EXPLORATORY RESEARCH

D3.2 Metamodels Requirements Specification

Deliverable ID:	D3.2
Dissemination Level:	CO
Project Acronym:	NOSTROMO
Grant:	892517
Call:	H2020-SESAR-2019-2
Topic:	SESAR-ER4-26-2019
Consortium Coordinator:	CRIDA
Edition date:	18/02/2021
Edition:	00.01.00
Template Edition:	02.00.02

Founding Members

EUROPEAN UNION EUROCONTROL

SESAR JOINT UNDERTAKING

Authoring & Approval

Authors of the document

Name/Beneficiary	Position/Title	Date
Francisco Antunes	DTU	02/12/2020

Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Javier Poveda Barbero	CRIDA	02/12/2020
Christoffer Riis	DTU	02/12/2020
David Mocholí	NOMMON	02/12/2020
Francisco Pereira	DTU	02/12/2020
Sandrine Molton	ISA	03/12/2020

Approved for submission to the SJU By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date
Mayte Cano	CRIDA	07/12/2020
Ricardo Herranz	NOMMON	Silent Approval
Gerard Gurtner	UoW	Silent Approval
Jordi Pons Prats	UPC	Silent Approval
Francisco Camara	DTU	07/12/2020
Sandrine Molton	ISA	Silent Approval

Rejected By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date
N/A		

Document History

Edition	Date	Status	Author	Justification
00.00.01	02/11/2020	Draft	DTU	Document creation for internal review
00.00.02	04/11/2020	Draft	DTU	Version for partners approval
00.00.03	07/12/2020	Final Draft	DTU	Version for SJU delivery
00.01.00	18/02/2021	Final	DTU	Version approved by SJU

NOSTROMO

NEXT-GENERATION OPEN-SOURCE TOOLS FOR ATM PERFORMANCE MODELLING AND OPTIMISATION

This deliverable is part of a project that has received funding from the SESAR Joint Undertaking under grant agreement No 892517 under European Union's Horizon 2020 research and innovation programme.

Abstract

This deliverable aims to specify the simulation metamodeling framework's main technical and modeling requirements to be deployed within WP3. Several considerations are drawn concerning the active learning strategy to be implemented, which eventually requires a constant link between the metamodel, the simulation model, and the data repository.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose of the Document

Simulation metamodels [1] are essentially input-output functions that approximate the true, usually much more complex, unknown function inherently defined by the simulation model. Such models are characterized by simple, functional formulations and evaluation speeds, allowing for a significant reduction of the computational burden associated with the intense exploration of simulation models' output behavior.

According to [2], simulation metamodels can be employed in four different situations, depending on their ultimate goal, namely 1) problem entity understanding, 2) simulation output prediction, 3) optimization, and finally, 4) verification/validation. This task will mostly focus on the first and second goals. Hence, the assumption is that the simulation model is perfectly verified/validated, optimized, and calibrated with respect to the studied real-world system.

Furthermore, Active Learning (AL) [3][4] schemes will be employed on top of the metamodeling framework. This learning paradigm aims to attain high accuracy performance with a few data points as possible. It proves to be particularly relevant for large-scale simulation models involving dozens of input variables and exhibiting reasonably high runtimes. In this sense, when labeled data is computationally expensive to obtain, AL can be used to minimize the acquisition costs (i.e., running the simulation model) while, on the other hand, aiming for a high prediction performance.

Despite their obvious advantages, simulation metamodels are not exempted from their particular drawbacks. As mentioned by [5], when the number of simulation input variables is too large, the required computational costs and complexity of the metamodels might not be worthy from a modeling perspective anymore. The well-known "curse of dimensionality" [6] is also a problem for metamodeling approaches. On the other hand, and within simulation experimental settings, active learning requires, by its turn, a constant communication link between the learning metamodel and the simulation model to work properly.

Several minimal requirements are considered and specified within this deliverable to address the above-mentioned issues and related conditions. These are meant to ensure that the proposed active learning simulation metamodeling strategy works as expected.

Due to the preliminary and exploratory nature of the underlying metamodel methodology, this deliverable is a living document that will be modified throughout the project to ensure that it is continuously aligned with its objectives. The final version will eventually be integrated as a part of Deliverable 3.4.

