is given to the values,
            from the highest to the lowest.
71  For example a list of weights as (0.5, 0.5, 0, 0, 0) is ignoring the 3 lowest values, and making
       an average of the two highest ones. A list like (0, 0, 1, 0 ,0 ) is calculating the median,
       while (0, 0, 0, 0, 1) is taking the minimum.
72  Notice that the sum of weights is required to be 1. In version 4.0 the weights have only one
       structure, where the user can specify her/his weights without using any identifier for each
       weight.
73      </description>
74    </documentation>
75      <xmcda tag="criteriaSetsValues">
76      <![CDATA[
77      <criteriaSetsValues>
78        <criteriaSetValues id="owa-weights" mcdaConcept="OWA weights">
79          <criteriaSetID>...</criteriaSetID>
80          <values>
81            <value>
82              <real>...</real>
83            </value>
84            <value>
85              [...]
86            </value>
87            [....]
88          </values>
89        </criteriaSetValues>
90      </criteriaSetsValues>
91        ]]>
92      </xmcda>
93    </input>
94
95    <input displayName="fuzzyNumbers" name="fuzzyNumbers" id="input4" isoptional="0">
96      <documentation>
97      <description>Definition of the fuzzy sets associated to the linguistic variable used for
            all the criteria. The semantics of the linguistic labels are given by a trapezoidal
```

```
        membership function, each membership function is represented as a group of four
        consecutive segments, each segment is descrived with two 2D points (head, tail). Into
        this version exists the possibility to define a segment using the tail of the previous
         segment (only in the same label) as the head point of the new one, avoiding the
        possibility of continuity errors of the segments. If the second segment is equal to
        third segment, the fuzzy is triangular. The values must be ordered increasingly. For
        each label in the linguistic domain (categoriesValues list), a fuzzy set must be
        defined. The labels must be ordered from the worst to the best performance (fi. Low,
        Medium, High, Perfect).</description>
98    </documentation>
99    <xmcda tag="criteriaScales">
100     <![CDATA[
101   <criteriaScales id="fuzzy-numbers">
102     <criterionScales>
103       <criterionID>...<\criterionID>
104       <!-- REQUIRED --> different ID than criteria supplied>
105       <scales>
106         <scale id="fuzzy-scales">
107           <qualitative>
108             <preferenceDirection>min</preferenceDirection>
109             <valuedLabels>
110               <valuedLabel>
111                 <label>...</label>
112                 <value>
113                   <fuzzyNumber>
114                     <piecewiseLinear>
115                       <segment>
116                         <head>
117                           <abscissa>
118                             <real>...<\real>
119                           <\abscissa>
120                           <ordinate>
121                             <real>...<\real>
122                           <\ordinate>
123                         <\head>
124                         <tail>
125                           <abscissa>
126                             <real>...<\real>
127                           <\abscissa>
128                           <ordinate>
129                             <real>...<\real>
130                           <\ordinate>
131                         <\tail>
132                       <\segment>
133                       <segment>
134                         <\head>
135                         <tail>
136                           <abscissa>
137                             <real>...<\real>
138                           <\abscissa>
139                           <ordinate>
140                             <real>...<\real>
141                           <\ordinate>
142                         <\tail>
143                       <\segment>
144                       <segment>
145                       [...]
146                       <\segment>
147                       <segment>
148                       [...]
149                       <\segment>
```

```
                    <\piecewiseLinear>
                  <\fuzzyNumber>
                <\value>
              <\valuedLabel>
              <valuedLabel>
                [...]
              </valuedLabel>
              [...]
            </valuedLabels>
          <\qualitative>
        <\scale>
      <\scales>
    <\criterionScales>
  <\criteriaScales>
    ]]>
  </xmcda>
</input>

<input displayName="orness" name="programParameters" id="input5" isoptional="1">
  <documentation>
  <description>The weights are associated to the values of the alternatives rather than to
      the criteria. In this way they can define different aggregation policies. In this
      option, the system computes the OWA weights vector from the degree of orness indicated
       as input. The orness values must be in the continuous range from 0 to 1. Notice that
      0.0 means andness, 0.5 neutrality, 1.0 corresponds to maximum orness.
</description>
  </documentation>
  <xmcda tag="programParameters">
  <![CDATA[
       <programParameters>
            <parameter !-- REQUIRED --> id="active">
                <values>
       <value>
         <boolean>...</boolean>
       </value>
      </values>
            </parameter>
            <parameter !-- REQUIRED --> id="orness">
                <values>
       <value>
         <real>...</real>
       </value>
      </values>
            </parameter>
       </programParameters>
  ]]>
  </xmcda>
</input>

