Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

AN OBJECT-ORIENTED FRAMEWORK FOR SIMULATING FULL TRUCKLOAD TRANSPORTATION NETWORKS

Manuel D. Rossetti
Shikha Nangia

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

ABSTRACT

In this paper, we discuss the design and use of an object-oriented framework for simulating full truckload (FTL) networks. We present a context for how the framework can be used through its application to an example trucking network. In addition, we describe the design by examining the major conceptual artifacts within the object-oriented model. The framework is built on a Java Simulation Library (JSL) and permits easy modeling and execution of simulation models. The example and discussion indicate the capabilities and flexibility of modeling with the framework. In addition, we summarize our future research efforts to model other transportation networks.

1 INTRODUCTION

Freight transportation plays a fundamental role in every modern supply chain as it allows production and consumption to take place at locations that are several hundreds or thousands of miles away from each other. It is essential to move raw materials from sources to plants, semi-finished products between factories, and final goods to customers and retail outlets. Transportation systems are complex organizations which require considerable human, financial and material resources. Transportation costs accounts for a signification part (often between one-third and two-thirds) of the logistics costs in several industries and have a major impact on the level of customer service (Ghiani et al., 1994). It is therefore not surprising that transportation plays a key role in logistic network design, analysis, and management.

There are many different types of transportation networks that can be utilized within a supply chain. Over the road transportation can be classified as full truckload (FTL) or less-than-truckload (LTL). FTL transportation moves a full load directly from its origin to its destination in a single trip. If the shipment adds up to much less than the vehicle capacity (LTL loads), it is more convenient to resort to several trucking services in conjunction with consolidation terminals rather than use direct shipments. In this paper, we concentrate on long-haul freight transportation in the form of a FTL transportation network.

The main contribution of this research is the prototype object-oriented software framework representing the key simulation elements within FTL networks. Booch *et al.* (1999), define a software design framework as “an architectural pattern that provides an extensible template for applications within a domain.” A framework provides a set of abstract and concrete classes that can be extended via sub-classing or used directly to solve a particular problem within a particular domain. In our case, we have designed a simulation framework that can be used to easily develop simulation models of FTL networks. A framework can be implemented in any object-oriented language. We implement the framework within the Java programming language build on top of the Java Simulation Library (JSL). The JSL is an open source object-oriented framework for discrete-event simulation in Java. An overview of the capabilities of the JSL can be found in Rossetti (2007).

An excellent overview of the use of simulation within logistics and transportation systems can be found in Manivannan (1998). The overview also makes it clear why simulation can be a powerful tool in strategic transportation decision making. It is interesting to note that during our literature review, we found many articles involving the application of optimization techniques to such logistics problems such as driver assignment, fleet management, dispatching rules, vehicle routing, etc. However, surprisingly, we have found very little detailed discussion of the simulation of FTL networks. We highlight a few of our findings in the following paragraphs.

The simulation of trucking is often a component in a larger supply chain simulation. For example, Dalal et al. (2003) describe the use of Simulation Dynamics Supply

Chain builder to model the movement of automobile shipments within a supply chain. The goal of the overall simulation was to reduce the order to delivery times within the supply chain. An interesting aspect of their modeling was to how to realistically initialize the network to test various alternatives. Similarly, Hamber (2003) describes the use of TLOADS within the a military supply chain. Chan (2006) models a distribution system in order to minimize the total traveling time and applied simulation to measure the effectiveness of the optimization results. This later application of simulation is a common theme within the literature. That is, to use simulation to evaluate the robustness of the logistics decisions made via an optimization algorithm or heuristic rule. This also makes apparent one of the key challenges in simulating logistics systems. A simulation model of a logistics system (e.g. FTL) may require sophisticated algorithms to be implemented in order to fully mimic the planning and dispatching of the vehicles.

Within an FTL context, we only found a few relevant references. Youngblood (2000) developed a simulation model in SIMNET II (SIMNET II is a discrete simulation language written in FORTRAN (Taha, 1992)) which investigated the effects of various dispatching methodologies between terminal cities in a truckload (FTL) trucking environment. Youngblood (2000) claimed that by reducing driver tour length via predetermined routes through the terminal cities, the driver retention can be improved rather than being subjected to the random nature of traditional dispatching methods. Petre (2000) used simulation to determine the possible synergies between truckload and inter-modal transportation. Petre (2000) also used SIMNET II to model the two different modes of freight transportation; over-the-road (OTR) trucking services and inter-modal services. Petre (2000) argued that by integrating over-the-road and inter-modal operation, greater operational flexibility, better balance, lower cost, and better customer service may be achieved over the use of the single modes independently.

