Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

KE TOOL: AN OPEN SOURCE SOFTWARE FOR AUTOMATED INPUT DATA IN DISCRETE EVENT SIMULATION PROJECTS

Panagiotis Barlas
Cathal Heavey

Enterprise Research Centre
University of Limerick
Limerick, IRELAND

ABSTRACT

Input data management is a time-consuming and costly for Discrete Event Simulation (DES) projects. According to research studies, the input data phase constitutes, on the average, can account for over a third of the time of an entire simulation project. This paper presents a newly developed Open Source (OS) tool, called the Knowledge Extraction (KE) tool that automates the input data management in DES projects enabling real-time simulation. The OS software reads data from several resources of an organisation; analyses it using statistical analysis and outputs it in a format that is applicable to be used by simulation software, all conducted in one automated process. We explain how the KE tool is developed using Python libraries, introduce its structure and provide insights of its employment.

1 INTRODUCTION

Manufacturing is of main importance for the prosperity and development of societies. There is a continuous need to keep the operations as flexible as possible to quickly take advantage of new industry trends. The need of rapid adaptation to changes either in industry trends or consumption patterns is one of the key components to competitive success in business. Unfortunately, this adaptation is done by significant investments instead of increasing utilization of existing equipment. Obviously, to increase competitiveness, it is of major importance to eliminate wastes, e.g. overproduction, waiting, motion, defects etc. The continuous improvement of production flows, which is actually the core of Lean Manufacturing, will lead to reduce the time it takes to finish an activity from start to finish and total costs. DES has proved itself to be an effective tool for such improvements. DES is an effective tool for planning, designing and improving material flows in production, thus the reduction of the aforementioned wastes.

However, despite its potential, manufacturing industry has been less than successful in the implementation of simulation as a decision support tool (McNally and Heavey 2004). One of the main weaknesses of operating DES is the extensive time-consumption of simulation studies. This leads to less detailed analysis, and by extension, to production systems designed for ideal circumstances, disregarding variation and disturbances. The reason for the extensive time-consumption is arguably the input data process. The time needed for the input data process is usually 10-40% of the total time of a DES project (Skoogh and Johansson 2008). This is the result of the low quality of data and the large amount of manual work to translate raw data into simulation input (Robertson and Perera 2002).

Therefore, research towards the development of new approaches that address the above issue of extensive time-consumption and enhance the input data process in DES is considered critical. The developed tool is an OS software, called the Knowledge Extraction (KE) tool, that can extract raw data from company's data sources, transforms this data to simulation input. The data formats that have been implemented are an XML (Extensible Markup Language) format that obeys the CMSD (Core Manufacturing Simulation Data) specification, a spreadsheet format, a data format that follows the Simul8 XML specification and a JSON (JavaScript Object Notation) format that follows the JSON API (Application Programming Interface) developed for exchanging DES data. The KE tool is part of an OS simulation based decision support platform developed within the FP7 project titled "Decision support in Real-time for Efficient Agile Manufacturing" (DREAM, http://dream-simulation.eu/), (Heavey et al. 2014).

The developed approaches targeting the automation of the input data process in DES projects are introduced in section 2. In the next section the description of the rationale behind the selection to build the tool in Python and utilize the Rpy2 library is given. Section 4 presents a description of the KE tool concept. Next we demonstrate the applied means and techniques to verify the KE tool's code and facilitate a consistent development. Section 6 presents an overview of three case studies that implement the tool. This paper ends presenting conclusions and future work.

2 LITERATURE REVIEW

We started our work by conducting a literature review in the input data process in DES projects (Barlas and Heavey 2016). In this section we briefly present the main studies in this area. Robertson and Perera (2002) realizing that the data collection is a tremendously time consuming process due to the manual orientation of the task, they stated that a potential automation of data collection process would be extremely beneficial.

Robertson and Perera (2002) described four different methodologies to input the required data to DES projects. The first two methodologies contain the manual involvement mainly in the collection of the simulation data, whilst the last two methodologies consist of automated steps. The third approach is based on an intermediary database that is connected to Corporate Business System (CBS) applications (e.g. ERP, MRP, CAD systems), with the intermediary database connected to a DES models. If the intermediary database integrates with the CBS, manual effort and errors can be dramatically reduced. In the fourth methodology, the model directly collects data from the CBS through an interface. By studying the literature the designed tools are mainly developed similar to the third methodology.

An example application is the Ford Assembly Simulation Tool (FAST) which is designed to build a WITNESS simulation model of powertrain assembly line from shop floor data from an Excel spreadsheet (Winnell and Ladbrook 2003). A set of VBA macros are embedded with an Excel file in order to generate and import simulation data file to the Lanner WITNESS software. The transmitted file from FAST has all required attributes to run simulation experiments. Moreover, it commands WITNESS to build customized machines with different behavior remotely by using the WITNESS Command Language (WCL).

Aufenanger et al. (2010) described the development of an application called Machine Data Acquisition (MDA) in order to enhance the material flow simulator d3FACT insight with a flexible interface for the adoption of machine data. They used the Devices Profile for Web Services (DPWS) for collection of raw data from manufacturing resources such as conveyor belts and machines. DPWS enables collection of raw data, which is then stored in a database called MDA database. Moreover the MDA application also includes algorithms for processing the raw data to provide as input to a d^{3}FACT insight simulation model. The paper demonstrates the successful accomplishment of real-time simulations using the application in a test implementation in a laboratory environment.

The Generic Data Management Tool (GDM-Tool) was presented by Skoogh et al. (2012) to enable automated data input and analysis management. In addition to data management, the GDM-Tool also exports an interoperable file to a simulator according to the Core Manufacturing Simulation Data (CMSD) standards. The GDM-Tool extracts data from various databases with different internal data structures allowing reuse of connections to databases and the pre-configured data from prior used simulation input data. The software operates by associating a sequence of small plug-ins together for execution.

