Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

SIMULATING INVENTORY SYSTEMS WITH FORECAST BASED POLICY UPDATING

Manuel Rossetti
Vijith Varghese
Mehmet Miman
Edward Pohl

Dept. of Industrial Engineering
4207 Bell Engineering Center
University of Arkansas
Fayetteville, AR 72701, USA

ABSTRACT

This paper presents an object oriented framework that facilitates modeling inventory systems whose policy updating is driven by forecast estimates. In an inventory system, the forecast estimates and the forecast error measures are used to set the inventory policy. A simulation approach can address questions regarding the choice of the forecasting technique and the frequency of updating the policy, especially in non-stationary demand scenarios. This paper discusses how the framework can be used to develop simulation models through which these questions can be addressed. In addition, two examples illustrate how to use the framework and how to analyze supply chains with forecast based policy updating.

1 INTRODUCTION

This paper describes an object oriented framework for developing simulation models of inventory systems that involve the updating of policy control parameters based on forecasts or other methods during the execution of the simulation. The framework allows for the implementation of multiple inventory policies, multiple forecasting techniques, multiple demand generation approaches, and multiple policy updating procedures all within a dynamic multi-item multi-echelon supply chain environment. The ease in which simulation models can be created will be demonstrated via two examples.

The software can also be used to study inventory systems with policy updating from a research and development perspective. In order to utilize the framework a basic knowledge of the Java programming language as well as object oriented concepts are required. In addition, users should have a strong working knowledge of simulation methods and practices. The software can assist in answering such important and practical questions as:

- What are the best inventory policies and inventory parameters in a dynamic non-stationary environment?
- Which forecasting techniques should be used for which items at which levels in the supply chain?
- How often should the policy parameters be updated under dynamic non-stationary demand conditions?

Many traditional inventory models found in textbooks assume a stationary demand process. It is well known that under these conditions and with a few additional technical assumptions that the (s, S) policy is an optimal policy. See for example Chapter 7 of Porteus (2002). However, in practice, demand processes are not necessarily stationary and many operational conditions necessitate the use of a wide variety of policies (e.g. reorder point, reorder quantity (r, Q), periodic order up to (R, S), etc.) In addition, the non-stationary behavior of demand has led to practical methods to update the policies as demand is realized.

A simple method that is traditionally used is to estimate the first two moments of the lead time demand distribution using the values obtained via a forecasting technique. The forecast estimate can be used as a reasonable estimate for the mean of the lead time demand and the forecast error in relationships for the variance of the lead time demand. This approach is suggested by Silver, Pike and Rein (1998), Axsäter (2006) and Johnston (1986), to parameterize the lead time demand distribution. Subsequently, the inventory policy parameters can be set based on operational performance measures, perhaps via an optimization model. There are several approaches in the literature to compute inventory policy parameters. The reader can refer to Silver, Pike and Rein (1998), Axsäter (2006) Johnston (1986), Simchi-Levi (2002), Strijbosch (2000), Janssen (1998), Flores (2003) for an overview of these methods.

The main purpose of this paper is to illustrate how the user can model inventory systems (irrespective of the policy) in which a forecasting technique drives the modeling

of the lead time demand distribution and the subsequent setting of the inventory policy parameters via a simulation modeling framework. The paper also illustrates through examples how some of the important questions can be examined via the framework. In addition, the paper offers some basic conclusions in this regard. The ease with which the user can implement other approaches is also illustrated.

The most recent work that is related to this paper can be found in Foote (1995) and Ingalls (2003, 2005). Foote (1995) and Ingalls (2003, 2005) address the frequency of policy updating, by a control based forecasting approach. It may be appropriate to update the policy as often as a new forecast is made; however, this may lead to high volatility in the inventory position. Ingalls (2005) showed the adverse effect of the volatility by investigating the increase in the bull-whip effect in the supply chain. He adopted Foote's algorithm (Foote 1995, Ingalls 2003) in which Foote proposed that the forecast estimate is a random component of the previous forecast and the forecast need not have to change if the Foote-proposed-statistical-test invalidates the current forecast. The tools and techniques discussed in this paper, especially the software, facilitate such studies.

Traditionally, inventory managers often select the forecasting technique whose forecast estimate is "nearly" unbiased and whose forecast error is low. Such a forecasting technique yields parameters for the lead time demand distribution and can be subsequently mapped to policy parameters. The intuition is that the operational performance measure associated with this inventory policy must be best across the different forecasting techniques. The literature is scant related to this intuition. However, there is research that shows that the choice of the forecasting techniques can considerably affect the inventory system. The reader may refer to Gardner (1990), Silver and Rahnama (1986), Jacobs and Wagner (1989) and Sani and Kingsman (1997) to better understand the relationship between the forecasting techniques and the operational performance measures based on an analytical approach. However, for a complex inventory system a simulation approach is often more practical. Gardner (1990) and Sani and Kingsman (1997) investigate these issues by using a simulation approach. This paper illustrates how this question can be addressed using the simulation framework to build simulation models of inventory systems. Then, specific forecasting techniques and policy updating procedures can be attached to the model. Subsequently, the operational performance measures can be estimated and compared across the different forecasting techniques.

