Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

EVENT GRAPH MODELING OF A HETEROGENEOUS JOB SHOP WITH INLINE CELLS

Donghun Kang
Hyeonsik Kim
Byoung K. Choi

Department of Industrial and Systems Engineering
KAIST
291 Daehak-ro, Yuseong-gu
Daejeon, 305-701, REPUBLIC OF KOREA

Byung H. Kim

VMS Solutions Co., Ltd.
Hanshin S-MECA #611
1359 Gwanpyeong-dong, Yuseong-gu
Daejeon, 305-509, REPUBLIC OF KOREA

ABSTRACT

In a flat panel display (FPD) production line, unlike a table-type machine that processes one glass at a time, an *inline cell* works simultaneously on several glasses unloaded from different cassettes in a serial manner and is divided into two types (*uni-inline cell* and *bi-inline cell*) according to the job loading and unloading behavior. In order to build a production simulator for this type of FPD production line, an object-oriented event graph modeling approach is proposed where the FPD production line is simplified into a job shop consisting of two types of inline cells, and the job shop is represented as an object-oriented event graph model. This type of job shop is referred to as a *heterogeneous job shop*. The resulting model is realized in a production simulator using an object-oriented event graph simulator and is illustrated with the experimental results from the production simulator.

1 INTRODUCTION

In flat panel display (FPD) production lines, the processing unit of the jobs is a large piece of glass known as the mother glass, which is later cut into smaller pieces to make the FPD product (Jang 2006). As a result of the fragility and high quality requirements of these glasses, FPD production lines have adopted fully automated material handling systems (AMHSs), such as inline stockers and conveyors, that connect the processing machines. In this system, the glasses are carried inside a container called a *cassette*, which means that the transportation unit and processing unit of the jobs are not the same. Therefore, in the FPD production line, there are ports in front of each processing machine where the glasses are loaded from the cassette into the processing machine or unloaded from the processing machine into the cassette.

In the FPD production line, unlike a table-type machine that processes one glass at a time, a typical processing machine works simultaneously on several glasses unloaded from different cassettes in either serial or parallel manners. Thus, the processing machines included in the FPD production line can be further divided according to the job loading/unloading and processing behaviors. In order to allow this heterogeneity in the behavior of a processing machine in an FPD production line, the authors proposed the concept of homogeneous and heterogeneous job shops in the previous study (Song, Choi, and Park 2012). A *homogeneous job shop* consists of each job that has its own operation sequences and a single type of processing machine that performs certain operations on the job, whereas a *heterogeneous job shop* consists of two or more types of processing machines that have different job loading/unloading or processing behaviors. The classical job shop system solely consisting of table-type machines is a typical homogeneous job shop system. However, in order to simulate an FPD production line, a heterogeneous job shop system consisting of various types of machines in the line is required.

Figure 1 depicts an example layout of an FPD production line of a thin-film transistor (TFT) stage. As seen in Figure 1, the production line consists of inline cells connected by the AMHS of inline stockers and conveyors. An *inline cell* is a typical processing machine that is commonly found in FPD production lines and consists of an inner conveyor that carries the glasses through the equipment and multiple subordinate processing machines along the conveyor belt (Song, Gu, and Choi 2010). According to the loading and unloading behaviors, the inline cells are classified into *uni-inline cells* and *bi-inline cells*. The loading and unloading of a cassette in a uni-inline cell occurs at the same input/output (I/O) port, whereas the bi-inline cell has two types of ports: *in-ports* for loading the glasses from a cassette and *out-ports* for unloading the finished glasses into a cassette. The in-ports and out-ports of a bi-inline cell are often connected to different inline stockers. If the material handling process can be neglected, the Fab in Figure 1 can be regarded as a typical heterogeneous job shop consisting of two types of inline cells.

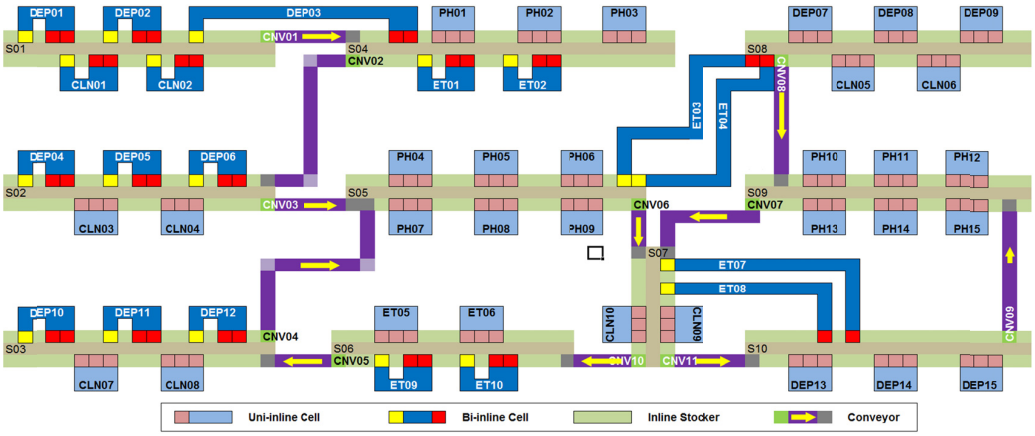

Figure 1: An example layout of a FPD TFT-stage Fab.