In their work towards the automated data collection focusing especially on SMEs (Small and Medium-sized Enterprises), Byrne et al. (2013) proposed a software adapter that integrates with data sources and/or fills data gaps using a data generator. This paper reports that the Cloud based solution allows SMEs to benefit from DES with decreased complexity leading to a low cost with higher result accuracy and validity.

The proposed concept with the use of the KE tool has the following unique characteristics:

- Publicly available OS tool, ready to be applied by simulation modelers.
- Ability to provide tailored solutions through the easily customizable features of the tool.

3 RPY2 AND OTHER PYTHON LIBRARIES

The second step, after the literature review, in our study was to perform a survey of OS data science tools. The main purpose of this study (Barlas, Lanning, et al. 2015) was to identify the most appropriate tool that would be the base for the development of the KE tool. Searching both on web and academic publications 70 OS data science tools revealed. In order to assess them, we defined different criteria under four group characteristics. These were:

- **General Characteristics** such as the type, the domain, the language used to develop the tool, the license released and the operating system. These are characteristics that may affect a user's decision on what is the most appropriate tool for their needs.
- **Project Activity** given that some packages may be more active having more frequent updates than others, provide more organized means of documentation than others etc.
- **Operational Characteristics** which define elements such as input data formats supported and manner of interaction as some may require or support more interaction with the user than the other others.
- **Data Mining Characteristics** different tools may execute different data mining tasks and use different data mining methods to achieve their goals.

This study resulted to a range of interesting points. First of all, the number of identified projects indicates that there is enormous interest in the OS area. Another interesting outcome is that five tools across the evaluation stood out among the others; RapidMiner, R, Weka and two Python libraries RPy2 and Scikit-learn. These tools offered the widest variety of data analysis tasks compared to others. In addition they exhibited good operational characteristics and remarkable project activity characteristics. It's interesting to note that RapidMiner incorporates R and Weka through extensions, providing learning schemes, models and algorithms from these two tools' scripts.

There are two choices to engineer an OS software, i) develop a new tool; ii) use an already available tool. From the five tools (RapidMiner, R, Weka and two Python libraries RPy2 and Scikit-learn) that stood out among the others, RPy2 is a Python based data analysis project and has been the software that we selected to base our development upon. Having an interface between both languages to benefit from the libraries of one language while working in the other appeared desirable. RPy2 has an active community of users and documentation is adequate providing to understand concepts and examples.

The main reason that we decided to use RPy2 was the fact that it is written in Python. Apart from RPy2, other Python libraries have been also used for specific purposes, e.g. pyodbc (https://code.google.com/p/pyodbc/) library has been applied in order to connect to databases. The versatility of Python and the variety of Python libraries have been a significant asset. Python has features that enhance the coding of a program for a user with software skills. It offers a range of attributes and methods for efficient list processing. The drawback is that Python, as a programming language, is slower than static languages such as C++ or Java. Another identified downside by a minority of users is that these Python libraries demand basic programming abilities in order to be used. However, the majority of users that follow the OS movement have basic programming abilities which reduce the impact of this drawback.

4 THE KE TOOL

In this section we describe how the KE tool is developed and how it can be applied to facilitate the IDM process in DES applications. At the beginning we describe the design and next we present the architecture. We continue by giving more details about the KE tool objects.

4.1 Design

Skoogh and Johansson (2008) focus on what they define input data management (IDM). IDM is described as "the whole process of arranging quality guaranteed, and simulation modified, depiction of all relevant input data factors for simulation models. This includes identifying relevant input parameters, collecting all information required to represent the parameters as appropriate simulation input, converting raw data to a quality assured representation, and documenting data for future reference and re-use". The IDM process starts with the collection of raw data and ends with providing processed data as useful information to a simulation model.

The KE tool design is inspired by the definition of IDM (Skoogh and Johansson 2008). The main scope behind the KE tool's design is to offer objects, i.e. classes containing code, which cover the three stages of the IDM process in DES projects. Therefore, the KE tool consists of three main components that represent the three stages. The three main components of the tool called:

- Data extraction;
- Data processing;
- Output preparation.

The user by applying the KE tool gets prefabricated objects from the three components which he or she is able to connect as black boxes in order to form the input data model. Since the outcome of the tool is a neutral file format with meaningful simulation information the KE tool can be connected to any simulation engine this requires either the adoption of the specific file format used by the simulation engine, e.g. SIMUL8 (http://www.simul8.com/) uses a schema developed in XML, called SIMUL8 XML, or the development of a translator between the KE tool's exported file format and the simulation engine.

4.2 Architecture

The KE tool architecture is illustrated in Figure 1. The architecture consists of three layers. The core consists of pre-existing technology that includes Python, RPy2 and other Python libraries like pyodbc that is used to import data from a database or xlrd and xlwt (http://www.python-excel.org/) that allow reading from and writing into spreadsheet (xls) format. It is essential to report that the KE tool controls all these dependencies. Therefore a modeler with a computer able to run Python can install the KE tool and its dependencies at once.

After the first layer there is the layer of abstract KE tool components. This layer includes the three main components (Data extraction; Data processing; Output preparation). Data extraction is a grouping that includes the developed objects used to import data to the KE tool, so the component itself is not used by the tool. The same happens with the other three components. The next level consists of the KE tool objects, which are used to form the KE tool model. These objects act like black boxes; by the term "black box" we mean that the user by simply calling this object gets the desired outcome without requiring any coding amendments.

