Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 25 (2014) 283 – 292

www.elsevier.com/locate/procedia

8th International Conference on Digital Enterprise Technology - DET 2014 – "Disruptive Innovation in Manufacturing Engineering towards the 4th Industrial Revolution

From COTS simulation software to an open-source platform: a use case in the medical device industry

Georgios Dagkakis[a,]*, Anna Rotondo[b], Ioannis Papagiannopoulos[a], Cathal Heavy[a], John Geraghty[b], Paul Young[b], Rob Holland[c]

[a]Enterprise Research Centre, University of Limerick, Limerick, Ireland
[b]Enterprise Process Research Centre, School of Mechanical and Manufacturing Engineering, Dublin City University, Dublin, Ireland
[c]Boston Scientific Cork Limited, Cork, Ireland

* Corresponding author. Tel.: +353-61-234200 ; fax: +353-61-213583. *E-mail address:* georgios.dagkakis@ul.ie

Abstract

The implementation of Discrete Event Simulation (DES) – based decision support tools in complex manufacturing environments could prove of invaluable help to industrial practitioners involved in cross-functional decision processes at multiple hierarchical levels. The increasing number of decision variables, their stochastic nature and the non-linearity of their mutual relationships theoretically make simulation a preferred modelling approach for a great variety of manufacturing systems as strict simplifying assumptions are not necessarily required and the models' detail level can be tuned according to the analysis purposes. However, recourse to Commercial Off-The-Shelf (COTS) simulation packages to develop and implement simulation-based solutions in real manufacturing environments usually presents significant cost-of-ownership (COO). Along with license costs, modelling flexibility and sustainability represent fundamental issues raised by industrial engineers that adopt COTS simulation packages. In order to promote the use of DES in production related decision making processes and reduce the associated COO for manufacturing companies, an open-source simulation platform, ManPy, has been developed. ManPy consists of a library of DES objects implemented in SimPy. ManPy's scope is to provide modellers with generic, highly customizable open-source simulation objects that can be connected to form a model in the same fashion of COTS simulation packages. ManPy's on-going development is based on guidelines provided by the analysis of real industrial use cases. Specific pilot models developed in SimPy are used to identify new objects and relevant features to be incorporated in ManPy in order to make it a highly flexible simulation tool. In this article, a use case based on a labour intensive serial production line operating in a medical device manufacturing plant is described. Insights for the transition from a COTS simulation model to a specific SimPy model and finally to generic ManPy objects are presented.

Keywords: Discrete event simulation; open-source software; SimPy; ManPy; modelling flexibility

1. Introduction

Modern manufacturing systems are very complex to manage and control [1]. The presence of multiple production steps, various product flows, machine-product-operators dedication and complex maintenance programs make it almost impossible to investigate the inter-relationships between the system decision variables by using straightforward analytical approaches. As a consequence, simulation is becoming a preferred modelling approach for a great variety of manufacturing systems as strict simplifying assumptions are not necessarily required and the models' detail level can be tuned according to the analysis purposes. However, the choice of simulation comes at a cost [2]. When simulation models are developed for supporting decision making in complex manufacturing environments, the Cost of Ownership (COO) of such models can be high as considerable effort and expertise is required [3,4]. System familiarisation, data mining and model development usually require long time periods and limit the applicability of the simulation approach to tactical or strategic

2212-8271 © 2014 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/3.0/).
Peer-review under responsibility of The International Scientific Committee of the 8th International Conference on Digital Enterprise Technology - DET 2014 – "Disruptive Innovation in Manufacturing Engineering towards the 4th Industrial Revolution"
doi:10.1016/j.procir.2014.10.040

decision problems [2]. Moreover, the natural evolution of a manufacturing system over time or even variations of the system configuration due to the implementation of solutions suggested by the simulation analysis tend to make simulation models quite obsolete fairly quickly after their early use.

By reducing the programming effort, Commercial off-the-self (COTS) DES software packages have contributed considerably to the diffusion of DES in the academic and industrial community. COTS software packages strength is that they provide the user with tools for modelling, debugging and experimentation [5]. The programming effort is, at least in standard cases, significantly reduced as prefabricated DES objects can be manipulated through a user friendly Graphical User Interface (GUI). However, due to the great variety of systems, in most real cases the user does have to write some code in order to model the occasional peculiarities. So most COTS DES tools offer an internal programming language, which is on one hand simplified but on the other hand specific for the users of the tools so support is more difficult to be found, no matter how good the documentation is.

Besides the COO of simulation models, the high license cost of COTS software packages is often the largest factor for organizations, especially Small and Medium Enterprises (SMEs) reluctance to adopt DES. Reusability [6] is also hindered because, in the case where the license is not renewed, past work cannot be exploited to further the system potential. Moreover, practice has shown that even with the use of COTS DES packages many simulation projects fail to achieve their goals. Even though the process is meliorated through the reduction of programming requirements, DES is a complex technique and high modeling expertise is always needed [7].

A more sustainable alternative to COTS DES packages, at least from a cost perspective, is represented by Open Source (OS) [8] DES software, which has the potential to overcome the aforementioned problems. Being available at zero license cost, the adoption of OS software should prove attractive to companies with limited financial resources. Also, for companies investigating the opportunity of deploying DES-based decision support OS software could be used to investigate test cases and assess the benefits deriving from simulation-based analyses. Nevertheless, our review of OS DES [9] has shown that most projects fail to attract contributors and remain inactive. While published literature provides numerous articles that relate to specific OS DES projects [10-12] and COTS DES tools are referenced in many papers [13-15], there are very few articles that compare different OS projects [9,16] or different COTS tools [17] and none which compare the development of a pilot model of a real world production line using both OS and COTS.

