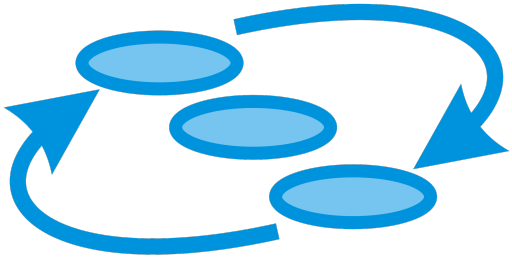
Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs



DSE in the INTO-CPS Platform

Deliverable Number: D5.3e

Version: 1.0

Date: 2017

Public Document

http://into-cps.au.dk

Contributors:

Carl Gamble, UNEW

Editors:

Carl Gamble, UNEW

Reviewers:

Julien Ouy, CLE
Etienne Brosse, ST
Frederik Foldager, AI

Consortium:

Aarhus University	AU	Newcastle University	UNEW
University of York	UY	Linköping University	LIU
Verified Systems International GmbH	VSI	Controllab Products	CLP
ClearSy	CLE	TWT GmbH	TWT
Agro Intelligence	AI	United Technologies	UTRC
Softeam	ST		

Document History

Ver	Date	Author	Description
0.0	2017-08-25	Carl Gamble	Skeleton Structure, content from D5.2d added
0.1	2017-11-02	Carl Gamble	Cloud DSE operation added
0.2	2017-11-05	Carl Gamble	Updated App interface added
0.3	2017-11-06	Carl Gamble	Implementation matrix added, version ready for review
1.0	2017-12-15	Carl Gamble	Internal review comments addressed

Abstract

There are two components to Design Space Exploration (DSE), the methodology of how to conduct a search of the design space and the scripts that then support performing the search. The methodological aspects may be found in deliverable D3.3a [FGP17] while this deliverable focusses on the scripts that actually perform the search. In this deliverable we discuss the needs of the industrial partners of the INTO-CPS project and then describe the scripts that comprise the tool support in terms of their function, status and future plans.

Contents

1 Introduction **6**

2 Related Work **6**

3 INTO-CPS Case Study DSE Goals **8**
3.1 Agro Intelligence 8
3.2 TWT Gmbh 9
3.3 ClearSy 10
3.4 United Technologies 12
3.5 Key Points from the Case Studies 14

4 DSE Component Status and Plans **15**
4.1 Module Overview 15
4.2 DSE Config File 16
4.3 Search Algorithms 25
4.4 COE Handler 27
4.5 Cloud support 27
4.6 Results Presentation 31
4.7 INTO-CPS Application Integration 32
4.8 Analysis Available to Single Simulations 36
4.9 Matrix of Capability Implementation 38

5 Conclusions **38**

6 List of Acronyms **41**

1 Introduction

After an introduction to the related work around DSE (Section 2) and the needs to the Work Package 1 industrial partners (Section 3), this document then focusses on the tool support for DSE, providing a description of the structure, the current status and future plans for DSE as part of the INTO-CPS association (Section 4).

2 Related Work

In the *DESTECS* project[1] DSE was supported by applying Automated Co-model Analysis (ACA), such as parameter sweep. The project also provided support for testing different model implementations. The project provided methodological guidelines for DSE in [BFG+12] and [FLPV13] and tool support for the Crescendo in the form of ACA [NBAR+12]. INTO-CPS will use the methods work from DESTECS as a baseline, extended with wider range of analysis techniques and including closed loop support.

The *Certainty* project[2] uses DSE in the DOL-Critical method; using the results of interference analysis reliability analysis to evaluate potential mapping and scheduling solutions of tasks to cores on multi-core platforms. The project includes several tools: "the EXPO tool is the central module of the framework. As an underlying multi-objective search algorithm, the Strength Pareto Evolutionary Algorithm (SPEA2) is used that communicates with EXPO via the PISA interface" [CER13a]. The project also proposes the "Mixed Criticality Mapping and Scheduling Optimisation (MCMSO) method" which implements a heuristic method based on simulated annealing [CER13b]. Both the implementation and methods developed by Certainty will influence DSE in INTO-CPS, in particular the use of simulated annealing and Pareto Front techniques. It is not clear to what extent this work is 'closed-loop', one focus of DSE in INTO-CPS.

As part of an integrated tool chain for high-level synthesis of high-performance FPGA systems, the *ENOSYS*[3] project uses two tools for DSE: *FalconML*[4] and *Jink*[5]. The Jink Design Space Explorer coordinates a design flow and its

[1] http://www.destecs.org/
[2] http://www.certainty-project.eu/
[3] https://sites.google.com/a/enosys-project.eu/www/home
[4] https://sites.google.com/a/enosysproject.eu/www/enosys-tools/falconml
[5] https://sites.google.com/a/enosys-project.eu/www/enosys-tools/jink

exploration engine searches over various parameters used in customising the soft core multi processor and in partitioning the UML design to the underlying architecture. Jink finally parses over the various logs and reports files produced by the tools during synthesis, compilation, simulation to extract various design characteristics and metrics. This work is limited to FPGA design and it is not clear to what extent this work is 'closed-loop', or which ranking or analysis methods are used. Further investigation is required to determine the extent of influence these outputs may have on INTO-CPS.

The ongoing *AXIOM*[6] project will provide DSE, with a deliverable on DSE due in M24 of the project (January 2017). DSE technologies will be used in the development of the software parts and the selection of the most appropriate hardware architecture and interconnect.

MADNESS[7] use traditional methods for DSE, alongside their "co-exploration" which uses different search algorithms for different dimensions and they report that "multidimensional co-exploration can find better design points and evaluates a higher diversity of design alternatives as compared to the more traditional approach of using a single search algorithm for all dimensions." We have been unable to obtain public deliverables.

In the *iCyPhy* project[8], effort is placed to reduce the design space for DSE through optimal architecture selection [FNSV15]. A routine is defined to optimise the continuous parameters of a CPS to decrease the number of simulations.

The *DARPA AVM META* project[9] defines the CyPhyML for the modelling of CPSs. The project uses the Design Space Exploration Tool (DESSERT) to prune the design space to a "manageable size". INTO-CPS should consider the DESSERT technology and its methods for design space reduction.

The Merlin project[10] produced a suite of tools collectively called the Strategic Decision Making Tool (SDMT) to facilitate exploring the design space within the rail domain. This suite consists of a core tool that orchestrates the DSE, an optimisation tool responsible for driving the DSE through the use of a genetic algorithm and pareto optimality analysis and a costing analysis tool tasked with computing the electrical costs part of the simulation results. This project makes explicit something akin to architectural aspects of the design

[6] http://www.axiom-project.eu
[7] http://www.madnessproject.org/
[8] http://www.icyphy.org/index.html
[9] http://cps-vo.org/group/avm/meta
[10] http://www.ncl.ac.uk/newrail/research/project/4392

space by allowing the use of clusters, where each cluster may have different sets of parameters and constraints but all are ultimately compared using the same objective values.

3 INTO-CPS Case Study DSE Goals

In this section the case study owners present details of their scenarios from four distinct viewpoints. These viewpoints consider the parameters that are present in each scenario, how each scenario will be measured, how the results of each design will be compared to rank the designs and how the results could be presented. These aspects will affect the direction of the DSE module scripts over the next two years of the project.

3.1 Agro Intelligence

3.1.1 Design Parameters

In the agricultural case study, we have two categories of parameters. The first category defines the robot and its operation conditions. The second category defines the parameters of the surrounding environments. The first category has internal dependencies, like total weight, wheel size/operation speed, sensors and control software. But there are also dependencies between the two categories, the wheel slip will affect the operating speed and the surface type will affect the wheel slip. Crop type will affect the width of the robot because the robot needs to fit the row distance for the current crop. Current placement of the robot in the environment has a significant impact on how the controller should operate in terms of movement and operational strategy.

3.1.2 Solution Objectives

The simulation goal for the agro-case is to determine viable candidate robot configurations for a given scenario. The scenarios are defined by the implement, crop, and surface type. The parameters that the robot design should be optimised for are navigation and implement response in the environment. The result should be a list of configuration parameters that can be used in the robot design.

3.1.3 Ranking of solutions

Similarly to the automotive case discussed in Section 3.2.3, the different robot designs will be compared by a cost function. The evaluation criteria of the cost function will be total cost and operational performance of the robot in the given scenario and configuration.

3.1.4 Results Presentation

The result of the simulation should be presented in an interactive manner, where the company developers can select the parameter he/she want to evaluate for a specific scenario. It should be possible to select several parameters, e.g. robot vehicle implement, wheel size, and sensory types. The result should be presented in a list and a graph so the user can see the how the result compares to each other. It should be possible to select the individual simulations to see a detailed description of the scenario.

