Evaluation of Scheduling Strategies by Modeling and Simulation

Eleonu Henry Chika
African University of Science and Technology
Computer Science Department
Km 10, Airport Road, Galadimawa, Abuja,
F.C.T., Nigeria
henryeleonu@yahoo.com

Mamadou Kaba Traoré
Université Blaise Pascal, Clermont-Ferrand
LIMOS
Campus des Cézeaux, B. P. 125
63173 Aubière, France
traore@isima.fr

Keywords: Evaluation of Scheduling Algorithms, Model-View-Controller, Design Patterns, Hoist Scheduling Problem, Software Engineering

ABSTRACT

The hoist scheduling problem is a critical issue in the design and control of many manufacturing processes. When the hoist number and tank numbers are very large, finding an optimal schedule is very hard. As a result of this, a lot of scheduling algorithms have been developed, and thus created a need to evaluate these algorithms. This calls for a cheap and efficient way of evaluating different hoist scheduling algorithms. To address this issue, we are proposing a generic simulator which will be a visual tool that will be developed with Java technology. The result of this work will help to reduce cost and also help to guarantee product quality in production lines.

1. INTRODUCTION

The hoist scheduling problem (HSP) is encountered in many production lines in many industries. This problem has been proven to be NP complete problem. Consequently many heuristic algorithms have been proposed by many researchers to solve this problem. Problem arises on the scheduling algorithm to adopt in an automated hoist system. As a result of the numerous algorithms, there is need to have a visual tool to explore, evaluate and compare these algorithms

We are proposing a visual tool (simulator) that can be use to create visual simulation that can evaluate different hoist scheduling algorithms. I am proposing that moves computed from hoist scheduling algorithm should be used as input to this simulator, so as to make the evaluation of the algorithm easy and less expensive. The moves can be in the form of a text file. We are also adopting a Model-View-Controller (MVC) architectural design pattern for this simulator.

The purpose of this work is to build a simulator which will have a graphical user interface that can evaluate hoist scheduling algorithms. The simulator will simulate hoist moves of different hoist scheduling algorithms, and also evaluate these algorithms. A move is represented by the tank numbers of the source and target (destination) tanks and the pick time (the time the carrier will be picked from the source). It also report violations of imposed constraints and also compares the algorithms to find which gives optimal scheduling. The simulator will be implemented with Java.

2. EXISTING TOOLS

No Visual tool has been identified to carry out the evaluations of hoist scheduling algorithms. To the best of our knowledge, this will be the first of its kind to be used.

3. HOIST SCHEDULING PROBLEM

3.1 Industrial Importance

The Hoist Scheduling Problem (HSP) deals with the scheduling of hoist that move product between tanks in electroplating facilities that perform chemical surface treatments. Electroplating lines are totally automated manufacturing systems that are used to cover parts with a coat of metal. They are encountered in many industries: mechanics, jewelry, electrical appliances, and printed circuit boards. Hoist system is also use in electroplating processes for the production of floppy disks, computer hard drives, communication network connectors and switches, Aerospace parts (airplane parts) etc. In particular, they allow the protection of parts from corrosion or give them some aesthetic properties. The hoist system is used in chemical processing, food processing, metal and pharmaceutical production (Manier and Bloch., 2003). Integrated circuit (IC) manufacturing is a new generation industries that try to increase their profits by improving the process technology. To do this, the hoist system is employed for material handling.

There are time windows constraints (a minimum and a maximum values) for the time spend at each tank. This time window is a quality constraint which must be adhered to ensure the quality of the products in the production line. In such production lines, any violation of the time window constraint will result in defective products, because standard must be adhered to. This translates to losses for the company involved. As no inventories are allowed, this soak time tolerance is the only source of flexibility. Tanks and hoists can only

process one job at a time. We also assume that one hoist must do all moves.

The control of the hoist's movements with respect to those constraints is known as the Hoist Scheduling Problem (HSP) (Lamothe et al, 1994).