In this study, the development and validation of a simulation model developed using two COTS and an OS DES simulation package is illustrated using an industrial use case. The OS DES application used, which is under ongoing development, is based on the simulation engine of a more comprehensive OS simulation platform being developed as part of the DREAM ("simulation based application Decision support in Real-time for Efficient Agile Manufacturing", http://dream-simulation.eu/) project, which also includes data mining tools and a GUI. This platform and its components will provide industrial practitioners with easy-to-use, reconfigurable and efficient simulation-based decision support tools for cross-functional decision processes at multiple hierarchical levels. The DREAM simulation engine is ManPy "Manufacturing in Python". ManPy's on-going development is based on guidelines provided by the analysis of actual industrial use cases. Specific pilot models developed in COTS DES software or SimPy are used to identify new objects and relevant features to be incorporated in ManPy in order to make it a highly flexible simulation tool.

The use case analysed in this study focuses on a labour intensive serial-parallel production line in a medical device fabrication plant. Other than serving as a fundamental application of the ManPy standard objects to a flow-shop manufacturing system, the analysis of this use case has also highlighted the need to develop simulation objects to model specific production flow control logic. In this case, restrictions of the production flow are dictated by statutory medical regulations. Moreover, the different production stations processing different sized batches; and this has inspired the development of batching and unbatching simulation objects whose behaviour has been validated against the results of simulation models built in Plant Simulation® and ExtendSim®.

The choice of using two different COTS packages to validate the ManPy model, is based on the consideration that the two packages chosen support two different modelling approaches. Plant is more flexible, as customisation of the logic within an object is facilitated by means of methods that can be added to the object through a graphical interface, providing a completely object oriented approach. In contrast, such modifications are more difficult in ExtendSim where methods can only be added to an object by accessing the object's inherent code. Often, a combination of standard objects which reproduce the logic is preferred, at the obvious cost of making the model more cumbersome.

The remainder of this paper is organised as follows. Details on the ManPy architecture are given in Section 2. The use case analysed in this study is described in Section 3. Sections 4 and 5 elaborate on the development of simulation models of the use case system using COTS software and ManPy, respectively. The models validation is discussed in Section 6; finally, conclusions are drawn in Section 7.

2. ManPy

ManPy is exclusively written in Python and takes advantage of SimPy's efficient use of Python generators via the SimPy.Process class. SimPy was chosen after a review of 23 OS DES projects [9]. The scope of ManPy is not to imitate or replace SimPy, but to offer something new and accommodate the needs of different levels of users. Highly customizable OS simulation objects that can be connected to form a model, in the same manner as COTS simulation packages, constitute the core of ManPy. The four industrial partners in the DREAM consortium provide relevant use cases to cover a wide range of manufacturing system related problems across four different industrial sectors. This ensures that the development of ManPy is focused toward industry use. The initial phase of the DREAM project focused on gathering industrial partners' requirements and identified three classes of user of the DREAM platform:

- **Super User (SU):** he/she can access the code directly, customize objects flexibly or create completely new ones. He/she needs to have good coding and modeling skills.
- **Industrial Engineer (IE):** he/she can use tailored objects in order to connect them and create a model. Limited coding experience is required at this level, but good modeling skills and knowledge of the system are essential.
- **End User (EU):** he/she takes a model which is tailored for their needs. Specification of certain values using forms, drop-down menus, spreadsheets etc. is the only customization required to set up the model for use. No software or modeling experience is required, only the ability to understand the results of the simulation, the presentation of which should also have been customized.

In order to address these needs, ManPy focuses on developing a repository of well-defined DES objects that can be connected like "black boxes" and form a model. It is worth noting that the SU and IE levels described above may be external consultants rather than internal members of the organization using the DES model.

Object oriented programming [18] is a natural approach in DES where, by default, objects coexist in a model. It is indicative that the ALGOL based simulation language Simula is historically the first that introduced the class concept, becoming the originator of object orientated programming [19]. Considering OS basis for this work and the implementation by independent developers, ManPy's ambition to have DES objects interacting with each other dictates the need for clean and efficient class architecture. The current architecture of ManPy is shown in Figure 1.

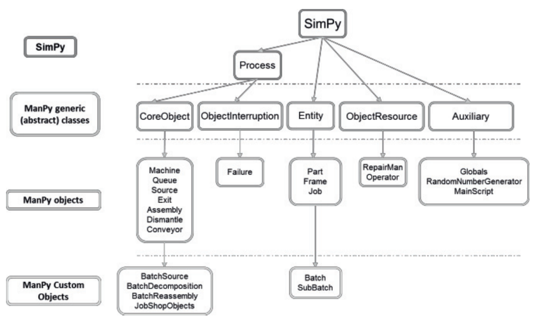

Fig. 1. ManPy architecture.

Figure 1 highlights how ManPy generic classes inherit objects and methods from SimPy and Python. In the next level, there is a basic core of DES objects that inherit from the generic classes and also customized objects that inherit from either existing ones in the basic core or other customized objects.

The repository of ManPy objects should be expandable and customizable. So users may create either completely new or customized objects and incorporate them into the platform and also obtain a repository of objects that might be generic, focused in a specific domain or tailored to a specific organization's needs.

The five generic classes of ManPy objects are:

- **CoreObject:** all the stations which are permanent for the model. These can be servers or buffers of any type and also entry and exit points.
- **ObjectInterruption:** all the objects that affect the availability of another object. For example failures, scheduled breaks, shifts etc.
- **Entity:** all objects that get processed by or wait in CoreObjects and are not permanent in a model. For example parts in a production line, customers in a shop, calls in a call centre etc.
- **ObjectResource:** all the resources that might be necessary for certain operations. For example repairman, operator, electric power etc.
- **Auxiliary:** auxiliary scripts that are needed for different simulation functionalities. For example a main script to create the objects and run the simulation, a script that contains global variables, methods for random number generation etc.