<output displayName="ulowaResults" name="alternativesValues" id="output0">
  <documentation>
  <description>Result obtained from the ULOWA aggregation on each alternative. It will be a
      linguistic label belonging to the domain defined in the fuzzyNumbers input file.</
      description>
  </documentation>
  <xmcda tag="alternativesValues"/>
</output>

<output displayName="messages" name="messages" id="output1">
  <documentation>
  <description>A status message.</description>
```

```
205       </documentation>
206       <xmcda tag="methodMessages"/>
207     </output>
208
209   </parameters>
210 </program_description>
```

A.3 OWADescriptors description

```
<?xml version="1.0" encoding="UTF-8"?>
<program_description xsi:noNamespaceSchemaLocation="http://sma.uni.lu/d2cms/ws/_downloads/description.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <program name="OWADescriptors" provider="URV" version="4.0" displayName="OWADescriptors"/>
  <documentation>
    <description>Compute measures of weights given to the Ordered Weighted Average. The Ordered Weighted Averaging operators, commonly called OWA operators, provide a parameterized class of mean type aggregation operators.
For OWA wights exists different measures to characterise a set of weights associated to an OWA operator. In this module we implement the measures: balance, divergence, entropy and ornes.</description>
    <contact>Aida Valls &lt;aida.valls@urv.cat&gt;</contact>
    <reference>A. Valls, The Ordered Weighted Averaging Operator, In: Proc. IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2010, IEEE Computer Society, Barcelona, Catalonia, 2010, pp. 3063-3070.</reference>
  </documentation>
  <parameters>
    <input displayName="weights" name="weightsOWA" id="input0" isoptional="0">
      <documentation>
      <description>The weights are associated to the values of the alternatives rather than to the criteria. In this way they can define different aggregation policies. Assuming that the values on the alternative will be sorted from the best to the worst, the list of weights must be ordered according to the importance that is given to the values, from the highest to the lowest.
For example a list of weights as (0.5, 0.5, 0, 0, 0) is ignoring the 3 lowest values, and making an average of the two highest ones. A list like (0, 0, 1, 0 ,0 ) is calculating the median, while (0, 0, 0, 0, 1) is taking the minimum.
Notice that the sum of weights is required to be 1. In version 4.0 the weights have only one structure, where the user can specify her/his weights without using any identifier for each weight.
      </description>
    </documentation>
      <xmcda tag="criteriaSetsValues">
      <![CDATA[
      <criteriaSetsValues>
        <criteriaSetValues id="owa-weights" mcdaConcept="OWA weights">
          <criteriaSetID>...</criteriaSetID>
          <values>
            <value>
              <real>...</real>
            </value>
            <value>
              [...]
            </value>
            [....]
          </values>
        </criteriaSetValues>
      </criteriaSetsValues>
        ]]>
      </xmcda>
    </input>

    <output displayName="balance" name="balance" id="output0">
      <documentation>
      <description>Result obtained apply balance calculation in weights table. It will be a single numeric value.</description>
      </documentation>
      <xmcda tag="criteriaValues"/>
    </output>
```

```
44
45      <output displayName="divergence" name="divergence" id="output1">
46        <documentation>
47        <description>Result obtained apply divergence calculation in weights table. It will be a
              single numeric value.</description>
48        </documentation>
49        <xmcda tag="alternativesValues"/>
50      </output>
51
52      <output displayName="entropy" name="entropy" id="output2">
53        <documentation>
54        <description>Result obtained apply entropy calculation in weights table. It will be a
              single numeric value.</description>
55        </documentation>
56        <xmcda tag="alternativesValues"/>
57      </output>
58
59      <output displayName="ornes" name="ornes" id="output3">
60        <documentation>
61        <description>Result obtained apply ornes calculation in weights table. It will be a single
               numeric value.</description>
62        </documentation>
63        <xmcda tag="alternativesValues"/>
64      </output>
65
66      <output displayName="messages" name="messages" id="output4">
67        <documentation>
68        <description>A status message.</description>
69        </documentation>
70        <xmcda tag="methodMessages"/>
71      </output>
72
73    </parameters>
74  </program_description>
```

A.4 Defuzzication description