3.2 TWT Gmbh

3.2.1 Design Parameters

For the automotive case study, two categories of parameters can be differentiated: The first group of design parameters that can be varied during an DSE experiment defines the vehicle: vehicle mass, aerodynamic drag coefficient c_w, rolling friction coefficient c_{rr}, battery capacity C, and the full load curve, defined by the maximum engine speed n_{max} and the maximum torque M_{max}. The second set of parameters defines the route the vehicle takes to get from the start position to its destination. These parameters can be described as a set of coordinates. For a typical DSE experiment in the context of INTO-CPS, the first set of parameters, defining the vehicle, is most likely the more relevant group.

The vehicle design parameters can depend on each other, e.g. the battery capacity has an influence on the total mass.

3.2.2 Solution Objectives

The simulation results that are of most interest for the automotive case study are relatively directly measurable. They include: the maximum acceleration

should not be higher than a specific value (e.g. $4ms^{-2}$), the time that it takes to travel a certain distance, the time it takes to achieve a temperature inside the vehicle within the comfort zone ($T_{min} < T < T_{max}$).

3.2.3 Ranking of solutions

Different designs (i.e. vehicle configurations) are compared by using a cost-function that has parameters such as total vehicle cost, energy consumption, space, and mass. In particular, electric vehicles are optimized using cost functions that include the battery capacity, energy consumption, mass, driving performance, efficiency per component, and energy at the tire. Hybrid vehicles have cost functions that include the battery capacity, power of the electric motor, power of the combustion motor, range, energy consumption, driving performance, and efficiency of components. These cost functions are however individual for each automobile manufacturer and depend on the specific problem that needs to be solved. Therefore, there are no universal cost functions or rules for ranking of results.

3.2.4 Results Presentation

The range of electric vehicles is typically presented as a bar diagram for different vehicle configurations. For hybrid vehicles, the results could be shown in a 3D-plot, with the different working points of the combustion motor as the second parameter axis.

3.3 ClearSy

3.3.1 Design Parameters

For the railway case study, two kinds of parameters can be differentiated. The first group of parameters correspond to real numbers (or function of real numbers) such as Kinetic energy, communication or physical movement delay, track length, track slope (function of position) or traction acceleration (function of speed), or breaking force.
The other group of parameters is rather a choice of decomposition of a whole track map into several distributed one, and the corresponding distributed interlocking. Thus, such parameters are a set of subsets of the track map database tuples. One can also consider a varying number of trains.

The parameters may be related, such as minimal and maximal kinetic energy, or minimal or maximal slope, or traction acceleration.

3.3.2 Solution Objectives

There are two kinds of measurement. The first kind of measurement are extremal values of monitored real number variables such as: train trip delay, kinetic energy, train availability. The other kind of measurement is whether one train will overrun another and collide, or whether two train collide because of an error in the interlocking PLCs. The two kinds may be dependent: one may want to measure availability but only in the case there is no collision.

3.3.3 Ranking of solutions

The preference value for solution objective may be the maximal or minimal value while such and such parameters vary (ex: track map distribution or traction vary and then the simulation tool would try to find the minimal train trip delay).

It is not clear what is the link between train trip delay and train availability. In this case, a curve with at least an extremal value (for instance the availability) in function of train trip delay could be fine (train trip delay computed with the other parameters the number of train, track map decomposition).

3.3.4 Results Presentation

The presentation for simple objective value would be a table or curve with a few significative simulations and showing the extremal objective value(s). For the case of a limit value is overrun (such as maximal allowed speed, overrun of train so positions overrunning, or collision with same), it could be interesting to show similar table or curve in the neighbourhood of this limit value. Finally, it should be interesting to show XY curve for showing tradeoff between two competitive objectives such as availability vs train trip delay.

3.4 United Technologies

3.4.1 Design Parameters

DSE is used in building automation to: (a) identify the optimal equipment and control settings for an existing building; (b) study the equipment scalability over different building thermal characteristics. In the following, we highlight the key design parameters used in the building automation case study:

- *Equipment Design Parameters:* tuning these parameters lead to identify the optimal thermal supply settings to a building using Fan Coil Units (FCUs).
  1. Maximum water flow rate: $\dot{m_{water}} \in [0.08 : 0.12]$
  2. Maximum air flow rate: $\dot{m_{air}} \in [0.4 : 0.6]$
  3. Water coil efficiency: $\epsilon_{coil} \in [0.1 : 1]$
- *Control Design Parameters:* tuning these parameters lead to identify the optimal PID control response to the building thermal load.
  1. Proportional set-point weighting: $K_p \in [0 : 1]$
  2. Derivative set-point weighting: $K_d \in [0 : 1]$
- *Plant Design Parameters:* these parameters are used to express different building heat dissipation characteristics. The building thermal parameters are varied based on ASHRAE fundamentals 2013 standard.
  1. Wall density: $\rho_{wall} \in [960 : 1600]$
  2. Wall thermal conductivity: $\lambda_{wall} \in [0.0865 : 0.1298]$

Considering that these parameters are independent, then the search space for optimizing equipment setting is $\dot{m_{water}} \times \dot{m_{air}} \times \epsilon_{coil} \times K_p \times K_d$. Whereas, the search space for equipment scalability study is $\rho_{wall} \times \lambda_{wall}$.

3.4.2 Solution Objectives

The objective of a building automation system is to maintain the user comfort, while minimizing the energy consumption. In this case study, the user comfort is represented as the room air temperature RAT, whereas more comfort metrics can be taken in account, such as CO_2 and humidity. The

automation system maintains the RAT in the comfort band identified as $RAT_{sp} \pm 1^oC$, where RAT_{sp} is the RAT set-point identified by the user or the building manager. Therefore, evaluating these performance metrics requires observation of the following dependent variables:

1. Room Air Temperature RAT
2. Room Air Temperature Set-point RAT_{sp}
3. Supplied Power Q_{in}

3.4.3 Ranking of solutions

In order to rank the search space solutions, we formulate two evaluation metrics as follows:

- User discomfort UD is calculated as the area between RAT and RAT_{sp} curves. In order to minimize the user discomfort, the RAT needs to respect the RAT_{sp}. RMSE (Root Mean Square Error) is used to quantify the user discomfort as follows, where N is the total number of samples:

$$UD = \sqrt{\frac{\sum_{k=1}^{N} \left[RAT_{sp}(k) - RAT(k)\right]^2}{N}} \tag{1}$$

- Energy Consumption E is calculated as the integration of the used power Q_{in} over the time. In order to calculate the energy consumption, Coefficient Of Performance (COP) of the heat pump (HP) required to be considered as follows (i.e. COP=2.6), where T is the sample duration :

$$E = \sum_{k=1}^{N} \frac{Q_{in}(k) * T}{COP} \tag{2}$$

Optimizing the building performance requires minimizing both metrics $Min(UD, E)$. However, decreasing one of them leads to increase the other, therefore a Pareto frontier is required to be evaluated in order to identify the optimal design parameters.

3.4.4 Results Presentation

Considering the building automation is a multi-objective optimization problem, then we present the DSE results in an nD plot, where n is the number

of the optimization criteria. In our case study, we optimize the building automation against two optimization criteria, i.e. E, UD. Therefore, the search space will be presented in a $2D$ plot that captures E and UD values for different configurations.

3.5 Key Points from the Case Studies

There are many key points from these descriptions that we can extract and take into account during the future development of the DSE model scripts.

- Each of the scenarios has a mixture of both design parameters of the system itself and parameters defining the environment in which the system is to operate. While these both contribute to the total design space to be explored it is important that we differentiate between them when, for example, grouping results by design.
- There are sometimes relations between simulation parameters meaning that not all combinations are valid, for example in the AI case study, the surface type of the ground affects the wheel slip parameter and in the TWT case study various parameters of the vehicles are linked, such as the battery capacity of an electric vehicle and its total mass. The parameter sweep should only visit parameter sets that respect these constraints.
- There is a range of different complexity levels when processing the raw simulation results to derive the objective measures needed to assess each simulation. Some are instantaneous measures that may be directly provided by the simulation outputs, such as the maximum acceleration of a vehicle, which others require more complex assessment. Examples include the time taken for the car cabin temperature to reach a comfortable level, the cumulative occupant comfort level in the UTRC scenario and computation of the turning radius in the AI study.
- There are also constraints over the variables in the simulations that must not be breached, an example of this is the detection of collision of two trains in the CLE case study. Such a constraint results in a boolean pass or fail that should be recorded amongst the objective results. It may be advantageous to terminate a simulation when a constraint is breached to reduce wasted CPU time but this is outside the current planned capabilities of the DSE module.

- The UTRC case study explicitly calls for a pareto optimal type analysis to compare and rank the design results the TWT case study calls for cost functions that take into account multiple design parameters and simulation results and are unique to each vehicle simulated.
- In terms of presentation the case studies propose a range of visualisations from being able to select a range of graph types such as bar graphs, 2 dimensional plots and 3 dimensional plots. These plots could show a range parameter and results or focus in on interesting areas such as the neighbourhood around the maximum speed of a train. These are alongside the ability to compare any two values on an XY plot style and also pareto optimal style plots.