1.2 Intended readership

This document is intended to be used by SESAR JU and NOSTROMO members.

1.3 Terminology and Acronyms

Term	Acronyms
AL	Active Learning
API	Application Programming Interface
ATM	Air Traffic Management
CSV	Comma-Separated Values
ER	Exploratory Research
KPA	Key Performance Area
KPI	Key Performance Indicator
NOSTROMO	Next-generation Open Source Tools for peRfOrmance Modelling and Optimisation
OS	Operating System
SESAR	Single European Sky ATM Research Programme
SJU Work Programme	The programme which addresses all activities of the SESAR Joint Undertaking Agency.
SESAR Programme	The programme which defines the Research and Development activities and Projects for the SJU.
SQL	Structured Query Language

2 Communication Link

Active Learning is a special case of machine learning in which the learning algorithm, or model, can interactively query a label provider, such as a simulation model. In this experimental setting, the labels ultimately refer to the simulation model's simulation output values.

As expected, this learning interaction requires a constant and active communication link between the learner and the label provider. There are two important data-related entities within active learning schemes: the labeled and unlabeled pools. Whereas the former essentially represents the training set, the latter refers to the input search space where the simulation model's behavior is being approximated. The unlabeled data points selected to be labeled by the simulation model are then added to the labeled pool, thereby expanding it interactively. A pictorial representation of this cyclic link is represented in Figure 1.

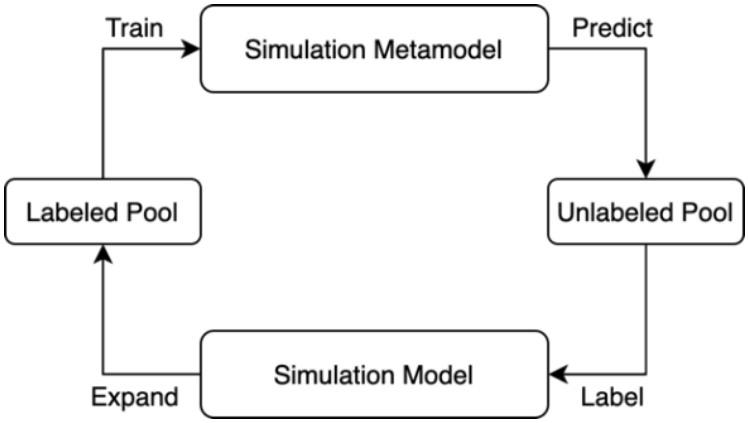

Figure 1 - Communication link between the simulation model and the metamodel.

Depending on the type of implementation, simulation models may use different strategies to access and store data, which include input data, output results, parameters, amongst other types. Databases, simple text files, or APIs are examples of data management that simulation designers can use. This, however, does not significantly alter the modeling algorithm per se but slightly affects the way it queries the simulation model for new labeled data points. Figure 2 summarizes a possible simulation request and output reading scheme.

Furthermore, although recommended, mainly to reduce communication lags, both the simulation and active learning models need not be in the same environment. Hence, if necessary, the simulation requests might be performed remotely.