```
<?xml version="1.0" encoding="UTF-8"?>
<program_description xsi:noNamespaceSchemaLocation="http://sma.uni.lu/d2cms/ws/_downloads/description.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <program name="Defuzzification" provider="URV" version="4.0" displayName="Defuzzification"/>
  <documentation>
    <description>This module implements three different methods to defuzzificate the result of a ULOWA operation (Unbalanced Linguistic Ordered Weighted Average). Aggregation operators for linguistic variables usually assume uniform and symmetrical distribution of the linguistic terms that define the variable. However, there are some problems where an unbalanced set of linguistic terms is more appropriate to describe the objects. ULOWA accepts a set of linguistic labels defined with unbalanced fuzzy sets. The fuzzy sets must define a fuzzy partition on the set of reference values. They can be defined by trapezoidal or triangular membership functions.
For this method we apply three different operations for every ULOWA alternative result: COG, COM and Ordinal.
    </description>
    <contact>Aida Valls &lt;aida.valls@urv.cat&gt;</contact>
    <reference>A. Valls, The Unbalanced Linguistic Ordered Weighted Averaging Operator, In: Proc. IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2010, IEEE Computer Society, Barcelona, Catalonia, 2010, pp. 3063-3070.</reference>
  </documentation>
  <parameters>
    <input displayName="linguistic scores" name="alternativesValues" id="input0" isoptional="0">
      <documentation>
        <description>A list of alternativesValue. Normally it will be the result of the ULOWA or other linguistic aggregation operation.</description>
      </documentation>
      <xmcda tag="alternativesValues">
        <![CDATA[
        <alternativesValues>
          <alternativeValue>
          <alternativeID>...</alternativeID>
          <values>
            <value>
              <label>...</label>
            </value>
          </values>
          </alternativeValue>
          [...]
        </alternativesValues>
        ]]>
      </xmcda>
    </input>

    <input displayName="fuzzyNumbers" name="fuzzyNumbers" id="input1" isoptional="0">
      <documentation>
      <description>Definition of the fuzzy sets associated to the linguistic variable used for all the criteria. The semantics of the linguistic labels are given by a trapezoidal membership function, each membership function is represented as a group of four consecutive segments, each segment is descrived with two 2D points (head, tail). Into this version exists the possibility to define a segment using the tail of the previous segment (only in the same label) as the head point of the new one, avoiding the possibility of continuity errors of the segments. If the second segment is equal to third segment, the fuzzy is triangular. The values must be ordered increasingly. For each label in the linguistic domain (categoriesValues list), a fuzzy set must be defined. The labels must be ordered from the worst to the best performance (fi. Low, Medium, High, Perfect).</description>
      </documentation>
      <xmcda tag="criteriaScales">
```

```
  <![CDATA[
<criteriaScales id="fuzzy-numbers">
  <criterionScales>
    <criterionID>...<\criterionID>
    <scales>
      <scale id="fuzzy-scales">
        <qualitative>
          <preferenceDirection>min</preferenceDirection>
          <valuedLabels>
            <valuedLabel>
              <label>...</label>
              <value>
                <fuzzyNumber>
                  <piecewiseLinear>
                    <segment>
                      <head>
                        <abscissa>
                          <real>...<\real>
                        <\abscissa>
                        <ordinate>
                          <real>...<\real>
                        <\ordinate>
                      <\head>
                      <tail>
                        <abscissa>
                          <real>...<\real>
                        <\abscissa>
                        <ordinate>
                          <real>...<\real>
                        <\ordinate>
                      <\tail>
                    <\segment>
                    <segment>
                      <\head>
                      <tail>
                        <abscissa>
                          <real>...<\real>
                        <\abscissa>
                        <ordinate>
                          <real>...<\real>
                        <\ordinate>
                      <\tail>
                    <\segment>
                    <segment>
                    [...]
                    <\segment>
                    <segment>
                    [...]
                    <\segment>
                  <\piecewiseLinear>
                <\fuzzyNumber>
              <\value>
            <\valuedLabel>
            <valuedLabel>
              [...]
            </valuedLabel>
            [...]
          </valuedLabels>
        <\qualitative>
      <\scale>
    <\scales>
```

```
99          <\criterionScales>
100       <\criteriaScales>
101         ]]>
102       </xmcda>
103     </input>
104
105     <output displayName="COG" name="defuzzificationCOG" id="output0">
106       <documentation>
107       <description>Result obtained from apply COG calculation for each alternative and their
              fuzzy label result.</description>
108       </documentation>
109       <xmcda tag="alternativesValues"/>
110     </output>
111     <output displayName="COM" name="defuzzificationCOM" id="output1">
112       <documentation>
113       <description>Result obtained from apply COM calculation for each alternative and their
              fuzzy label result.</description>
114       </documentation>
115       <xmcda tag="alternativesValues"/>
116     </output>
117     <output displayName="Ordinal" name="defuzzificationORD" id="output2">
118       <documentation>
119       <description>Result obtained from apply Ordinal calculation for each alternative and their
               fuzzy label result.</description>
120       </documentation>
121       <xmcda tag="alternativesValues"/>
122     </output>
123
124     <output displayName="messages" name="messages" id="output3">
125       <documentation>
126       <description>A status message.</description>
127       </documentation>
128       <xmcda tag="methodMessages"/>
129     </output>
130
131   </parameters>
132 </program_description>
```

A.5 fuzzyLabelsDescriptors description