In order to achieve the interconnection of objects, all ManPy classes have to follow a well-defined naming convention of methods and attributes, which define how ManPy objects interact. The fundamental methods required for this model are:

- **__init__:** this is the python constructor method. This method is executed only once, when the instance (for example a specific instance of the type "Queue") is created.
- **initialize:** this method initializes the object for a simulation replication. Not to be confused with the constructor method above, this must be invoked at the start of every replication.
- **canAccept:** returns true if the object is in a state to receive an Entity.
- **haveToDispose:** returns true if the object is in a state to give an Entity.
- **canAcceptAndIsRequested:** returns true only when both conditions are satisfied: the active object is in the state of accepting an Entity and also another object is waiting to give an Entity to it, i.e. its haveToDispose returns True when it is called by the active object. Only when this method returns true the main simulation logic of the object is started.
- **getEntity:** gets the Entity from another CoreObject.
- **removeEntity:** removes an Entity from the CoreObject.
- **createEntity:** creates new Entities in the CoreObject. Most usually used by entry points such as the ManPy Source object, which creates Entities with a defined interarrival time, but every object can potentially use it.
- **calculateProcessingTime:** calculates the processing time of the object.
- **run:** this is a generator method and it is the one where the logic of the progress of the object in simulation time is

implemented. For this reason run requires that the user has also knowledge of SimPy in order to customize.

ManPy, along with other DREAM related code, has recently been released in GitHub (https://github.com/nexedi/dream) under the terms of the GNU Lesser General Public License (LGPL) [20], which means that it can also be used in proprietary DES projects for specific companies. ManPy can cooperate with other DREAM modules such as the Graphical User Interface (GUI), but it can also be used as a standalone project. In the root of the GitHub repository, a ManPy user-manual is also available. The methods described above are the ones that needed to be overridden in order to make tailored objects for our use case. A more comprehensive list of methods can be found in [21]; for further details on both ManPy and its methods, the reader is referred to the software documentation available in GitHub.

3. Use case – system description

Several production lines operate in the medical device fabrication facility involved in DREAM. The pilot line chosen as a use case for supporting the expansion and validation of ManPy is located in a clean room where other lines also operate; the selection of the pilot line was based on the relatively uncomplicated variations between the different product types. Even though there are dimensional variations in the products fabricated on this line, the pilot line can be considered product dedicated, which means the production flow is the same for the all products. The product in question is a medical device for use in operating theatres. It consists of a two part stent; a 1.8m flexible shaft, and a balloon capable of inflation within a human vein. The length of the shaft impacts on product storage capacity along the line as the shaft is kept straight for most of its processing. Items are produced in batches of 100 units. Figure 2 presents a process flow diagram for the pilot line.

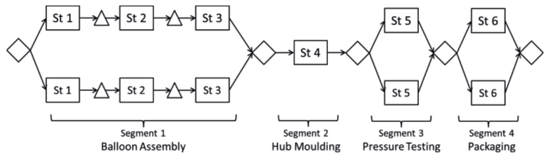

Fig. 2. Schematic layout of the pilot line.

The line can be considered as an assembly line and different tasks can be assigned to each station for balancing purposes. The process flow consists of four main segments;

1. Balloon assembly ;
2. Hub moulding ;
3. Unit pressure testing ;
4. Unit packaging.

The first segment comprises two serial lines that operate in parallel, each having three consecutive stations across which a production batch can be distributed. A production batch is generally divided into several sub-batches which are passed to the downstream buffer once completed. However, due to regulatory compliance a line clearance constraint applies. Line clearance prevents units belonging to different production batches from accidental mixing by imposing a clear and physical segmentation of the line with respect to production batches [22]. This constraint might cause blocking at a station if the buffer between two consecutive stations contains units belonging to a different batch. Before leaving segment 1 a batch must be complete. More generally, distribution of sub-batches across different sections is not allowed; hence, if a section consists of only one station, the entire batch must be taken from the upstream buffer and its processing completed before it can be transferred to the following buffer. It is obvious that in this case, for processing purposes, sub-batches can be created and processed separately within a station but they cannot be transferred to the next station until the entire batch has been completed. As an example, in segment 2, the moulding machine can operate on maximum 2 units at a time; hence, sub-batches of 2 units are effectively created; however, once processed, the sub-batches remain at the station and are not immediately passed to the following buffer. Due to the presence of both batches and sub-batches, it is worth noting that a distinction is made between unit buffers, which hold completed units, and batch buffers, which hold completed batches. In Figure 2, unit buffers are represented by triangles whereas batch buffers are represented by diamonds.

As happens for most of the lines in the plant, the process is labour intensive; each station requires one or more operators. The presence of more than one operator per station is mandatory in the last segment (e.g. packaging) where three operators are required to run the two parallel stations. In some sections, the presence of a number of operators greater than the number of stations enables processing time reductions whenever work division is possible. As an example, in segment 3, unit labelling and pressure tests can be carried out separately; hence, the presence of an additional operator at one of the two stations would help speed up the process.

The production is carried out based on two daily shifts running on weekdays; in order to facilitate Work-In-Process (WIP) balancing, production capacity is opportunely adjusted at different shifts; as an example, being Segment 2 the line bottleneck, one of the sub-lines in Segment 1 is shut down during the evening shift in order to reduce the WIP built up at Segment 2 during the day shift.