4 DSE Component Status and Plans

4.1 Module Overview

The DSE module comprises multiple python scripts, configuration files, analysis files and files used to store DSE progress and results. Figures 1 & 2 show the entities associated with DSE. Within an INTO-CPS project folder, the primary element is the *dseConfig* which contains details defining the design space to be searched, how to search it, how to measure the results and how to compare results. Paths to the *dseConfig* and the *mm.json* file describing the model to be used are passed to the *search algorithm* script to start the DSE process. The *search algorithm* creates a *date-time* folder to store results and creates a subfolder for each simulation it needs to run. The subfolder contains a *config* file containing the complete multi-model details including the specific parameters for that simulation. A *COE handler* script sends the contents of the *config* files to the COE, launches the simulation, retrieves the raw result and saves it in the *results.csv* file.

With the raw results in place, the *search algorithm* invokes the analysis defined in the *dseConfig* which making use of the built in *simple objective* scripts and/or user defined scripts, represented here by *analysis.py*. In the case of the user defined analysis, the script is passed a path to a scenario folder, in this case *scen. 1* in which it will find a data file related to that scenario if needed, such as the map for a line following robot, here represented by *analysis data*. Both types of analysis store their results (objective values) in the *objectives.json* file appropriate for the simulation. The results in the *objectives.json* files are used by the *ranking script*, to compare all results,

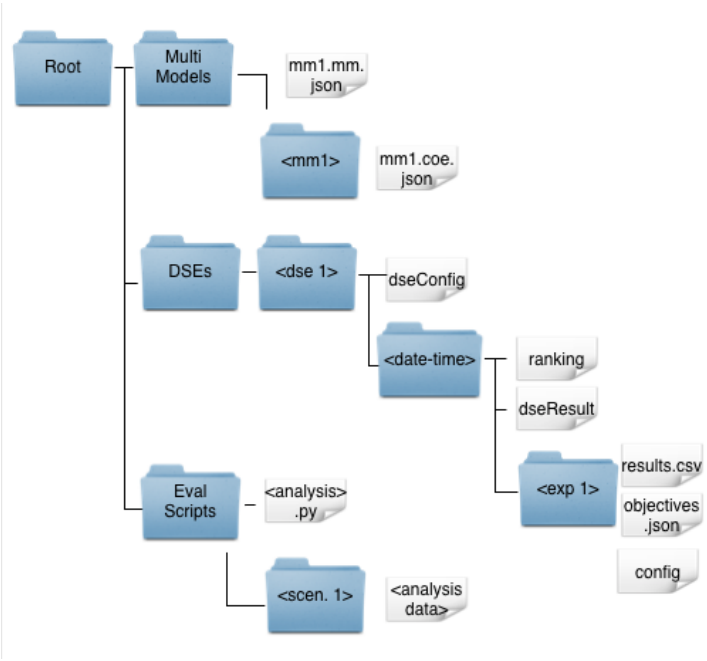


Figure 1: Folders and file used by DSE in an INTO-CPS project

with the resulting rankings over the whole DSE experiment being stored in the *ranking* file. Finally, the stored rankings are used by the *output* scripts which generates the *dseResult* that is presented to the user.

4.2 DSE Config File

The *<name>.dse.json* files contain seven sections, each defining a separate part of the DSE to be performed, Figure 3, these will now be explained.

4.2.1 Parameters

Parameters are core to defining the space over which a DSE should search. Figure 4 shows an example of parameter definition for the line follower robot. Parameters are currently defined explicitly as a list.

The future plan here is to allow a set comprehension such as defining lower value, upper value and step.

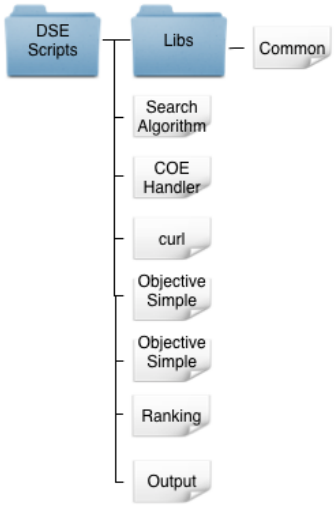


Figure 2: Outline scripts in the DSE module

Parameter Constraints Parameter constraints allow the user to define conditions that must be true for a set of design parameters to be valid and therefore worth simulating. In the line follower example, Figure 4, we define parameters that represent the co-ordinates of the left and right line follow sensors. Each sensor has two possible x and y coordinate values, giving each sensor 4 possible position and therefore 16 possible combinations of the two sensors. Only four combinations of the sensor positions represent symmetrical designs and so the engineer may want to apply constraints so that the non-symmetrical designs are not simulated. Figure 5 shows the constraints that ensure the line follow sensors only adopt symmetrical positions.

These constraints are written as a list of boolean equations in python, which are processed to map the names given to the data structure used in the DSE scripts and then passed to the Python *eval* method where it is evaluated. This method of defining parameter constraints has been tested using the line follower model and the three water tank model and has proven itself to be adequate in these cases. There are no immediate plans to develop this feature further unless shown to be needed by any of the case studies.

4.2.2 Objectives

Objective is the name given to the characterising metrics we may derive from the raw simulation results [Deb12]. There are two methods available

```
{
    "algorithm": {},
    "objectiveConstraints": {},
    "objectiveDefinitions": {
        "externalScripts": {
            "lapTime": {
                "scriptFile": "lapTime.py",
                "scriptParameters": {
                    "1": "time",
                    "2": "{bodyFMU}.body.robot_x",
                    "3": "{bodyFMU}.body.robot_y",
                    "4": "studentMap"
                    }
                },
            "meanCrossTrackError": {
                "scriptFile": "meanCrosstrackError.py",
                "scriptParameters": {
                    "1": "{bodyFMU}.body.robot_x",
                    "2": "{bodyFMU}.body.robot_y"
                    }
                }

        },
        "internalFunctions": {}
    },
    "parameterConstraints": [
        "{sensor1FMU}.sensor1.lf_position_y == {sensor2FMU}.sensor2.lf_position_y",
        "{sensor1FMU}.sensor1.lf_position_x == - {sensor2FMU}.sensor2.lf_position_x"
    ],
    "parameters": {
        "{sensor1FMU}.sensor1.lf_position_x": [
            0.01,
            0.03
        ],
        "{sensor1FMU}.sensor1.lf_position_y": [
            0.07,
            0.13
        ],
        "{sensor2FMU}.sensor2.lf_position_x": [
            -0.01,
            -0.03
        ],
        "{sensor2FMU}.sensor2.lf_position_y": [
            0.07,
            0.13
        ]
    },
    "ranking": {
        "pareto": {
            "lapTime": "-",
            "meanCrossTrackError": "-"
        }
    },
    "scenarios": [
        "studentMap"
    ]
}
```

Figure 3: An example DSE config file for the line follower robot

```
"parameters": {
    "{sensor1FMU}.sensor1.lf_position_x": [
        0.01,
        0.03
    ],
    "{sensor1FMU}.sensor1.lf_position_y": [
        0.07,
        0.13
    ],
    "{sensor2FMU}.sensor2.lf_position_x": [
        -0.01,
        -0.03
    ],
    "{sensor2FMU}.sensor2.lf_position_y": [
        0.07,
        0.13
    ]
},
```

Figure 4: Example parameters from the line follower robot example

```
"parameterConstraints": [
    "{sensor1FMU}.sensor1.lf_position_y == {sensor2FMU}.sensor2.lf_position_y",
    "{sensor1FMU}.sensor1.lf_position_x == - {sensor2FMU}.sensor2.lf_position_x"
],
```

Figure 5: Parameter constraints

for evaluating objectives and we refer to these as internal, so called because it is built in to the DSE module, and external analysis, so called because it is not part of the supplied DSE scripts but is developed by the engineer and may make use of data and applications external to the DSE scripts.

Internal analysis allows the user to define objectives that are based upon simple functions of individual variables logged during simulation. These functions are currently limited to finding the maximum, minimum or mean value of some logged variable. Such functions could be used to evaluate an objective such as finding the peak measurement of the current drawn through a component or the minimum fluid level measured in a water tank. Figure 6 shows an example of how internal analysis is declared. Each instance of the analysis is declared within the *internalFunctions* section. The definition begins with the name of the objective which is also the name that will be used to record the objective in the *objectives.json* file. The definition contains two fields, *columnID* which defines the name of the variable to be tracked and *objectiveType* which states the function to be applied to that column of data where the options are *max*, *min* and *mean*.