Ervin and Harris (2004) developed discrete-event simulation in Arena for evaluating the effect of over the road and hours of service rules for a FTL trucking fleet. The model allows for the determination of fleet utilization, cycle times, and customer service to guide company decisions. The model incorporates demand generation, load and truck assignments, capacity management, customer pick-up and delivery, as well as transport execution.

The ability to more easily evaluate the algorithms or rules within an FLT network is one of the main motivators for this research. When performing an analysis, such at that in Ervin and Harris (2004) or in evaluating dispatching rules, the user of a transportation simulation model will be interested in understanding the effect on the loaded miles versus empty miles. At the same time the user will be concerned with customer and driver satisfaction. From the driver's standpoint, the company needs to keep track of the average distance traveled per day, as wages depend on the miles traveled. From a customer's standpoint, the average delay in receiving the load will be the main concern. The economic advantage of a transportation network can be evaluated by considering many criteria such as equipment utilization, total circuitry, total unloaded miles, total loaded miles, and percentage of on time pick-ups (Taha and Taylor, 1994). Our framework facilitates the building of FTL networks and the estimation of such quantities. Our research is also part of a larger effort to build frameworks for simulating supply chains, see for example Rossetti et al. (2006). Of which, this research constitutes the transport layer. The other layers include the inventory and facility layers.

In what follows, we first give a description of the structure and functionality of the transport layer. Due to space limitations, we will concentrate on providing an understanding of the modeling issues so that the reader can understand what the framework can model. This should also give an idea of its capabilities. A detailed discussion of the implementation (Java coding) of the framework is beyond the scope and space limitations of this paper. We then describe a simple example FTL model to illustrate the output capabilities of the architecture. We wrap up with a summary of our efforts and present ideas for future research.

2 TRANSPORT LAYER

An object-oriented analysis begins with the identification of the key elements within the system, their roles, attributes, relationships with each other, and modeling and implementation issues. We identified the following as the key conceptual elements needed within a generic FTL simulation model: physical network (locations, lanes, etc.), loads and their generation, dispatcher and driving tasks, truck, trailer, and driver. These conceptual elements were embodied in Java classes: *Network, Location, Lane, Load, LoadGenerator, Dispatcher, DispatchTask,* and *Driver*. Table 1 provides a list of the classes and interfaces within the framework. The concept of a trailer and a truck were modeled as resources within a physical spatial frame. In particular, the JSL has the notion of a spatial resource (SpatialResource). A spatial resource is a resource that has a physical initial position and can be moved within a spatial model.

The Network class is the spatial representation of the transportation network with a valid geometric coordinate system. The Network consists of many Locations. A transport demand placed by a customer is represented as a Load. A Load moves from one Location (origin) to another (destination). Therefore, a Location plays the role of both an origin and a destination in the Network. Since the

Location in the truckload transportation model emulates the concept of an actual physical location, it is represented by means of co-ordinates. The distances between locations are supplied by the underlying spatial model (subclass of the abstract base class SpatialModel) attached to the network.

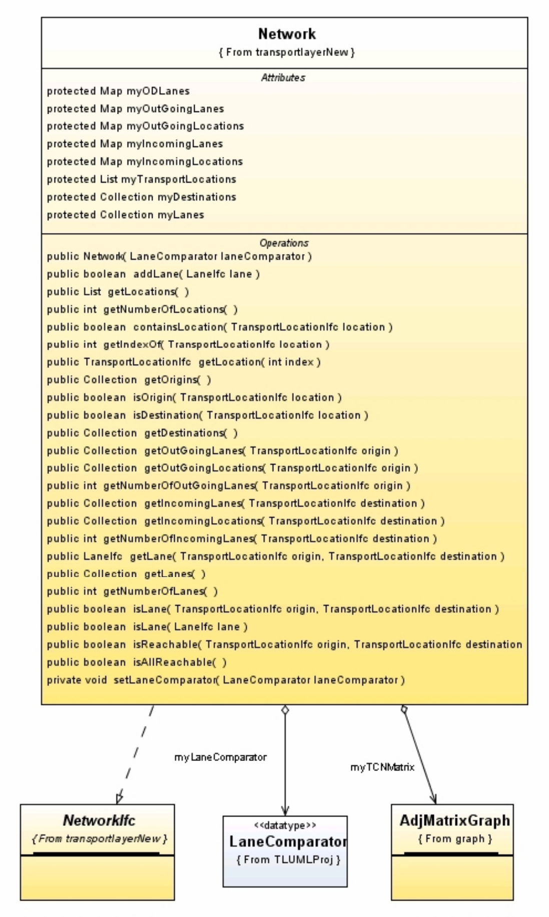
Table 1: List of Classes and Interfaces in Framework