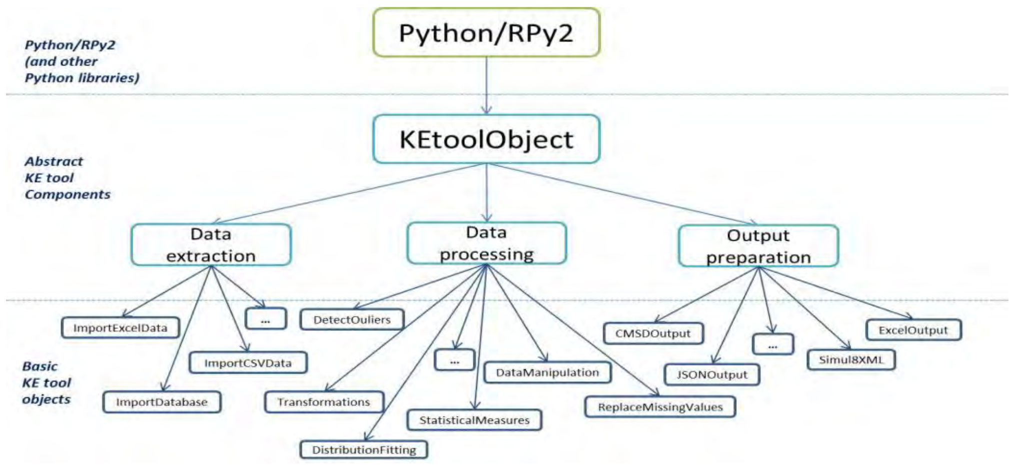


Figure 1: Schematic figure showing the KE tool architecture.

The KE tool was developed so that it is modular, extensible, OS and flexible. Every object has its own attributes encapsulated in its code and outputs its results when is called to be executed. The set of these objects is the core of the KE tool. These objects give the basic guidelines of how the tool is structured. More information about the developed objects is available in GitHub (https://github.com/nexedi/dream/tree/master/dream/KnowledgeExtraction). The repository contains developed documentation, objects and different examples. The code kept under version control with Git (http://git-scm.com/), which the user can clone and work with the different versions of the code through the project repository in GitHub.

4.3 The KE tool Main Script

The tool is built in such a way that the data input, the processing of this data and the output preparation, are conducted by either one script or several scripts in a row, depending on how many objects are required in a given case. We refer to these scripts as "main script". The main script is the only one to be changed in order to read data from different files and transform the raw data in order to be handled by the Data processing objects of the tool. The objects of the above three components can be connected as "black boxes" in order to form the main script. Therefore, this main script calls different objects in order to give as an output the actual selected data exchange file format, updated with the processed data. In this way there is the possibility for a developer to configure the main script so that it reads data from a Graphical User Interface (GUI) or a company's data resource. As a prerequisite for the configuration of the tool is the existence of operational data in a format that can be extracted. The KE tool main script performs the following generic operations:

1. Imports the needed objects in order to run the main script;
2. Calls the Data Extraction objects in order to input the data to the tool;
3. From the imported data select the required data and put them in separate lists;
4. Calls the Data pre-processing objects (if needed) in order to perform the required pre-processing to the data sets;
5. Calls the Data processing objects in order to conduct the required processing to the data sets;
6. Calls the Output preparation objects to export the already processed data to the selected file.

The KE tool main script is different from the other developed objects, so modelers may form their own main script based on their needs, which may constitute an interface with other software, e.g. a drag and drop GUI or an Excel spreadsheet. The KE tool main scripts can also be part of Python code similar to the example reported below. This example will present how the main script of core KE tool objects can be developed and ran. We will examine an assembly and dismantle model, this model contains two sources, one assembly station, one machine, one dismantling station and two exits. Figure 2 illustrates the graphical representation of the model. The machine in this example is subject to failures and again it is assumed that historical data regarding processing times and the times to failures and repairs are available. The data are recorded in an xls file.

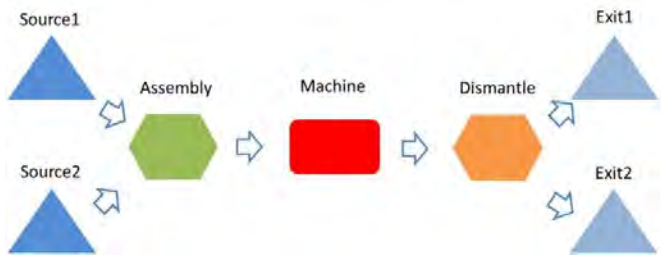
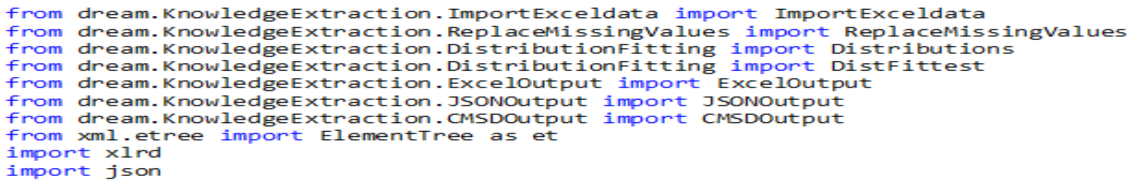


Figure 2: Assembly and dismantle model.

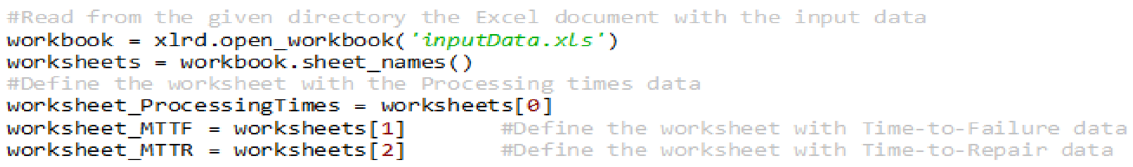
Five objects are used in this model. The steps conducted in the main script described are listed below:

1. Import the needed objects in order to run the main script (see Figure 3). The user can also use `from dream.KnowledgeExtraction.imports import*` to import all the KE tool objects, but it is generally lighter to import only what is required. Also, we imported the xlrd and json Python libraries, these libraries are used to read the Excel document and export the results in the JSON file of the model, respectively.
2. Read the necessary data from the given directory and define the worksheets with the different data sets (processing times, time-to-failure and time-to-repair data) (see Figure 4).
3. Call the ImportExceldata object and by using its method *Input_data* with arguments the above defined worksheet and workbook we import the data sets to the tool as Python dictionaries (Figure 5).
4. From the imported data (python dictionaries) select the required data and put them into separate lists (see Figure 6).
5. Call the ReplaceMissingValues object and apply its method *ReplaceWithMean*, which replaces the missing values with the mean of the non-missing values in the list (see Figure 7).
6. Call the *Distributions* (Maximum Likelihood Estimation) and *DistFittest* (Kolmogorov-Smirnov fitting test) objects and apply them to the data. A Kolmogorov-Smirnov test on the processing times data is carried out and it shows that an Exponential distribution is used for the MTTF and MTTR data (Figure 8).
7. Call the *CMSDOutput* object to export the already processed data (statistical distributions of processing times, MTTF and MTTR) to the CMSD file of the model (see Figure 9).
8. Call the *JSONOutput* object to export the processed data to the developed JSON file of the model (see Figure 10).
9. Call the *ExcelOutput* object and using its methods export the statistical analysis and distribution fitting results of the three data categories (see Figure 11).



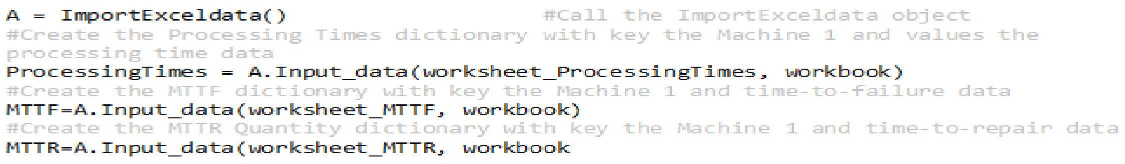
```
from dream.KnowledgeExtraction.ImportExceldata import ImportExceldata
from dream.KnowledgeExtraction.ReplaceMissingValues import ReplaceMissingValues
from dream.KnowledgeExtraction.DistributionFitting import Distributions
from dream.KnowledgeExtraction.DistributionFitting import DistFittest
from dream.KnowledgeExtraction.ExcelOutput import ExcelOutput
from dream.KnowledgeExtraction.JSONOutput import JSONOutput
from dream.KnowledgeExtraction.CMSDOutput import CMSDOutput
from xml.etree import ElementTree as et
import xlrd
import json
```

Figure 3: Schematic figure showing the code to import the needed objects.



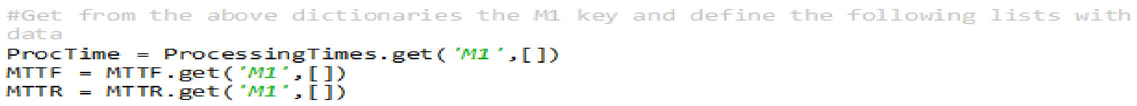
```
#Read from the given directory the Excel document with the input data
workbook = xlrd.open_workbook('inputData.xls')
worksheets = workbook.sheet_names()
#Define the worksheet with the Processing times data
worksheet_ProcessingTimes = worksheets[0]
worksheet_MTTF = worksheets[1]       #Define the worksheet with Time-to-Failure data
worksheet_MTTR = worksheets[2]       #Define the worksheet with Time-to-Repair data
```

Figure 4: Schematic figure showing the code to define the worksheets in the Excel document.



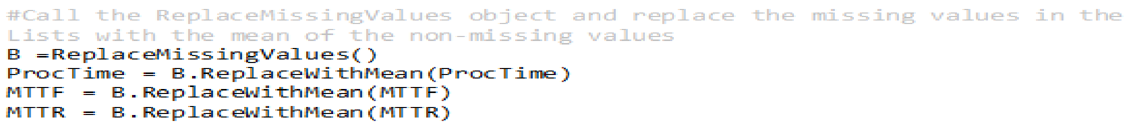
```
A = ImportExceldata()                    #Call the ImportExceldata object
#Create the Processing Times dictionary with key the Machine 1 and values the
processing time data
ProcessingTimes = A.Input_data(worksheet_ProcessingTimes, workbook)
#Create the MTTF dictionary with key the Machine 1 and time-to-failure data
MTTF=A.Input_data(worksheet_MTTF, workbook)
#Create the MTTR Quantity dictionary with key the Machine 1 and time-to-repair data
MTTR=A.Input_data(worksheet_MTTR, workbook
```

Figure 5: Schematic figure showing the code to import data.



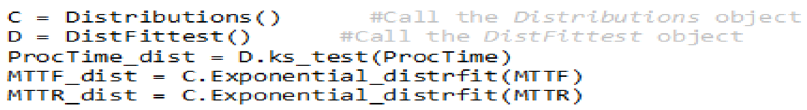
```
#Get from the above dictionaries the M1 key and define the following lists with
data
ProcTime = ProcessingTimes.get('M1',[])
MTTF = MTTF.get('M1',[])
MTTR = MTTR.get('M1',[])
```

Figure 6: Schematic figure showing the code to select the required data.



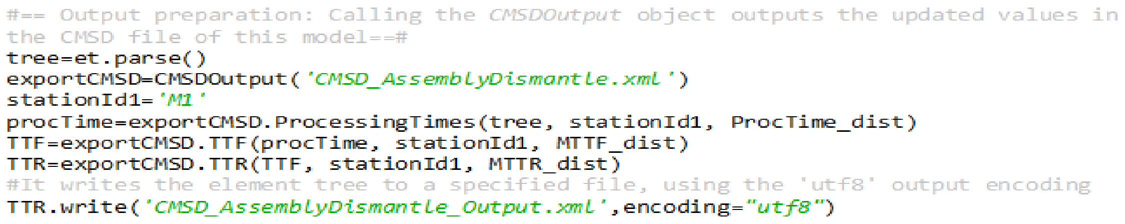
```
#Call the ReplaceMissingValues object and replace the missing values in the
lists with the mean of the non-missing values
B =ReplaceMissingValues()
ProcTime = B.ReplaceWithMean(ProcTime)
MTTF = B.ReplaceWithMean(MTTF)
MTTR = B.ReplaceWithMean(MTTR)
```

Figure 7: Schematic figure showing the code to replace the missing data with the mean value.