This paper builds on the work presented in Rossetti et al. (2006) and Rossetti, Miman and Varghese (2008). The former illustrates the modeling of supply chain networks using an object oriented framework integrated with the Java Simulation Library (JSL). The JSL is an open source simulation library based on Java. The interested reader may refer to Rossetti (2008) for further information on the design and use of the JSL.

Rossetti, Miman and Varghese. (2008) subsequently improved the ease and flexibility in modeling within the inventory layer, by introducing a state pattern design. The demand is the primary entity whose events drive the simulation. The various states of the demand were identified and the transition of the states of the demands are constrained by the new design. The transitions of the states can be sensed and suitable responses can be implemented. This enables sense and respond logistics concepts, which serve as an underpinning for the design of the framework presented in this paper.

Section 2 presents an overview of the design of the framework. This involves both the presentation of the abstract classes that form the basis of the framework and example concrete classes. Section 3 provides examples to illustrate how the classes can be used to address key managerial decisions. Finally, the results are summarized and future work is described.

2 DESIGN AND IMPLEMENTATION

Three Java packages within the JSL form the basis for the modeling framework presented in this paper:

- *jsl.research.supplychain.inventorylayer* – Provides simulation constructs for modeling the use and location of inventory within general supply chain structures.
- *jsl.utilities.forecasting* – Provides basic forecasting algorithms (e.g. moving average, exponential smoothing, etc.) and interfaces for using the algorithms.
- *jsl.research.supplychain.forecasting* – Provides abstract and concrete classes that permit forecast based policy updating within a supply chain simulation.

This section overviews each of these packages with brief code examples and presents how they can be used. The next sub-section presents the modeling of the inventory layer.

2.1 Inventory Layer Package

This paper utilizes the inventory layer abstractions within the JSL for supply chain modeling. The JSL also has other abstractions for supply chain modeling: facility layer and transport layer. The reader may refer to Rossetti et al. (2006) and Rossetti, Miman and Varghese (2008) for further details. The *inventorylayer* package consists of 33 interfaces, 7 abstract classes and 51 concrete classes of which Table 1 lists the key classes with respect to this paper.

Table 1: Classes in *inventorylayer* Package

Class Name	Description
ItemType	Describes items and products
Demand	Represents a request for inventory
DemandState	Represents the status of the request for the inventory
InventoryPolicyAbstract	Abstraction of the inventory policy
BackLogPolicyAbstract	Abstraction of the backlog policy
DemandFillerIfc	Represents objects that fill requests for inventory
DemandFillerFinderIfc	Represents objects that find a filler to fill the demand
DemandSenderIfc	Represents objects that send requests for inventory
DemandGenerator	Encapsulates the demand creation
Inventory	Something that holds inventory
InventoryHoldingPoint	IHP, Represents locations that stock inventory items (sub class from InventoryHolderAbstract)
LeadTimeDemandFiller	Represents locations that produce items and fills demand requests after a time delay
DemandArrivalListenerAbstract	Abstraction of the listener that monitors demand arrival and responds

For simplicity in the presentation, objects that implement an interface or that are instances of a particular class are referred to by the lower case nouns related to the class or interface names whenever the context is clear. For example, item type and demand filler refer to instances of ItemType and DemandFillerIfc, respectively.

Rossetti, Miman, and Varghese (2008) illustrates how to create an inventory system with an (r, Q) inventory policy using the inventory layer package. Inventory models built on the inventory layer package consist of objects/things that send demand (implementing the DemandSenderIfc) and those that fill these sent demands (implementing the DemandFillerIfc). The demand fillers can be directly assigned or indirectly assigned through the demand filler finder. Rossetti et al. (2006) illustrates how easy it is to assign the demand filler to each through a demand filler finder. The demand undergoes the various state changes as discussed in Rossetti, Miman, and Varghese (2008).

Whenever a demand is filled by the inventory or whenever a replenishment arrives at the inventory, the inventory position is updated. The checking of the inventory position and the subsequent responses are handled by the inventory policy classes. In addition to this, whenever a demand is filled, the inventory may react in various other ways, for example the demand may be collected and stored. The abstraction of such reactions are modeled by DemandArrivalListenerAbstract. Whenever the demand request is accepted to be filled, the listener is notified. The listener may respond by collecting the demands and aggregating them across periods of time. Section 2.3 discusses some of the concrete classes of DemandArrivalListenerAbstract.