As the level of automation in the line is very low, machine failures are generally not a significant limiting factor in terms of productivity. Preventive maintenance is normally carried out during off-shift hours in order to avoid disrupting production. On the contrary, the production is constrained by human operators' availability. Absenteeism might cause shut-downs of different stations and, as a consequence, limit the productivity of the line. In order to avoid this, a number of operators greater than those actually needed to run the line is planned to be available during each shift. In an ideal situation, any possible excess operator available is employed for training purposes.

The main productivity related performance measures used in the company are:

- **Throughput:** this is measured in terms of number of batches produced per hour or per shift;
- **Average unit departure rate:** this is calculated as the average time interval between the completion of two consecutive units;
- **Cycle time:** in this production context, cycle time is defined as the elapsed time between the introduction of a batch into the line and the completion of the batch at the end of the line;
- **Average daily line attainment:** this metric expresses the production quantity achieved in a day in relative terms with respect to the production target set for that day;
- **Line attainment:** this measures the ability to meet daily production targets. It is a binary variable which is set equal to 1 if the daily production target is met, to 0 otherwise.

Moreover, Work in Process (WIP) at the intermediate buffers and stations status related statistics (e.g. working, waiting and blocking %) are monitored during production in order to trigger line rebalancing actions.

4. Development of COTS simulation models

In this section, the assumptions based on which simulation models of the production line described in the previous section have been developed using ExtendSim® and Plant Simulation® are described.

4.1. Modelling assumptions

The models developed in this study aim to capture relevant features of the system described in Section 3 so that ManPy's functionality could be expanded. Conceptual models were developed to facilitate the correct representation of the production flow logic. Production flow constraints (e.g. line clearance) and variation of batch size along the line proved to be novel elements within the ManPy framework and objects able to deal with these features were therefore developed in ManPy.

The models developed here are intended to be used as a tool to facilitate industrial engineers in tactical decision processes related to production line design issues. These issues include line re-balancing (e.g. assignment of tasks to workstations), definition of optimal WIP levels and operating batch sizes at the various stations. For this reason some aspects of the system not closely related to the purpose of this simulation analysis have been neglected in the models. In particular, due to both the high availability of the machines used in the pilot line and the specific preventive maintenance programme adopted, the machines are assumed to be failure free. More generally, the stations are assumed to be always available, this means that at least one operator per machine is available for the entire duration of each shift. The operators' availability constitutes a fundamental problem at an operational level; a reduced number of operators in the line might disrupt productivity. This problem will be addressed separately using an operational decision support tool, where the presence of operators and the associated assignment logic will be introduced in the simulation models.

The production is assumed to be defect free and scrapping of units is not modelled. This assumption is reasonable as the production yield observed in the production line is greater than 95%. It is worth noting that even though the validation experiments ignore the presence of quality defects, ManPy objects able to model scrapping of defective units have been developed.

Starvation due to the lack of raw material at the various stations (e.g. components of the stent) is not considered in the models as it is rarely experienced in the real line; a kanban-based inventory system is adopted in the clean room where the pilot line operates and ensures that both production unit components and disposable tools required to complete the tasks at a station are always available. This also applies to the first station in the line; hence, the system operates based on a pull production logic. As a consequence, production entities are infinitely generated in the simulation models. In order to make the computation more efficient, the production entities modelled consist of the smallest transfer sub-batches observed in the line in accordance to the line clearance constraint. In this case, sub-batches of 25 units, which are used in the first segment of the line, are generated and processed in the models; the number of units available in each production entity is represented as an entity attribute. The sub-batches are batched and unbatched based on the stations' processing requirements; when sub-batches cannot be transferred to the following stations once completed (e.g. segments 2, 3 and 4), the entities are batched and transferred as a 4 entity-batch. Processing times are calculated based on the number of units that a production entity (e.g. batch or sub-batch) at a station carries with it. Deterministic values are used to model processing times; this is based on the practice adopted in the company to treat times as constant values at tactical decision levels. Stochastic processing times will be used when simulation applications for operational decision support tools will be developed.

4.2. ExtendSim Model

Based on the assumptions above, the pilot production line has been modelled using ExtendSim standard blocks. Several libraries of blocks are available in ExtendSim to handle most modelling needs; these blocks can be linked using connectors and assembled in hierarchical blocks of sub-systems. The model layout as appears in the graphical interface at a high hierarchical level is shown in Figure 3.

Following the logical flow in Figure 3, production entities (e.g. sub-batches of 25 units) are generated at deterministic time intervals so that the initial buffer of the line is always at full capacity except during the simulation start; attributes are assigned to the entities which are then batched into batches of 4 entities. The initial batch is created so as to facilitate the handling of common properties across the sub-batches that form a batch. A batch waits in the initial buffer until one of the two unbatch blocks that precede the machines at the first station of Segment 1 becomes available. The unbatch block separates the full batch into 4 sub-batches and is used as a temporary storage for these production entities; the sub-batches are released one by one as soon as the machine downstream of the unbatch block becomes available. When the last sub-batch of a batch leaves the unbatch block another full batch is pulled from the initial buffer. During the batch and unbatch operations the entities' uniqueness is preserved so that the properties assigned to each single entity prior the batch assembly are not overwritten by the batch's properties.

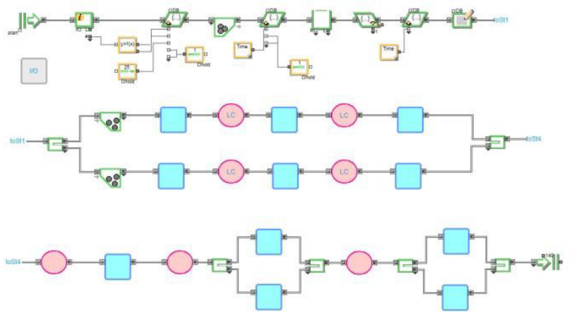

Fig. 3. ExtendSim model layout.