External analysis permits the user to define their own methods for calculating objectives for use during DSE. The definition is in two parts, the script that performs the calculation and the definition of the *dseConfig* of the parameters needed to invoke the script. Figure 7 shows the definition of two external

```
"energyConsumed": {
    "columnID": "{bodyFMU}.body.total_energy_used",
    "objectiveType": "max"
    }
}
```

Figure 6: Definition of an internal objective which captures the energy consumed during a simulation

```
"externalScripts": {
    "lapTime": {
        "scriptFile": "lapTime.py",
        "scriptParameters": {
            "1": "time",
            "2": "{bodyFMU}.body.robot_x",
            "3": "{bodyFMU}.body.robot_y",
            "4": "studentMap"
            }
        },
    "meanCrossTrackError": {
        "scriptFile": "meanCrosstrackError.py",
        "scriptParameters": {
            "1": "{bodyFMU}.body.robot_x",
            "2": "{bodyFMU}.body.robot_y"
            }
        }

},
```

Figure 7: Definition of external objectives for the line follower robot

analysis that evaluate the lap time and cross track error for the line following robot. Each analysis definition is given a name and contains two fields, the *scriptFile* field contains the name of the python script containing the analysis and the *scriptParameters* field contains and ordered list of arguments to be passed to the script when launched. The parameters passed can contain any string and in the example they contain the name of variables to be used by the script and also a constant value which represents a target value to be used in the cumulative deviation script. When launching an external script there are three arguments that are passed by default and before those the user defines. These arguments pass the script its objective name, the path to the folder containing the simulation results and the path to a scenario folder.

Figure 8 shows an example of an external analysis script that calculates the cumulative deviation of the water level from a target level in the water tank example. The script can be separated into two parts. The common part can be used in all scripts and it extracts the objective name, results folder path and scenario folder path, it also contains methods to determine the column index for any variables it should use and a method to write the final objective result into the *objectives.json* file. The second part of the script contains the

scripting that is required to compute the associated objective value and to extract the arguments added by the user in the definition.

There are no plans currently to develop the internal analysis at this point unless the need for more simple functions is derived from any of the case studies. There are plans to develop the external analysis in two ways. First, the common part of the analysis is to be made a library that can be imported to reduce the need to copy and paste code. The second is with regard to the values returned by an objective. It has been observed that it is not always possible to return a value from the analysis function that has meaning, for example, if the line follower robot fails to complete a lap in the simulation time it is not possible to calculate the lap time. To this end we will introduce keyword values that may be returned to indicate that it was not possible to compute a value and this will be used in place of the very large default values that have been used up till this point.

4.2.3 Objective Constraints

Objective constraints are not yet implemented but it is expected that they will follow the same structure as used for parameter constraints. These will be boolean expressions over objectives values defining conditions that must be true for a result to be considered further. Such expressions could include objective values being considered against constant values, such as '$maximum\_power < 200$' based upon multiple objectives such as '$max\_deviation\_A + max\_deviation\_B < 10$'. These constraints may be used both by closed loop DSE algorithms so that bad results are not considered further and also in the presentation of results so only acceptable results are presented.

4.2.4 Ranking

Ranking is the process of determining the relative fitness of the simulated designs to meet some goals of the user. There is a single ranking method implemented at this time, this is the Pareto method which returns a non-dominated set of results that represent the set of best compromises of a pair of objectives. The non-dominated set contains all the points where it is not possible to improve the value of one objective without degrading the other. Figure 9 shows the results of Pareto ranking the results of running a DSE on the water tank example, with the non-dominated set shown in

```python
import csv,os, sys, json, io

def getColumnFor(colName, row):
    index = 0
    for thisName in row:
        if thisName.strip() == colName.strip():
            return index
        else:
            index +=1
    return index

def writeObjectiveToOutfile(key, val):
    parsed_json = {}
    if os.path.isfile(objectivesFile):
        json_data = open(objectivesFile)
        parsed_json = json.load(json_data)
    parsed_json[key] = val
    dataString = json.dumps(parsed_json, sort_keys=True,indent=4, separators=(',', ': '))
    with io.open(objectivesFile, 'w', encoding='utf-8') as f:
        f.write(unicode(dataString))

resultsFileName = "results.csv"
resultsFile = sys.argv[1] + os.path.sep + resultsFileName
objectivesFileName = "objectives.json"
objectivesFile = sys.argv[1] + os.path.sep + objectivesFileName
objectiveName = sys.argv[2]
scenarioDataFolder = sys.argv[3]
csvfile = open(resultsFile)
csvdata = csv.reader(csvfile, delimiter=',')
```

Common Section

```python
levelColumnID = sys.argv[4]
targetLevel = float(sys.argv[5])

cumulativeDeviation = 0.0
levelColumn = 0
stepSizeColumn = 0
firstRow = True

for row in csvdata:
    if firstRow:
        levelColumn = getColumnFor(levelColumnID, row)
        stepSizeColumn = getColumnFor('step-size', row)
        firstRow = False
    else:
        level = float(row[levelColumn])
        stepSize = float(row[stepSizeColumn])
        cumulativeDeviation += abs ((level - targetLevel)*stepSize)

writeObjectiveToOutfile(objectiveName,cumulativeDeviation)
```

Script Specific Section

Figure 8: Definition of an internal objective which captures the energy consumed during a simulation

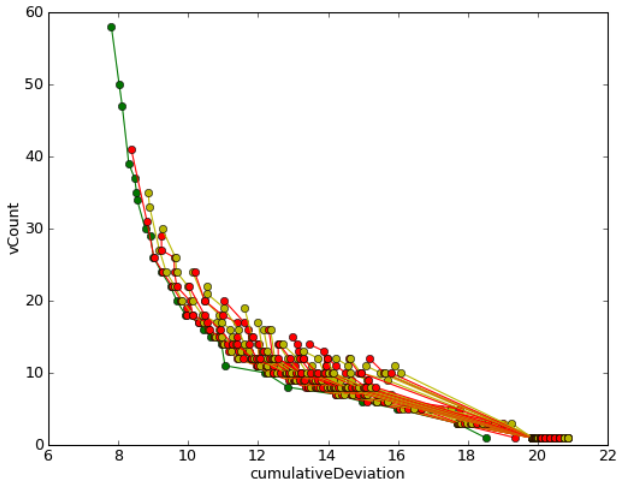


Figure 9: The results of running a DSE on the water tank model, the Non-dominated set represented in green

green. Here the objectives *vCount* and *cumulativeDeviation* were being used to rank designs with the goal of minimising both values.

The current implementation of the Pareto ranking method supports ranking results based upon two objective values where they can independently have their preferred direction (maximising or minimising) defined. The plan here is to extend the current algorithm to support n variables since we can imagine wanting to perform a ranking of the *quality, cost & delivery* type.

Pareto is of course not the only type of ranking that a user may want to utilise, indeed TWT expressed a desire to be able to define a cost function for ranking (Section 3.2.3). Thus ranking efforts as the INTO-CPS association starts will be to implement methods to support such ranking functions. It is not decided yet what implementation method(s) will be used to extend this functionality. If the cost functions take the form of a single equation such as the Weighted Additive Method [Bak05] then allowing the user to write the equation in the *dseConfig* and then processing it using the Python *eval* appears feasible, however if the cost functions are more complex such as the Enumeration and Scoring method [Bak05] then *eval* may not be suitable and an approach similar to the external analysis scripts used for measuring objectives may permit the user the flexibility they need.

```
"scenarios": [
    "studentMap"
]
```

Figure 10: Specification of scenario names to be explored

4.2.5 Scenarios

Scenarios permit the user to make use of different text environments and conditions when assessing the performance of a system under test, for example we may want to evaluate performance of the line following robot using a variety of tracks that assess different capabilities or in the case of a vehicle model we may want to assess how it performs with different driver models.

There are multiple ways in which scenarios may be changed in an INTO-CPS multi-model depending on how and where the scenario is defined in the model. Changing a scenario may change any or all of the following:

FMU parameter(s) If an FMU contains aspects of more than one scenario such as a line follow sensor containing the data for more than one map, it may possible to select which map is used via a parameter that may be set using the same mechanism as used for design parameters.

FMU implementation An alternative to FMUs containing data for multiple scenarios is to use multiple FMUs each representing one scenario. This is the current method used for the line follow robot sensors. Here scenarios would be changed by changing the instance of FMU referenced in the multi-model.

Multi-Model configuration While there have, so far, been no examples of this in any of the pilot studies, it is conceivable that a different scenario may require the use of a different multi-model.

External analysis data file Finally, the external analysis scripts may require data specific to a scenario to support their analysis. For example the cross track error analysis for the line follow robot requires a data file containing a representation of the path to be followed.

The DSE configuration contains a section where the user may specify in a list the name of each scenario to be used in a DSE (Figure 10). The DSE drivers currently used this list both in the naming of results directories along with the design parameters and it is also used to define a path passed to the external analysis so it may find the required data file.