2.2 Utilities Forecasting Package

This package is an abstraction of forecasting algorithms. It consists of basic forecasting techniques (e.g. moving average, exponential smoothing, etc.) and interfaces for using the algorithms. The *utilities.forecasting* package consists of 1 interface, 1 abstract class and 6 concrete classes of which Table 2 lists the relevant classes in this package.

Table 2: Classes in *utilities.forecasting* Package

Class Name	Description
ForecasterAbstract	Abstraction of forecasting technique
Croston	Implementation of Croston's forecasting technique
MovingAverage	Implementation of moving average
SimpleExponentialSmoothing	Implementation of simple exponential smoothing
Syntetos	Implementation of Syntetos' forecasting technique
HoltsWintersNonSeasonal	Implementation of the non seasonal forecasting technique proposed by Holts and Winters
ForecastErrorIfc	Represents the logic that estimate error

The ForecasterAbstract is an abstraction representing forecasting techniques. The user can sub-class from ForecasterAbstract and implement any forecasting technique. The SimpleExponentialSmoothing class is a sub-class of ForecasterAbstract that implements the simple exponential smoothing forecasting technique. In the SimpleExponentialSmoothing class, the method checkSufficientHistory checks whether there is at least one demand to initialize the forecast; the method initializeForecast initializes the forecast with this single demand; while the method makeForecast makes the forecast according to simple exponential smoothing. All five methods are inherited abstract methods from ForecasterAbstract. The following shows how an instance of SimpleExponentialSmoothing is created:

```
mySimpleExponentialSmoothing = new SimpleExponentialSmoothing(0.25, this+" SES");
```

The ForecastErrorIfc processes the request for the implementation of the evaluation of the performance of the forecasting technique; its concrete implementation may make the error estimates. There are several different fore-

casting error measures and the user can implement these error measures by implementing this interface. The class ForecasterAbstract has the reference to the concrete implementation of the ForecastErrorIfc; it is assigned within the constructor. The class which has the information of the demand and the forecast estimate, (i.e. ForecasterAbstract and its subclasses) is responsible for triggering the setErrorPerformance method of the concrete implementation of the ForecastErrorIfc to evaluate the forecast error. In the next section, we illustrate how the forecast error interface is used.

2.3 Supply Chain Forecasting Package

This package provides abstract and concrete classes that permit forecast based policy updating within a supply chain. This package consists of 3 interfaces, 2 abstract classes and 5 concrete classes. The relevant classes in this package are listed in Table 3. The package consists of several concrete listeners. These are the sub-classes of the abstract demand arrival listener in the inventory layer which was discussed in Section 2.1. In addition to demand arrival, a forecast arrival event may require a particular response and the package has another listener that listens to the arrival of forecasts. These listeners implement the ForecastArrivalListenerIfc interface.

Table 3: Classes in *supplychain.forecasting* Package

Class Name	Description
ForecastManager	Describes the manager that triggers forecasting and error estimation
ForecastArrivalListenerIfc	Represents objects that monitor forecasts and respond
UpdatableIfc	Describes an inventory model whose policy can be updated

2.3.1 Concrete listeners of demand and forecast

This section briefly discusses the listeners that monitor demand and forecast arrivals. This feature is illustrated using the DemandAndForecastArrivalListenerDefault class, a concrete implementation of DemandArrivalListenerAbstract. The following code listing illustrates how a listener can be attached to an inventory:

```
myListener = new DemandAndForecastArrivalListenerDefault(this, myInventory, this,
this+"DemandForecastArrivalListener", myDemandAggregationLevel);
```

This listener listens to the demand arrival for the inventory and the implementation of its inherited abstract method demandArrived() collects the demand. It can schedule the demand aggregation at user specified levels and subsequently make a forecast. This is achieved through the inherited method schedulePeriodicDemandProcessing(). The forecast arrival listener inherits its behaviors from the interface ForecastArrivalListenerIfc. DemandAndForecastArrivalListenerDefault is a concrete implementation of ForecastArrivalListenerIfc. It also inherits from DemandArrivalListenerAbstract and hence it is both a forecast arrival listener and a demand arrival listener. It implements the response when a forecast is made in its inherited method forecastArrived(). In the listener, DemandAndForecastArrivalListenerDefault the response to the forecast arrival is to store the current forecast estimate.

The other demand arrival listeners are DemandArrivalListenerDefault and DemandArrivalListenerWithForecaster. The former collects demand and aggregates it at the user specified interval and subsequently writes it into a file. The latter is similar to the DemandAndForecastArrivalListenerDefault with the exception that it does not have the behavior of a forecast arrival listener. It collects demand and aggregates it at the user specified level and then makes a forecast. The user can easily create other demand arrival listeners with other behaviors by implementing the demandArrived() method of DemandArrivalListenerAbstract class.