The machines operating at a station are modelled using hierarchical blocks, which are represented as blue squares in Figure 3. The layout of a Machine block is shown in Figure 4. An initial gate limits the number of entities that can enter a machine. For most machines this gate closes when an entity enters the Machine block; the gate will re-open when the entity leaves the block so that only one entity at a time is allowed in the block. There are exceptions to this logic for the machines of station 3 where sub-batches are re-assembled in a full batch before they leave the station. In this case, 4 sub-batches are allowed in the block before the gate closes; the gate will re-open when the full batch leaves. The choice of introducing a gate at each Machine block is related to the choice of separating the blockage status of a machine from its waiting and working status. In order to realise this, a "blockage" buffer of capacity 1 has been introduced immediately after the Activity block (e.g. the block where the processing delay is imposed) in the Machine block. The reason for this choice is twofold. Firstly, due to modelling requirements, a "dummy" buffer is required before the conditional gate at the entrance of the Line Clearance (LC) buffers (e.g. buffer that follows the Machine block); this prevents production entities not allowed in the LC buffers from being held in a block (e.g. gate) that has no corresponding element in reality. Secondly, the presence of a "blockage" buffer proves quite useful at the machines where batch re-assembly is required after processing (e.g. Station 3) in order to separate the machine waiting time from its blockage time. When batch re-assembly is required, the completed sub-batches are held in the batch block, which acts as a temporary storage; even though the presence of the initial gate in the Machine block would prevent any other sub-batch from entering the block, it would be complex to distinguish the waiting time of sub-batches in the batch block, which does not impact the machine status, from the blocking time of the full batch in the same block, which corresponds with the machine's blocking time. The presence of the blockage buffer allows re-assembled batches to immediately leave the batch block so that the blockage time at the associated machine can be calculated as the queuing time in the "blockage" buffer. The remaining blocks in the Machine block are used to update and record information about the entities. Among these information blocks, there are blocks used for calculating the processing time required for any given production entity based on the current production step, the unit processing time (e.g. this is read from an MS Excel worksheet) and the number of units that form the entity currently processed (e.g. batch or sub-batch).

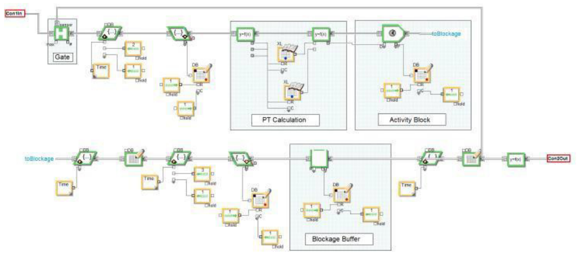

Fig. 4. Machine block layout.

The intermediate buffers in the ExtendSim model are also modelled as hierarchical blocks and represented as pink circles in Figure 3. The buffers where the Line Clearance concept applies with its strictest meaning are those placed between Segment 1 stations (labelled 'LC' characterizes these buffers in Figure 3); the graphical layout of the LC buffer block is reported in Figure 5.

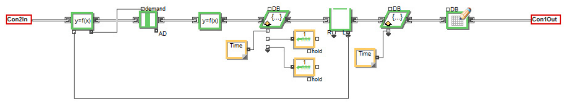

Fig. 5. Line Clearance Buffer layout.

A conditional gate regulates the flow of production entities into the finite capacity buffer. This gate remains open if the buffer is empty; when the buffer holds some entities but it is not at full capacity, the passage of entities through the gate is allowed provided that the entity going through belongs to the same batch as the entities currently held in the buffer. This prevents entities belonging to different batches from mixing, as the Line Clearance constraint requires. When the buffer is full no entity can enter the block even though the gate is open. In order to keep track of the batch ID of the batches held in the buffer, a database has been created where the batch ID of any entity that passes through the buffer is recorded. In the upstream buffers for Segments 2, 3 and 4 the implementation of the Line Clearance constraint is looser as the units belonging to a batch are confined in storage boxes; hence, there is no risk that these units could be mixed. These buffers can hold different batches and the only restriction to the upstream flow is given by their maximum capacity. In terms of ExtendSim block layout, these buffers appear the same as the block in Figure 5 with the only exception that the gate and the preceding equation block are not included for obvious reasons.

Information on the entities schedule and the machines' status are recorded in internal databases and exported to MS Excel worksheets at the end of each simulation run. The simulation data obtained are post-processed in MATLAB V7.12® where a more readable format for the batches' schedule and relevant statistics are generated.

4.3. Plant Simulation model

Plant Simulation is a very powerful DES tool developed by Siemens, providing an Object Oriented approach, where the user can make highly customized new objects inheriting their properties either from the prefabricated ones that Plant Simulation provides or from other customized objects. The model layout, as it appears in Plant Simulation GUI is (Figure 6) is straightforward. A Source creates the Batches, which are queued before they pass along one of the sub-lines and then they follow their route in the system. Two methods are invoked when the model is reset and at the end of the simulation. *reset* ensures that the initial state of a run is not affected by the previous one by resetting certain statistics and control variables that were added for the needs of custom objects and *EndSim* outputs results at the end of the run.

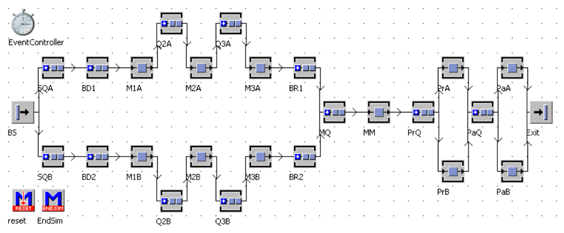

Fig. 6. Plant Simulation model layout.

