Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

THE OBJECT-ORIENTED DISCRETE EVENT SIMULATION MODELING: A CASE STUDY ON AIRCRAFT SPARE PART MANAGEMENT

Haobin Li

Institute of High Performance Computing
Department of Computing Science
1 Fusionopolis Way, 138632, SINGAPORE

Yinchao Zhu
Yixin Chen

National University of Singapore
Department of Industrial and Systems Engineering
1 Engineering Drive 2, 117576, SINGAPORE

Giulia Pedrielli

National University of Singapore.
Centre for Maritime Studies
15 Prince George's Park, 118414, SINGAPORE

Nugroho A. Pujowidianto

Hewlett-Packard Singapore
Business Printing Division
138 Depot Road, 109683, SINGAPORE

ABSTRACT

Object–Oriented DES (O^2DES) is an effort to implement the object oriented paradigm in the scope of ease the development of discrete event simulation models in both education as well as industrial settings. In particular, O^2DES offers several functionalities which support the integration of the tool with optimization techniques, thus making it easier to the students to understand the concept of simulation–optimization. It also supports the application of different variance reduction techniques such as budget allocation and time dilation. In order to do so, the provided toolkit exploits the C# language and the .NET Framework and it guarantees the efficient generation of DES models, as well as the effectiveness of the developed models in being integrated with sampling solutions. We propose a case study related to the aircraft spare part management problem to show case the main functionalities of the proposed tool.

1 INTRODUCTION

In the past decade, Discrete event simulation (DES) has established in many industrial realities as the main tool for evaluation of the performance of critical portions of the system. More recently, it has been integrated with optimization in many commercial software, in order to support decision makers in the improvement/redesign of industrial systems. Examples of commercial software packages providing this functionalities are Arena (Kelton et al. 1998), AutoMod (Muller 2011), FlexSim (Nordgren 2002). These software packages provide a graphical interface to support the users building the model and they offer animation features important to understand the dynamics of the system resulting from the generated model. Simulation is also a key topic in education and DES in particular. Despite the fact that DES is event based, most of the simulation software provided to students to start creating their first simulation models are process–based, i.e., they do not explicitly expose the event and the events list (Kelton et al. 1998). On the contrary, purely event –based simulation paradigm typically do not expose the state of the system, which is, instead, implicitly defined through the events sequence (Schruben and Yucesan 1993).
In relationship to this, O^2DES represents the effort to provide students with a toolkit enabling, simultaneously,

the description of the sequence of events as well as the status of the entities of the model, thus mirroring the contents of a simulation class.

In addition to this, we recognized that, despite optimization and variance reduction have been since long integrated within the simulation class material, still the tools used by the student to develop models do not enable interactive linkage with the optimization as well as some variance reduction functionality (Nelson 2013). Besides these functionalities, advancing technologies have influenced the latest developments in the DES community. As computing capability increases, due to the emergence of parallel high-performance and cloud infrastructures, not only the evaluation of significantly complex and large systems has been enabled, but the active integration of simulation and optimization is a practical alternative to more traditional search algorithms based on approximated analytical models of the system under analysis. Due to its object–oriented characteristic, O^{2}DES simplifies the incorporation of such techniques making it even more useful to student not only from engineering, but also from computer science background.

In fact, if optimization is considered in the setting of simulation, one of the main criticality raises from the need to evaluate a large number of configurations when a solution needs to be identified (Fu 2002). Hence, calling the simulator has to be fast. In particular, the set–up times due to the interaction of the simulation with the optimization have to be reduced to the minimum (Fu et al. 2014). Nevertheless, this interaction is only one of the main source of inefficiency in the current implementation of simulation–based optimization. As an example of this inefficiency, several optimization architectures use a database to store a great amount of simulation results, to make them available to the search procedure. Such a decoupled approach can largely hinder the efficiency of the procedure. To facilitate the communication between simulator and other layers in the optimization architecture, it is desired to have all parties to talk through interfaces under a common framework. Certainly, one possible choice is to select a common programming language that is robust to develop all parties.

Besides, the simulation modeling has to be more flexible and modular in order to enable the following functionalities in an optimization setting: (1) starting from an initial configuration, new configurations should be automatically generated driven by the optimization procedure; (2) simulation parameters such as the replication length, the experiment time scale or the number of replications might be modified dynamically and or interactively as the simulation progresses under to reduce the output variance.

The second point is strongly related to the problem of improving efficiency of simulation in optimization. For the past two decades, many contributions have been proposed on how to efficiently allocate replications for DES in the process of identifying optimal solutions (Chen et al. 1997). Almost at the same time, Schruben (1997) proposed the concept of event time dilation, also aiming to improve the efficiency of the optimization procedure. Recently, some have also initiated the research on parallel optimization to utilize the advantages of multi-core computers and cloud computing (Fu et al. 2014). Despite the aforementioned research effort, few commercial simulation modeling paradigms can support all the advanced features mentioned above, because many simulators are rigid, and the interaction with the simulator while the simulation is running is not a provided functionality.