The DemandAndForecastArrivalListenerDefault implements ForecastArrivalListenerIfc and thus has the behavior of a forecast arrival listener. The ForecastArrivalListenerDefault is another forecast arrival listener that implements ForecastArrivalListenerIfc through an abstract forecast arrival listener ForecastArrivalListenerDefaultAbstract. It is notified whenever a forecast is made and it responds by writing the forecast estimates and the forecast error into a file. The user can easily create other forecast arrival listeners with other behaviors by implementing the forecastArrived() method of ForecastArrivalListenerIfc interface. The next subsection presents how these listeners are integrated with the forecasting package.

2.3.2 Managing the forecasters

Section 2.2 described the implementation of forecasting techniques. The purpose here is to model an inventory location whose policy is updated by the forecasts. This requires a simulation model element that integrates the forecasting technique with the events in the inventory model. This is handled by the ForecastManager which triggers forecasting and error estimation. The ForecastManager holds a forecasting technique.

The listener that was created in Exhibit 2, DemandAndForecastArrivalListenerDefault holds a ForecastManager. Within the method schedulePeriodicDemandProcessing(), the forecast manager requests the update of the forecast. Thus, the demand arrival listener interacts with the forecast manager. When the forecasting technique makes the forecast, the manager will notify the forecast arrival listener that forecast was made. The forecast arrival listener in this case requests that the policy be updated. The

following code listing illustrates how to assign a forecast manager so that it holds the forecaster that was previously created with the listener.

```
myListener.setDemandForecaster(new ForecastManager(this, mySimpleExponentialSmoothing, this+" SES ME"));
```

The interface UpdatableIfc can be implemented by inventory models to behave as an updatable inventory model. It receives parameters through the receiveParameters() method. The forecast arrival listener may trigger the updating for such an inventory model or the inventory model that creates the inventory and the listeners and the forecaster can trigger the policy updating. Example 1 in the next section illustrates the implementation of the UpdatableIfc interface.

The previous discussion illustrated how to create an (s, S) inventory model updated by the forecasts from a simple exponential smoothing forecast. It follows the following sequence.

- Create an inventory with its backlog policy and inventory policy
- Create the forecasting technique and a forecast manager and hook them together
- Create the demand arrival listener and assign the forecast manager to it
- Create the forecast arrival listener and assign the forecast manager to it

The next section illustrates how to build and analyze models for two different examples.

3 EXAMPLES

The first example illustrates a simple reorder point, order up to level (s, S) inventory system where a forecast will be used to update the policy parameters (s, S) during the simulation. The second example illustrates the framework in the context of a simple but interesting multi-echelon supply chain. The performance across different update periods and forecasting techniques is considered.

3.1 Example 1

The purpose of this example is to illustrate the ability of the software packages to model an inventory system with forecast-based updates. Additionally, the software displays the performance metrics necessary to analyze such systems. A simple inventory system using a (s, S) control policy is modeled and its performance measured for two different demand scenarios: a stationary demand scenario (a simple Poisson) and a non-stationary demand scenario (Non Homogeneous Poisson Process). The inventory system is analyzed for different updating policies: no policy updating and monthly updating based on estimates from a simple exponential smoothing forecast SES (0.15). The s (reorder point) and S (order up to level) are updated based on a simple heuristic mentioned in Silver, Pike and Rein (1998). The reorder point is updated based on the forecast estimate, the forecast error, and a user assigned customer service level. The order up to level is updated based on the reorder point and the order quantity estimated using the EOQ formula.

Exhibit 1 provides a code listing to illustrate the ease of use of the package to simulate and run such an inventory system. A non homogenous Poisson process is created in line 1 where the rate linearly increases from 1 to 100 through 1825 time units and linearly decreases back to 1 in the next 1825 time units.

```
1  PiecewiseRateFunction f = new
   PiecewiseLinearRateFunctin(1.0, 1825.0, 100.0);
   f.addRateSegment(1825.0,1.0);
   NHPPTimeBtwEventRV myNHPPTimeBtwEventRV
2  double initforecastest = 1.0;
3  DistributionIfc lt = new Exponential(1.0);
4  ItemType myItemType = new ItemType(m, "Type-A");
5  int level = 3;int s = 1;int S = 3;
6  int myAggregationLevel = 30;
7  ForecastBasedContinuousUpdatingsSInventoryModel
   myModel =new
   ForecastBasedContinuousUpdatingsSInventoryModel
   (m, myItemType, myNHPPTimeBtwEventRV,
   myNHPPTimeBtwEventRV,
   lt, level, s, S, myAggregationLevel,
   initforecastest,
   "ForecastBasedPeriodicUpdatingsSInventoryModelCase", true);
```