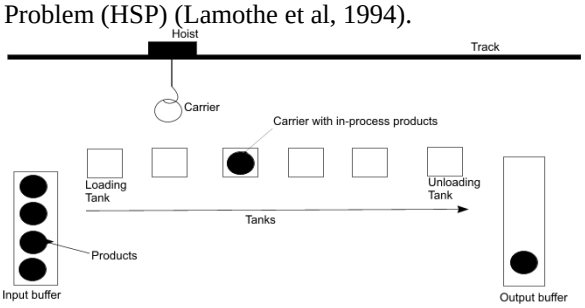

Figure 1: An example of a hoist system

The *scope* of the study is in the area of single hoist scheduling with multiple tanks. See Figure 1. The objective is to minimize the total production time and also have fewer products that are defective. Efficient scheduling of such hoists can improve production throughput dramatically. In many production lines, violations of the constraints can lead to defective job. Thus the need for efficient hoist scheduling algorithms becomes very important. This has lead to many hoist scheduling algorithms being proposed.

As the number of tanks increase and the number of carriers that can be in the system at any point in time also increases it becomes very difficult to schedule the hoist to do all the moves, at the same time obeying all the time window constraints and other constraints that may be imposed on the system. Generally, the problem to determine the scheduling for operations done by a hoist with the objective to optimize the productivity appears as a NP-complete problem (Lei and Wang, 1989). The generic hoist scheduling problem is NP-hard and arises from automated manufacturing lines (Riera and Yorke-Smith, 2002).

3.2 HSP Solutions

Solutions to HSP are mainly heuristic algorithm. The first solutions to the hoist scheduling problem used mathematical programming (Phillips and Unger, 1976). Another technique that used local search and constraint logic programming (CLP) were applied (Baptiste et al., 1994; Lam, 1997). More recently, a hybrid technique that combines MIP and CLP has been developed (Rodoˇek and Wallace, 1998) and many other algorithms that we have not mentioned. So a problem arises on the hoist scheduling algorithm to adopt in an automated hoist system, and hence a need for an evaluation tool to evaluate the numerous hoist scheduling algorithms that may be an option for an automated hoist system.

3.3 Need for Evaluation Tool

The material-handling operations (i.e., the operations to move jobs between stages or tanks) are performed by a computer-programmed robot. The programs that run on these computers are based on some of these hoist scheduling algorithms. These numerous algorithms needs to be evaluated, to enable a company choose the best option that will help it to maximize profit. This calls for the need for a visual interactive tool that will simulate the hoist system and evaluate different moves that result from different algorithms.

Our aim is to model, design and develop a simulator (visual tool) that can be use to simulate some of these hoist system classes and also evaluate some of the heuristic algorithms that have been proposed as solutions to the hoist scheduling problems of these hoist systems.

4. MODEL-VIEW-CONTROLLER

4.1 Design Patterns

A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate. (Buschmann et al., 1996). Three categories of patterns defined by (Buschmann et al.)

• Architectural patterns
• Design patterns
• Idioms

Architectural Patterns: Architectural Pattern is a high-level structure for software systems that contains a set of predefined sub-systems. The responsibilities of each sub-system are defined and detail the relationships between sub-systems. The Model View-Controller pattern falls under this category.

Design Patterns: Design Pattern is a Mid-level construct which is Implementation independent and Designed for ‘micro-architectures’ – somewhere between sub-system and individual components. It is a documented best practice or core of a solution that has been applied successfully in multiple environments to solve a problem that recurs in a specific set of situations. Some authors don’t make this distinction. Most times, they are all referred to as design patterns. In this paper, when we say, design pattern, we shall assume all the categories.

Some of the important qualities of design patterns are that it Help improve the quality of the software in terms of the software being reusable, maintainable, extensible, etc and it Patterns provide a way to do “good” design and are used to help design frameworks. (Kuchana, 2004)

Most design patterns also make software more modifiable. The reason for this is that they are time-tested solutions. Therefore, they have evolved into structures

that can handle change more readily than what often first comes to mind as a solution.

The design patterns in this work are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context, as defined by Gamma and others (Gamma et al, 1995). In this work, we are adopting a Model-View-Controller (MVC) design pattern.

4.2 MVC Pattern

One of the frequently cited frameworks was the Model-View-Controller framework for Smalltalk (Krasner and Pope, 1988), which divided the user interface problem into three parts. The parts were referred to as a data model which contains the application object or computational parts of the program, the view, which presented the user interface, and the controller, which interacted between the user and the view. Before MVC, user interface designs tended to lump these objects together. MVC decouples them to increase flexibility and reuse.