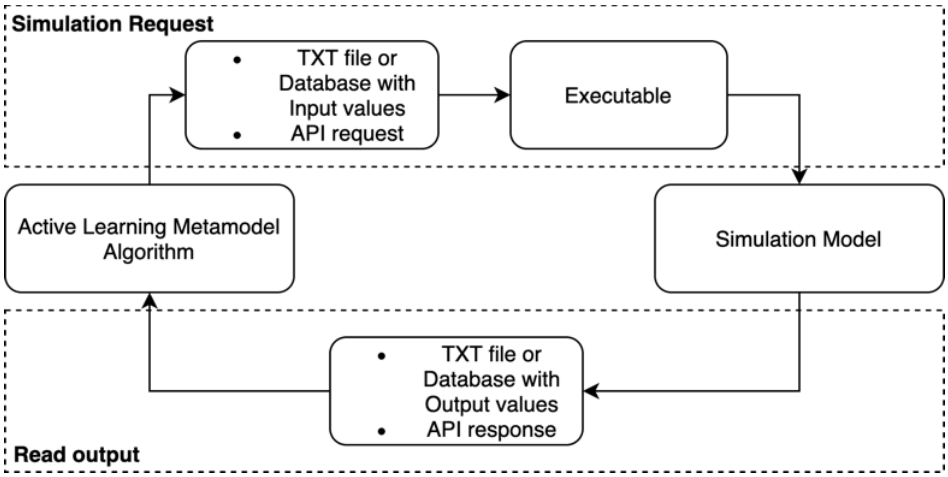

Figure 2 - Simulation run request and output reading flow.

The essential idea of active learning is to minimize the number of simulation runs otherwise required with its absence, making the communication link between both entities crucial in any active learning metamodeling approach.

Nevertheless, this can be remedied somehow using a fixed data set encompassing the results of previously conducted simulation experiments. The data set can then be split into the training and the prediction pools. The latter will also play the simulation model's role, consequently working as a proxy for the label provider. This setting is far from ideal: it tends to render the active learning scheme useless since all the computational effort has already been utilized a priori. Thus, no efficiency is actually being achieved. In any case, it can be used for comparison and benchmarking purposes.

3 Input/Output Variable Selection

Traditionally, simulation models encompass a multitude of input variables and a set of output variables used to assess the performance of the underlying system being studied. This is particularly evident in the cases of Flitan and Mercury ATM simulators used in this project.

It is virtually unfeasible to consider the entire range of input/output variables simultaneously from a metamodeling perspective. This would make the metamodel less interpretable and computationally demanding due to the high dimensionality, eventually rendering it less attractive as a base framework for an active learning strategy. Hence, to address this shortcoming, it is essential to select a manageable subset of input variables varying within reasonable value windows (see Figure 3.2.3). The output dimension does not pose such a similar modeling hindrance since it is on the simulation input space that the exploration process is conducted and in which the simulation model is queried over.

Different subsets of input variables, not necessarily disjoint and possibly associated with different planning scenarios and solutions, should be considered. Ultimately, these variables should represent the factors with the most significant predictive impact on the KPI's of interest within the decision-making process. For this reason, domain knowledge and expert advice are crucial for the variable selection procedure. One metamodel can be developed per variable subset defined within each scenario. The different subsets can also be used for benchmark and comparison purposes, and the combination of variables, disjointly defined within each scenario, is also a possibility.

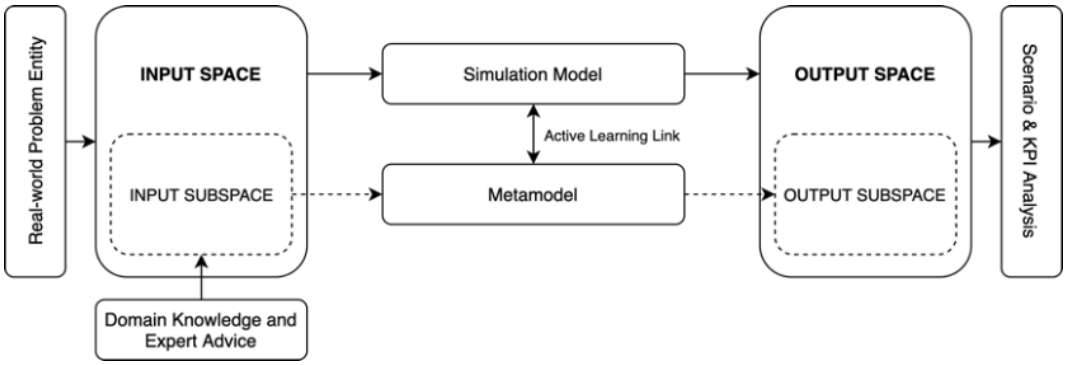

Figure 3 - Input-Output Subspaces used for metamodeling.

4 Simulation Models' Documentation

Well-documented simulation models constitute a crucial component within the metamodeling framework. Although most of these approaches treat the simulation models as pure black-boxes, external expert domain knowledge and guidance is highly advisable, as it can significantly ease the ramp-up of the modeling process.