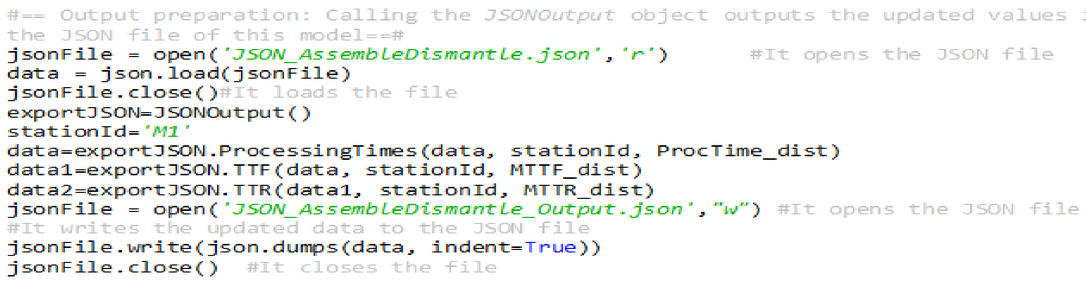
```
C = Distributions()       #Call the Distributions object
D = DistFittest()       #Call the DistFittest object
ProcTime_dist = D.ks_test(ProcTime)
MTTF_dist = C.Exponential_distrfit(MTTF)
MTTR_dist = C.Exponential_distrfit(MTTR)
```

Figure 8: Schematic figure showing the code to apply Kolmogorov-Smirnov test and fit Exponential distribution to the data sets.



```
#== Output preparation: Calling the CMSDOutput object outputs the updated values in
the CMSD file of this model==#
tree=et.parse()
exportCMSD=CMSDOutput('CMSD_AssemblyDismantle.xml')
stationId1='M1'
procTime=exportCMSD.ProcessingTimes(tree, stationId1, ProcTime_dist)
TTF=exportCMSD.TTF(procTime, stationId1, MTTF_dist)
TTR=exportCMSD.TTR(TTF, stationId1, MTTR_dist)
#It writes the element tree to a specified file, using the 'utf8' output encoding
TTR.write('CMSD_AssemblyDismantle_Output.xml',encoding="utf8")
```

Figure 9: Schematic figure showing the code to export the results to CMSD file.



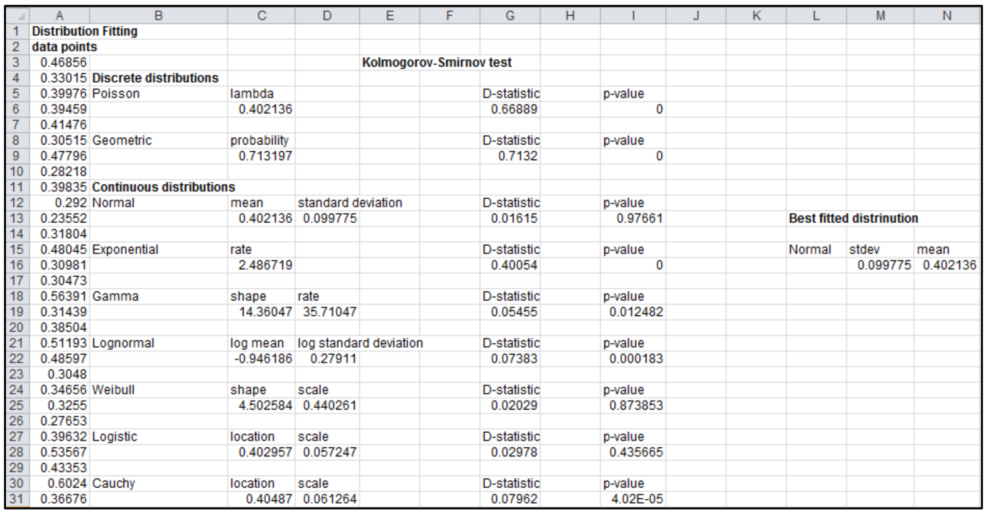
```
#== Output preparation: Calling the JSONOutput object outputs the updated values in
the JSON file of this model==#
jsonFile = open('JSON_AssembleDismantle.json','r')      #It opens the JSON file
data = json.load(jsonFile)
jsonFile.close()#It loads the file
exportJSON=JSONOutput()
stationId='M1'
data=exportJSON.ProcessingTimes(data, stationId, ProcTime_dist)
data1=exportJSON.TTF(data, stationId, MTTF_dist)
data2=exportJSON.TTR(data1, stationId, MTTR_dist)
jsonFile = open('JSON_AssembleDismantle_Output.json',"w") #It opens the JSON file
#It writes the updated data to the JSON file
jsonFile.write(json.dumps(data, indent=True))
jsonFile.close()  #It closes the file
```

Figure 10: Schematic figure showing the code to export the results to JSON file.

```
#== Calling the ExcelOutput object, it exports the outcomes of the statistical
analysis in .xls files ==#
C=ExcelOutput()
C.PrintStatisticalMeasures(ProcTime,'ProcTime_StatResults.xls')
C.PrintStatisticalMeasures(MTTR,'MTTR_StatResults.xls')
C.PrintStatisticalMeasures(MTTF,'MTTF_StatResults.xls')
C.PrintDistributionFit(ProcTime,'ProcTime_DistFitResults.xls')
C.PrintDistributionFit(MTTR,'MTTR_DistFitResults.xls')
```