Adopting an object–oriented paradigm can largely simplify the development of a simulation model and enable the interaction and integration with the optimization component, by creating objects that the simulation can expose to the optimization in order to be modified.

In this paper, we propose the first steps towards the development of a new paradigm for Object-Oriented DES modeling, O^{2}DES, which we developed in C#. In our study, all the advanced features mentioned above were taken developed and implemented. We hope that this work could inspire a new alternative and serve as a reference for the DES simulation & Optimization community.

The remainder of the paper is structured as follows: section 2 provide the basis language and the main reference literature at the basis and motivating our work. Section 3 presents the object–oriented simulation paradigm, whose main components are described in section 4, while the main functionalities are the subject of section 5. Section 6 shows a case study on the aircraft spare part management problem.The case study shows the object–oriented procedure to generate a discrete event simulation model, while proving the

easiness of incorporating the generated simulation model with optimization algorithms and two advanced simulation budget control techniques. Finally, section 7, closes the paper.

2 BACKGROUND

DES models the dynamics of a system by executing by an ordered sequence of discrete events which defines, at any point, the simulation events' list (Banks and John S. Carson 1986, Schriber et al. 2014). In a DES, an event occurs at a particular point in time and it may determine changes in the system state. Specifically, a discrete event system might change its state *only* when an event triggered and the system dynamics evolves according to the scheduled events list. DES is widely used in areas of optimization, process improvement, and network analysis. In manufacturing industries, it is the main technique for the analysis of reliability, capacity and maintenance improvements, as well as for evaluations of alternative designs (e.g., plant expansions, capital investment options, or cycle time reduction and safety (Sharda et al. 2011)).

For DES modeling, many high–level simulation languages have been introduced due to the fact that simulation programs are comparatively difficult to write in machine languages. A history of simulation including the DES software development can be found in Nance and Sargent (2002). As an example, SIMULA is programming language that was extended from ALGOL 60, a language specifically developed for simulation, facilitating the formal description of layout and rules of operation of systems. SIMULA uses the method of quasi–parallel processing for performing operations in "active phases" or "events" (Dahl and Nygaard 1966).

Many commercial softwares have been developed enhancing the user interface and promoting the spread of simulation in education and industries. As an example, Arena©, developed based on SIMAN simulation language and CINEMA graphic libraries, uses a hierarchical approach to provide the user with an object-oriented simulation language and flexible system definition for end-user (Hammann and Markovitch 1995). In addition to the generation of simulation–specific languages and commercial software, several open source libraries have also been released developed in general purpose programming languages. Examples are adevs in C++, MASON in Java and SimPy in Python. In particular, C# has been used to develop DES software (Choi and Kang 2013). Some examples of C# DES software are SharpSim, DEVS#, Activity Cycle Executor (ACE). However, the focus of the aforemetioned platform is performance evaluation, whereas optimization is not of concerns.

As a result, despite numerous contributions have been provided in the field of performance evaluation by means of DES, there is a need to develop a tool for efficient DES modeling which, at the same time, enables incorporating the recent advances in the simulation optimization.

In particular, a new optimization–oriented simulation framework is required which try to satisfy the following needs: 1) efficient and automated generation of alternative configurations; 2) run–time accessibility to database to store scenario data and candidate solutions, as well as (3) for designing and incorporating heuristics and various optimization algorithms; and 4) run–time access to simulation parameters for output accuracy on-line control.

In light of these requirements, we propose O^2DES as an alternative way to build DES model in C# language.

3 THE GENERAL FRAMEWORK

Any DES model consists of two major components: a system clock that indicates the time of the simulated system, and a future events list (FEL) that stores all events scheduled to happen at a future clock time. When a simulation model is running, the *head event* (i.e., the one with the earliest scheduled time) in the FEL is selected for execution and the system clock is updated to the head event time. Specifically, when an event is executed, some status property of the simulation model might change and new events may

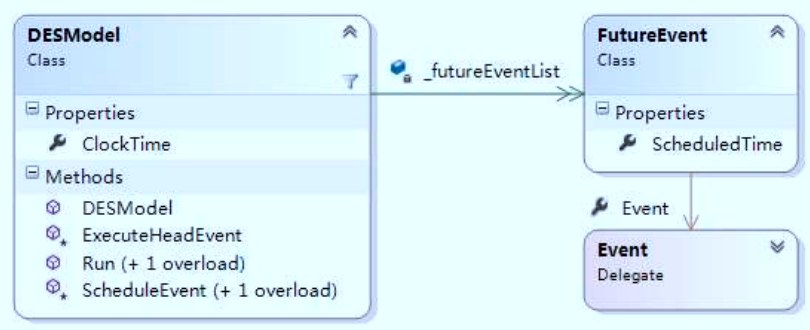

Figure 1: Class diagram of a DES framework.