Such documentation should include, to the greatest extent possible, a detailed list of the most relevant input/output variables as well as a set of features that characterize them from both mathematical and interpretation perspectives:

- Type of variable, i.e., continuous vs. discrete.
- Possible range of values (theoretically).
- Accepted range of values (in practice).
- Impossible combination of values (both theoretically and in practice).
- Default range of values (e.g., the "average" standard value).
- Diagram of causal dependency, not only between input and output variables but also within the input and output dimensions.
- Brief description and interpretation within the entire simulation environment.

5 Requirements Specification

In theory, and purely from a modeling point-of-view, most of the metamodels' requirements rely on the correct specification of its inputs and output variables, which in turn generally correspond to subsets of the original simulation model's input and output dimensions, respectively. However, from an implementation perspective, that is to say, in practice, specific technical aspects must be taken into account.

5.1 Data Sets Formatting

Generally, training sets are organized in matrix form and denoted by (X, Y). Here, X is an n x D1 design matrix, with D1 representing the number of input dimensions (or independent variables, regressors) and n the number of observations. Thus, each column represents a different input variable, whereas each line corresponds to the different simulation instances or runs. Similarly, Y is an n x D2 matrix, where n has the same meaning as in the latter, but D2 is now the dimensionally of the output space (or target space, dependent variables). A concrete example: consider a training set with 100 different simulation results, 15 input variables, and three output variables. X contains 100 x 15 elements, whereas Y has 100 x 3. The whole training set (X, Y) encompasses 100 x 18 entries. The result of the metamodel training is, trivially, a trained metamodel with prediction capabilities. Finally, this data set is rewritten to (Xtr, Ytr) to highlight its training purposes.

The prediction stage follows the training stage. Both the testing and validation are omitted in this text, as they share similar specifications as the training stage. During prediction, the values of the dependent variable are not available. Thus, the trained metamodel only receives an unlabeled data set, i.e., a data set containing only the independent variables values, denoted by Xpred. After prediction, an estimative for the unknown Y, Ypred, is obtained. Thus, the generated prediction data set is represented by (Xpred, Ypred). This data set is precisely used to explore the simulation input space by generating predictions for the simulation output results.

The simplest, most straightforward, and hassle-free way to store and manage all the above-mentioned data sets is via plain text-based files, namely, the Comma-Separated Values (CSV) format, using "," as a separator. By adopting such a format, the correspondence between the matrix format and the CSV becomes trivial and direct. Each line corresponds to different simulation runs, whereas each column corresponds to the input and output variable dimensions. For example, in the case of (Xtr, Ytr), and assuming M simulation runs, the corresponding CSV version would be akin to Table 1.

Note, however, that the first column displaying the simulation number is not required in practice. It serves merely for illustration purposes within this report. The same is valid for the values, which were arbitrarily set. By default, the metamodel will automatically identify each line of the CSV as a different simulation result. On the other hand, the headers should identify and distinguish the input and output variables.

Similar CSV structures can be easily derived for the remaining data sets.

Table 1 – Data Set Structure.

	Input 1	Input 2	...	Input D1	Output 1	Output 2	...	Output D2
Simul run 1	100	0	...	0.1	300	10.5	...	50.3
Simul run 2	150	0.5	...	0.2	350	15.8	...	75.8
...	...	...	...	...	...	...	...	...
Simul run M	300	1	...	0.3	400	30.9	...	100.4

5.2 Exploration Limits

Another important requirement is the specification of the input search/exploration regions. This can be achieved by defining specific lower and upper thresholds for each independent input variable, which in turn will depend upon the case studies' nature and details. The corresponding CSV should be similar to the following table.

Table 2 - Exploration Limits Set Structure.

	Lower limit	Upper limit
Input 1	0	300
Input 2	0	1
...	...	...
Input D1	25	125

These limits do not necessarily correspond to the input variables' possible ranges but rather to subsets of these ranges. For example, if Input 1 represents the taxi time (for all flows, for the sake of simplicity), in theory, the possible values will definitely lie in [0, +infinity]. However, for a particular case study, the metamodeling exploration region might be restricted to [100, 300]. This means that values outside this interval are not a priori relevant to the study in question. Note, however, that these input threshold specifications do not serve as inputs for the metamodel itself. Instead, they help define the simulators' input space exploration regions from a modeling and active learning perspective.