View: it renders the contents of the model; it is responsible for mapping graphics onto a device. A view typically has a one to one correspondence with a display surface and knows how to render to it. A view attaches to a model and renders its contents to the display surface.

Controller: Serves as a bridge between the View and the Model. A controller offers facilities to change the state of the model. The controller interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate. It is the piece that manages user interaction with the model. It provides the mechanism by which changes are made to the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

Model: The model is the piece that represents the state. It manages the state and conducts all transformations on that state. The model has no specific knowledge of either its controllers or its views. A model can have more than one view. The model represents real world objects.

4.3 Why MVC

It allows for modular separation of function. Software developed using this design pattern is easier to maintain. It can allow change of Interface from Swing to three dimensions OpenGL or to another graphic API. The model can also be changed from UML model to DEVS (atomic and coupled) model without changing the interface or controller. The MVC pattern can open up new levels of robustness, code reuse, and organization.

4.4 Problem with Swing Application

In Applying MVC to swing applications, splitting the controller from the view didn't work well in practical terms because the view and controller parts of a component required a tight coupling (for example, it was very difficult to write a generic controller that didn't know specifics about the view). So Sun Microsystems collapsed these two entities into a single UI (user-interface) object (Fowler, 2010). In this Simulator, we have been able to provide logically separation at the application level amongst the model view and controller.

5. DESIGN AND IMPLEMENTATION OF SIMULATOR

5.1 Problem Parameter and Description

The hoist system to be modeled has the following characteristics:

- ✓ it has a single hoist
- ✓ it has multi -tank(multi -stage)
- ✓ No-wait or the no-wait-in-process constraints, this means that the products must be taken to the next tank immediately it is removed from a tank, without waiting.
- ✓ multi-carrier(multi-barrel)
- ✓ multi-product, different products can be processed in the system
- ✓ all the product in a carrier are identical and each product type require the same processing sequence
- ✓ processing time in a tank is within upper and lower bound(time window constraint)
- ✓ the tanks are in a single straight line
- ✓ Multifunction tanks.
- ✓ There is no storage for the carriers near the facility or at the load/unload stations. Then empty carriers must remain on the line and be moved from tank to tank so as to prevent them from interfering with loaded carriers.

The hoist can carry only one carrier at a time. Each tank can only take one carrier at a time, and each carrier can take many products to be treated. The first tank serves as the loading tank where products are loaded to the carrier. Empty carriers have to be taken to the first tank to be loaded. The last tank is the unloading tank. Full carriers must be taken to the last tank as the last stage of treatment of the products, for the carriers to be unloaded. It has both input and output buffer. The robot will travel to the specified tank/location, wait if necessary, lift the job, travel to the next tank on the route and then release the job. After that, the hoist is ready for the next scheduled material handling operation. The hoist should have enough time to travel between the starts of successive operations, which are called the *traveling time* constraints.

5.2 MVC STRATEGY

Figure 2 shows the MVC design for this simulator.

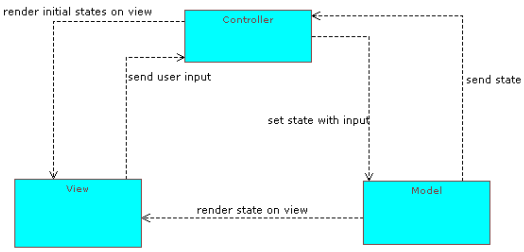

Figure 2 MVC of Simulator

The MVC design pattern strategy has enabled us to separate the problem into three separate problems, the view, the model and the controller. This has enabled us to focus on one subsystem at a time, there by avoiding the confusions involved in handling the whole system as one. Much effort on MVC is directed towards web applications, and not much is done on swing standalone application. The design of the simulator focused on how to use MVC in swing applications, and devising techniques on how to achieve this. Our aim is to make a clear separation between Model, view and controller.