Due to their extensive modeling power, job shop models have been widely accepted to represent various manufacturing systems including serial assembly lines (Lee, Cheng, and Lin 1993) where every job has the same operation sequence, special job shops, which are often referred to as *flow shops* (Garey 1976), and semiconductor lines where several types of jobs with different operation sequences are processed (Balas, Simonetti, and Vazacopoulos 2008). The production system of an FPD line exhibits some similarities to that of a semiconductor line because each job has a series of processing steps and a processing machine can perform more than one processing step. Therefore, a job shop model can also be used when analyzing the problems related to FPD production lines.

Vinod and Sridharan (2009) used a simple discrete event simulation model in their study in order to propose a meta-modeling approach to finding the optimal dispatching rules for scheduling stochastic dynamic job shops. Yin and Chen (2009) also proposed a discrete-event simulation framework for real-time online job shop scheduling, which naturally has dynamic and stochastic characteristics. Mahdavi, Shirazi, and Solimanpu (2010) considered a more complex job shop that has flexibility and they built a real-time decision support system. In addition, several other studies that have considered real-world job shop cases have adopted discrete-event simulation models to represent their target systems (Parthanadee and Buddhakulsomsiri 2010, Legato and Mazza 2001). The common feature of the job shop models used in the abovementioned studies is that the machines included in the models are assumed to process one job at a time; that is, they are simple table-type machines. However, it is difficult to accept this assumption when modeling an FPD production line due to certain features of the line, as explained above.

In this paper, a heterogeneous job shop model of an FPD production line in the form of an object-oriented event graph is proposed. An event graph is a formal model, which is quite simple yet extremely

powerful and natural to represent a discrete-event system (Buss and Sanchez 2002). In event graphs, the influence of events on the state variables is represented using vertices and the relationships between events are represented as directed edges between the vertices (Schruben 1983). One of the enrichments in the event graph is a parameterization of event vertices in which similar events are represented using a single vertex with different parameter values (Schruben and Schruben 2006). The parameterized event graph model is one of the most efficient modeling formalisms that is used to represent the behaviors of the processing machine in an FPD production line (Song, Gu, and Choi 2010; Song, Choi, and Park 2012).

However, in order to develop a production simulator for an FPD production line that consists of different types of processing machines, the parameterized event graph lacks the ability to model the entire FPD production line at a manageable level. As a heterogeneous job shop involves more various types of processing machines or even the AMHS equipment (e.g. inline stocker and conveyor), it becomes more complex and more difficult to understand and manage the model. In this paper, therefore, an object-oriented event graph modeling approach to developing a production simulator is presented for the FPD production line of a heterogeneous job shop.

Buss and Sanchez (2002) proposed an object-oriented modeling approach to the event graph, which enables small models to be encapsulated in reusable modules that are linked together using a listener pattern. In this paper, the encapsulation of events are adopted in order to model each type of processing machine and a simplified linking mechanism is introduced.

The remainder of this paper is organized as follows. Section 2 presents the encapsulated event graph models of homogeneous job shops for each type of inline cell; these are merged into a single encapsulated event graph model of a heterogeneous job shop in order to represent the entire FPD production line of Figure 1 in Section 3. Section 4 presents the simulation execution of the encapsulated event graph model, and Section 5 presents the experimental results drawn from a production simulator prototype that implements the proposed model. Conclusions and discussions are given in the final section.

2 EVENT GRAPH MODELING OF INLINE CELLS

In this section, the object-oriented event graph models of two homogeneous job shops of inline cells, which are sourced from the parameterized event graph models presented in previous work (Song, Gu, and Choi 2010; Song, Choi, and Park 2010), are presented. In the following subsections, a *cassette* object and *port* object are introduced in order to provide a compact model description. In each inline cell, they are declared as a *record variable* (cst) for cassette information and P[m] for the status of ports at a processing machine, which is identified by m. The attributes of these variables are summarized in Table 1.

Table 1: Record variables declared for use in event graph models of job shops.

Variable		Type/Value	Description
cst	j	int	Job type of the glasses in cassette *cst*
	p	int	Processing step of the glasses in cassette *cst*
	d	string	ID of the equipment for the next processing step of cassette *cst*
	n	int	Number of glasses in cassette *cst*
P[m]	x	0 ~ port capacity	Number of *empty ports* at the I/O Port of an inline cell m
	rx	0 ~ port capacity	Number of *reserved empty ports* at the I/O Port
	f	0 ~ port capacity	Number of *full-cassette ports* at the I/O Port
	e	0 ~ port capacity	Number of *empty-cassette ports* at the I/O Port

The admissible states of a point in the I/O Port of an inline cell are occupied by a *full cassette* (f); occupied by an *empty cassette* (e); not occupied but *reserved* (rx); and *not occupied* nor reserved (x). The following functions are defined in order to update the state of a port:

- P[m](rx→f): change the state of a port from 'reserved' to 'full-cassette'.
- P[m](f→e): change the state of a port from 'full-cassette' to 'empty-cassette'.
- P[m](e→x): change the state of a port from 'empty-cassette' to 'no-cassette'.
- P[m](x→rx): change the state of a port from 'no-cassette' to 'reserved'.

2.1 Modeling of a Uni-inline Job Shop

Figure 2 presents the reference model of a uni-inline cell with typical processing equipment commonly found in FPD production lines. It consists of several *I/O ports* (P), a *track-in robot* (R), and processing machines, as well as the conveyor. It has a distinguished feature of the glasses of the cassette being loaded at the uni-inline cell return to the same cassette after processing.