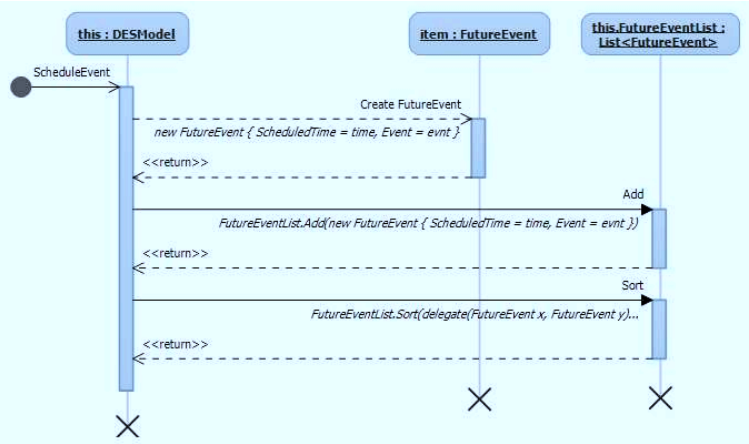

Figure 2: Sequence diagram for scheduling a future event.

be scheduled in another future clock time. The procedure is repeated until the FEL becomes empty, or a terminating condition for the simulation is met, e.g., a specified clock time, or number of events is reached.

The class diagram of the DES framework implemented in C# is shown in Figure 1. And we have also attached the sequence diagrams for the core functions in the framework, i.e., scheduling a future event (Figure 2) and executing a head event (Figure 3). Although it is simple and compact, the framework could constitute the fundamentals for all DES models implemented with Object Oriented Programming (OOP) paradigm.

In addition to this basic features, we further defined the main O^{2}DES components and functionalities to meet the highlighted challenges at the end of section 2.

4 BASIC COMPONENTS

O^{2}DES provides four main components the user can adopt to build a simulation–optimization model: (1) *Scenario* and *Static Components*, (2) *Status* and *Loads*, (3) *Event*. It is noteworthy that, due to the use of C#, we did not need to create objects for distributions or for the definition of queues (at least up to this point). In fact, distributions as well as random number generation engines, are provided by means of the C# library MathNet.Numerics provided in Visual Studio NuGet Packages. Also, queues can be modeled by directly referring to the native libraries using the *queue*, *stack* or *list* objects.

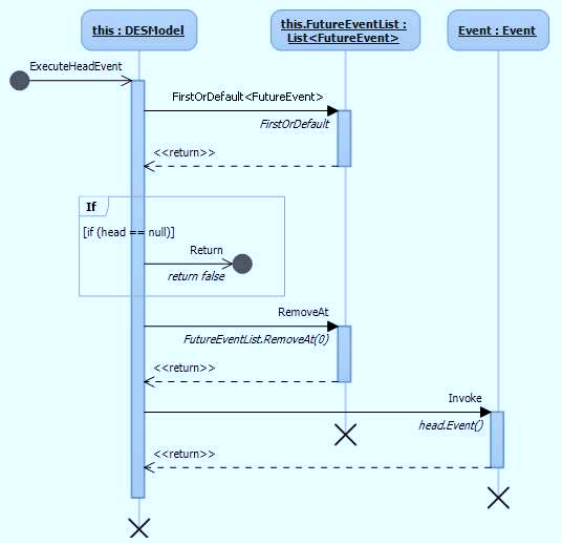

Figure 3: Sequence diagram for executing a head event.

4.1 Scenario & Static Components

The *static* components refer to all entities whose properties do not change during the simulation run. For encapsulation purposes, in O^{2}DES, we also defined the *Scenario* object for each simulation model as a means to contain the collection of static components.

As an example, refer to an M/M/n queue, in which we have two static components, i.e., customer_type and server_type. The customer_type is defined through its *static* inter–arrival time distribution, while the server_type is characterized by the distribution of the service times and the number of servers in the considered problem. Then the *Scenario* object is created and it contains these static components and fully characterizing the system configuration under analysis.

To the knowledge of the authors there is no modeling framework proposing the scenario class. Instead, the system configuration is implied by the definitions of the simulation objects (e.g., resources, processes). Nevertheless, it can be argued that the implicit scenario definition can make it more difficult, or hinder, the possibility to define, and study, multiple scenarios. This is particularly relevant in an optimization setting, where several alternatives might be defined and compared.

4.2 Status & Loads

Differently from the previous case, the *Status* and the *Load* are dynamic components, i.e., they refer to the objects whose properties are updated during the simulation run, and transient objects (i.e., entities that leave the system), respectively. Together, they represent the simulation runtime information. Usually they are associated to one or more static components so that during the runtime, the *Scenario* properties can be easily referred.

Specifically, if the simulated entity has a life cycle in the simulation which is shorter than the simulation run length, it is referred to as a load and its class contains all the runtime properties of the entity, whereas its static properties are encapsulated in a static component class.