AbstractDispatcher	Driver
AbstractDispatcherFTL	LoadGenerator
DispatcherFTL	Location
DispatchTask	Lane
Network	Load
TransportationNetwork	NetworkIfc
DriverSelectionRule	TrailerSelectionRuleIfc
DispatchLoadSelectionRuleIfc	DriverSelectionRuleIfc
TransportLocationIfc	LoadReceiverIfc
TruckSelectionRuleIfc	LoadSenderIfc
TransportTaskReceiverIfc	

A SpatialModel is a representation for physical space within a JSL simulation. For example, a GreatCircleDistanceSpatialModel class can supply distances based on using the great circle distance between two points on the earth. The elements contained in a SpatialModel are called spatial elements. A SpatialModel has methods to add and remove spatial elements from the spatial model, provide default positions to the spatial elements if not supplied by the user, set coordinates for the spatial elements if supplied, compare two spatial elements if they are at the same coordinates, get the coordinates of the spatial element, get the distance between two spatial elements etc.

The Locations in the Network are connected by means of Lanes. A Lane is an object which connects two Locations and is directional; which means that a Lane connecting Location A to Location B is a different object from the Lane object connecting Location B to Location A. One Location in a lane serves as an origin and the other Location serves as a destination. A Location has the ability of setting their loading and unloading distribution and can notify the driver about the loading and unloading times it might take at that location.

In a broader sense, the Network in the framework resembles a directed graph in which Locations serve as the vertices and Lanes serve as the edges. Figure 1 illustrates some of the attributes and methods of the Network class. The Network class encapsulates much of the functionality of a graph representation including for example the checking of reachability. A Location may have outgoing lanes and incoming lanes. A Lane is defined as an incoming lane for a Location if that Location happens to be its destination. Similarly, a Lane is defined as an outgoing lane for a Location for which that Location happens to be its origin. The concept of incoming and outgoing lanes are important in deadheading or bobtailing resources from a different location when all resources are not available at the origin of the Load. Bobtail implies that the driver (and truck) travels unloaded without a trailer and deadhead implies that the driver (and truck) travels unloaded with a trailer.



```
protected Map myODLanes
protected Map myOutGoingLanes
protected Map myOutGoingLocations
protected Map myIncomingLanes
protected Map myIncomingLocations
protected List myTransportLocations
protected Collection myDestinations
protected Collection myLanes
```

```
public Network( LaneComparator laneComparator )
public boolean  addLane( LaneIfc lane )
public List  getLocations( )
public int  getNumberOfLocations( )
public boolean  containsLocation( TransportLocationIfc location )
public int  getIndexOf( TransportLocationIfc location )
public TransportLocationIfc  getLocation( int index )
public Collection  getOrigins( )
public boolean  isOrigin( TransportLocationIfc location )
public boolean  isDestination( TransportLocationIfc location )
public Collection  getDestinations( )
public Collection  getOutGoingLanes( TransportLocationIfc origin )
public Collection  getOutGoingLocations( TransportLocationIfc origin )
public int  getNumberOfOutGoingLanes( TransportLocationIfc origin )
public Collection  getIncomingLanes( TransportLocationIfc destination )
public Collection  getIncomingLocations( TransportLocationIfc destination )
public int  getNumberOfIncomingLanes( TransportLocationIfc destination )
public LaneIfc  getLane( TransportLocationIfc origin, TransportLocationIfc destination )
public Collection  getLanes( )
public int  getNumberOfLanes( )
public boolean  isLane( TransportLocationIfc origin, TransportLocationIfc destination )
public boolean  isLane( LaneIfc lane )
public boolean  isReachable( TransportLocationIfc origin, TransportLocationIfc destination
public boolean  isAllReachable( )
private void  setLaneComparator( LaneComparator laneComparator )
```

Figure 1: Network Class

A Load in this framework represents a request to move material from one location (origin) to another location (destination). A Load is sent by a specific sender to be completed by the network. Every Load in the network has attributes like its network, sender, origin, and destination and possibly a due date. The Loads are generated by the LoadGenerator class in the framework, which is similar to CREATE modules found in other simulation languages. In addition, the load generator can generate loads for the network based on a general origin/destination probability specification. A basic overview of the load processing is as follows:

- LoadGenerator generates Load
- LoadGenerator sends the Network the message to create the Load

- Network creates the Load
- Network sends the Load to the Dispatcher for processing
- Dispatcher will create a DispatchTask for the load
- Dispatcher checks for the availability of the resources, seizes them and allocates them to the Load
- Next it notifies the Driver to execute the DispatchTask.
- Finally the Driver will transport the Load to its destination.
- Resources get released at the destination of the load
- Driver notifies the Dispatcher

This is also illustrated by the sequence diagram given in Figure 2.