While the graphics reflect the layout of the line and appear simple, the actual model contains hidden complexity. As shown in Figure7, new classes of Plant Simulation objects with their own custom methods and attributes were required. These are either instances or children of Plant Simulation's *MUs* (Mobile Units that flow in the simulation model), *SingleProc* (a single station for processing an *MU*), *PlaceBuffer* (lines up several *MUs* of the same kind one after the other), *Source* (produces *MUs* in a single station) and *Drain* (removes *MUs* from the model). More info on Plant Simulation objects can be found in [23].

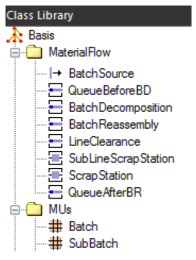

Fig. 7. Plant Simulation objects implemented in the model.

The new classes created are:

- **BatchSource:** A *Source* that creates *Batches.*
- **Batch:** An *MU* that holds a number of units.
- **SubBatch:** An *MU* that contains a number of units extracted from a parent *Batch*. Its attributes include the id of the parent *Batch*.
- **BatchDecomposition:** A *PlaceBuffer* that takes a batch and separates it into sub-batches.
- **BatchReassembly:** A *PlaceBuffer* that takes a number of sub-batches and reassembles the original batch.
- **LineClearance:** A *PlaceBuffer* that can take a sub-batch only if it is completely empty or the sub-batches that it already holds are derived from the same parent batch as the one that requests entry.
- **ScrapStation:** A *SingleProc* that can scrap some units of the MU it processes. Its processing time is calculated according to the number of units in the MU. Scrapping of units has being modelled both in Plant Simulation and ManPy in order to be used in future work, but in the experiments presented here scrappage is not considered.
- **SubLineScrapStation:** Almost identical to the *ScrapStation* class with the addition of extra coding to interact with the *LineClearance* class.
- **QueueBeforeBD:** A *PlaceBuffer* with modifications to allow communication with the downstream *BatchDecomposition* object.
- **QueueAfterBR:** A *PlaceBuffer* with modifications to communicate with the upstream *BatchReassembly* object.

Table 1. Object types and instances used in the model shown in Figure 6.

Object Type	Objects Used in the Model
BatchSource	BS
BatchDecomposition	BD1, BD2
BatchReassembly	BR1, BR2
LineClearance	Q1A, Q2A, Q1B, Q2B
ScrapStation	M3A, M3B, MM, PrA, PrB, PaA, PaB
SubLineScrapStation	M1A, M1B, M2A, M2B
QueueBeforeBD	SQA, SQB
QueueAfterBR	MQ
PlaceBuffer	PrQ, PaQ
Drain	Exit

Table 1 shows the instances of the object types used in the model. These correspond with the objects shown in Figure 6. Despite the introduction of new classes, some further customization was needed for specific objects in the model. It was found that the PlaceBuffer (MQ), which is placed after the two BatchReassembly objects (BR1, BR2), should accept the item which has been waiting the longest in those buffers. This is the default behaviour in Plant Simulation, but the customizations negated it. For this reason some methods needed further modifications. In general, this is considered poor Object Orientation since methods were customised for specific instances. In complex situations it is often quite difficult to identify a straightforward model development approach. In this specific case, there may have been an easier way to model this feature but, in general, this is indicative of how complex modeling of specific processes can become.

5. ManPy model development

ManPy aims in providing the user with a selection of CoreObjects that exchange Entities during simulation time. In the initial development of ManPy, Entities were constructed to flow as separate objects or parts that could get loaded into and unloaded from frames. This pilot case has supported the extension of ManPy so that Entities can flow in batches, be decomposed into sub-batches and then reassembled to the parent batch. Some other particular features involve the line clearance concept, scrappage (even thought this feature is not included in the validation process) and also that the processing times in station is dependent on the number of units the processed Entity holds.

In order to model such a case new ManPy objects had to be implemented:

- **Batch:** A ManPy Entity that holds a number of units.
- **SubBatch:** A ManPy Entity that contains a number of units and is derived from a parent Batch. In its attributes it also holds the parent Batch.

Also we needed two new CoreObject types:

- **BatchDecomposition:** A CoreObject that takes a batch and decomposes it to sub-batches. It overrides CoreObject's _ *init__, initialize, canAccept, hasToDispose, canAcceptAndIsRequested* and *run* methods in order to implement the new logic.
- **BatchReassembly:** A CoreObject that takes a number of sub-batches and reassembles the original batch. It overrides CoreObject's _ *init__, initialize, canAccept, hasToDispose, canAcceptAndIsRequested* and *run* methods in order to implement the new logic.

Finally customizations in already existing CoreObjects were made:

- **BatchSource:** A CoreObject that inherits from Source and creates Entities that hold a number of units. It overrides the *__init__* method of the Source in order to accept the number of units per batch as an argument and also the *createEntity* method so that it creates the Entity containing this number of units.
- **BatchScrapMachine:** A CoreObject that inherits from Machine. It overrides *init__* in order to use the scrappage distribution, *removeEntity* in order to scrap some units of the Entity when it is ready to leave the object and also *calculateProcessing* time in order to define the processing time according to the number of units that the Entity had when it entered.
- **LineClearance:** A CoreObject that inherits from Queue and overrides its *canAccept* and *canAcceptAndIsRequested* methods, so that it can take a sub-batch only if it is completely empty or the sub-batches that it already holds are derived from the same parent batch as the one that requests to enter.
- **BatchDecompositionStartTime**: A CoreObject that inherits from BatchDecomposition and overrides its *removeEntity* method in order to assign the start time of the Entity as the simulation time that the Entity got into this object.