Exhibit 1: Building Forecast-based Update Model

In Exhibit 1, lines 1 – 4 illustrate how to construct the non-stationary Poisson process, the lead-time distribution, the item type, and the parameters for the (s, S) policy. Line 7 instantiates the class ForecastBasedContinuousUpdatingsSInventoryModel where the user specifies all the information required to make the required inventory system. The constructor will create the inventory system with a (s, S) policy and updating based on a user specified forecasting technique (or the default forecasting technique SES with alpha 0.15). The user can declare the item type, demand arrival process, update period, whether to update or not, initial demand rate etc. The simple heuristics to estimate s and S are encapsulated within a class named DefaultsSInventoryOptimizer. An instance of this class will be created within the constructor of ForecastBasedContinuousUpdatingsSInventoryModel.

In this system, the inventory-position of the inventory items constitutes the state of the system. The inventory-position summarizes the state variables on-hand, on-order and the backlog. Since the performance of the system over a finite horizon is desired (the demand pattern), the simulation does not have to be treated as a steady state simulation. However, it may be unrealistic to initialize the system "empty and idle". This can be mitigated by running the simulation prior to the start of the demand scenario pe-

riod (in essence warming up the simulation) or by more intelligently initializing the state. This paper takes the latter approach.

For this paper, all of the simulations are started such that the inventory on hand at the beginning of the run (initial inventory level) will be set equal to the order up to level. This initialization of the on-hand inventory represents a system that begins as if the inventory system has just received replenishment. The policy will be initialized at a very high customer service level 99.9% in order to minimize the risk of a back log at the beginning of the replication. The initial demand rate is assigned in line 3 based on the demand process created in line 1. The initial values specified in line 6 are used only for the definitions of the inventory elements while the initial demand rate is used to estimate the initial (s, S) policy. Then, the initial level of the inventory is set to the initial order up to level.

Currently, the software package has a variety of forecasting techniques including simple exponential smoothing, moving averages, Non-seasonal Winters and intermittent demand forecasting techniques such as Croston, Syntetos and Average Demand. The default updating is based on the simple heuristics discussed previously. However, the object-oriented structure enables the user to easily plug-in other forecasting techniques and other error measures and their own procedures to update the policies.

The software package includes a wide variety of performance measures: the conventional operational performance measures like fill rate, expected backorder, average inventory, number of replenishments, on order etc., as well as volatility measures. In this study we have introduced volatility measures to analyze the inventory systems in terms of the robustness of the control policies. The volatility measures quantify the effect of the increments and the decrements of the policy. For example, when there is a reorder point increment, the system has to make an order which results in an ordering cost. The ordering cost depends on the number of such increments. When there is a decrement in the order up to level, there is a possibility of surplus inventory which results in an increase in holding cost. This holding cost of inventory can be measured by: the average surplus inventory due to decrements as well as the probability of surplus inventory due to decrements. At the end of the planning horizon, there is the possibility of surplus inventory resulting from the policy decrements. This surplus inventory may have to scrapped or sold, resulting in a disposal cost. This cost can be quantified by the average surplus inventory at the end of the planning horizon due to increments and the probability associated with having a surplus. The costs associated with the volatility measures are as listed in Table 5 along with the other volatility measures.

The model was executed for a run length of 3650 days (10 years) with 30 replications. These results are tabulated in Tables 4 and 5. According to Table 4, when there is no update, the non-stationary case improved the on-hand inventory with its smaller value of s, set at the beginning of the simulation. Meanwhile the fill rate, as well as expected backorders, deteriorates dramatically when the demand is non-stationary when there is no updates in the non-stationary case. This is because the policy parameters are set based on the known rate at the start of the simulation, but are not updated as the rate increases throughout the time horizon. The number of replenishments increases in the non-stationary case as the system attempts to catch up with the increasing demand rate. When there are updates, the variance of the statistics decrease considerably for the non-stationary demand scenario. Expected backorders are slightly higher when compared to the stationary demand profiles when updating of the policy is allowed. While fill rate and number of replenishments are less than those in the stationary cases. As expected, lack of knowledge of the demand profile and for non-stationary profiles in general, negatively affect the ability to manage the inventory system. At the same time, as shown in the example, updating policies are more robust to demand profile changes and uncertainty.