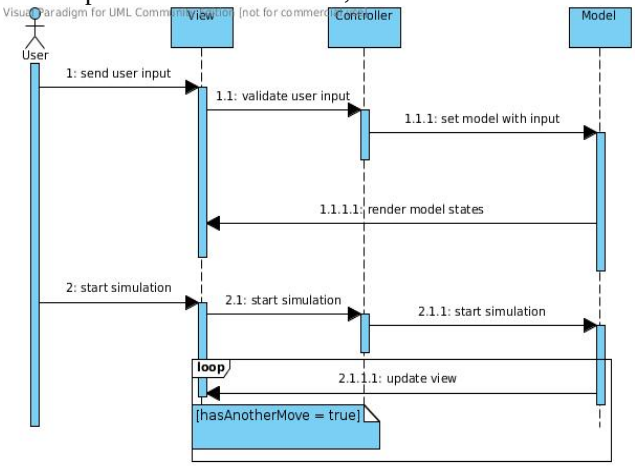

Figure 3 high level sequence diagram of simulator

Figure 3, shows a high level sequence diagram of the hoist system simulator. The user can only interact with the system from the view which is the user interface as we have said before. The view takes user inputs and then passes it on to the controller which validates the inputs and then passes it on to the model, thus changing the state of the model. The model responds by passing its states to the view to be rendered. When simulation starts, the model continuously changes its states and continuously updates the view.

5.2.1 USER INTERFACE

The main window is the interface which has the canvas on which the simulation takes place. On the canvas, the track, hoist, tanks and carriers are displayed. It has menus from where other child windows could be launched. A screen shot of this window is shown in figure 4, with two carriers with one of the carriers with a product being processed. The panel that appears under each tank shows for any bath taking place in that tank, the minimum bath time, the maximum bath time and also the soak time. If the soak time is below the minimum time, the colour is orange, if it is within the time window, the colour is green, and if it is above maximum time, the colour is red. If the product is being soaked in the wrong tank, the colour becomes black. This visually gives the user an instant report on the situation of the bath in the tank

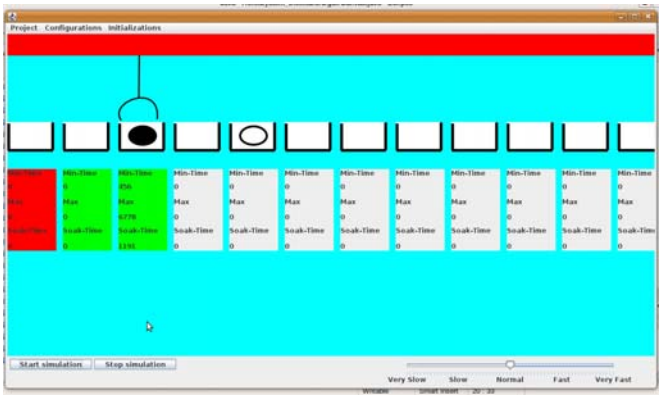

Figure 4: the main window of the simulator

5.3 IMPLEMENTATION

The implementation is Java based. The simulator is made up of three packages, the gui, which serves as the view and the model, which is the model and the controller which is the controller. Two libraries (Application programming interface (API's)) that were use amongst others are swing and Business Intelligence and Reporting Tools (BIRT).

The Swing classes (part of the Java™ Foundation Classes software) implement a set of components for building graphical user interfaces (GUIs) and adding rich graphics functionality and interactivity to Java applications. The GUI components used for developing the simulator are part of this swing API. Graphical components ranging from buttons, tables, text fields etc. are all from swing.

The Business Intelligence and Reporting Tools (BIRT) API is an open source API that provides reporting and business intelligence capabilities for rich client and web applications, especially those based on Java and Java EE. BIRT was used to implement the time window violation report and the treatment error report.

The eclipse and netbeans Integrated Development Environment (IDE) where use as the development environments. The GUI of some of the child windows were developed under netbeans, while the other parts of the software were developed on the eclipse IDE.

6. TESTING AND RESULTS

6.1 TESTING AND RESULTS

The total number of good baths within the time windows is a parameter that will be used to measure the performance of an algorithm. Three different sets of moves will be use to test the simulator with two sets of treatments. A set of treatments shows the sequence of tanks that a product will be processed, and also the time window constraints. A system with 12 tanks and 2 carriers will be use for this test. The simulation starts with one carrier in the first tank while the second carrier in the last tank. Table 1 and Table 2 show the treatments that will be used for the tests.