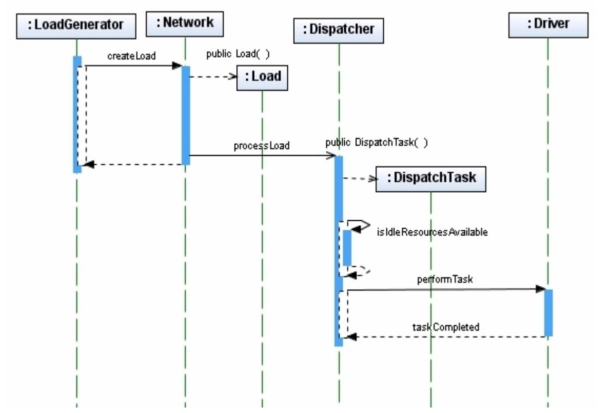
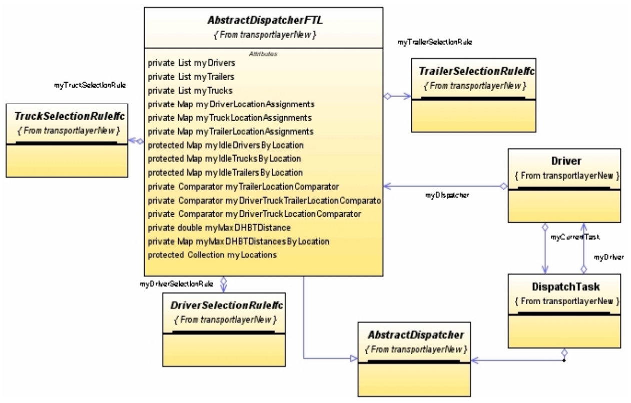


Figure 2: Load Handling Sequence Diagram

The transportation network must have an object to allocate the resources (drivers, trucks, trailers) to the loads that require movement between origins and destinations. In the framework, AbstractDispatcher represents the most general dispatcher class, which has the most basic behaviors that have been identified for dispatching loads. A client using this framework can subclass from AbstractDispatcher and add more functionality to make it more specific for their requirements. For instance, we have constructed an AbstractDispatcherFTL class which subclasses from AbstractDispatcher and is more specific to full truckload transportation network modeling. Further downstream, there is even a more specific class DispatcherFTL sub-classing from AbstractDispatcherFTL in which we have implemented our own specific searching and dispatching algorithms for resources. A client may want to use this framework for building their truckload transportation network but may not necessarily want to select and allocate the resources through our default mechanisms. In that case they can always subclass from these dispatcher classes and override the methods and selection and allocation rules.


AbstractDispatcherFTL
{ From transportlayerNew }
Attributes
private List myDrivers
private List myTrailers
private List myTrucks
private Map myDriverLocationAssignments
private Map myTruckLocationAssignments
private Map myTrailerLocationAssignments
protected Map myIdleDriversByLocation
protected Map myIdleTrucksByLocation
protected Map myIdleTrailersByLocation
private Comparator myTrailerLocationComparator
private Comparator myDriverTruckTrailerLocationComparato
private Comparator myDriverTruckLocationComparator
private double myMaxDHBTDistance
private Map myMaxDHBTDistancesByLocation
protected Collection myLocations
myTruckSelectionRule
TruckSelectionRuleIfc
{ From transportlayerNew }
myTrailerSelectionRule
TrailerSelectionRuleIfc
{ From transportlayerNew }
Driver
{ From transportlayerNew }
myDispatcher
myCurrentTask
myDriver
DispatchTask
{ From transportlayerNew }
myDriverSelectionRule
DriverSelectionRuleIfc
{ From transportlayerNew }
AbstractDispatcher
{ From transportlayerNew }
Figure 3: AbstractDispatcherFTL Class Diagram

As can be seen in Figure 3, the modeler can supply their own rules for selecting the next truck, trailer, or driver for when a load requires a resource. During the transportation, the resources move from one location to another. As the resources move they get released at a different location and get added to the idle resource lists of the destination and removed from the idle resources list of the origin. These data structures can be used to select the next resource (driver, truck, or trailer). AbstractDispatcherFTL has variety of methods for searching locations with resources. These methods are useful in identifying locations that can be used to initiate a bobtail or deadhead. Some of these methods are as follows:

- findLocationWithMostIdleResources(): Facilitates finding locations with most idle resources.
- findLocationWithMostIdleDriversAndTrucks() : Facilitates finding locations with most idle driver and an idle truck combinations.
- getLocationsSortedByMostIdleResources(): It is the method by which we can get a collection of locations sorted by most idle resources.
- getLocationsSortedByMostIdleTrucksAndDrivers(): It is the method by which we can get a collection of locations sorted by most idle truck and idle trailer pairs.
- getLocationsSortedByMostIdleTrailers(): It is the method by which we can get a collection of locations sorted by most idle trailers.
- findLocationWithIdleDriverAndTruckWithMinDeliveryTime(): This method finds the driver which will take the minimum expected delivery time to complete the bobtail task in case there is more than one driver available at a location.
- findLocationWithIdleResourcesWithMinDeliveryTime(): This method finds the driver which will take the minimum expected delivery time to complete the deadhead task in case there is more than one driver available at a location.

After being asked to handle a load, the dispatcher creates a dispatch task for allocation to a driver. A DispatchTask represents a task that a driver must perform within the network. There are five basic tasks:

- Transport – Transport task is created when combination of an idle driver, trailer and truck are all available for transporting a load.
- BobTailToLoad – This task is created when an idle trailer is available at the origin of the load but either an idle driver or an idle truck or both are not available.
- DeadHeadToLoad – This task is created when all three resources i.e., an idle truck, an idle trailer and an idle driver are not available together at the origin of the load. A combination of all these three idle resources is searched and the selected driver at the deadhead location is assigned this task and asked to deadhead to the origin of the load to pick up the load for transport.
- BobTailNoLoad - This task is created when a driver is asked to bobtail to another location and there is no load required to be picked up for transport.
- DeadHeadNoLoad - This task is created when a driver is simply asked to deadhead to another location and there is no load required to be picked up for transport.

The DispatchTask knows its driver, truck and trailer to which it has been assigned. The driver knows its current task that it has been assigned. The dispatcher asks the DispatchTask to allocate and release resources. The DispatchTask makes a request for a single unit of its driver, truck and trailer for allocation and releases the driver, truck and trailer requests when it is done. A task also knows about its current origin and destination. In addition, every DispatchTask has an associated expected task time, i.e. the time expected to complete that task. The expected task time plays an important role in the selection of the driver when there is more than one idle driver available in the network. A user may want to select the driver which takes the minimum expected time to deliver the load.

We have implemented a concrete sub-class of AbstractDispatcherFTL called DispatcherFTL based on some standard allocation rules. DispatcherFTL utilizes a number of queues and other data structures to facilitate the allocation of resources to loads via a task. For example, the following queues are available:

- Queues of the loads at their origins waiting to be dispatched
- Queues of the loads at their origins waiting for a bobtail
- Queues of the loads at their origins waiting for a deadhead
- Queues of the loads at their origins waiting to be transported to their destinations
- Queues of the loads at their destinations that have been loaded and wait in the queue until their transport is completed
- A queue holding all loads waiting to be dispatched.

The task is enqueued in the overall dispatch queue (list) of the network and dispatch queues at the load's origin. The dispatcher next checks if there is an idle truck, trailer and driver at the load's origin which can facilitate its transportation to its destination. If there are idle resources available, then they are selected based on the selection rules provided in the framework. The default selection rule for truck and trailer is to simply select the first idle truck and trailer available from the idle list of trucks and trailers at the load's origin. In the case of the driver, our model provides two rules for driver selection. The first is to select the first idle driver from the idle list of drivers at the load's origin. The second is to select the driver which will take minimum expected time to complete the transport of the load. The default driver selection rule is set as the one using the minimum expected delivery time. The framework facilitates the user to plug in their own selection rule to select a resource at a particular location. If the user supplies no selection rule then the default rules are executed.

Based on the availability and requirement of the resources, we can classify the dispatching mechanism into 3 categories. The first is where all the resources are available for the load; in the second case the load has only an idle trailer at its origin but no idle driver and truck pair. Lastly, none of the resources are available for the load at its origin. In the first case, all the three resources are available at the origin of the load. The resources are selected according to the selection rules and are set for the task. Dispatch task is now set as transport type. Once, the task is set as a transport task, the dispatcher removes the load from the overall dispatch queue and the dispatch queue of the load's origin and places it in the transport queue of the origin. Next, the resources are allocated. The allocated driver is asked to transport the load from its origin to its destination. After loading, the task is removed from the transport queue and added to the incoming load queue at the destination of the load. After the loading is completed, the driver travels to the destination of the load. After transport is completed the load is unloaded. After unloading, the driver will notify the dispatcher and the dispatcher will remove the task from the incoming load queue of the destination. The driver, trailer and truck are now released and added to the idle list of resources at the destination. The dispatcher will then get an opportunity to reallocate the resources in the network in order to process other tasks pending in the network.