4.3 Search Algorithms

At this point there are two DSE search scripts implemented, exhaustive search, and a genetic algorithm.

4.3.1 Exhaustive Search

The first algorithm is an open loop exhaustive algorithm which, as the name suggests, will search through the entire design space, testing each and every combination of design parameters, computing the objective values as it goes and finally determining the ranking of those results. This is a simple algorithm with no parameters to adjust its behaviour and it guarantees to find the optimal design with the design space defined. However, its weakness is that it may not be feasible to run simulations for all combinations of design parameters and so it is only applicable to 'small' design spaces.

4.3.2 Genetic Algorithm

The second script contains two variants of a genetic algorithm [Deb12] that have two parameters to tune their behaviour. The variations offer choices of both how the initial population is generated and how parents are selected to produce each subsequent generation. The parameters affect the size of the initial population and how long the algorithm will continue if no better results are being found. The strength of a closed loop search, such is this genetic search, is that they will perform a search of the design space without testing each design and so require less CPU time than an exhaustive search [FGPL17]. This strength comes at the cost of guaranteeing of finding globally optimal designs, the search may instead find some set of local optimums and return those. The following subsections outline the steps take by the genetic algorithm.

Initial population generation The first step in the genetic algorithm is to generate an initial population of designs. The size of this initial set is a parameter the user may set and there is ongoing work that is described in D3.2a [FGPP16] that aims to provide guidance on what this size should be. It is on the generation of this initial set that the two genetic algorithms differ. One version of the script produces an entirely random set of designs and then proceeds to the next step, while the other version attempts to produce a set of designs that is evenly distributed across the design space.

Again the experimentation in D3.2a aims to provide guidance about which of these options should be used and when.

Evaluation and ranking The second step in the genetic algorithm is to evaluate the new designs according to the objectives in the DSE config file (section 4.2.2) and then to place them in a partial order according to the ranking defined (section 4.2.4).

Progress assessment With the whole population evaluated and ranked it is possible to determine if the fitness of the best designs is improving or not, where fitness is a function of each design's objective values and the ranking method. This is done by looking at the population of the non-dominated set of designs to determine how long, in generations, it has been since the one or more new designs were added to this set. If the number of generations since this set changed is above a threshold then the algorithm assumes that an optimal design has been found and the genetic algorithm halts, returning the graph and table of results to the user. The number of generations the algorithm will proceed to the next step without seeing any improvement is a parameter the user may define for the algorithm and once again this is being investigated so that guidance may be provided.

Parent selection and offspring generation If the algorithm decides to proceed, the next step is to select a pair of parents from the whole population. Here the parents are weighted according to the rank they achieved in the evaluation step such that those in rank 1 are more likely to be selected than any of those in rank 2 and any design in rank 2 is more likely to be selected than any in rank 3 and so on. With a pair of parents selected the algorithm places their design parameters in an ordered list, and randomly chooses a place to cross them, producing two offspring, such that each offspring has some parameters from each parent, with a small probability that each parameter may mutate to a different but valid value. Once the offspring have been defined, the process moves back to the evaluation and ranking step and the loop continues until the progress and assessment step determines that no progress is being made.

4.4 COE Handler

The COE handler is the script that connects the DSE scripts with the COE and orchestrates the running of the simulation and retrieval of the simulation results. Aside from the occasional changes imposed by changed in the COE itself, this script it has remained largely stable since the first version was produced. The most significant change made during Year 3 has been to remove a dependency on the application *curl*. This application was used to communicate the HTTP messages between the COE Handler and the COE itself. Since Windows does not contain an implementation of curl by default, this required the application to be distributed with the DSE scripts. The COE Handler now uses a Python library to handle the HTTP communications meaning it is now entirely Python based.

One change that would be desirable is the addition of simulation progress monitoring to the COE Handler. While it is not expected that a user will sit and watch a DSE running, it is important that feedback be given on the progress of this activity. As will be discussed later in Section 4.7, it is most important that the user can perceive progress on the level of the DSE search rather than an individual simulation, however if individual simulation progress were available then this would provide the user with extra information regarding how long the DSE is likely to take. There is a mechanism currently in the INTO-CPS Application that performs this simulation progress function, it will therefore be investigated if this may be leveraged for use in the DSE section of the application.

4.5 Cloud support

While the INTO-CPS DSE scripts now support closed loop searching in the form of the genetic algorithm (Section 4.3.2) and this does improve the efficiency of the search, there is still a non-trivial cost in terms of the number of simulations run, to exploring a large design space [FGPL17]. One means to partly address this issue is to make use of parallelism by running multiple simulations of different designs at the same time, thereby reducing the time to complete a search. However, performing parallel simulation may require access to more spare computation resources than a small organisation may have available and so making use of cloud based resources may help democratise this capability.

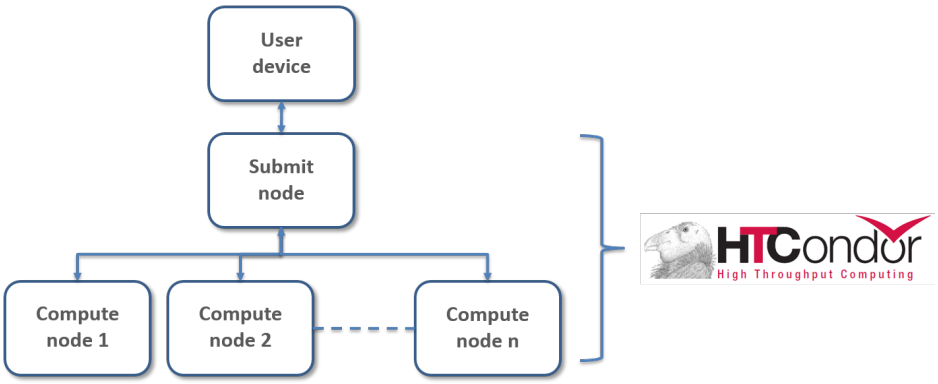


Figure 11: The condor system

The INTO-CPS DSE scripting makes use of the HTCondor[11] high throughput computing platform. This is an open source 'workload management system for compute-intensive jobs' developed by the University of Wisconsin-Madison. In essence, HTCondor consists of a submit node and some number of compute nodes, Figure 11. A user interacts with HTCondor by submitting a job to the compute node. A job consists of a software package the user wishes to execute and a configuration file that, among other things, the environment the job requires, how to launch the job, how many instances of the job should be launched and what results should be returned to the submit node once a job is complete. The submit node is then responsible for ensuring that all instances of the job are executed by sending them to the compute nodes when they become available, monitoring their progress and retrieving the required results when they complete.

Implementing DSE on the cloud required significant extension of the existing DSE search scripts to cover the transfer of simulation artefacts to and from the HTCondor submit node, communications with the submit node and also handling of 'straggler' jobs. The issue of stragglers will be discussed shortly. The outline of the operation both the exhaustive and genetic search scripts are shown in Figures 12 and 13. Apart from the differences in the open-loop and close-loop approaches, both scripts follow the same sequence when a simulation is to be run: they first create the appropriate multi-model configuration, pass the path of the multi-model configuration to a COE Handler script and then await the results of this single simulation.

The interactions when using HTCondor to deploy parallel simulations is somewhat different and is outlined for the exhaustive search in Figure 14.