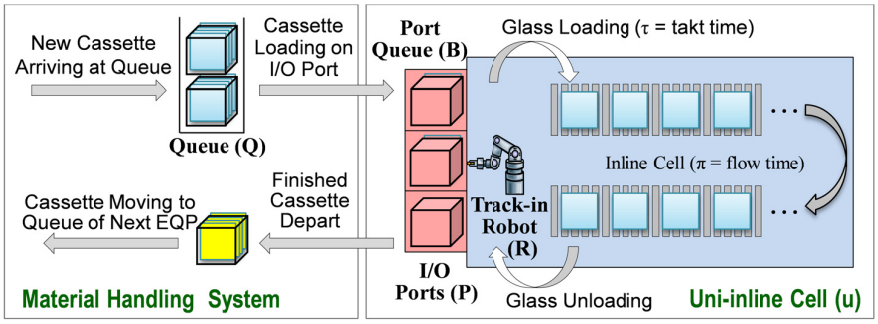

Figure 2: Reference model of a uni-inline cell.

A cassette with new glasses that is stored in the *queue* (Q) is loaded on a port in the I/O ports (P); this process is called *Cassette Loading*. The glasses are then loaded into the Inline Cell using the *track-in robot* (R), with one glass being loaded at every *takt time* (τ), which is called *Glass Loading*. It takes a *flow time* (π) for the glass to reach the end of the cell where it is unloaded into the *same cassette* that it was loaded from at the I/O port. The cassette departs when it is filled with finished glasses and moves to the queue of a processing machine for the next processing step.

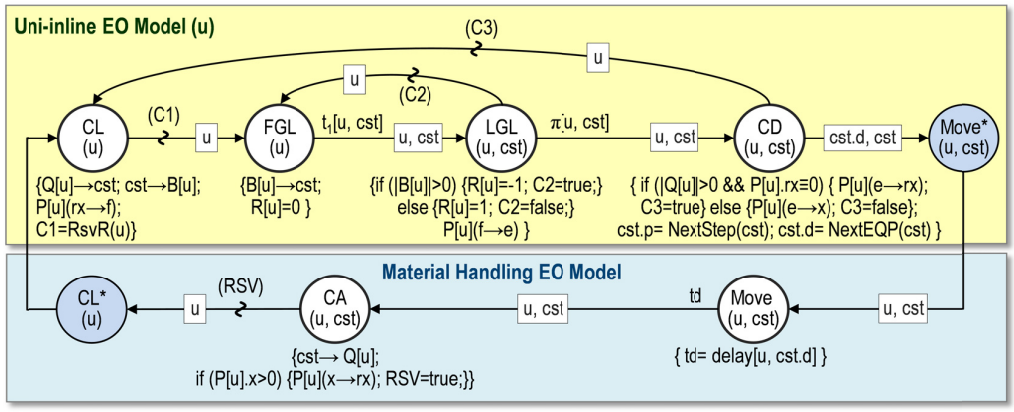

Figure 3: Encapsulated event graph model of a uni-inline job shop.

Figure 3 illustrates an encapsulated event graph model of a *uni-inline job shop*, which is a job shop consisting of uni-inline cells. Used in the encapsulated event graph model are the *state variables*: (1) Q[u],

a list of cassettes waiting for processing at a uni-inline cell u; (2) B[u], a list of cassettes loaded on the ports at a uni-inline cell u; (3) R[u], status of the track-in robot at a uni-inline cell u; and (4) P[u], a record variable for the I/O port of a uni-inline cell u. Also, the *time-related variables* t_1[u, cst], π[u, cst], and delay[u, nu] denote the cycle time for a cassette (cst) of glasses, the flow time of the cell, and the transportation delay time from a uni-inline cell *u* to other cell *nu*, respectively. The en-queue function (cstQ[u]) and de-queue function (Q[u]→cst) are used to manage the cassette queues (e.g. Q[u], R[u]).

The encapsulated event graph model depicted in Figure 3 consists of two event object (EO) models: a uni-inline EO model and a material handling EO model. The *event object model* is an independent parameterized event graph model that encapsulates a group of events that describe the state changes made at an object that is comprised of a discrete-event system.

The material handling EO model simplifies the material handling process of the FPD production line depicted in Figure 1 into the direct transportation of a cassette from a processing machine to another machine. The following events are encapsulated in the material handling EO model:

- CA (u, cst): (1) en-queue an arriving cassette (cst) into Q[u], (2) reserve an empty port if any, and (3) schedule a CL event if a port is reserved.
- Move (u, cst): (1) schedule a CA event to occur after delay[u, cst.d] (moving to the next machine).

The uni-inline EO model also encapsulates the following events related to the loading, inline processing, and unloading of the glasses:

- CL (u): (1) de-queue a cassette (cst) from Q[u], (2) en-queue the cassette (cst) into B[u], (3) set the status of the port to a full-cassette port (P[u](rx→x)), and (4) schedule an FGL event if the track-in robot R[u] is available (C1=RsvR(u)).
- FGL (u): (1) de-queue a cassette (cst) from B[u], (2) set the status of the track-in robot R[u] to busy, and (3) schedule an event LGL to occur after t_1[u, cst].
- LGL (u, cst): (1) set the status of the track-in robot R[u] to idle, (2) change the port status to an empty-cassette port, (3) schedule an FGL event if B[u] is not empty, and (4) schedule a CD event to occur after π[u, cst].
- CD (u, cst): (1) set the port status to a reserved port (P[u](e→rx)) if Q[u] is not empty and no port is reserved, if else, set the port status to an empty port (P[u](e→x)) and (2) the cassette (cst) is updated with next processing-step ID (cst.p) and next processing-equipment ID (cst.d) using the job-routing functions: (1) NextStep(cst) returns the next processing-step ID of a job and (2) NextEQP(cst) returns the next equipment ID that will process a cassette (cst).