The status, instead, refers to a set of properties that describes a snapshot of the simulated system (e.g., waiting queue, server status).

Once the loads and status classes have been defined, we can instantiate a scenario and link them to the scenario. From an OOP perspective, the scenario encapsulates the load and the status instances, together with the common method that manipulates the status when events are triggered (refer to section 4.3). Thus, the modeling of events will focus on the logical dependency among the status changes. This has an effect

on the coding of the simulation model, in that it makes the modeling clean and tight, improving readability and maintainability of the code.

As an example, following the M/M/n queue previously described, although the customer_type is static, the individual customers (i.e., the occurrences) are transient entities as they arrive and leave the system as simulation is running and each of them might have associated a certain statistics (e.g., cycle time, waiting time). A buffer is a status entity in that its level varies as the simulation progresses. Note that the buffer is also a static component, however, as such it is described by its, static, capacity attribute.

The .NET framework provides a rich library for organizing status variable in a well defined data structure. For example, the classes of List, Stack and Queue as in the standard library, can be directly applied to implement different types of waiting queues.

4.3 Events

The event is not a static neither a dynamic component since it only describes the procedures (methods) to update the instances of load and status classes and it has not static neither dynamic properties.

Once scenario, load and status are defined, the event specifies a set of status changes following certain logical statements, and it schedules one or more new events at a future time when necessary. In particular, an event generating function takes as the input one or more occurrences of loads, and status, and it modifies them by calling the status manipulating methods. As mentioned in Section 3, each generated event encapsulates the entire procedure into a delegate object and puts into the FEL after appending a time stamp corresponding to the scheduled execution time.

Each event can be decomposed into several sub-procedures that may happen at the same time, but the execution of which depends on the status condition. In such cases, each sub-procedure is an *atomized event* and it is triggered together with other atomized events for immediate execution. The example in Section 6 will clarify this point.

5 MAIN FUNCTIONALITIES

With respect to the provided definitions, a DES run instance, in order to be initialized, requires only two sets of information: the scenario, i.e., the static components; and, for stochastic simulation, the random seed (through the MathNet.Numerics library). O^2DES exposes several functionalities which were developed to facilitate the integration of the models in an optimization environment and that we explain in the following.

- Generation of multiple simulation models by defining multiple scenario instances. This feature facilitates the integration of recursive stochastic search algorithms such as MO-COMPASS (Li et al. 2015), that samples alternative configurations of a base scenario to be simulated (examples to be shown in Section 6);
- Generation of multiple simulation replications by giving different random seeds as input, while the scenario object remains the same across the replications. This functionality represents a fundamental requirement to integrate techniques such as Optimal Computing Budget Allocation to allocate replications to candidate solutions (i.e., scenario) in a sequential manner (Chen et al. 1997).
- Concurrent simulation. Multiple simulation instances are simultaneously managed as a unique model due to the synchronous central controller. It enables the utilization of Time Dilation (Schruben 1997) as an efficient variance reduction technique.

With O^2DES, both simulator and optimizer program can be compiled in the same executable, therefore all communication are made through memory level and no file operation is required at all. In such a way, the efficiency of the optimization infrastructure tremendously increases.

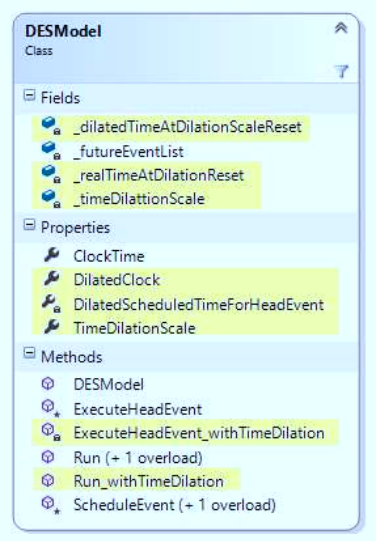

Figure 4: Extended base class for DES model with support of time dilation.

5.1 Optimal Computing Budget Allocation

Although OCBA (Chen et al. 1997) has been proposed for nearly two decades, the technique has not been integrated, to the knowledge of the authors, in any simulation package performing optimization tasks. In fact, the integration of OCBA requires a control of the simulation execution: an initial simulation run is conducted for all candidate solutions and the remaining budget is iteratively assigned to the promising candidates *sequentially allocating* simulation replications as opposed to the traditional user–defined allocation performed in commercial simulation packages. The proposed O^{2}DES could integrate OCBA making it transparent to the user.

5.2 Time Dilation

The purpose of the concurrent simulation with time dilation (Schruben 1997) is similar to OCBA, i.e., to improve the efficiency of the simulation optimization. However, it requires each underlying DES model to expose its future events list and system clock to a centralized controller which works according to a common clock time. The central controller is able to allocate the computational effort by executing more often events from the events list of system configurations showing better performance.