[11] `https://research.cs.wisc.edu/htcondor/`

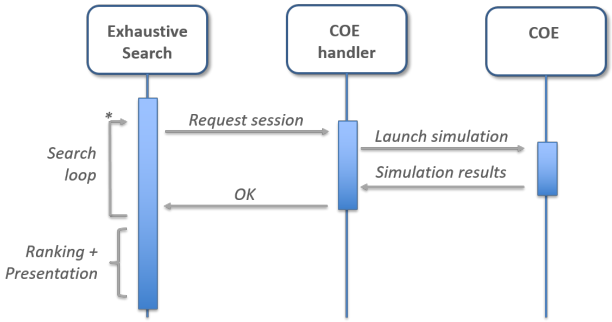


Figure 12: Outline sequence diagram showing local execution of exhaustive DSE

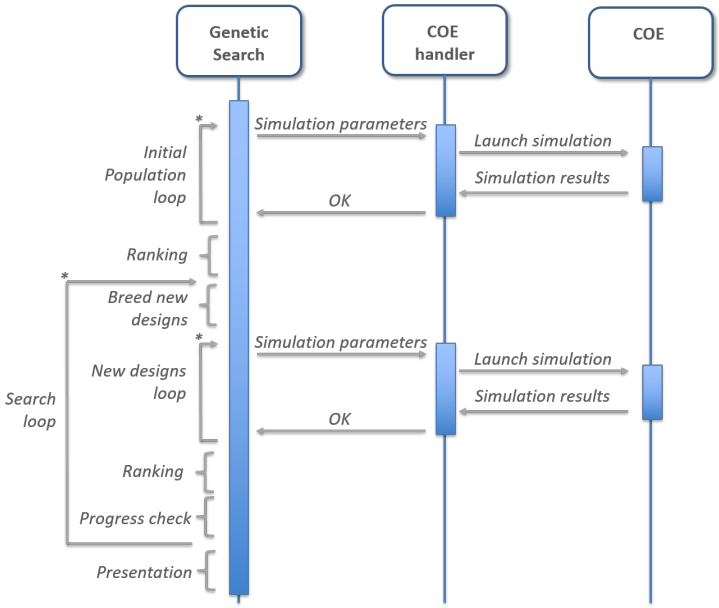


Figure 13: Outline sequence diagram showing local execution of genetic DSE

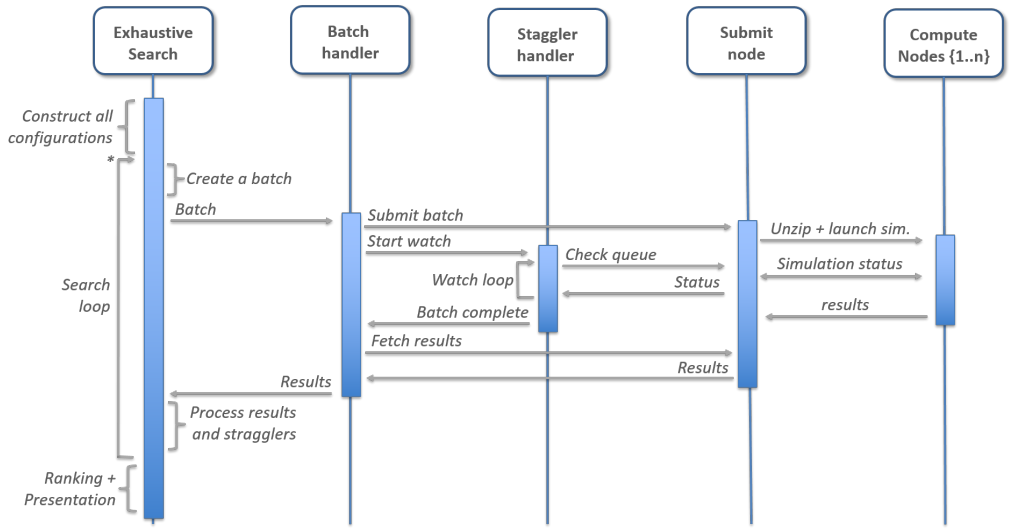


Figure 14: Outline sequence diagram showing execution of an exhaustive DSE using HTCondor

First of all, the user is expected to have created an archive containing the simulation FMUs and the objective scripts and transferred these to the HTCondor submit node. This is done once as these files are common to all simulations and so this reduces network traffic. This archive is termed the *simulation package*.

With the simulation package in place, the key difference between the local and cloud search implementation arise from the need to paralellise the simulations. Rather than generating a configuration, running the simulation and then moving on, the script now generates exhaustively all configurations before commencing simulation. When all simulations are created, the script then proceeds to create a 'batch' of configurations to run, the size of this batch should match the number of compute nodes that are available for simulation. The batch is then zipped up and transferred to the HTCondor submit node. On the submit node the configurations are added to the simulation package, before copies of this package are distributed among the compute nodes and each is given a process number to identify it among the batch. Each node then performs a single simulation, using its process number to select the required multi-model configuration.

A script called the straggler handler monitors the execution of the simulations and determines when the simulation of the batch is complete. The objective results, which are returned to the submit node from the computation nodes, are then zipped up and returned to the search exhaustive search script where

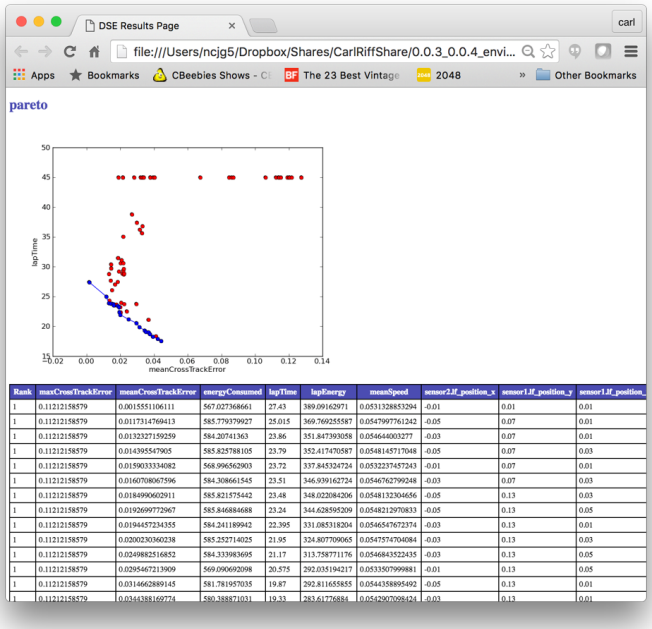
they are placed into a folder structure using the same naming conventions as employed when DSE is performed locally. In the case of the exhaustive search, the next batch is then packaged up and sent to the submit node for the next round of simulations, while in the case of a genetic search, it now computes the ranking of the results seen so far and then breeds the next generation of simulations.

The concept of stragglers has been mentioned and needs explanation. During the testing of cloud DSE at Newcastle University using our HTCondor installation, we have found that simulations do not always complete. In one test with 200 identical copies of the line follower robot using identical multi-model configurations, several copies of the simulation failed to terminate. This problem is likely due to the nature of our HTCondor facility, which makes use of idle computing cluster machines which, although they have identical software installations, have had different usage histories and so behave differently. To address this the DSE scripts include a straggler handling strategy that allows it to retrieve the completed simulation results when certain conditions arise. At this time the strategy employed returns results when either a minimum percentage of the simulations has terminated successfully or a time-out is reached, whichever occurs first. When the simulation results are returned, the search script places the multi-model configurations for any simulations that did not complete correctly back into the pool to be run again later. Using this method the scripts eventually return all simulation results despite the slightly unreliable infrastructure. It is possible that a dedicated computation facility may not exhibit this unreliable behaviour.

4.6 Results Presentation

The results presentation is the result of two scripts, the *ranking* script is responsible for producing the raw ranking data and the graph that will be displayed (in png format). The *output* script is responsible for producing a static html file including the png graph and extracting the details of the ranked designs for a table, the results of which are shown in Figure 15.

The page contains the information that would allow the user to determine which are the best designs, but while the graph highlights the best designs with different coloured points and the table lists all designs with their rank number, it is still a manual process to link the points on the graph with the rows in the table. It also does not lend itself to the identification of trends by, for example, highlighting all designs with a particular value of a design parameter. The page also presents all design results, even those with the very



Rank	maxCrossTrackError	meanCrossTrackError	energyConsumed	lapTime	lapEnergy	meanSpeed	sensor2.lf_position_x	sensor1.lf_position_y	sensor1.lf_position_
1	0.11212158579	0.0015551106111	567.02736861	27.43	389.09162971	0.053132885329	-0.01	0.01	0.01
1	0.11212158579	0.011731476941	585.779379927	25.015	369.769255587	0.0547997761242	-0.05	0.07	0.01
1	0.11212158579	0.013232759259	584.20741363	23.86	351.847393058	0.054644003277	-0.03	0.07	0.01
1	0.11212158579	0.014395547905	585.825788105	23.79	352.417470587	0.0548145717048	-0.05	0.07	0.03
1	0.11212158579	0.0159033334082	568.996562903	23.72	337.845324724	0.053223745724 3	-0.01	0.07	0.01
1	0.11212158579	0.0160708067596	584.308661545	23.51	346.939162724	0.0546762799248	-0.03	0.07	0.03
1	0.11212158579	0.0184990602911	585.821575442	23.48	348.022084206	0.0548132304656	-0.05	0.13	0.03
1	0.11212158579	0.0192699772967	585.846884688	23.24	344.628595209	0.0548212970833	-0.05	0.13	0.05
1	0.11212158579	0.0194457234355	584.241189942	22.395	331.085318204	0.0546547672374	-0.03	0.13	0.01
1	0.11212158579	0.0200230360238	585.252714025	21.95	324.807709065	0.0547574704084	-0.03	0.13	0.03
1	0.11212158579	0.0249882516852	584.333983695	21.17	313.758771176	0.0546843522435	-0.03	0.13	0.05
1	0.11212158579	0.0295467213909	569.090692098	20.575	292.035194217	0.053507999881	-0.01	0.13	0.05
1	0.11212158579	0.0314662889145	581.781957035	19.87	292.811655855	0.0544358895492	-0.05	0.13	0.01
1	0.11212158579	0.0344388169774	580.388871031	19.33	283.61776884	0.0542907098424	-0.03	0.13	0.01

Figure 15: A page of DSE results

high default values when an objective could not be computed, on the same graph, this has the effect of distorting the results such that the real spread of results become obscured. The plans for Year 3 include making the page more dynamic to help identify trends and the position of results in objective space and also to filter the simulation results into the acceptable results, those that were evaluated but did not meet the objective constraints and those that could not be evaluated for some reason. Additionally to this, since DSE simulations are not run with either 3D or live stream output, it is desirable if a user will be able to relaunch a simulation from the results such that the simulated behaviour may be observed using these other features.