Table 1: Treatment 1 (T1)

Tank Number	Minimum Time(milliseconds)	Maximum Time(milliseconds)
0	655	5566
1	456	7677
2	456	7778
5	333	8333
4	333	8333
5	333	9333
6	1234	23444
7	123	9456
1	345	8785
3	567	7894
10	123	8376
11	453	8679

Table 2: Treatment 2 (T2)

Tank Number	Minimum Time(milliseconds)	Maximum Time(milliseconds)
0	454	7345
1	456	8754
2	234	6778
3	222	8456
4	337	8333
5	334	7456
6	123	9342
7	445	9567
8	341	7229
9	567	9667
10	556	10055
11	451	7445

6.1.1 First Test

The moves file to be use is shown in the table below.

Source Tank	Target Tank	Pick Time(milliseconds)
0	1	100
11	0	200
1	2	500
0	1	1500
2	3	2500
1	2	3500
3	4	4500
2	3	5700
4	5	6700
3	4	7666
5	6	8333
4	5	9333
6	7	14000
5	6	15000
7	1	16000
6	7	17000
1	4	18000
7	8	33555
4	10	38000
8	9	40999
10	11	41000
9	10	51344
11	2	52233
10	11	53000

After the simulation run, the following reports were generated

Good Bath

Tank Number	Product Name	Minimum Time(milliseconds)	Maximum Time(milliseconds)	Soak Time
0	T1-1	655	5566	1346
1	T1-1	456	7677	7466
1	T2-1	456	8754	8013
4	T1-1	333	8333	7994
3	T2-1	222	8456	7833
5	T1-1	333	9333	7780
4	T2-1	337	8333	7813
6	T1-1	1234	23444	7784
7	T1-1	123	9456	7764
6	T2-1	123	9342	7726
1	T1-1	345	8785	7671
7	T2-1	445	9567	7746
10	T1-1	123	8376	7683
9	T2-1	567	9667	8223
10	T2-1	556	10055	7543

Number of Good Baths: 15

Time Window Violation

Tank Number	Product Name	Minimum Time(milliseconds)	Maximum Time(milliseconds)	Soak Time	Description
0	T2-1	454	7345	7795	soak time is above
2	T1-1	456	7778	7878	soak time is above
2	T2-1	234	6778	7789	soak time is above
5	T2-1	334	7456	7770	soak time is above
8	T2-1	341	7229	7606	soak time is above

Number of Violations: 5

TREATMENT ERROR

Current Index	Product Name	Expected Tank Number	Processing Tank Number
3	T1-1	5	3
9	T1-1	3	4

Treatment Error Count: 2

6.1.2 Second Test

The moves file to be use for this test is shown in the table below.

Source Tank	Target Tank	Pick Time
0	1	100
11	0	200
1	2	500
0	1	1500
2	5	1900
1	2	2500
5	4	4500
2	3	5700
4	5	6700
3	4	7666
5	6	8333
4	5	8433
6	7	12000
5	6	13000
7	1	16000
6	7	17000
1	3	18000
7	8	33555
3	10	38000
8	9	40000
10	11	41000
9	10	51344
11	2	52233
10	11	53000

The reports generated for this test are shown below.

Good Bath

Tank Number	Product Name	Minimum Time(milliseconds)	Maximum Time(milliseconds)	Soak Time
0	T1-1	655	5566	1407
2	T1-1	456	7778	7591
1	T2-1	456	8754	7599
5	T1-1	333	8333	7761
4	T1-1	333	8333	8096
3	T2-1	222	8456	7886
5	T1-1	333	9333	7751
4	T2-1	337	8333	7798
6	T1-1	1234	23444	7897
7	T1-1	123	9456	7794
6	T2-1	123	9342	7906
1	T1-1	345	8785	7807
7	T2-1	445	9567	7671
3	T1-1	567	7894	7863
10	T1-1	123	8376	7742
9	T2-1	567	9667	8223
10	T2-1	556	10055	7632

Number of Good Baths: 17

Time Window Violation

Tank Number	Product Name	Minimum Time(milliseconds)	Maximum Time(milliseconds)	Soak Time	Description
1	T1-1	456	7677	7953	soak time is above
0	T2-1	454	7345	8014	soak time is above
2	T2-1	234	6778	8032	soak time is above
5	T2-1	334	7456	7649	soak time is above
8	T2-1	341	7229	7933	soak time is above

Number of Violations: 5

TREATMENT ERROR

Current Index	Product Name	Expected Tank Number	Processing Tank Number

Treatment Error Count 0

6.1.3 Third Test

The moves file to be use for this test is shown in the table below.