Table 4: Results for Traditional Measures

	Stationary Case a		**Non-stationary Case b**	
Performance Measures	**w/o updates**	**w/ updates**	**w/o updates**	**w/ updates**
Expected Backorders	0.01 (0.00)	0.02 (0.02)	46.12 (0.08)	0.15 (0.01)
Average Inventory On Hand	56.42 (0.14)	54.75 (0.17)	0.10 (0.00)	55.04 (0.14)
Fill Rate	1.00 (0.00)	1.00 (0.00)	0.00 (0.00)	0.98 (0.00)
Number of Replenishments	18251.97 (17.77)	17312.77 (95.61)	92172.30 (86.72)	16784.03 (30.73)

Forecasting-based updates are traded-off with the robustness of the inventory control policies. To capture the effects of volatility on items as well as robustness of the control policy, we have introduced a variety of performance metrics, some of which are tabulated in Table 5. Note that these measures are not applicable when there are no updates. Table 5 indicates that non-stationary processes require more dynamic inventory control. It also increases the likelihood that the system will have surplus inventory during or at the end of the planning horizon. The modeling and analysis of the cost of these trade offs are beyond the scope of this paper. Rather, our goal is to illustrate the value of forecast based update systems as well as the flexibility of our software package, which captures the essence of cost modeling through the performance metrics.

Table 5: Results for Additional Volatility Measures

Performance Measures	**Case a w/ up-dates**	**Case b w/ up-dates**
Number of re-order level increments, N^{RIR}	26.03 (1.12)	57.80 (0.36)
Size of re-order level increments, S^{RIR}	1.34 (0.33)	2.78 (0.01)
Number of re-order level decrements, N^{RDR}	17.90 (1.09)	59.47 (0.46)
Size of re-order level decrements, S^{RDR}	1.73 (0.35)	3.16 (0.03)
Size of order up to level increments, S^{SIR}	1.36 (0.22)	2.92 (0.01)
Number of order up to level decrements, N^{SDR}	23.20 (1.12)	59.47 (0.46)
Size of order up to level decrements, S^{SDR}	1.75 (0.24)	3.36 (0.03)
Number of order quantity increments, N^{QIR}	9.57 (1.02)	8.03 (0.07)
Size of order quantity increments, S^{RIR}	1.02 (0.04)	1.00 (0.00)
Number of order quantity decrements, N^{QDR}	10.00 (1.07)	10.07 (0.13)
Size of order quantity decrements, S^{QDR}	1.03 (0.05)	1.20 (0.01)
Number of order up to level increments, N^{SIR}	32.10 (1.19)	57.80 (0.36)
Average surplus inventory due to order up to level decrement, I^S	0.10 (0.06)	0.30 (0.02)
Probability of surplus inventory due to order up to level decrement, W^S	0.06 (0.02)	0.14 (0.01)
Average surplus inventory at the end of a planning horizon due to order up to level decrement, IR^S	0.03 (0.07)	1.13 (0.42)
Probability of surplus inventory at the end of planning horizon due to order up to level decrement, WR^S	0.03 (0.07)	0.57 (0.18)

3.2 Example 2

Example 2 considers a multi echelon system with a single item (Type-A). The demand arrives at 4 retailers each of which have distinct demand arrival patterns. The retailers are replenished by a warehouse and the warehouse by a factory. Inventory at the warehouse and at each retailer and is controlled by (s, S) policies and updated individually by the heuristic described in example 1. For this example, the policies are updated at each location at the same time. Retailer 1 receives a non stationary decreasing trend, retailer 2 has an increasing trend. Retailer 3 receives non stationary demand which initially has an increasing trend, followed by decreasing trend. Retailer 4 has stationary demand. Figure 1 illustrates the structure of the system.

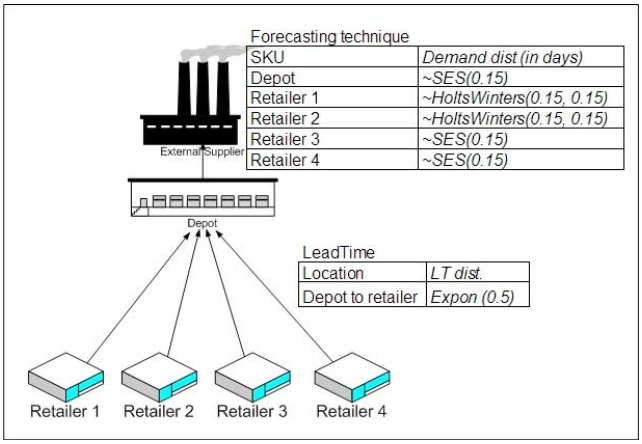

Figure 1: Example2 Network for Type-A Items

The UpdatableSSInventoryNetwork class facilitates the modeling of a multi item multi echelon inventory network with a tree-like structure. This class facilitates the adding of updatable inventory models (as in example 1 to multiple inventory locations. The arborescent structure modeling is similar to the multi item multi echelon network example discussed in Rossetti et al. (2006). Due to limited space the details of this modeling has been omitted.