O^{2}DES, as with OCBA, makes the implementation of time dilation particularly easy and transparent to the user. Specifically, several fields and properties were defined at the level of the class of DES model: we store the relevant information on the time series of frequencies in order to perform the updates (Figure 4). Two static functions (*execute head event with time dilation*, and *run with time dilation*) were defined under the base class, taking as input *multiple* DES objects and selecting the different events list to simulate based on their performance.

6 A CASE STUDY

In this section, we illustrate a case study on the aircraft spare part management to demonstrate how the proposed modeling paradigm can be used for a real industrial problem. It is a complex scenario, where many types of entities and their interactions are considered, and multiple operational rules are involved in the formulating the process. Therefore, with O^{2}DES we have an explicit and systematic way to describe the simulation model as following.

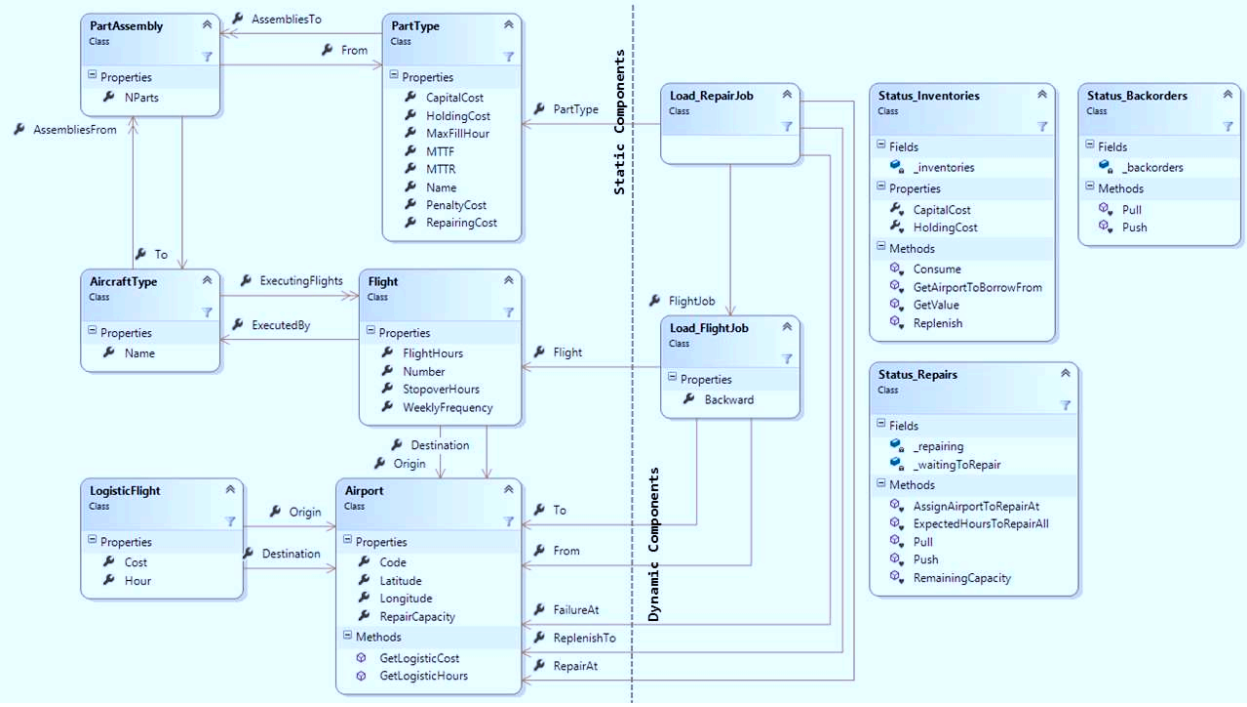

Figure 5: Class diagram of the DES model for aircraft spare part management.

6.1 Simulation

Firstly, we define the *static* components of the DES in the left portion of the class diagram in Figure 5 represents all the static entities and their relationships. Herein, we defined the relationships between **flight schedules**, their executing **aircrafts** and the **spare-part** rotables associated to the aircrafts, the airports serving as both the origins and destinations of flights, as well as the location to store and repair the failures. Since the model allows the spare-parts to be transferred between different **airports**, we also model the **logistic flights** connecting pairs of the airports. Besides, the class diagram clearly shows the attributes of each entity which are concerned with the DES model.
The right portion of Figure 5 represents, instead, the *dynamic* components of the model. As we can observe, there are two types of logical entities with life-cycle (i.e., the loads) in the system, one is called the **flight job**, i.e., an instance of flight trip, and the other is the **repair job**, i.e., the main entity to be processed in the system. In addition, three sets of status variables were modeled, i.e., **inventories**, **backorders**, and **repair jobs** for all types of spare–parts and at all airport locations. Correspondingly, we created three objects to encapsulate each set of status variables and prepare the common updating methods for the event calls.