Source Tank	Target Tank	Pick Time
0	1	100
11	0	200
1	2	300
0	1	400
2	5	500
1	2	900
5	4	1500
2	3	2700
4	5	3700
3	4	4666
5	6	5333
4	5	5433
6	7	7000
5	6	8600
7	1	9000
6	7	10000
1	3	11000
7	8	12555
3	10	13000
8	9	14000
10	11	20000
9	10	21344

11	2	22233
10	11	31000

The reports generated for this test is shown below

Good Bath

Tank Number	Product Name	Minimum Time(milliseconds)	Maximum Time(milliseconds)	Soak Time
0	T1-1	655	5566	1889
1	T1-1	456	7677	7660
2	T1-1	456	7778	7633
1	T2-1	456	8754	7881
5	T1-1	333	8333	7726
4	T1-1	333	8333	7676
3	T2-1	222	8456	7728
5	T1-1	333	9333	7910
4	T2-1	337	8333	7845
6	T1-1	1234	23444	7897
7	T1-1	123	9456	7813
6	T2-1	123	9342	7818
1	T1-1	345	8785	7796
7	T2-1	445	9567	7994
3	T1-1	567	7894	7843
10	T1-1	123	8376	7665
9	T2-1	567	9667	8391
10	T2-1	556	10055	7761

Number of Good Baths: 18

Time Window Violation

Tank Number	Product Name	Minimum Time(milliseconds)	Maximum Time(milliseconds)	Soak Time	Description
0	T2-1	454	7345	7713	soak time is above
2	T2-1	234	6778	7653	soak time is above
5	T2-1	334	7456	7930	soak time is above
8	T2-1	341	7229	7774	soak time is above

Number of Violations: 4

TREATMENT ERROR

Current Index	Product Name	Expected Tank Number	Processing Tank Number

Treatment Error Count 0

6.2 EVALUATION AND FINDINGS

6.2.1 Evaluation

The total number of good baths within time windows is a parameter that will be used to measure the performance of an algorithm. From the three test performed in section 5.1, we can see that the first test had 2 treatment error, which shows that two baths were done in the wrong tank. Treatment error is a very critical error which should be avoided. This is an indicator that the moves are not good enough and that the algorithms that produced the moves does not give a good scheduling.

The second test does not have any treatment error and the total number of good baths is 17, while the total number of time window violations is 5. This moves produced a better scheduling than the moves from the first test.

The third test did not produce any treatment error, and the total number of good baths is 18 and a total number of time window violations of 4. Since the total number of good bath are higher for the third test when compared to the second test, we can say that the algorithm for the last test is the best algorithm for the hoist system parameters we have adopted for the test.

6.2.2 Findings

We have been able to show that given a set of moves based on some hoist scheduling algorithm, it is possible to use these moves as input to a visual simulator that can simulate the hoist system operations and thus enable the evaluation of the algorithm.

We were able to identify finite states that the hoist and tank in a hoist system can be in any point in time. These finite states where derived by grouping the infinite states into finite states that could be used to model and then implement the simulator. These states can be adopted by those who want to model the hoist system. These states can also be adopted while modeling the hoist system using the Discrete Event systems specification **(**DEVS) formalism or also while modeling with DEVS graphical notations like DEVS Driven Modeling Language (DDML) (Traoré, 2009).

We were able to device a programming technique that enabled the separation of the controller from the view at the application level by initializing the reference of the GUI components in the controller class. Making a separation between the controller and the view has always been a very difficult task in swing application, but this technique has proved to be very effective. This will make it very easy to modify the view, using swing or other APIs. This technique will create a new insight in software engineering and most especially in the application of MVC pattern in swing applications.