4.7 INTO-CPS Application Integration

The integration of DSE into the INTO-CPS Application has moved from a simple launch of an already configured DSE as reported in Year 2, to a facility that permits the editing and launching of DSE. The outline of the user's interaction with a DSE configuration are shown here, full details are described in the user manual, D4.3a [BLL+17].

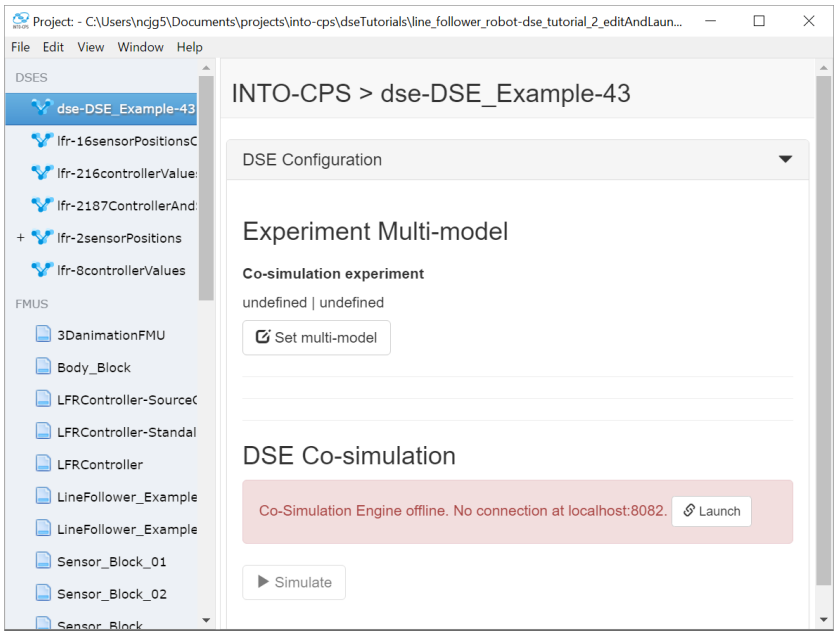


Figure 16: Double clicking on a DSE configuration in the app takes the user to the DSE view

The outline process for using DSE is that the user first selects a DSE configuration by double clicking its icon in the project browser (Figure 16). The user must then select the multi-model that the DSE will use as a basis for the simulations that will be performed (Figure 17). The DSE configuration is then parsed with the multi-model configuration in mind and will only open if the two are compatible in terms of the FMUs in the DSE configuration existing in the multi-model configuration (Figure 18). The user may then edit the details of the DSE configuration (See D4.3a for details here), when this is done the user must then save the DSE configuration and launch the COE (Figure 19). Finally the user may then launch the DSE process and await the results (Figure 20). When complete the user may view the results as shown in the previous section in Figure 15.

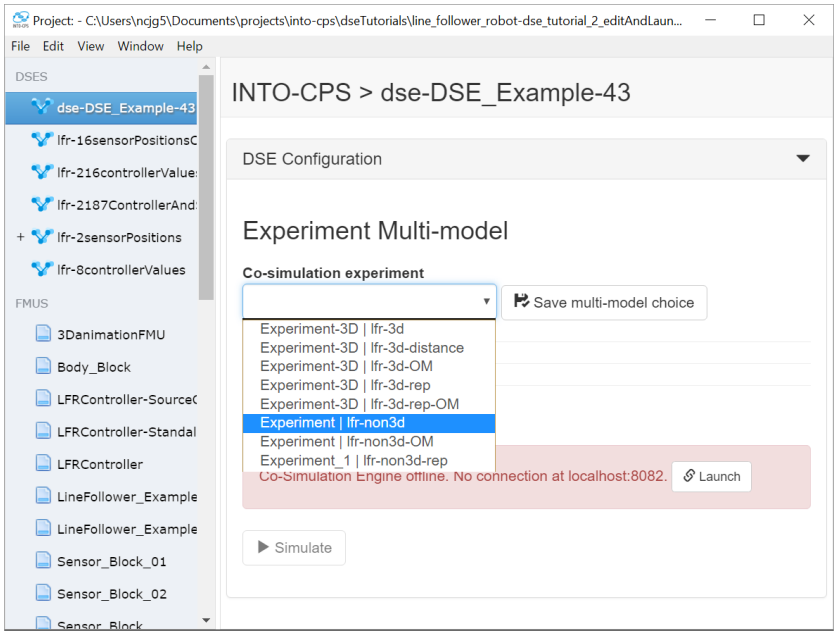


Figure 17: With a configuration selected, the user then selects which multi-model the DSE will act upon

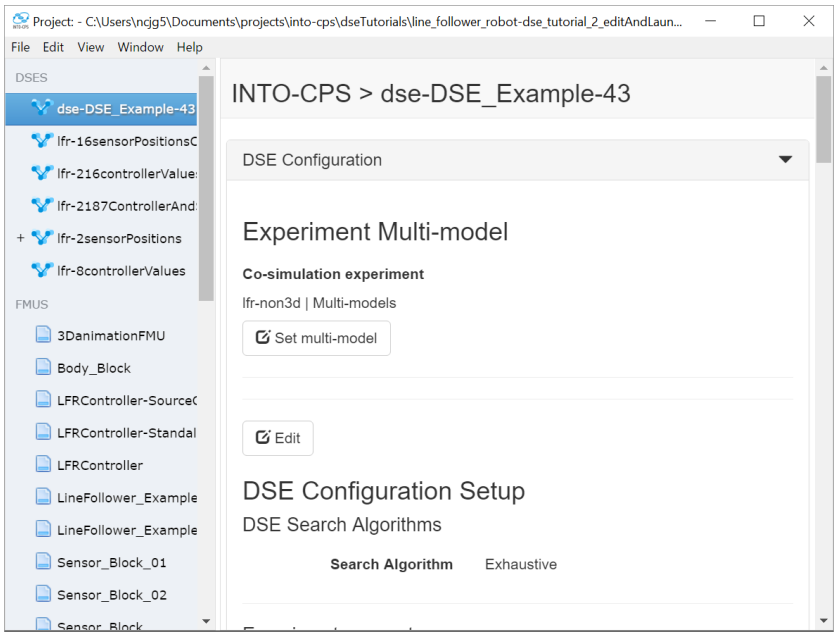


Figure 18: The DSE configuration is then opened and parsed, and the user may then edit the details.

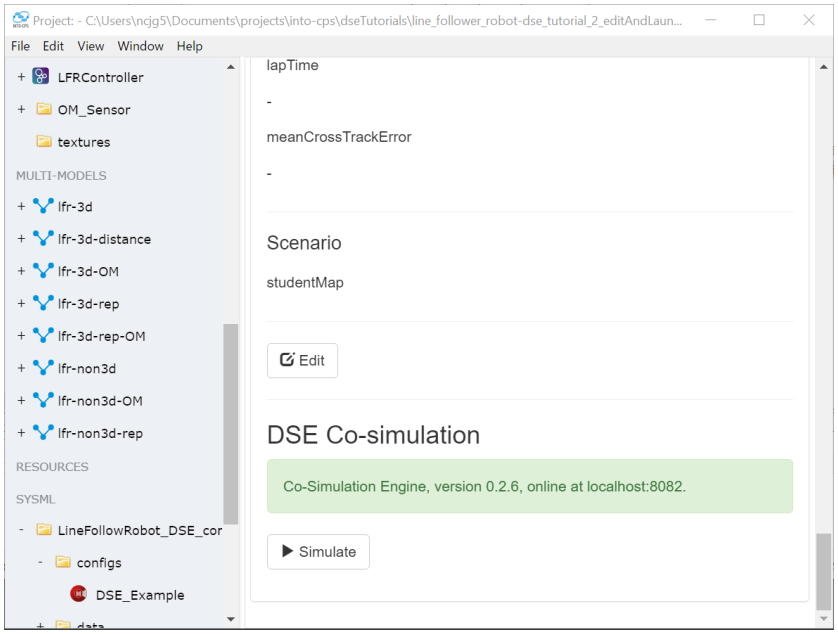


Figure 19: The DSE configuration is saved once edited and the COE is launched

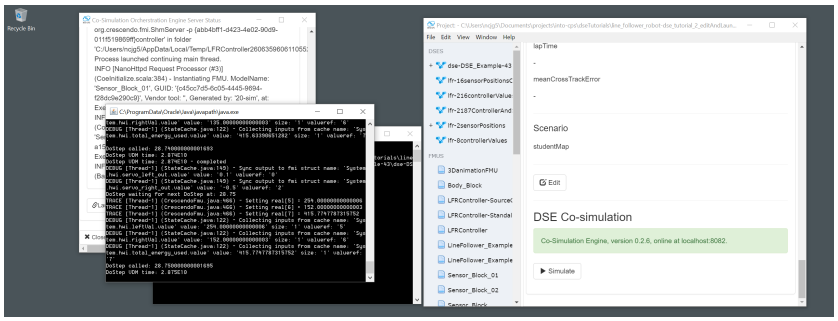


Figure 20: The DSE may be launched once the configuration is saved.