Figure 6 enumerates all the nine events defined in the model. Two of them on the right portion of the digram are for the **flight jobs**; and the remaining seven events are for the **repair jobs**. The arrows show the triggering relationships among all the events. Note that, among the nine events, the ones colored in green utilize the encapsulated functions of the status variables (Figure 7) in which operational rules (as in light blue) are define and could be replaced. To be more specific, we recognized the following nine main events:

- *Depart*: The a flight job is departing, it triggers an arrival event to be scheduled after its flight hours. And if the flight job is forward direction, it also triggers a returning departure concerning both flight hours and the stop over time, together with the next departure on the forward direction concerning the flight frequency. For simplicity, we assume that the scheduled departure will not be affected by any part failure;

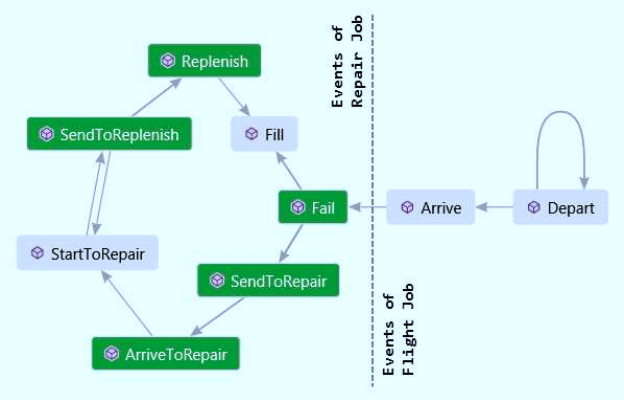

Figure 6: Code map of event generators for aircraft spare part management.

- *Arrive*: When a flight job arrives, according to certain probabilities that are functions of the flight hours and mean time to failure (MTTF) of each part, it triggers failures of parts which are associated to the executing aircraft. Multiple repair jobs could be created and the "fail" event is invoked immediately for each of them;
- *Fail*: Once a part failure is observed, it tries to consume an inventory of corresponding spare part at the local airport for replacement, then the failure is filled instantly. If not successful, it tries to look for surrounding inventories considering certain rules (as in the right-top portion of Figure 7, the method colored in light blue can be implemented by different rules), and find an inventory to be consumed and transferred to the local airport, so that the failure is filled at a future time. In the case of no finding, the repair job is pushed into a backorder locally for future process (right-bottom portion of Figure 7). In any cases, the repair job triggers an event of "sent to repair" immediately after the failure happens;
- *Send to Repair*: The event calls a status encapsulated method to assign an airport to repair the failed part (left portion of Figure 7, the method colored in light blue can be easily replaced). Basically, the method look into the available repairing capacities at all the airports and the expected repairing and transportation time, from which it selects the one costs the least. Then it sends the part and schedule an event of "arrive to repair" for the underlying repair job;
- *Arrive to Repair*: When a repair job arrives at the repairing location, it is pushed into the status object for the repairs (left portion of Figure 7). Then, based on the availability of repairing capacity, either it invokes a "start to repair" or it is joined into a local queue (also encapsulated in the status object) for future process;
- *Start to Repair*: The event triggers nothing more than a finishing event of the underlying repair job called "send to replenish", at a future time according to the mean time to repair (MTTR) of the part;
- *Send to Replenish*: The event pulls the repair job out of the status object. According the logistic time, it schedules a "replenish" event in future. And if there is still repair job in the corresponding queue, the "start to repair" event is invoked for it;
- *Replenish*: Refer to the right portion of Figure 7. The replenished part will first look into the local backorders, and make one unit of existing backorder filled by consuming itself. Otherwise, it is pushed into the local inventory; and
- *Fill*: The repair job is time stamped and archived.