5.3 Simulation-Metamodel-Repository Links

As seen in section D3.2, the metamodel requires a communication link to the simulation model. Another important link is the connection to a data repository or database (MySQL, for example), which in turn is used to store and manage both training and prediction (metamodel results) datasets. A dedicated API might not be entirely necessary for the database access, as the metamodel script can natively connect to it and execute SQL-like requests.

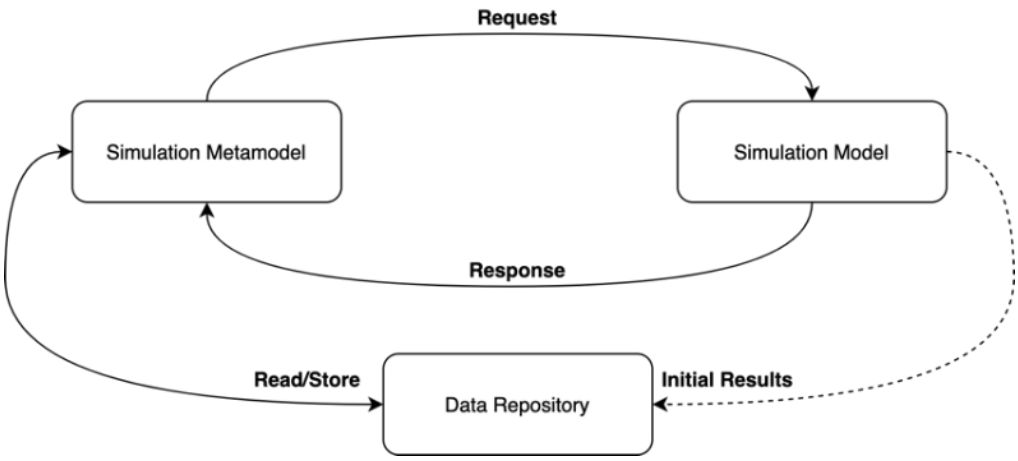

Figure 4 - Links connecting the metamodel, simulation model, and data repository.

The simulation metamodel starts its learning process with an initial set of prior simulation results accessible from the data repository. Afterward, this initial training set gets iteratively expanded via simulation requests as the active learning algorithm evolves. Here notice that a simulation request does not encompass the transfer of data per se but merely queries to run the simulation model with specific input data points locations.

When the stopping criteria are met, the simulation metamodel's final results are stored in the data repository. Figure 4 briefly describes the involved data flows. The dashed line connecting the simulation model and the data repository means that the depicted data flow occurs only once, i.e., it is not part of the cyclic active learning algorithm.

Ideally, the metamodel should request and obtain simulation results via API. If this is not possible, direct manipulation of text-based files, which are then read by the simulator as inputs, constitutes a straightforward compromise.

Additionally, for traceability and logging purposes, each active learning iteration can be stored in the data repository, along with the metamodel's fitting parameters and several prediction performance indicators. This tracing can be provided by the metamodel script after the algorithm has finished, along with the metamodel's results.

5.4 Metamodel High-Level Requirements Summary

As the project evolves, it is expected that some of these requirements might be either expanded or revisited in the future. Most of the potential adjustments are likely to depend on the simulation models' final concrete implementations and data repository, especially in terms of their API specifications. The following table summarizes these requirements.

Table 3 - Metamodel High-Level Requirements Summary.