For the second dispatching case, if there is an idle trailer but no idle driver and truck pair at the origin, the dispatcher sets the task as a bobtail task and searches for a location within its defined reachable range from where it can get an idle driver and truck pair. The task is removed from the overall dispatch queue and the dispatch queue of the load's origin and added to the bobtail dispatch queue at the origin. The framework provides three options by which the user can select a bobtail location. First we can select the location based on the minimum expected delivery time. We search all the locations and find the location which has an idle driver which takes the minimum expected time taken to complete the transport. The location must also have an idle truck. The second option is to select the location based on most number of idle driver and truck pairs available. The location which has the maximum number of idle (driver, truck) pairs is selected. The third option is a simple one where we just select the first location available with an idle driver and truck pair during the search. Once, an idle driver, truck and trailer are selected, they are allocated to the load. The selected driver is asked to bobtail from the bobtail location selected to the load's origin.

Finally, in the third case for dispatching, when there are no idle resources available at the load's origin and there are no inbound loads, the dispatcher sets the task as a deadhead task and searches for idle resources within the defined reachable range. The logic for searching the deadhead location is same as that of bobtail using any of the described three options. Only now in addition to checking for the availability of an idle driver and an idle truck, we will also check for the availability of an idle trailer at the location.

The Driver resource is represented by the Driver class in the framework. Driver extends SpatialResource. Driver is the resource which is responsible for completing the task assigned to it by the dispatcher. It is responsible for transporting the load from its origin to its destination in case of the transport task or just completing the transport in case of the bobtail and the deadhead task. The Driver class in this framework has been modeled in accordance with the Department of Transportation (DOT) regulations for truck drivers and their hours of service rules. According to the DOT regulations, a driver's "tour of duty" cannot go beyond the 14th hour after coming on-duty, following 10 hours off-duty. After the completion of 10 consecutive hours off duty, the driver shall not drive after the 14th hour from the start of "tour of duty". This is also called the 14-Hour rule. A driver shall not drive for more than 11 hours following 10 consecutive hours off duty. This is also called the 10-Hour rule. Three hours in the 14 hour work window covers inspections, fueling, and loading or unloading time. Any delay over 3 hours immediately begins to erode the 11 hours the driver has available to drive. A driver shall not drive after having been on duty for 70 hours during a consecutive 8 day period. A driver may restart the 8 day period after a consecutive 34 hour off duty period. This is also called the 70-Hour rule. Table 2 summarizes these basic rules.

Table 2: Summary of Driver Hours of Service

	Regulations
Driving Time	11 hours
Total On-Duty Time	14 consecutive hours
Off-Duty Time	10 consecutive hours
Total Time (Driving + On-Duty, Not Driving + Off-Duty)	24 hours
Cumulative On-Duty	70 hours in 8 days
Cumulative On-Duty "Restart" Period	34 hour restart at any point in a driver's 7 or 8 day cycle

Based on the above rules and regulations, the Driver class was modeled using discrete event scheduling and a rigorous state transition model as illustrated in Figure 4. A driver can either be on-duty or off-duty. In either case, they can be currently allocated a task. Thus, at any point of time, a driver can be in only one of the following 4 states – on duty without task (idle), on duty with task (busy), off duty with task, or off duty without task. For simplicity, we assume that if a driver must go off duty while assigned a task, that the task stays with the driver.