The purpose of this example is to illustrate the analysis of a multi-echelon inventory network with updatable inventory policies. In addition, the improvement in performance measures when the policy update period and the forecasting technique are changed is investigated. For the convenience of analysis, the aggregate performance measures across each location will be considered. In the base case, the warehouse and the retailers will be assigned a forecasting technique appropriate to its demand. Four more cases are considered as follows:

- Case 1 (base case): Appropriate forecast techniques (as in Figure 3) at each location; policy update period is 30.0 days
- Case 2: Appropriate forecast techniques (as in Figure 3) at each location; policy update period is 120.0 days
- Case 3: Same forecast technique (SES with smoothing constant 0.15) at each location; policy update period is 30.0 days
- Case 4: Same forecast technique (SES with smoothing constant 0.15) at each location; policy update period is 120.0 days
- Case 5: No policy updating

The operational performance measures (Fill rate, *FR*, On hand, *OH*, On order, *OO*) and cost-related volatility measures are considered for the comparison of each case.

The above five experiments were replicated 10 times and each simulation executed for 3650 days. The number of replications was arbitrarily chosen. All these metrics are aggregated across each location in the network and the results are shown in Table 6. The initialization of the simulations were handled in the same manner as described in Example 1. All of the simulations are started such that at each location, the on hand inventory at the beginning of the run (initial inventory level) was set equal to the order up to level. This initialization of the on-hand inventory represents a system that begins as if each location has just received a replenishment. The policy is initialized at a very high customer service level 99.9% in order to minimize the risk of a backlog at the beginning of the replication. The initialization is based on the initial demand rate.

Comparison of case 1 to 2 and case 3 to 4 helps in making inferences on the effect of update period. Here it is observed that when the update period is increased (i.e. less frequent updating), the fill rate decreases and the on hand inventory increases. Comparing case 5 (where there is no updating) to the other cases shows considerable improvement in the fill rate. This justifies the use of forecast based policy updating in non stationary demand scenarios.

The measures related to the volatility shows that the ordering cost related to increments, the holding cost associated with surplus inventory due to policy decrements, and the disposal cost at the end of the planning horizon increases, when the updating is done less frequent. It can be inferred that longer update periods are less desirable. This software can be integrated in the future with a package that has simulation optimization features and can be used to find an optimal update period.

Comparison of case 1 to 3 and case 2 to 4 enables us to make inferences on the effect of the forecasting technique. Case 1 and 2 considers the instance in which the inventory has chosen its forecasting technique based on the demand series. In this example, when the inventory is observing a demand with a trend it chose Winters non seasonal approach. The smoothing parameters are arbitrarily chosen. In case 3 and 4 all the locations choose the same forecasting technique irrespective of the demand pattern. These scenarios are realistic, when the inventory managers ignore the appropriateness of the forecasting technique and want to avoid the cost involved in choosing the best forecasting technique. Here in this example it is seen that the choice of the forecasting technique has no statistically significant difference based on fill rate, on order and on hand. However the cost associated with volatility is reduced if the "nearly" best forecasting technique is chosen. This shows that the inventory managers can use this software to choose the best forecasting technique. This result confirms intuition as well as results found in previous literature.

4 SUMMARY

In this paper, we have presented a flexible object oriented modeling and analysis framework for multi-item, multi-echelon inventory systems. This framework is unique in that it enables one to model a variety of demand scenarios in an environment where the impacts of inventory policy changes based on inventory forecasting models can be studied. In the simple examples presented in the paper, we demonstrated that the use of forecasting tools to update inventory policies has a positive impact on the performance of the inventory system. This effort is part of an ongoing research effort in the area of sense and respond logistics. Future research efforts include developing optimal strategies for policy selection as well as developing techniques for selecting the most appropriate forecasting model for various demand scenarios. All of the models and code developed as part of this effort are available via the JSL as open source software at: <http://www.uark.edu/~rossetti>.

Table 6: Operational Performance Measures Across Each Case

Case	Fill Rate	On Hand	On Order	N^{PIR}	I^S	W^S	IR^S	WR^S
1	0.95 (0.00)	158.30 (3.87)	100.74 (0.14)	1.50 (0.02)	0.79 (0.04)	0.56 (0.04)	0.26 (0.08)	0.24 (0.07)
2	0.92 (0.01)	429.01 (33.97)	98.98 (0.31)	2.52 (0.03)	1.65 (0.08)	0.79 (0.07)	0.52 (0.16)	0.30 (0.06)
3	0.95 (0.00)	158.48 (3.81)	100.75 (0.15)	1.56 (0.03)	0.90 (0.06)	0.59 (0.04)	0.34 (0.08)	0.24 (0.06)
4	0.92 (0.01)	432.68 (34.41)	98.98 (0.30)	2.91 (0.05)	1.88 (0.11)	0.92 (0.08)	0.60 (0.16)	0.36 (0.09)
5	0.51 (0.00)	286.47 (31.60)	98.73 (0.31)	n/a	n/a	n/a	n/a	n/a

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Air Force Office of Sponsored Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Air Force.