From the final object in the list above we can see that it was necessary to customize one of the newly created objects, in order to calculate the cycle time as defined in section 3. This makes sense since BatchDecomposition can be a generic object to be used in multiple cases, while BatchDecompositionStartTime is just needed in this specific model where the start of the lifecycle must be recorded in these stations.

As expected, the most time consuming part of the implementation was the coding of BatchDecomposition and BatchReassembly. These are brand new CoreObjects in which also the Python generator *run* had to be overridden. These methods run in parallel, making it more difficult to debug them. In fact, it is the overriding of the *run* method that discriminates a completely new CoreObject from a customization of an already existing one. Indeed BatchDecomposition and BatchReassembly were the only new CoreObjects to directly inherit from the generic class. The generic class is abstract and does not include a specific implementation of *run*. So every object that directly inherits from the generic class has to have a *run* method developed for it. Also, it is not obligatory, but in most cases it seems that if a new CoreObject needs to override *run* of the parent object, then its logic is peculiar, so it makes sense to inherit directly from the generic class.

It is important to mention that during the implementation of the new objects no modifications to the underlying ManPy code were needed. So the ManPy repository was considered as a black box, where we have been able to create new objects and use them in models together with the pre-existing objects. This fact is essential to demonstrate the expandability of the platform.

The procedure described so far lead to the implementation of tailored objects for this use case. Effectively, this is the repository of objects that an Industrial Engineer user should have in order to model the specific problem defined by the industry partner. More objects, such as operators, will be needed in order to get the full model of the line. This is left for future work; the scalability of this development approach offers the opportunity to progressively gain confidence in the objects developed hereinbefore.

In order to use ManPy DES objects in a model, a separate Python script should be used. This is an auxiliary script and we refer to it as “main script”. Main script is responsible for creating, connecting, activating and initializing the DES objects, defining general simulation attributes, running the simulation and outputting results. The main script may be raw Python code in the way it is presented in examples of ManPy documentation, but there are also other ways to define the model. One of them is reading from a JSON [24] representation which in this case is used for interaction with the DREAM GUI. The definition of the pilot line was chosen to be implemented using the JSON representation to combine this model with the DREAM GUI. Generally the main script is separate of the actual simulation code, so future ManPy users can implement their own approaches to input and output definition.

6. Models validation and comparison

Validation and verification are complex issues that should be taken care all through a simulation project [25]. Validation aims at ensuring that a real system is validly captured in a model whereas verification has the purpose to confirm that a model is correctly coded as a computer program [26]. In order to define the scope of the study and gain understanding of the logic of the simulated system a workshop took place at the facility where the production line operates. Following the workshop, regular communication with the industrial engineers familiar with the pilot line guided the development of the conceptual model and ensured that the modelling assumptions were correct. A formal questionnaire and a Request For Information (RFI) were also used as support documents. In particular, details and peculiarities of the pilot line, such as the line clearance concept, were extensively clarified before the models were developed. A prototype model of the pilot line was illustrated to the industrial engineers; the production flow logic implemented, inputs and outputs were discussed, whilst the simulation logic and results were considered valid, the need to include further features of the system so that the model could be used to support operational decisions was expressed. More specifically, the presence of operators in the line should be modelled and allocations algorithms should be developed in order to assist the line supervisor in assigning operators to workstations in case of absenteeism. This will be done in the near future.

To further assess the validation of our modeling, we plan its review by experts of the system within the company. This will be firstly done using the COTS models, since the medical device company where the pilot line operates are familiar with using COTS simulation packages and also the GUI of these packages is at a more mature state to enhance a structured walkthrough. Nevertheless, if a COTS model is adequately validated and ManPy model is verified against this COTS model, then this is considered enough for the validation of the ManPy model. The development of the model using three different tools, one OS and two COTS, was followed in order to establish the verification concept. Firstly, the models were built and debugged for a specific set of inputs. It is worth mentioning, that debugging revealed not only errors in new ManPy objects, but also mistakes in the modeling in COTS. Both COTS tools used are in a more advanced state than ManPy, but the deep level of control we had in the latter, was sometimes essential in identifying problems and solutions.

A structured experimental plan was developed to validate the models from a numerical perspective. Realistic data were used as access to historical data proved difficult at this stage of the project; significant effort is required to interpret the data available in historical database and data mining tools are being developed to automate inputting relevant simulation data in the models. Deterministic input data were used since the purpose of the experiments was the verification of models and code. The experimental plan was conceived so that the new objects developed in ManPy could be verified under different operating conditions. Two sets of experiments were run; each set consisted of seven experiments. In the first six experiments each of the six stations populating the line was set as a bottleneck, respectively; in the last experiment the production flow was made synchronous so that the system would not present any bottleneck. In order to achieve these conditions, the unit processing times were opportunely set. The second set of experiments replicated the first set using a different inter-arrival time for the production batches; this was done to assess impact on waiting time at the different stations. The performance measures considered in the validation process were the daily throughput, the average cycle time and the machine status percentages (e.g. waiting, working and blockage).

For all the experiments the results proved identical; hence the validity of ManPy in simulating flow shop systems subjected to both variation of production batch sizes and production flow constraints has been demonstrated. It is worth noting that as a result of the validation process, one bug in ManPy that would cause a premature interruption of the simulation run has been identified and fixed. This shows the importance of comparing ManPy performance against COTS packages; comparison analyses based on sensible experimental plans prove fundamental in order to verify ManPy's logic and set effective development guidelines.

In terms of speed ManPy proved slower than COTS alternatives. This issue was also demonstrated in [21] where the results obtained are similar to those obtained here. The pursuit of generality of OS code does induce an overhead of computations. Nevertheless, ManPy can be less expensively used in a cluster of PCs, since there is no license needed in order to install it. That can enhance the speed significantly using a web-based distributed simulation in cases where we have multiple replications of stochastic models or scenario analysis. It is in the scope of DREAM to research such methodologies in order to provide a high performance DES framework.