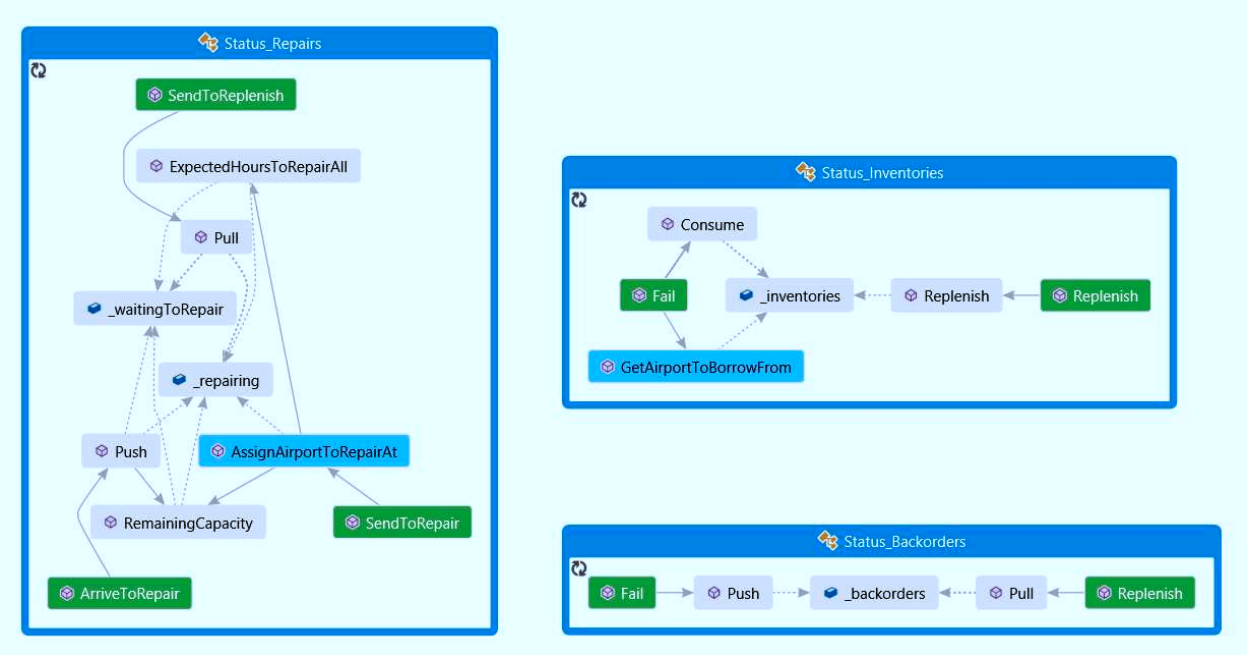

Figure 7: Code map of events with status objects for aircraft spare part management.

6.2 Optimization

The simulation model just presented can be used to efficiently solve the well-known spare parts allocation problem. Specifically, in the spare parts allocation problem, the decision variables are $I_{i,j,t}$, $t = 0$, i.e., the amount of inventory to allocate to the site i, for the part type j, at time 0, i.e., the initial spare parts inventory level for each part type. The objectives are typically two:

- $\max \vartheta_1$, maximize the service level defined as the percentage of repair jobs which are re–filled within a maximum interval specified as an attribute of its underlying part type;
- $\min \vartheta_2$, minimize the total cost defined as the sum of capital costs for purchasing the spare–parts, holding costs, logistic costs, repairing costs and penalty costs for excess of the maximum fill hours;

The optimization technique suggested exploits the functionalities of O^{2}DES. In fact, due to the possibility of easily manipulating the *scenario* with the initial inventory level at all the airports as well as to access the load information in run-time, both a stochastic selection procedure such as OCBA considering several initial inventory configurations to choose the best, as well as a recursive procedure as MO–COMPASS can be used in this scenario.

We firstly try a simple heuristic algorithm to solve the optimization problem. Starting from the solution of zero inventories for each part at all locations, we abstract additional information from the simulator to find out which part type and location cumulates the maximum number of unsatisfactoriness. Then in the next solution, one additional inventory is allocated to the location for the specific part type. Until the observed service level reaches 100%, we stop the procedure and report the Pareto solutions with either higher service level or the lower cost. For all the three pilot run scenarios, the heuristic optimization can be completed within a few minutes.

Then, we apply the MO-COMPASS (Li et al. 2015) as the optimizer. To form the integration, we simply fetch the quantitative values sampled from the optimizer, and cast them into the *Scenario* object which can be taken in by the simulator for evaluation. Then, the quantitative values from the simulation results are fed back to the optimizer. The iteration repeats until satisfying results are obtained.

It is observed that until the optimal performance converges, compared with the heuristic results with the similar service level, the MO-COMPASS is able to further reduce the cost by 20%. However, it takes longer time to complete the optimization.

7 CONCLUSION

This paper proposes a compact and flexible paradigm for building O^{2}DES model with C# programming language. Such an effort results in an open toolkit for teaching DES simulation as well as simulation–based optimization. The underlying framework as well as the three main components of the modeling paradigm are discussed and the main functionalities enhancing the framework for simulation–optimization are reported as well. O^{2}DES is demonstrated by a case study of aircraft spare part management problem.

With the case study, we also illustrated that the modeling paradigm facilitates the integration of the simulator within a simulation optimization architecture, and make it feasible to incorporate advanced search techniques as MO-COMPASS.

Future studies will focus on two areas. One is to apply the paradigm to model further complex DES systems, e.g., health care, warehouse, transportation and other logistic problems. Another direction is to enrich the general framework by integrating with more practical features concerning simulation–optimization. We hope this study inspires a wide range of practical research on simulation modeling, and benefit the industrial practitioners by providing them an efficient tool to analyze and improve productivities.

REFERENCES

Banks, J., and I. John S. Carson. 1986. "Introduction to discrete-event simulation". In *Proceedings of the 18th conference on Winter simulation*, 17–23. 318253: ACM.