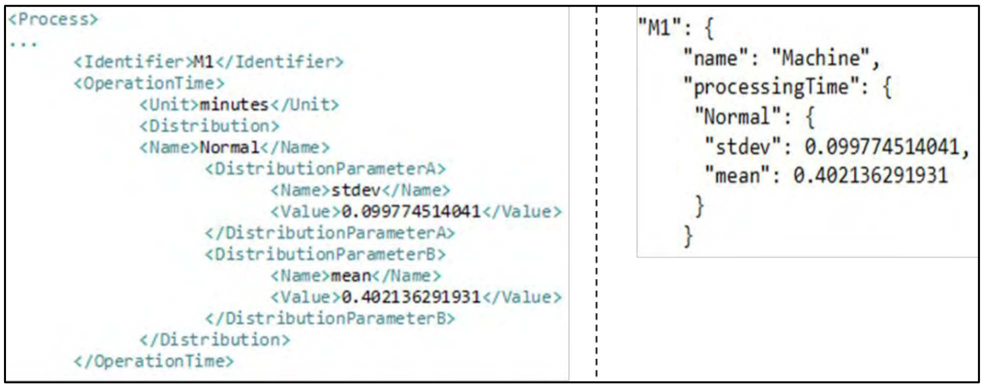
Figure 11: Schematic figure showing the code to export the results to Excel files.

The example along with the required files for the execution can be found in GitHub and also it is reported in the KE tool documentation. Figure 12 illustrates the exported Excel spreadsheets showing the results from the distribution fitting analysis for the data set. As we can see the exported results follow a specific template created for better visualization and ease of understanding. Figure 13 illustrates parts of the CMSD and JSON file showing the statistical distribution for the processing time of Machine.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	**Distribution Fitting**													
2	**data points**													
3	0.46856				**Kolmogorov-Smirnov test**									
4	0.33015	**Discrete distributions**												
5	0.39976	Poisson	lambda				D-statistic		p-value					
6	0.39459		0.402136				0.66889		0					
7	0.41476													
8	0.30515	Geometric	probability				D-statistic		p-value					
9	0.47796		0.713197				0.7132		0					
10	0.28218													
11	0.39835	**Continuous distributions**												
12	0.292	Normal	mean	standard deviation			D-statistic		p-value					
13	0.23552		0.402136	0.099775			0.01615		0.97661			**Best fitted distrinution**		
14	0.31804													
15	0.48045	Exponential	rate				D-statistic		p-value			Normal	stdev	mean
16	0.30981		2.486719				0.40054		0				0.099775	0.402136
17	0.30473													
18	0.56391	Gamma	shape	rate			D-statistic		p-value					
19	0.31439		14.36047	35.71047			0.05455		0.012482					
20	0.38504													
21	0.51193	Lognormal	log mean	log standard deviation			D-statistic		p-value					
22	0.48597		-0.946186	0.27911			0.07383		0.000183					
23	0.3048													
24	0.34656	Weibull	shape	scale			D-statistic		p-value					
25	0.3255		4.502584	0.440261			0.02029		0.873853					
26	0.27653													
27	0.39632	Logistic	location	scale			D-statistic		p-value					
28	0.53567		0.402957	0.057247			0.02978		0.435665					
29	0.43353													
30	0.6024	Cauchy	location	scale			D-statistic		p-value					
31	0.36676		0.40487	0.061264			0.07962		4.02E-05					

Figure 12: Exported Excel spreadsheet with the distribution fitting analysis of the processing times data of Machine.



```
<Process>
...
      <Identifier>M1</Identifier>
      <OperationTime>
            <Unit>minutes</Unit>
            <Distribution>
            <Name>Normal</Name>
                  <DistributionParameterA>
                        <Name>stdev</Name>
                        <Value>0.099774514041</Value>
                  </DistributionParameterA>
                  <DistributionParameterB>
                        <Name>mean</Name>
                        <Value>0.402136291931</Value>
                  </DistributionParameterB>
            </Distribution>
      </OperationTime>
```

```
"M1": {
    "name": "Machine",
    "processingTime": {
     "Normal": {
      "stdev": 0.099774514041,
      "mean": 0.402136291931
     }
    }
```

Figure 13: Parts of the exported CMSD file (left) and the exported CMSD file (right) showing the processing or operation time distribution for the Machine.

5 SOFTWARE VERIFICATION

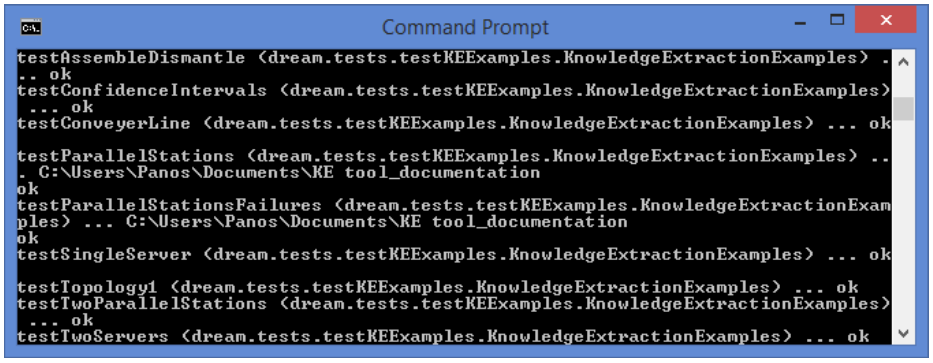
The main goal of software verification is to assure that that software fully satisfies all the expected requirements. In the case of extremely large simulation models, in which the user needs to import data from different data resources and use several "Data processing" objects the code will be large and sometimes difficult to debug.