7. Conclusions

ManPy is the simulation engine of the DREAM platform whose ultimate objective is to provide industrial practitioners with easy-to-use, expandable and efficient simulation based decision support tools for industrial problems at different decision levels. In this study, the application of ManPy to a flowshop system characterized by batch production, varying batch size and production flow constraints has been illustrated. The ManPy model has been successfully validated against the simulation results of two versions of the same model developed using COTS software. Being under development, the validation of newly introduced ManPy objects against corresponding objects available in COTS software represents a fundamental step for proving that ManPy is able to offer a valid license free alternative to more commonly used simulation packages. Based on both validation results and the identification of relevant industrial use case features, effective development guidelines have been set to enhance ManPy's functionalities. ManPy's most significant advantage with respect to other OS simulation packages available consists of the fact that objects similar to those available in COTS simulation packages are ready to use and can be connected with minimal programming effort.

Future work regarding this pilot case and DREAM in general include: the integration of the model with the tailored GUI that will enhance the human/system interaction and the DREAM Knowledge Extraction (KE) tool that will automate of the ManPy simulation inputs. To model the system at an operational level, new ManPy objects will be needed and more specifically an operator object since the efficient allocation of operators is regarded crucial by the company. Moreover, the model will be combined with a scenario analysis approach, so that it can be used for decision support. If this is proven to be computationally expensive, then research will be conducted, so that ManPy is deployed over a cluster of computers.

Acknowledgements

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

References

[1] Ramirez YM, Nembhard DA. Measuring knowledge worker productivity. J of Intellectual Capital 2004;5:4602-628.
[2] Carson JS. Introduction to modeling and simulation. In: Proceedings of the 36th Winter Simulation Conference 2004. p. 9-16.
[3] Salman MR, Geraghty J, Brady M. The development of a discrete event simulation framework for complex manufacturing environments, In: Proceedings of EUROSIS-ETI conference 2010.
[4] McLean C, Leong S. The expanding role of simulation in future manufacturing, In: Proceedings of the 33rd Winter Simulation Conference; 2001. p.1478-1486.
[5] Pidd M, Carvalho A. Simulation software: Not the same yesterday, today or forever. J of Simulation 2006;1:17-20.
[6] Paul RJ, Taylor SJ. What use is model reuse: Is there a crook at the end of the rainbow?, In: Proceedings of 2002 Winter Simulation Conference; 2002. p. 648-652.
[7] Kuljis J, Paul RJ. An appraisal of web-based simulation: Whither we wander? Simul Pract Theory 2001;9:137-54.
[8] Fitzgerald B. The transformation of open source software. Mis Quarterly 2006;587-598.
[9] Dagkakis G, Heavey C, Papadopoulos CT. A review of open source discrete event simulation software, In: Proceedings of 9th Conference on Stochastic Models of Manufacturing and Service Operations; 2013. p. 27-35.
[10] Varga A, Hornig R. An overview of the OMNeT++ simulation environment, In: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops; 2008. p.60.
[11] King D, Harrison HS. Discrete-event simulation in java: A practitioner's experience, In: Proceedings of the 2010 Conference on Grand Challenges in Modeling & Simulation.
[12] Rossetti MD. Java simulation library (JSL): An open-source object-oriented library for discrete-event simulation in java. International J of Simulation and Process Modelling 2008;4:169-87.
[13] Bengtsson N, Shao G, Johansson B, Lee YT, Leong S, Skoogh A, Mclean C. Input data management methodology for discrete event simulation. In: Proceedings of the 2009 Winter Simulation Conference.
[14] Zhang SS, Peng H, Liu C, Yao LL. Simulation of genetic algorithm scheduling on extendsim based on COM technology, In: Proceedings of the 2012 International Conference on Modelling, Identification and Control.
[15] Bergmann S, Stelzer S, Strassburger S. On the use of artificial neural networks in simulation-based manufacturing control. J of Simulation 2014;876-90.
[16] Weingartner E, Vom Lehn H, Wehrle K. A performance comparison of recent network simulators, In: Proceedings of 2009 IEEE International Conference on Communications (ICC'09).
[17] Tewoldeberhan TW, Verbraeck A, Valentin E, Bardonnet G. Software evaluation and selection: An evaluation and selection methodology for discrete-event simulation software, In: Proceedings of the 34th Winter Simulation Conference; 2002. p. 67-75.
[18] Smith B. Object-oriented programming, AdvancED ActionScript 3.0: Design Patterns, Springer; 2011.
[19] Rentsch T. Object oriented programming. ACM Sigplan Notices 1982;17:951-57.
[20] Ueda M. Licenses of open source software and their economic values, In: Proceedings of the 2005 Symposium on Applications and the Internet Workshops.
[21] Dagkakis G, Heavey C, Robin S, Perrin J. ManPy: An open-source layer of DES manufacturing objects implemented in SimPy, In: Proceedings of the 8th EUROSIM Congress on Modelling and Simulation; 2013.
[22] http://www.gpacmfg.com/ISO_13485_Line_Clearance.html
[23] Bangsow S. Manufacturing Simulation with Plant Simulation and Simtalk: Usage and Programming with Examples and Solutions. Springer; 2010.
[24] Severance C. Discovering JavaScript object notation. Computer, 45:46-8.
[25] Law AM, Kelton WD. Simulation modelling and analysis. McGraw Hill Boston, MA; 2000.
[26] Shannon RE. Introduction to the art and science of simulation. In: Proceedings of the 1998 Winter Simulation Conference, p.7-14.