In order to encapsulate a group of events into an event object model, a *mirror event* is created for each *boundary event* at the receiving side. In Figure 3, the CL event of the uni-inline EO model and the Move event of the material handling EO model are the *boundary events* of the receiving sides. Thus, their mirrors events CL* and Move* are introduced in the respective event object models. Note that the encapsulated event graph model of the uni-inline job shop presented in Figure 3 is embellished from the parameterized event graph model of the uni-inline job shop presented in Song, Gu, and Choi (2010).

2.2 Modeling of a Bi-inline Job Shop

Figure 4 presents the reference model of a bi-inline cell, which has the distinctive feature of the loading and unloading of cassettes being separated into in-ports and out-ports. In the bi-inline cell, the underlying behavior of processing a cassette is essentially the same as that of a uni-inline cell. As the in-port and out-port are separated, the empty cassette is placed at the in-port when the loading is finished and the out-port requires an empty cassette to unload the finished glasses. Thus, in order to prevent the equipment from the blocking, a mechanism for handling the empty cassettes must be provided.

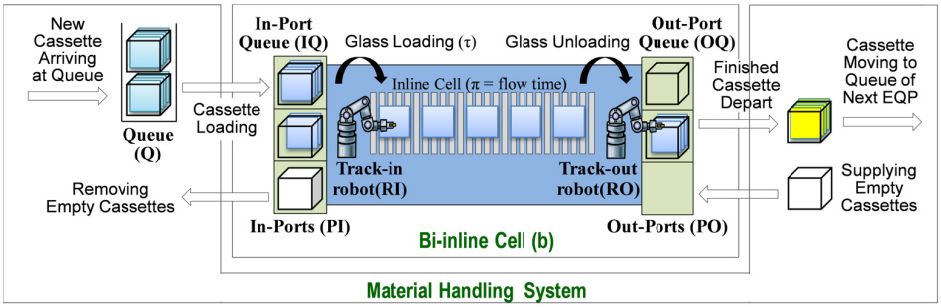

Figure 4: Reference model of a bi-inline cell.

Figure 5 illustrates an encapsulated event graph model of a bi-inline job shop, which is embellished from the parameterized event graph model of the bi-inline job shop presented in Song, Choi, and Park (2012) through grouping events into two event object (EO) models: a bi-inline EO model and a material handling EO model. Similar to that of the uni-inline job shop, two mirror events (CL* and Move*) are introduced in order to encapsulate the event objects.

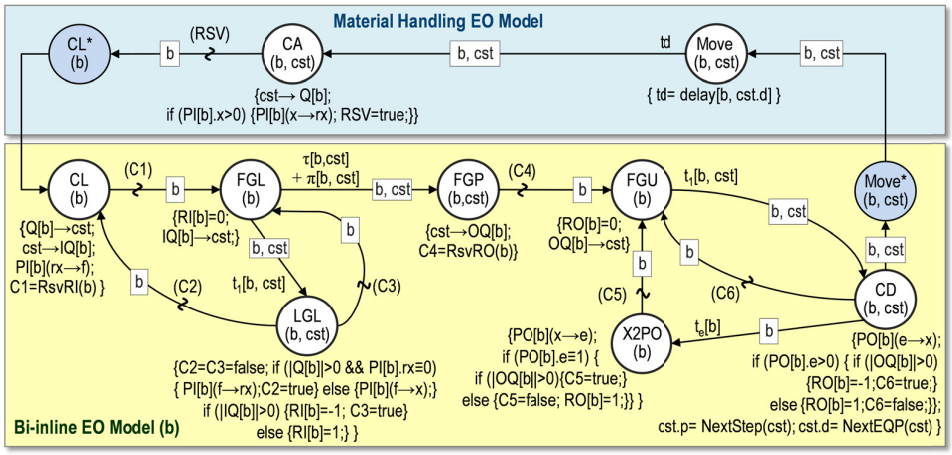

Figure 5: Encapsulated event graph model of a bi-inline cell.

Note that Figure 5 introduces two new events of FGP and X2PO with a number of state variables (PI and PO replacing P, RI and RO replacing R, and IQ and OQ replacing B of the uni-line job shop model) in order to handle the empty cassette. In the LGL event, when the cassette is empty, it is removed from the in-port; then, the in-port scans its queue (Q) for a cassette to load. In an FGP event, if the track-out robot (RO) is idle, it is reserved to unload the first glass. In a CD event, an empty cassette is supplied to the out-port by scheduling an X2PO event to occur after $t_e[b]$ (time to supply an empty cassette). The FGL event schedules an FGP event to occur after the first glass is processed ($\tau[b,cst] + \pi[b, cst]$), where $\tau[b,cst]$ is the *takt time* for a bi-inline cell (b) to process a cassette (cst) with a job-type (cst.j) and processing step (cst.p).

3 EVENT GRAPH MODELING OF A HETEROGENEOUS JOB SHOP

A heterogeneous job shop refers to a job shop consisting of different types of processing equipment. Figure 6 presents an encapsulated event graph model of a heterogeneous job shop with uni-inline cells and bi-inline cells, and with a material handling system. The encapsulated event graph model is

constructed by joining the two models in Figures 3 and 5, and it is a network of three event object (EO) models: uni-inline EO, bi-inline EO, and material handling EO models.