5.1 Comparison Against COTS

COTS (Commercial off-the-shelf) data science tools have been around for few decades. These products exist in a highly competitive market, constantly looking to increase their performance Whenever we build a new KE tool object in the "Data processing" component, we compare the results against a COTS alternative. The methods from the *ReplaceMissingValues*, *DetectOutliers*, *Transformations*, *DataManipulations*, *StatisticalMeasures* and *DistributionFitting* objects have all been tested against Microsoft Excel, EasyFit (http://www.mathwave.com/easyfit distribution-fitting.html), STATISTICA (http://www.statsoft.com/Products/STATISTICA/) and Minitab (http://www.minitab.com/en-us/). It is verified that there is no difference between the two approaches. Of great help in this verification process is the ExcelOutput object, where the results from the statistical and distribution fitting analysis are exported into the Excel spreadsheet.

5.2 Unit Testing

Unit testing (Zhu et al. 1997) is used in the KE tool in order to ascertain the code's reliability in different versions. Unit testing will detect errors introduced in different versions each time a model is sufficiently verified (using the approaches described above). This is done by storing the outcome from the KE tool in a test dump folder. A particular test Python script can automatically execute all the tests that are already committed in the project (see Figure 14). Basically this script runs all the test models and compares their results to the expected ones. For every test there are three possible results: "error" that means the code crashed, "fail" that means that the outcome was different than the expected and "ok" that means the model yielded the expected outcome. In the case of errors the cause is also appeared in the console at the end of the test. Unit testing is used to check the outcome of the data processing objects, the simulation results and the syntax of the exported file.



```
testAssembleDismantle (dream.tests.testKEExamples.KnowledgeExtractionExamples) .
.. ok
testConfidenceIntervals (dream.tests.testKEExamples.KnowledgeExtractionExamples)
 ... ok
testConveyerLine (dream.tests.testKEExamples.KnowledgeExtractionExamples) ... ok

testParallelStations (dream.tests.testKEExamples.KnowledgeExtractionExamples) ..
. C:\Users\Panos\Documents\KE tool_documentation
ok
testParallelStationsFailures (dream.tests.testKEExamples.KnowledgeExtractionExam
ples) ... C:\Users\Panos\Documents\KE tool_documentation
ok
testSingleServer (dream.tests.testKEExamples.KnowledgeExtractionExamples) ... ok

testTopology1 (dream.tests.testKEExamples.KnowledgeExtractionExamples) ... ok
testTwoParallelStations (dream.tests.testKEExamples.KnowledgeExtractionExamples)
 ... ok
testTwoServers (dream.tests.testKEExamples.KnowledgeExtractionExamples) ... ok
```

Figure 14: Example of the KE tool unit testing.

5.3 XML and JSON Validators

XML and JSON validation is the procedure of testing XML and JSON files, respectively, to verify that they are well-structured. A well-structured file follows the basic syntactic rules of XML, which are the same for all XML documents. A valid document also respects the rules dictated by a particular Document Type Definition (DTD) or XML/JSON schema, according to the application-specific choices.

Possible errors in XML or JSON files will cause errors stopping the execution of the simulation model. Therefore, it is of great importance to validate the exported XML or JSON output from the KE tool. This can be done as there are available online validators for both exported file types. For the exported XML files, i.e. CMSD (SISO 2010) and Simul8 XML we use the XML validators provided by the w3schools.com (http://www.w3schools.com/xml/xml_validator.asp) and Validome (http://www.validome.org/xml/). For the exported JSON files, we use the JSONLint validator (http://jsonlint.com/). In these validators all you have to do is to simply copy and paste in the provided window the XML/JSON document and then click the "Validate" button. Syntax errors or the message that the document is valid appear after clicking the button.

6 CASE STUDIES

The KE tool was used in three industrial cases. One case study came from a Multinational Corporation (MNC) and the other two from SMEs. A summary description of the three companies can be seen in Table 1.

Table 1: Overview of the pilot cases in which the KE tool is applied.

Company	**Sector**	**Size**	**Production Processes**	**Data**
1	Medical	[1500-2000] Employees, MNC	Approximately 63 product lines and 857 end product codes, described as a diverse product range. Total output in 2012 was 5.9m units.	Data rich company, MES and ERP are used to store process data, spreadsheet for scheduling.
2	Prototyping	[10-20] Employees, SME	A contract manufacturing service bureau with a wide range of innovative manufacturing technologies for early models and prototypes in different development phases up to small series production.	Information about customers and projects stored in a database, spreadsheet for data concerning the stock levels.
3	Textile	[150-200] Employees, SME	The company receives orders for both machines and lines. It is quite common for a client to order a full line, which is built in its entirety at the company.	Manual paper based data recording, and accounting software not used extensively.

Below we list the approaches developed by the KE tool for these case studies.

- Case 1 was an ideal chance to validate the tool, since it is a data rich company. The test included comparison of the time consumption and data quality to a traditional industrial approach to IDM. This manual procedure consists of: manual data extraction, data filtering and cleaning and data transformations using MS Excel, and distribution-fitting using a commercial software solution. Also, the data quality was validated using hypothesis testing, with a level of significance equal to 95%, on the output results of the simulation model in three Key Performance Indicators (KPIs). Based on the validation results, the use of the KE tool reduced the needed time for arranging input simulation data from 350 minutes (5 hours and 50 minutes) to 65 minutes (1 hour and 5 minutes), a reduction of 81% contrasted to the traditional approach (Barlas, Heavey, et al. 2015). Furthermore, the quality of input data using the KE tool is also tested using statistical hypothesis test and it is proven to be similarly reliable as for the manual approach.

- Cases 2 and 3 as regular SMEs, where the use of DES has received a low level of research by academics and simulation practitioners. In our efforts to support the use of DES in SMEs we developed an approach that facilitates not only the IDM process but also the data collection process since the majority of SMEs don't collect operational data. Figure 15 illustrates a high level representation of the developed approach. The numbered labelled arrows demonstrate the sequence of the generic steps. This approach makes use of the OS modules of the DREAM simulation platform including the KE tool, DREAM GUI (Graphical User Interface) and SE (Simulation Engine). Additionally, an OS relational database, a templating system (Workplan) developed in MS Excel and a GUI to update the database are used. From the description of the required tools, special care was given in order to apply tools that are either already available in a typical SMEs network environment (MS Excel) or easily accessible and customizable since they are OS.