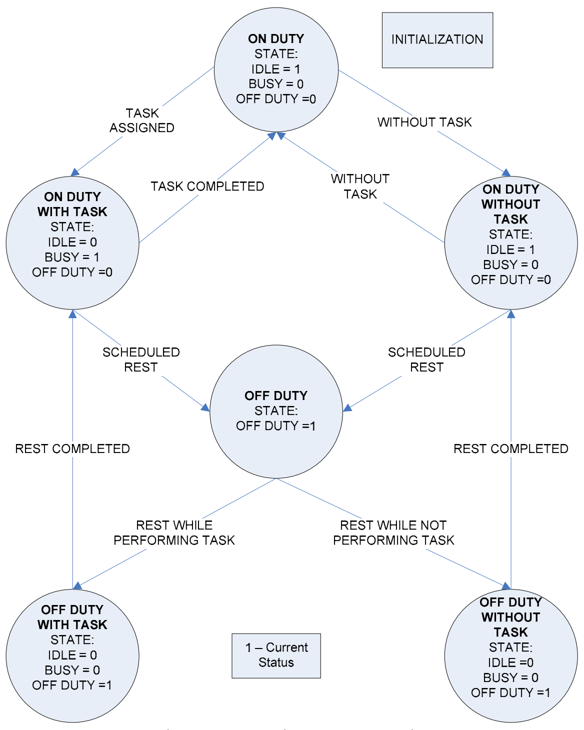

ON DUTY STATE: IDLE = 1 BUSY = 0 OFF DUTY =0
INITIALIZATION
TASK ASSIGNED
WITHOUT TASK
TASK COMPLETED
WITHOUT TASK
ON DUTY WITH TASK STATE: IDLE = 0 BUSY = 1 OFF DUTY =0
ON DUTY WITHOUT TASK STATE: IDLE = 1 BUSY = 0 OFF DUTY =0
SCHEDULED REST
SCHEDULED REST
OFF DUTY STATE: OFF DUTY =1
REST COMPLETED
REST COMPLETED
REST WHILE PERFORMING TASK
REST WHILE NOT PERFORMING TASK
OFF DUTY WITH TASK STATE: IDLE = 0 BUSY = 0 OFF DUTY =1
1 – Current Status
OFF DUTY WITHOUT TASK STATE: IDLE =0 BUSY = 0 OFF DUTY =1

Figure 4: Driver State Diagram

The modeling of the transportation network involved developing Java classes using the ModelElement or SchedulingElement base classes from the JSL. The implementation of the framework was tested on small cases for which the exact sequence of events and resource allocations were known in advance. This verified that the Java coding represented what was intended. We then developed a larger case study for testing purposes and to illustrate the statistical quantities captured by the model. This is briefly discussed in the next section.

3 EXAMPLE CASE STUDY AND RESULTS

In this section, we illustrate the output available from the framework. It is not our intention here to fully analyze this system. Rather we are using this example to simply illustrate some of the capabilities of the framework. A full analysis is beyond the scope and page limitation of this paper.

The framework was tested using some of the data from a case study by Taylor et al. (2004). The case study addressed several problems in transportation, specifically driver dispatching and tour formation in full truckload trucking. The case study involved North American freight movements for JB Hunt (JBHT). The network was composed of 11 high freight density terminal cities within the JBHT terminal city network. The terminal city network and freight lanes (defined as a city-to-city pairing) used in the case study are illustrated in Figure 5.

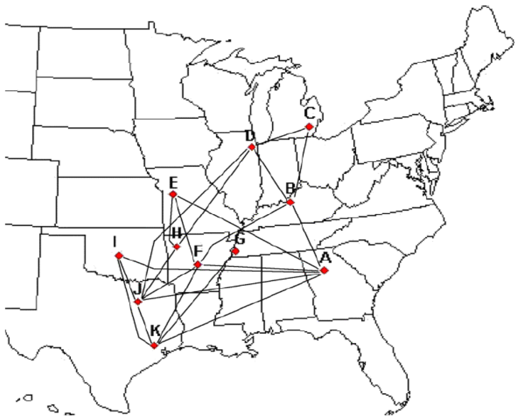


Figure 5: Terminal City Network (Taylor et al. (2004))

For this example, we assumed that the loading and unloading time distributions are constant with a value of 1 hour. In addition, we assumed that the allowable distance for deadheading and bobtailing was 1000 miles to make sure that the locations on all the lanes are included in the search for a driver. The time, at which the first load is generated, and the time between load generation was assumed to be exponential with the mean value of 1 hour. We assumed that the average speed of the driver was 60 miles/hour. The driver selection rule was based on selecting the driver which takes minimum expected time to deliver the load. The truck and trailer selection rule was based on selecting the first idle truck and trailer in the idle lists. The bobtail and deadhead location were selected as the ones which were closer to the origin within the defined range of 1000 miles (Petre's (2000) thesis was also based on selecting the deadhead or bobtail location within the user defined range). The origin probability for all the origins was calculated as the percentage of freight generated from each of the cities acting as an origin. The destination probability was calculated as the percentage of freight generated for all origin-destination pairs. The starting allocation of resources in the network is given in Table 3.

Table 3: Resource Distribution for Case Study

Region	City	#drivers	#trucks	#trailers
A	Atlanta	8	12	14
B	Louisville	7	11	13
C	Detroit	7	11	13
D	Chicago	8	12	14
E	Kansas City	6	10	12
F	Little Rock	10	14	16
G	Memphis	3	7	9
H	Lowell	7	11	13
I	OK City	6	10	12
J	Dallas	9	13	15
K	Houston	9	13	15

The framework collects summary statistics at the network, location and resource levels. These statistics include such quantities as: average number of loaded and unloaded miles (by driver, network, and sub-divided into deadhead/bobtail miles), resource utilization (driver, truck, trailer), average number of loads waiting at origins and in the network, average time spent waiting by loads, the average system time for loads, and the throughput of the network, etc.