ID	Description	Comment
NOSTROMO-MM-REQ-01	Metamodel receives a CSV (txt) file containing the input-output simulation results.	Corresponds to the training data set - (Xtr, Ytr).
NOSTROMO-MM-REQ-02	Metamodel receives a CSV (txt) file containing only input variables values.	Corresponds to the prediction set - independent variables only - Xpred's
NOSTROMO-MM-REQ-03	CSV (or simple .txt format) using “,” (comma) as a value separator.	n.a.
NOSTROMO-MM-REQ-04	Headers should be present to identify and distinguish the input from the output variables.	The identification via header can be made, for example, by appending "_input" or "_output" to the end of the variables names.
NOSTROMO-MM-REQ-05	The CSV file lines refer to different simulation runs, and columns refer to the input/output variables used.	n.a.
NOSTROMO-MM-REQ-06	Metamodel provides to the data repository a CSV (txt) file containing the input-output predicted results. Each line refers to a different simulation run, and columns refer to the input/output variables used.	Corresponds to the entire prediction set - (Xpred, Ypred)
NOSTROMO-MM-REQ-07	Metamodel provides to the data repository a CSV (txt) containing the prediction history split by iteration. The structure should be similar to that of NOSTROMO-MM-REQ-06	In practice, it corresponds to each iterative (Xpred_i, Ypre_i), where i is the iteration number. Hence, each CSV file should be generated by iteration. Moreover, the index i should be appended to the end of the file's name using, for example, "_iter_i".
NOSTROMO-MM-REQ-08	Metamodel provides to the data repository a CSV (txt) containing the history of its own parameters, which should vary from iteration to iteration.	The structure of this file is highly dependent on the type of metamodel used. In the case of the simplest approach of Gaussian Processes, only

		two hyper-parameters are used. Hence, an n x 2 matrix would suffice, where n is the number of iterations.
NOSTROMO-MM-REQ-09	Metamodel provides to the data repository a CSV (txt) containing the history of its prediction performance or other kinds of relevant metrics.	This should be an n x K matrix, where n is the number of iterations and K the number of metrics.
NOSTROMO-MM-REQ-10	Each input/output variable should be unidimensional and can be both continuous or discrete.	For example, if an input variable is a bidimensional array containing the mean and standard deviation of a Gaussian distribution, it should be split into two different columns.
NOSTROMO-MM-REQ-11	Metamodel receives the lower and upper search bounds for each input dimension via a CSV (txt).	This data can be represented by a D1 x 2 matrix, where each line corresponds to each input variable. The two columns encompass the lower and upper bounds, respectively. This helps the metamodel to define the input search/exploration space of interest.
NOSTROMO-MM-REQ-12	Metamodel requires a communication link to query (or run) the simulation model in specific input points to obtain the real simulation results (or labels). This can typically be achieved via Application Programming Interface (API) on the simulation model's side.	Each query's result should be in the form of CSV (txt) as in NOSTROMO-MM-REQ-01. This link is crucial for the active learning part of the metamodeling methodology. Many APIs typically work with JSON files. If required, minor adaptations can be made so that the metamodel receives that type of file.
NOSTROMO-MM-REQ-13	Metamodel requires a connection to a database from which it can read the initial simulation results.	The initial simulation results encompass prior simulation experiments and constitute the starting point for the active learning process.

NOSTROMO-MM-REQ-14	Metamodel requires a connection to a database where it can store its results.	The metamodel stores its final results in the database for further analysis.
NOSTROMO-MM-REQ-15	Metamodeling methodology is implemented in Python.	Other scripting languages can be used, such as R or Matlab, although it is improbable.
NOSTROMO-MM-REQ-16	Operating System (OS): Unix/Linux.	As Python is cross-platform, the OS can be changed if any infrastructure requirement demands it. A few minor adjustments are likely to be undertaken.
NOSTROMO-MM-REQ-17	High-level causal dependency among the simulation variables/parameters.	Examples: workflow diagrams, pseudo-algorithms, and dependency graphs.

6 References

[1] L. W. Friedman, The simulation metamodel. Springer Science & Business Media, 2012.

[2] Jack PC Kleijnen and Robert G Sargent. A methodology for fitting and validating metamodels in simulation. European Journal of Operational Research, 120(1):14–29, 2000.

[3] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[4] Xizhao Wang and Junhai Zhai. Learning with Uncertainty. CRC Press, 2016.

[5] Wang, G. Gary; Shan, Songqing. Review of metamodeling techniques in support of engineering design optimization. 2007.

[6] Köppen, Mario. The curse of dimensionality. In: 5th Online World Conference on Soft Computing in Industrial Applications (WSC5). 2000. p. 4-8.