As seen in Figure 6, the material handling EO model is revised in order to handle the cassette arrival in the queue of different types of processing equipment where an in-port reservation is used. Therefore, the CA event is revised by defining the following function RsvP(m) that returns a Boolean variable RSV, which indicates whether the in-port of an inline cell m (either a uni-inline or bi-inline cell) is reserved or not. Note that the I/O ports of a uni-inline cell are represented by a state variable P[u] and the in-ports of a bi-inline cell are represented by a state variable PI[b]. Here, the set variables U and B denote a set of IDs of uni-inline cells and a set of IDs of bi-inline cells, respectively.

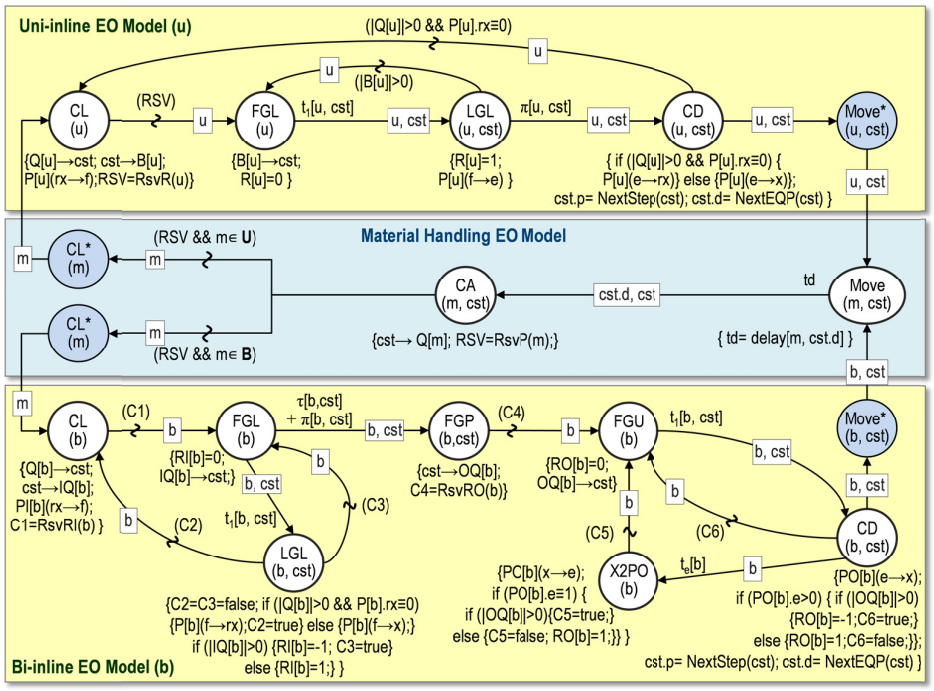

Figure 6: Encapsulated event graph model of an heterogeneous job shop.

4 SIMULATION EXECUTION OF AN ENCAPSULATED EVENT GRAPH MODEL

The encapsulated event graph model of the heterogeneous job shop presented in Figure 6 consists of three event object models with scheduling edges that connect the mirror events of an event object to the boundary events of another event object, as seen in Figure 7(a). The simulation execution of this encapsulated event graph model is conducted using the object-oriented event graph simulator in Figure 7(b). The object-oriented event graph simulator consists of a simulator coordinator and three event object (EO) simulators: one for each event object model in the encapsulated event graph model of Figure 7(a). An EO simulator is similar to the traditional event graph simulator that implements the next-event methodology with the event routines except that the EO simulator does not schedule future events by itself.

The simulation coordinator maintains the *simulation clock* and *future event list* (FEL) as the traditional event graph simulator does, and it also maintains another list, named the *local event list* (LEL), of the *local events* scheduled by the EO simulators. A local event (e) is an event record that carries three attributes: *ObjectID*, *Name*, and *Time* denote the ID of the EO simulator that sent the local event, the

name of the local event, and its scheduled time to occur, respectively. As depicted in Figure 7(b), an EO simulator sends a local event (e) to the simulation coordinator by calling the function ScheduleLocalEvent(e) of the simulation coordinator. Then, the simulation coordinator (1) stores the local events in the LEL, (2) selects the next local event from the LEL, and (3) sends it back to the respective EO simulator through calling the function ExecuteLocalEvent(e) of the EO simulator.

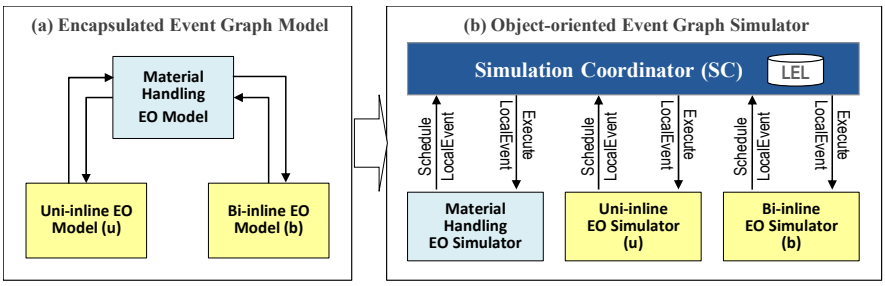

Figure 7: (a) An encapsulated event graph model and (b) its object-oriented event graph simulator.

4.1 Simulation Coordinator

Figure 8 presents an event graph model of the simulation coordinator, which is a single server system that processes the local events in the order of the scheduled event time. The event graph model has two state variables: LEL represents a *buffer* of local events and SC denotes the status of a *machine*, i.e. simulation coordinator (–1 = reserved, 0 = busy, and 1 = idle).