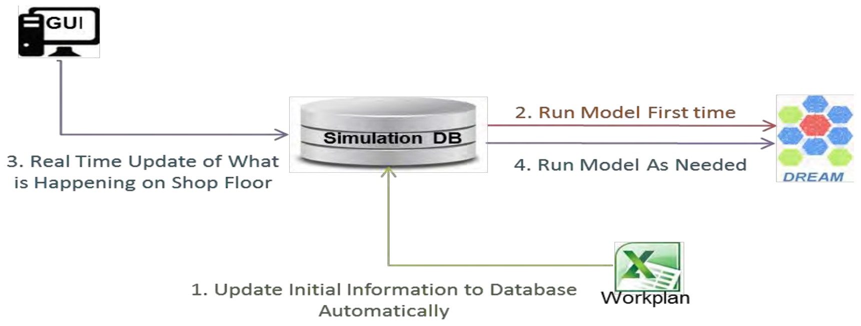

GUI
Simulation DB
2. Run Model First time
4. Run Model As Needed
DREAM
3. Real Time Update of What is Happening on Shop Floor
1. Update Initial Information to Database Automatically
Workplan

Figure 15: Schematic figure showing a high level representation of the designed approach.

7 CONCLUSIONS AND FUTURE WORK

We presented a developed OS tool that offers features that cover the three phases of the IDM process (Data collection, Data processing and Data interface). The paper justifies why we developed the tool in Python and used the RPy2 library. The KE tool which is OS consists of objects that can be used by developers customized for automation of input data management, by integrating the developed objects into a main python script. The tool can output data to an XML format that follows the CMSD format, a spreadsheet customized for SME application, a data format that uses the SIMUL8 XML specification and a JSON format. The OS tool has documentation and is available in GitHub where different versions can be tested using a unit test facility. As we stated above an essential precondition for the employment of the tool is the existence of an IT data repository. Barlas and Heavey (2016) noted that a limitation for the automation of DES applications is the lack of a well-structured database management systems. Future work would be the development of a GUI to facilitate configuration of the tool.

ACKNOWLEDGMENTS

The research leading to the results presented in this paper has received funding from the European Union Seventh Framework Programme (FP7-2012-NMP-ICT-FoF) under grant agreement n° 314364.

REFERENCES

Aufenanger, M., A. Blecken, and C. Laroque. 2010. "Design and Implementation of an Mda Interface for Flexible Data Capturing." *Journal of Simulation* 4 (4):232-241.

Barlas, P., and C. Heavey. 2016. "Automation of Input Data to Discrete Event Simulation for Manufacturing: A Review." *International Journal of Modeling, Simulation, and Scientific Computing.*

Barlas, P., C. Heavey, and G. Dagkakis. 2015. "An Open Source Tool for Automated Input Data in Simulation." *International Journal of Simulation Modelling* 14 (4).

Barlas, P., I. Lanning, and C. Heavey. 2015. "A Survey of Open Source Data Science Tools." *International Journal of Intelligent Computing and Cybernetics* 8 (3):232-261.

Byrne, J., P. Byrne, and A. M. Ivers. 2013. Towards a Cloud Based Sme Data Adapter for Simulation Modelling. In *Proceedings of the 2013 Winter Simulation Conference*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl. 147-158. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Heavey, C., G. Dagkakis, P. Barlas, I. Papagiannopoulos, S. Robin, M. Mariani, and J. Perrin. 2014. Development of an Open-Source Discrete Event Simulation Cloud Enabled Platform. In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, S. D. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller. 2824-2835. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

McNally, P., and C. Heavey. 2004. "Developing Simulation as a Desktop Resource." *International Journal of Computer Integrated Manufacturing* 17 (5):435-450.

Robertson, N., and T. Perera. 2002. "Automated Data Collection for Simulation?" *Simulation Practice and Theory* 9 (6):349-364.

SISO. 2010. "Simulation Interoperability Standards Organization (Siso) Standard For : Core Manufacturing Simulation Data — Uml Model.".

Skoogh, A., and B. Johansson. 2008. A Methodology for Input Data Management in Discrete Event Simulation Projects. In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Monch, O. Rose, T. Jefferson, and J. W. Fowler. 1727-1735. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Skoogh, A., B. Johansson, and J. Stahre. 2012. "Automated Input Data Management: Evaluation of a Concept for Reduced Time Consumption in Discrete Event Simulation." *Simulation: Translations of the Society for Modeling and Simulation International* 88 (11): 1279–1293.

Winnell, A., and J. Ladbrook. 2003. Towards Composable Simulation: Supporting the Design of Engine Assembly Lines. 17th European Simulation Multiconference ESM2003, June.

Zhu, H., P. A. Hall, and J. H. May. 1997. "Software Unit Test Coverage and Adequacy." *ACM Computing Surveys (CSUR)* 29 (4):366-427.

AUTHOR BIOGRAPHIES

PANAGIOTIS BARLAS received his M. Eng in Mechanical Engineering from the Aristotle University of Thessaloniki in 2010. He completed his M.Sc. in Logistics Management at the same University in 2012. He holds a Ph.D. in Operations Research from the University of Limerick. His research interests are in Discrete Event Simulation modelling and Data Analytics. His email address is panagiotis.barlas@ul.ie.

CATHAL HEAVEY is Professor in the Department of Design & Manufacturing Technology at the University of Limerick. He is an Industrial Engineering graduate of the National University of Ireland (University College Galway) and holds a M. Eng.Sc. and Ph.D. from the same University. He has published in the areas of queuing and simulation modeling. His research interests includes, simulation modeling of discrete-event systems; modeling and analysis of supply chains and manufacturing systems; process modeling; component-based simulation and decision support systems. His email address is cathal.heavey@ul.ie.