Due to MVC design pattern adopted for building the simulator, it will be possible to change the view and use a view that is based on a 3D interface based on the OpenGL graphic API. The model can also be changed by adopting a framework such as simStudio or DEVSJAVA, which are java implementations of DEVS formalism. This will be possible since the hoist system is a Discrete Event System.

7. CONCLUSIONS AND FURTHER RESEARCH

The result of our work will interest those who are seeking to develop and test algorithms for the HSP and also manufacturing outfits that make use of the hoist systems.

The MVC pattern adopted has proved to be a very good design pattern for the development of the hoist scheduling simulator. This simulator can be used as building blocks for more sophisticated hoist system simulators in the future. Clearly, this work will be a valuable asset to many people who are seeking to evaluate their HSP algorithms.

Further work can be done on this simulator with less work, because of the MVC design pattern adopted for this simulator, since the view, model and controller are neatly separated. Work can be done on only the view to have a GUI that has three dimensional (3D) capabilities, based on OpenGL API. Work can also be done on the model to adopt a Discrete Event Specification (DEVS) modeling. A Java based API for DEVS such as SimStudio can be use for this implementation. Further work could also be done to develop a version of this simulator that can run on the web.

7. REFERENCES

(Lamothe et al, 1994) Lamothe, J., Correge, M., Delmas, J. hoist scheduling problem in a real time context, 11th International Conference on Analysis and Optimization of Systems Discrete Event Systems

(Lei and Wang, 1989) Lei, L. and Wang, T. J. A proof: the cyclic hoist scheduling problem is NP-complete, Working paper #89-0016, Rutgers University.

(Riera and Yorke-Smith, 2002) Riera, D. and Yorke-Smith, N. *An Improved Hybrid Model for the Generic Hoist Scheduling Problem*. Annals of Operations Research 115, 173-191, September 2002.

(Phillips and Unger, 1976) Phillips, L.W., Unger, P. S. Mathematical Programming Solution of a Hoist Scheduling Program, AIIE Transactions, 1976, 8, n. 2, 219-225.

(Baptiste et al., 1994) Baptiste, P., Legeard, B., Manier, M. A. A Scheduling Problem Optimisation Solved with Constraint Logic Programming, In: Proc. of the Intl. Conf. on Parallel Architectures and Compilation Techniques

(Lam, 1997) Lam, K. A Heuristic Method for Multiple Hoist Scheduling Problems by using Simulated Annealing and Local Search'. Working Paper (1997)

(Rodoˇek and Wallace, 1998) Rodoˇek, R. and Wallace, M. A Generic Model and Hybrid Algorithm for Hoist Scheduling Problems , In Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520

(Maria, 1997) Anu Maria, Introduction to Modeling and Simulation, *Proceedings of the 1997 Winter Simulation Conference*

(Fishman, 2001) Fishman, G. S., Discrete-Event Simulation modelling' programming and analysis, Springer Series in Operations Research

(Wainer, 2009) Wainer, G. A., Discrete-Event Modeling and Simulation, a practitioner's approach. Taylor & Francis Group.

(Gamma et al, 1995) Gamma, E., Helm, R., Johnson, R. and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, 1995.

(Krasner and Pope, 1988) Krasner, G.E. and Pope, S.T. (1988). A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, 1(3):26-49.

(Traoré, 2009)Traoré, M. K., A Graphical Notation for DEVS, SpringSim '09 Proceedings of the 2009 Spring Simulation Multiconference

(Buschmann et al., 1996) F. Buschmann, R. Meunier, H. Rohnert, P.Sommerlad, M. Stal, Pattern-Oriented Software Architecture A System of Patterns, John Wiley and Sons Ltd, Chichester, UK, 1996 ISBN 0-471-95869-7

(Kuchana, 2004) Kuchana, P., Software architecture design patterns in Java, Auerbach Publications.

(Manier and Bloch., 2003) M. Manier and C Bloch, A Classification for Hoist Scheduling Problems, The International Journal of Flexible Manufacturing Systems, 15, 37–55, 2003

(Zeigler et al, 2000)Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. Theory of modeling and simulation, 2nd. ed. New York: Academic Press.

(Fowler, 2010), Amy Fowler, A Swing Architecture Overview, an Article in Sun Developer Network (SDN), http://java.sun.com/products/jfc/tsc/articles/architecture/