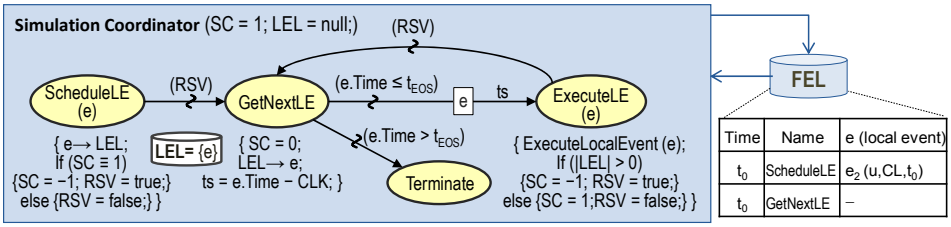

Figure 8: Event graph model of the simulation coordinator.

A brief description of the events used in the event graph model in Figure 8 is as follows:

- ScheduleLE(e): The role of this event is the same as that of the Arrive event in a single server system where the EO simulator sends an enabled local event (e) to the simulation coordinator.
- GetNextLE: This event functions as the Load event in a single server system. The simulation coordinator becomes busy (SC=0) and the next local event (e) is retrieved from LEL (LEL→e).
- ExecuteLE: This event functions as an Unload event in a single server system where the loaded job, local event (e), is ready. The local event (e) is executed on its EO simulator through calling the function ExecuteLocalEvent(e).
- Terminate: This event stops the simulation when the simulation clock exceeds the EOS time.

The event graph model of the simulation coordinator in Figure 8 is executed using the next-event scheduling algorithm (Schruben and Schruben 2006) that maintains the simulation clock (CLK) and the future event list (FEL). The FEL is an ordered list of future events {Time, Name, e}, where Time is the event time, Name is the event name, and e is the local event.

Figure 9 illustrates the main program and event routines of the simulation coordinator in a pseudo-code. The main program of Figure 9(a) implements the next-event scheduling algorithm using four event

routines. The event routine is a subprogram that describes the changes in the state variables and how the future events are scheduled. One event routine is required for each event in the event graph model. Figure 9(b) presents the event routines of four events with an initialize routine. Furthermore, the simulation coordinator defines two functions for handling the local events: ScheduleLocalEvent(e) that stores a local event in the LEL and ExecuteLocalEvent(e) that executes the event routine of a local event e.

(a)

```
Main Program of Coordinator
Begin
  Clock = 0;
  Execute-Initialize-Routine(Clock);
  While (Clock < EOSTime) {
    Retrieve-event (TIME, E-NAME, e);
    Clock = TIME;
    case E-NAME of {
      ScheduleLE: Execute-ScheduleLE-Routine(e, Clock);
      GetNextLE: Execute-GetNextLE-Routine(Clock);
      ExecuteLE: Execute-ExecuteLE-Routine(e, Clock);
      Terminate: Execute-Terminate-Routine(Clock); }
  }
End

ScheduleLocalEvent (e) {
  Schedule-event ( ScheduleLE, e, Clock); }

ExecuteLocalEvent (e) {
  ObjectList [e.ObjectID].ExecuteLocalEvent (e); }
```

(b)

```
Execute-Initialize-Routine (Now) {
  SC=1; LEL = null;
}

Execute-ScheduleLE-Routine (e, Now) {
  e→ LEL;
  if (SC≡1) { SC= -1; RSV= true;} else {RSV= false;}
  if (RSV≡ true) Schedule-event (GetNextLE, Now); }

Execute-GetNextLE-Routine (Now) {
  SC=0; LEL→ e;
  if (e.Time ≤ t_EOS)
    Schedule-event(ExecuteLE, e, e.Time);
  if (e.Time > t_EOS) Schedule-event(Terminate, Now); }

Execute-ExecuteLE-Routine (e, Now) {
  ExecuteLocalEvent (e);
  if (|LEL|>0) {SC=-1; RSV=true;} else {SC=1; RSV=false;}
  if ( RSV≡ true) Schedule-event(GetNextLE, Now); }

Execute-Terminate-Routine (Now) { LEL = null; }
```

Figure 9: Simulation coordinator: (a) main program and (b) event routines.

```
Uni-inline EO Simulator:
  Execute-CL-Routine (u, Now) {…}
  Execute-FGL-Routine (u, Now) {
    B[u]→cst; R[u]=0;
    e = new LocalEvent (u, LGL,  Now + t1[u, cst], u, cst);
    Coordinator.ScheduleLocalEvent (e); } }
  …
  ExecuteLocalEvent (e) { // invoked from SC
    case e.Name of {
      CL: Execute-CL-Routine ( e.ObjectID, e.Time);
      FGL: Execute-FGL-Routine ( e.ObjectID,  e.Time);
      LGL: Execute-LGL-Routine ( e.ObjectID, e.cst, e.Time);
      CD: Execute-CD-Routine ( e.ObjectID, e.cst, e.Time);
      Move: Execute-Move-Routine ( e.ObjectID, e.cst, e.Time); } }
```

Event Routines

ExecuteLocalEvent(e)

Figure 10: Event object simulator for a uni-inline EO model.