Chen, H.-C., C.-H. Chen, L. Dai, and E. Yucesan. 1997. "New development of optimal computing budget allocation for discrete event simulation". In *Proceedings of the 1997 Winter Simulation Conference*, 334 – 41.

Choi, B. K., and D. Kang. 2013. *Modeling and Simulation of Discrete Event Systems*. John Wiley & Sons.

Dahl, O.-J., and K. Nygaard. 1966. "SIMULA: an ALGOL-based simulation language". *Commun. ACM* 9 (9): 671–678.

Fu, M. C. 2002, July. "Optimization for Simulation: Theory vs. Practice". *INFORMS Journal on Computing* 14 (3): 192–215.

Fu, M. C., G. Bayraksan, S. G. Henderson, B. L. Nelson, W. B. Powell, I. O. Ryzhov, and B. Thengvall. 2014. "Simulation optimization: A panel on the state of the art in research and practice". In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, 3696–3706. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hammann, J. E., and N. A. Markovitch. 1995. "Introduction to arena". *Winter Simulation Conference Proceedings*:519 – 523. Arena simulation system;Simulation analysis program;.

Kelton, W. D., R. P. Sadowski, and D. A. Sadowski. 1998. *Simulation with ARENA*, Volume 47. WCB/McGraw-Hill New York.

Li, H., L. H. Lee, E. P. Chew, and P. Lendermann. 2015. "MO-COMPASS: A Fast Convergent Search Algorithm for Multi-Objective Discrete Optimization via Simulation". *IIE Transactions* (just-accepted).

Muller, D. 2011. "Automod - providing simulation solutions for over 25 years". In *Simulation Conference (WSC), Proceedings of the 2011 Winter*, 39–51.

Nance, R. E., and R. G. Sargent. 2002. "Perspectives on the Evolution of Simulation". *Operations Research* 50 (1): 161–172.

Nelson, B. 2013. *Foundations and methods of stochastic simulation: a first course*, Volume 187. Springer Science & Business Media.

Nordgren, W. 2002. "Flexsim simulation environment". In *Simulation Conference, 2002. Proceedings of the Winter*, Volume 1, 250–252 vol.1.

Schriber, T. J., D. T. Brunner, and J. S. Smith. 2014. "Inside Discrete-event Simulation Software: How It Works and Why It Matters". In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, 132–146. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Schruben, L., and E. Yucesan. 1993. "Modeling paradigms for discrete event simulation". *Operations Research Letters* 13 (5): 265–75. 4525298 modeling paradigms discrete event simulation graph theory simulation graphs model-based problem-solving environment.

Schruben, L. W. 1997. "Simulation optimization using simultaneous replications and event time dilation". In *Proceedings of the 29th conference on Winter simulation*, 177–180: IEEE Computer Society.

Sharda, B., S. J. Bury, J. Banks, and J. S. Carson. 2011. "Best practices for effective application of discrete event simulation in the process industries Introduction to Discrete-Event Simulation.". In *Proceedings of the 2011 Winter Simulation Conference (WSC 2011)*, 2315 – 24.

AUTHOR BIOGRAPHIES

HAOBIN LI is a scientist in Institute of High Performance Computing, under Agency for Science, Technology and Research (A*STAR) of Singapore. He received his B.Eng. degree (1st Class Honors) in 2009 from the Department of Industrial and Systems Engineering at National University of Singapore, with minor in computer science; and Ph.D. degree from the same department in 2014. He has research interests in operation research, simulation optimization and designing high performance optimization tools which are ready for practical industrial use. His email address is lihb@ihpc.a-star.edu.sg.

YINCHAO ZHU is a research engineer in National University of Singapore. He received his B.Eng. degree (1st Class Honors) and B.Sci in 2012 from Engineering Science Program and Department of Mathematics at National University of Singapore. He is currently pursuing his PhD degree in the Department of Industrial and Systems Engineering. He has research interests in operation research and simulation optimization. His email address is zy@nus.edu.sg.

GIULIA PEDRIELLI is Research Fellow for the Centre for Maritime Studies at the National University of Singapore. She received her Ph.D. degree from the Mechanical Engineering Department in Politecnico di Milano in 2013 (with Honors). Her research focuses on stochastic simulation based-optimization in both single and multiple–objectives framework. Her email address is cmsgp@nus.edu.sg.

NUGROHO A. PUJOWIDIANTO is an R&D Writing System Engineer in the Business Printing Division at Hewlett-Packard Singapore. He received his B.Eng. (Mechanical Engineering) degree from Nanyang Technological University in 2006 and his Ph.D. degree from the Department of Industrial and Systems Engineering, National University of Singapore in 2013. His research interests include simulation optimization and its application in health care. His email address is nugroho@hp.com.

YIXIN CHEN is a final year undergraduate student in the Department of Industrial and Systems Engineering at National University of Singapore. His email address is chen_yixin29@u.nus.edu.