REFERENCES

Axsäter, S. 2006. *Inventory control.* 2nd ed. New York: Springer.

Foote, B. L., and B. Leon. 1995. On the Implementation of a Control-based Forecasting System for Aircraft Spare Parts Procurement. *IIE Transactions* 27(2) 210-216.

Garcia-Flores, R., X. Z. Wang, and T. F. Burgess. 2003. Tuning Inventory Policy Parameters in a Small Chemical Company. *Journal of the Operational Research Society* 54(4) 350-361.

Gardner Jr, E. S. 1990. Evaluating Forecast Performance in an Inventory Control System. *Management Science* 36(4) 490-499.

Ingalls, R. G., and B. L. Foote. 2003. Control-based lifecycle forecasting. *IEEE Transactions on Electronics Packaging Manufacturing, (see also IEEE Transactions on Components, Packaging and Manufacturing Technology, Part C: Manufacturing)* 26 (1):5-13.

Ingalls, R. G., B. L. Foote, and A. Krishnamoorthy. 2005. Reducing the bullwhip effect in supply chains with control-based forecasting. *International Journal of Simulation and Process Modelling* 1(1/2) 90-110.

Jacobs, R. A., and H. M. Wagner. 1989. Reducing Inventory System Costs by Using Robust Demand Estimators. *Management Science* 35(7) 771-787.

Janssen, F., R. Heuts, and T. de Kok. 1998. On the (R, s, Q) Inventory Model when Demand is Modelled as a Compound Bernoulli Process. *European Journal of Operational Research* 104(3) 423-436.

Johnston, F. R., and P. J. Harrison. 1986. The Variance of Lead-Time Demand. *The Journal of the Operational Research Society* 37(3) 303-308.

Porteus, E. L. 2002. *Foundations of stochastic inventory theory.* Stanford, California: Stanford Business Books, an imprint of Stanford University Press.

Rossetti, M. D. 2007. JSL: An Open-Source Object-Oriented Framework for Discrete-Event Simulation in Java. *International Journal for Simulation and Process Modeling*.

Rossetti, M. D., M. Miman, V. M. Varghese, and Y. Xiang. 2006. *An Object-Oriented Framework For Simulating Multi-Echelon Inventory Systems.* Piscataway, New Jersey ed. Institute of Electrical and Electronic Engineers.

Rossetti, M. D., M. Miman, and V. M. Varghese. 2008. An Object-Oriented Framework for Simulating Supply Systems. *Journal of Simulation* (Accepted for publication).

Sani, B., and B. G. Kingsman. 1997. Selecting the best periodic inventory control and demand forecasting methods for low demand items. *Journal of the Operational Research Society* 48(7) 700-713.

Silver, E. A., and M. R. Rahnama. 1986. The Cost Effects of Statistical Sampling in Selecting the Reorder Point in a Common Inventory Model. *The Journal of the Operational Research Society* 37(7) 705-713.

Silver, E.A., D. F. Pyke, and P. Rein. 1998. *Inventory management and production planning and scheduling.* 3rd ed. New York: Wiley.

Simchi-Levi, D., P. Kaminsky, and E. Simchi-Levi 2003. *Designing and managing the supply chain : concepts, strategies, and case studies.* 2nd ed. Boston: McGraw-Hill/Irwin.

Strijbosch, L. W. G., R. M. J. Heuts, and E.H.M. van der Schoot. 2000. A Combined Forecast - Inventory Control Procedure for Spare Parts. *Journal of the Operational Research Society* 51(10) 1184-1192.

AUTHOR BIOGRAPHIES

MANUEL D. ROSSETTI is an Associate Professor in the Industrial Engineering Department at the University of Arkansas. He received his Ph.D. in Industrial and Systems Engineering from The Ohio State University. He serves as an Associate Editor for the International Journal of Modeling and Simulation and is active in IIE, INFORMS, and ASEE. He will serve as co-editor for the WSC 2009 conference. He is also author of the forthcoming textbook, *Simulation Modeling and Arena* to be published by John Wiley & Sons.

VIJITH M. VARGHESE is a Ph.D. Candidate in Industrial Engineering at the University of Arkansas.

MEHMET MIMAN is a Ph.D. Candidate in Industrial Engineering at the University of Arkansas.

EDWARD POHL is an Associate Professor of Industrial Engineering at the University of Arkansas and holds the John L. Imhoff Chair in Industrial Engineering. He is currently the Director of the Operations Management Program. Ed received his Ph.D. in systems and industrial engineering from the University of Arizona. His primary research interests are in repairable systems modeling, reliability, decision making under uncertainty, engineering optimization, and systems engineering. Ed is a Senior member of IIE, a Senior member of ASQ, a Senior member of IEEE, a member of INFORMS, MORS and INCOSE.