4.2 Event Object Simulator

For each event object (EO) in an encapsulated event graph model, the event object simulator is constructed with the event routines and a public function ExecuteLocalEvent(). Figure 10 presents an event object simulator for the uni-inline EO model of Figure 6 in pseudo-code form. The event object simulator consists of two parts: event routines for each event vertex and a function ExecuteLocalEvent(e). Due to space limitations, only the event routine for an FGL event is presented: (1) change the state variables and (2) schedule a new local event LGL to occur after t_1[u, cst]. Here, the local event (e) is defined using an Object ID (u), Event Name (LGL), Event Time (Now + t_1[u, cst]), and Parameter Variables (u, cst). The function ExecuteLocalEvent(e) is invoked by the simulation coordinator at the event ExecuteLE, and then it calls the respective event routine for a given local event (e) using the event name (e.Name).

5 ILLUSTRATIVE EXAMPLE

Figure 11 presents the system architecture of the production simulator for an FPD Fab, which consists of three modules: Input Data module, Object-oriented Event Graph Simulator module, and Output Report module. As seen in Figure 11, two more event object (EO) simulators (FabIn and FabOut) are introduced in the Object-oriented Event Graph Simulator module in order to control the cassette release under the CONWIP policy (Spearman, Woodruff, and Hopp 1990): the FabIn EO simulator releases a new cassette into the Fab when another cassette has completed the last processing step and it maintains a constant level of work in progress. The RTD handles the job selection and machine selection at the CL and CD events of the inline cells, respectively.

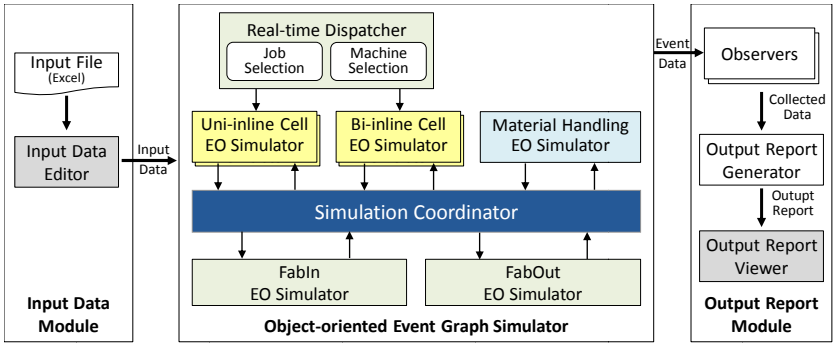

Figure 11: System architecture of a production simulator.

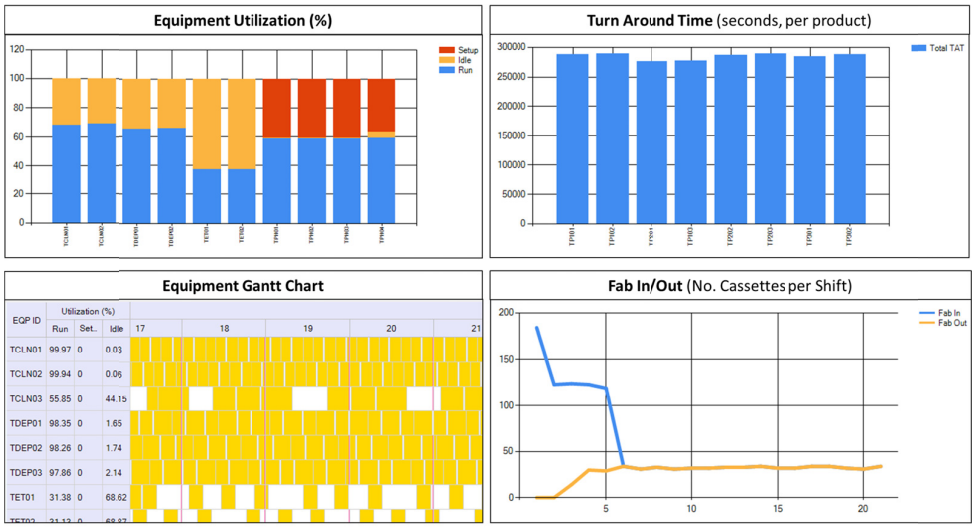

Figure 12: Simulation results of a FPD Fab.

In the Input Data module, a set of data should be prepared in order to execute the production simulation: Product, Equipment, Equipment Port, Bill of Process, Loadable Set, Processing Time, FabOut Plan, and Moving Time. In the Output Report module, a set of observers (Gamma et al. 1994) subscribe to the events defined in the event object models in order to collect the event data generated from the event object simulators and to calculate the performance measures at the end of the simulation run.

Figure 12 presents a simulation output report of the FPD Fab described in Figure 1 under the CONWIP policy of a 600 cassette limit. From the Fab In/Out report, thirty cassettes are released per shift (the Fab operates in three eight-hour shifts) from the sixth shift (excluding warm-up periods), which amounts to 1350 glasses per day. If the CONWIP size is adjusted to 300 cassettes, the turn-around time of the cassettes is decreased by 20.5%.

6 CONCLUSION

This paper presents an object-oriented event graph modeling approach to developing a production simulator for an FPD production line that is a heterogeneous job shop of inline cells and a material handling system. Firstly, the concept of an encapsulated event graph model was provided in order to model the uni-inline job shop and bi-inline job shop. The encapsulated event graph models of two homogeneous job shops are combined into a single encapsulated event graph model that consists of a uni-inline EO model, a bi-inline EO model, and a material handling EO model to represent the entire FPD production line. The resulting model was realized in a production simulator using an object-oriented event graph simulator that consists of the simulation coordinator and the event object simulators for each EO model. The presented encapsulated event graph model successfully captured the heterogeneous properties of the FPD production line with a valid simulation execution method.