4.8 Analysis Available to Single Simulations

There is a distinct difference in the analysis included provided by the DSE internal and external analysis scripts along with the ranking methods and the analysis within the INTO-CPS Application. The INTO-CPS Application analysis supports a live stream of selected variables and there is also the 3D visualisation option if engineers create such a view for their models. The INTO-CPS Application analysis is well suited for understanding how a model is behaving during simulation, for fault analysis and for communicating with non-technical stakeholders, especially through the 3D visualisation. At the same time the DSE analysis abstracts away the detail generated during a simulation and concentrates on producing the objective values that characterise performance and allow comparison of competing designs. Making the DSE type of analysis available to single simulations as well as DSE simulations would allow them to be compared on common grounds

As can be seen earlier in Figure 3, the definitions of both the DSE search space and algorithm are in the same file as the definition of the objectives and ranking definition, this structure does not make it obvious that the analysis could be used outside of DSE, also it leads to the objective and ranking definitions being duplicated across multiple DSE configurations when it only needs to be defined once.

The proposal for Year 3 then is to dissect the DSE configuration such that the search and analysis sections are separate and then to move the analysis portion up the to a higher level in the INTO-CPS Application project structure so it is more logically accessible to both DSE and single simulations. Figure 21 shows a proposed change to the file and folder structures around DSE. Note the new *analysis* folder at the bottom of the tree, it contains a new file the *name.analysis.json* containing the description of the objectives and ranking method. The external analysis scripts and their data files are moved under the new analysis folder. The *dseConfig* still exists but contains only the sections defining the search parameters and algorithms.

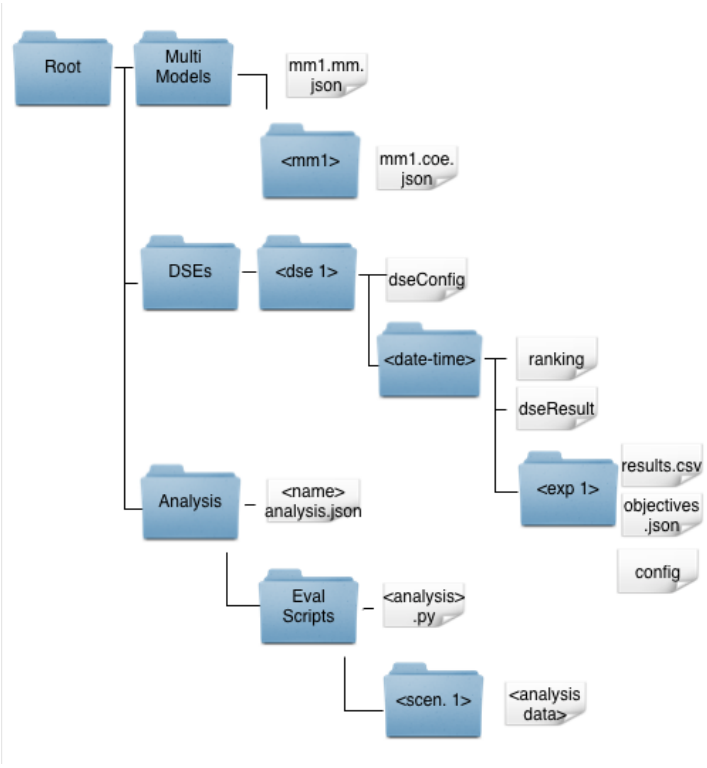


Figure 21: Proposed adjustment to the INTO-CPS Application project structure around DSE

4.9 Matrix of Capability Implementation

The different DSE search algorithms, Exhaustive/Genetic and Local/Cloud execution currently exist a four individual scripts that share some common functions via a library. This parallel development has resulted in not all features being available to all versions of the search. This section then describes which features are available to which search scripts.

Table 1 presents a view of each of these scripts and which features are implemented in each, indicated by a ✓. There are two comments regarding the implementation of both scenario sweeping and architecture sweeping that are common to all search scripts, these comments now follow.

**1: Scenario sweeping is currently under development and is being driven by the 'Little Yellow Bot' that will be added to the examples compendium. This work will be completed by UNew as their first action in the INTO-CPS association.

**2: Architecture sweeping has been discussed with colleagues at the INCOSE international symposium and work on exploring the description of architecture change points and options. UNew aim to collaborate with the interested party on this subject in the first half of 2018.

Table 1: The four DSE search scripts and the features implemented in each

		Local DSE		Cloud DSE	
		Exhaustive	Genetic	Exhaustive	Genetic
Parameters:	range	✓	✓	✓	✓
	constraints	✓			
Objectives:	definition	✓	✓	✓	✓
	constraints				
Sweeping:	scenarios	**1			
	architectures	**2			

5 Conclusions

This deliverable has described the state of DSE support in scripts and also in the INTO-CPS Application. It identifies that there is certainly much more work that can take place in terms of DSE support and this is a key part of

the UNew contribution to the activities that follow on from the INTO-CPS project in the INTO-CPS association.

References

[Bak05] Rachel Edith Baker. *An Approach for Dealing with Dynamic Multi-Attribute Decision Problems*. PhD thesis, Department of Computer Science, University of York, UK., 2005.

[BFG+12] Jan F. Broenink, John Fitzgerald, Carl Gamble, Claire Ingram, Angelika Mader, Jelena Marincic, Yunyun Ni, Ken Pierce, and Xiaochen Zhang. Methodological guidelines 3. Technical report, The DESTECS Project (INFSO-ICT-248134), October 2012.

[BLL+17] Victor Bandur, Peter Gorm Larsen, Kenneth Lausdahl, Casper Thule, Anders Franz Terkelsen, Carl Gamble, Adrian Pop, Etienne Brosse, Jörg Brauer, Florian Lapschies, Marcel Groothuis, Christian Kleijn, and Luis Diogo Couto. INTO-CPS Tool Chain User Manual. Technical report, INTO-CPS Deliverable, D4.3a, December 2017.

[CER13a] CERTAINTY. Modelling languages and models. Technical Report Deliverable D2.3, EU FP7 288175 CERTAINTY, 2013.

[CER13b] CERTAINTY. Preliminary methodology. Technical Report Deliverable D8.2, EU FP7 288175 CERTAINTY, 2013.

[Deb12] Kalyanmoy Deb. *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd., 2012.

[FGP17] John Fitzgerald, Carl Gamble, and Ken Pierce. Method Guidelines 3. Technical report, INTO-CPS Deliverable, D3.3a, December 2017.

[FGPL17] John Fitzgerald, Carl Gamble, Richard Payne, and Benjamin Lam. Exploring the Cyber-Physical Design Space. In *Proc. INCOSE Intl. Symp. on Systems Engineering*, volume 27, pages 371–385, Adelaide, Australia, 2017.

[FGPP16] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. Method Guidelines 2. Technical report, INTO-CPS Deliverable, D3.2a, December 2016.

[FLPV13] John Fitzgerald, Peter Gorm Larsen, Ken Pierce, and Marcel Verhoef. A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems. *Mathematical Structures in Computer Science*, 23(4):726–750, 2013.

[FNSV15] John B. Finn, Pierluigi Nuzzo, and Alberto Sangiovanni-Vincentelli. A mixed discrete-continuous optimization scheme for cyber-physical system architecture exploration. In *International Conf. Computer-Aided Design*, 2015.

[NBAR+12] Yunyun Ni, Jan F. Broenink, Kenneth G. Lausdahl Augusto Ribeiro, Frank Groen, Ken Pierce Marcel Groothuis, Carl Gamble, and Peter Gorm Larsen. Design space exploration tool support. Technical report, The DESTECS Project (INFSO-ICT-248134), December 2012.

6 List of Acronyms

AU	Aarhus University
CLE	ClearSy
CLP	Controllab Products B.V.
DSE	Design Space Exploration
ENUM	Enumeration and Scoring
PROV-N	The Provenance Notation
ST	Softeam
TWT	TWT GmbH Science & Innovation
UNEW	University of Newcastle upon Tyne
UTRC	United Technology Research Center
UY	University of York
VSI	Verified Systems International
WAM	Weighted Additive Method
WP	Work Package