```
<?xml version="1.0" encoding="UTF-8"?>
<program_description xsi:noNamespaceSchemaLocation="http://sma.uni.lu/d2cms/ws/_downloads/description.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <program name="FuzzyLabelsDescriptors" provider="URV" version="4.0" displayName="FuzzyLabelsDescriptors"/>
  <documentation>
    <description>Two types of uncertainty in fuzzy sets are recognized: (1) specificity, related to the measurement of imprecision, which is based on the cardinality of the set, and (2) fuzziness, or entropy, which measures the vagueness of the set as a result of having imprecise boundaries. Specificity and fuzziness refer to two different characteristics of fuzzy sets. Specificity (or its counterpart, non-specificity) measures the degree of truth of the sentence: Containing just one element. Fuzziness measures the difference from a crisp set.</description>
    <contact>Aida Valls &lt;aida.valls@urv.cat&gt;</contact>
    <reference>A. Valls, The Unbalanced Linguistic Ordered Weighted Averaging Operator, In: Proc. IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2010, IEEE Computer Society, Barcelona, Catalonia, 2010, pp. 3063-3070.</reference>
  </documentation>
  <parameters>
    <input displayName="fuzzyNumbers" name="fuzzyNumbers" id="input0" isoptional="0">
      <documentation>
      <description>Definition of the fuzzy sets associated to the linguistic variable used for all the criteria. The semantics of the linguistic labels are given by a trapezoidal membership function, each membership function is represented as a group of four consecutive segments, each segment is descrived with two 2D points (head, tail). Into this version exists the possibility to define a segment using the tail of the previous segment (only in the same label) as the head point of the new one, avoiding the possibility of continuity errors of the segments. If the second segment is equal to third segment, the fuzzy is triangular. The values must be ordered increasingly. For each label in the linguistic domain (categoriesValues list), a fuzzy set must be defined. The labels must be ordered from the worst to the best performance (fi. Low, Medium, High, Perfect).</description>
      </documentation>
      <xmcda tag="criteriaScales">
        <![CDATA[
      <criteriaScales id="fuzzy-numbers">
        <criterionScales>
          <criterionID>...<\criterionID>
          <scales>
            <scale id="fuzzy-scales">
              <qualitative>
                <preferenceDirection>min</preferenceDirection>
                <valuedLabels>
                  <valuedLabel>
                    <label>...</label>
                    <value>
                      <fuzzyNumber>
                        <piecewiseLinear>
                          <segment>
                            <head>
                              <abscissa>
                                <real>...<\real>
                              <\abscissa>
                              <ordinate>
                                <real>...<\real>
                              <\ordinate>
                            <\head>
                            <tail>
                              <abscissa>
```

```
                          <real>...<\real>
                        <\abscissa>
                        <ordinate>
                          <real>...<\real>
                        <\ordinate>
                      <\tail>
                    <\segment>
                    <segment>
                      <\head>
                      <tail>
                        <abscissa>
                          <real>...<\real>
                        <\abscissa>
                        <ordinate>
                          <real>...<\real>
                        <\ordinate>
                      <\tail>
                    <\segment>
                    <segment>
                    [...]
                    <\segment>
                    <segment>
                    [...]
                    <\segment>
                  <\piecewiseLinear>
                <\fuzzyNumber>
              <\value>
            <\valuedLabel>
            <valuedLabel>
              [...]
            </valuedLabel>
            [...]
          </valuedLabels>
        <\qualitative>
      <\scale>
    <\scales>
  <\criterionScales>
<\criteriaScales>
  ]]>
</xmcda>
</input>

<output displayName="specificity" name="specificity" id="output0">
  <documentation>
  <description>Result obtained from apply Specificity calculation for each fuzzy Number from
       the list.</description>
  </documentation>
  <xmcda tag="alternativesValues"/>
</output>
<output displayName="fuzziness" name="fuzziness" id="output1">
  <documentation>
  <description>Result obtained from apply Fuzziness calculation for each fuzzy Number from
      the list.</description>
  </documentation>
  <xmcda tag="alternativesValues"/>
</output>

<output displayName="messages" name="messages" id="output2">
  <documentation>
  <description>A status message.</description>
  </documentation>
```

```
99        <xmcda tag="methodMessages"/>
100     </output>
101
102   </parameters>
103 </program_description>
```