Although the inline cells are typical processing machines that are found in FPD production lines, other types of processing machines are also found, including chamber-type and oven-type machines. In a future study, these types of processing machines should be modeled in an event object model and would be easily integrated into the encapsulated event graph model of the heterogeneous job shop presented in this paper.

Furthermore, the AMHS is a core facility of FPD production lines, which handle all material flows between the processing machines through inline stockers and conveyors. In this paper, however, the AMHS is simplified into a material handling EO model that only considers the transportation delay times between the processing machines. As the demands for FPD products increase, the capacity of the AMHS becomes more important to production planners. In order to manage this requirement, the event object model for each type of AMHS machine should be incorporated into the encapsulated event graph model.

REFERENCES

Balas, E., N. Simonetti, and A. Vazacopoulos. 2008. “Job Shop Scheduling with Setup Times, Deadlines and Precedence Constraints.” *Journal of Scheduling* 11(4):253−262.

Buss, A., and P. Sanchez. 2002. “Building Complex Models with LEGOS (Listener Event Graph Objects).” In *Proceedings of the 2002 Winter Simulation Conference*, edited by E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 732−737. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Garey, M. R., D. S. Johnson, and R. Sethi. 1976. “The Complexity of Flowshop and Jobshop Scheduling.” *Mathematics of Operations Research* 1(2):117–129.

Gamma, E., R. Johnson, R. Helm, and J. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Jang, Y. J., and G. H. Choi. 2006. “Introduction to Automated Material Handling Systems in LCD Panel Production Lines.” In *Proceedings of the 2006 International Conference on Automation Science and Engineering*, 223−229.

Lee, C. Y., T. C. E. Cheng, and B. M. T. Lin. 1993. “Minimizing the Makespan in the 3-Machine Assembly-Type Flowshop Scheduling Problem.” *Management Science* 39(5):616−625.

Legato, P., and R. M. Mazza. 2001. “Berth Planning and Resources Optimisation at a Container Terminal via Discrete Event Simulation.” *European Journal of Operational Research* 133(3):537−547.

Parthanadee, P., and J. Buddhakulsomsiri. 2010. "Simulation Modeling and Analysis for Production Scheduling using Real-time Dispatching Rules: A Case Study in Canned Fruit Industry." *Computers and Electronics in Agriculture* 70(1):245−255.

Schruben, L. 1983. "Simulation Modeling with Event Graphs." *Communications of the ACM* 26(11):957−963.

Schruben, D. L., and L. W. Schruben. 2006. *Event Graph Modeling Using SIGMA*. 5th ed. Custom Simulations.

Spearman, M., D. Woodruff, and W. Hopp. 1990. "CONWIP: A Pull Alternative to Kanban." *International Journal of Production Research* 28(5):879−894.

Song. E., S. J. Gu, and B. K. Choi. 2010. "Event Graph Modeling of Electronics Fab with Uni-inline Cells." In *Proceedings of the 2010 Spring Joint Conference of KIIE and KORMS*, Jeju.

Song, E., B. K. Choi, and B. Park. 2012. "Event Graph Modeling of a Homogeneous Job Shop with Bi-inline Cells." *Simulation Modelling Practice and Theory* 20(1):1−11.

Mahdavi, I., B. Shirazi, and M. Solimanpu. 2010. "Development of a Simulation-based Decision Support System for Controlling Stochastic Flexible Job Shop Manufacturing Systems." *Simulation Modelling Practice and Theory* 18(6):768−786.

Yin, J., and B. Chen. 2009. "The Simulation Expert System for Job Shop On-line Scheduling based on G2." In *Proceedings of 16th International Conference on Industrial Engineering and Engineering Management*, 860−863.

Vinod, V., and R. Sridharan. 2009. "Simulation-based Metamodels for Scheduling a Dynamic Job Shop with Sequence-dependent Setup Times." *International Journal of Production Research* 47(6):1425−1447.

AUTHOR BIOGRAPHIES

DONGHUN KANG is a postdoctoral researcher in the Department of Industrial and Systems Engineering at KAIST in Daejeon, Republic of Korea. He holds a Ph.D. from KAIST in Industrial Engineering. His research interests lie broadly in the discrete-event system modeling and simulation. His email address is donghun.kang@kaist.ac.kr.

HYEONSIK KIM is a Ph.D. candidate student in the Department of Industrial and Systems Engineering at KAIST, Daejeon, Republic of Korea. He holds a B.S. and a M.S. from KAIST in Industrial and Systems Engineering. His current research area is system modeling and simulation. His email address is hyeonsik.kim@vmslab.kaist.ac.kr.

BYOUNG K. CHOI is a Professor in the Department of Industrial Engineering at KAIST in Daejeon, Republic of Korea. He is also affiliated with the Faculty of Communication and Information Technology at King Abdulaziz University in Jeddah, Saudi Arabia. He holds a Ph.D. in Industrial Engineering from Purdue University. His current research interests are system modeling and simulation, BPMS, simulation-based scheduling, and virtual manufacturing. His email address is bkchoi@kaist.ac.kr.

BYUNG H. KIM is the president of VMS Solutions Co., Ltd. He holds a B.S. from Sungkyunkwan University, a M.S. from KAIST, and a Ph.D. from KAIST, all in Industrial Engineering. His main interests are simulation-based scheduling and planning, manufacturing information systems, BPMS, and virtual manufacturing. His email address is kbhee@vms-solutions.com.