We configured the model to run for 10 replications of 10000 time units (hours) with a warm up period of 500 hours. When possible we attempted to validate the results as compared to the results in Petre (2000). In our model, a driver can be in one of four states – Busy (on duty with task), Idle (on duty without task), off duty with task, and off duty without task. If we add up the results for all the drivers in these states, it should be close to the total number of drivers in the system. In our case, the drivers in these states add up close to the total number of drivers i.e. 80 (see Table 4). The sum of number of off duty drivers with and without task is close to the total number of drivers off duty.

In addition, the number of off duty drivers should be close to half of the total number of drivers because a driver's day consists of a maximum of 14 hours in a day or 11 hours if all the time was spent in driving. The number of idle drivers suggests that there shouldn't be many tasks lined up in the dispatch queue. The number of tasks in the dispatch queues as seen in Table 4 is very low. Out of 10000 loads generated in the system 9973 have been processed. Around 14 are still being processed as seen by the number of loads in IncomingLoadsQueue. The average time spent in the transport queue is expected to be around the time spent in loading which is around 1 hour, which it is.

Table 4: Summary Results for Test Case (cont.)

Time Weighted Average number of Busy Drivers	10.4880
Time Weighted Average number of Idle Drivers	27.7946
Time Weighted Average number of Off Duty Drivers (Average #Off Duty Drivers With Task + Average #Off Duty Drivers Without Task)	41.7173
Time Weighted Average number of Off Duty Drivers With Task	7.8755
Time Weighted Average number of Off Duty Drivers Without Task	33.8417
Time Weighted Average number of Busy Trucks	18.3636
Time Weighted Average number of Busy Trailers	18.3636
Time Weighted Average number in Dispatch Pick Up Queue	0.0908
Time Weighted Average number in Transport Pick Up Queue	1.568
Time Weighted Average number in BobTail Pick Up Queue	1.685
Time Weighted Average number in DeadHead Pick Up Queue	0.6766
Time Weighted Average number in Incoming Loads Queue	14.43
Average Time in Dispatch Pick Up Queue	0.090
Average Time in Transport Pick Up Queue	1.565
Average Time in BobTail Pick Up Queue	10.510
Average Time in DeadHead Pick Up Queue	10.959
Average Time in Incoming Loads Queue	14.405

Table 5: Sample Network Statistics

Average BobTail (unloaded) miles per Load	334.224
Average DeadHead (unloaded) miles per Load	351.620
Average Transport (loaded) miles per load	425.830
Average Total miles per load	1111.674
Average Load System Time	18.421
Total Number of Loads processed	9973.0

The illustrated statistics in Tables 4 and 5 are similar to those obtained in Petre (2000) when available. Although not reported here, the framework collects all standard statistical summary measures (e.g. average, standard deviation, half-width, minimum, maximum, etc) and can write the values to Excel or any database automatically.

4 SUMMARY AND FUTURE RESEARCH

The purpose of this research was to analyze and identify the fundamental elements necessary for modeling a generic Full Truck Load transportation network via simulation. We classified and organized the modeling elements into a coherent set of objects having attributes, behaviors, and inter-relationships to form a framework. We provided a standardized model of the object-oriented transportation framework using the UML for documentation and dissemination of the framework. We used the framework to simulate a realistic truckload network using the data from a realistic test case. The performance statistics indicate that the framework can simulate such realistic networks. Using this framework, the user can develop and simulate their own truckload networks. This research has provided a prototype by which additional transportation elements can be modeled.

Future work should involve the modeling of additional dispatching algorithms (e.g. based on a dynamic driver assignment mathematical model), the development of decision and selection rules based on customer due dates, an improvement in the underlying connection to spatial models (e.g. maps), the incorporation of cost models, the modeling of LTL networks, and the modeling of vehicle routing networks.

ACKNOWLEDGMENTS

This material is based upon work supported by the Air Force Office of Sponsored Research and the Air Force Research Laboratory. 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 Air Force.

REFERENCES

Booch, G., J. Rumbaugh, and I. Jacobson., 1999. *The Unified Modeling Language User Guide*. Addison-Wesley.

Chan, F. T. S. 2006. "Design and evaluation of a distribution network: a simulation approach", *International Journal of Advanced Manufacturing Technology*, 26, pp. 814-825.

Dalal, M. A., Bell, H., and Denzien, M. 2003. "Initializing a distribution supply chain simulation with live data", In *The Proceedings of the 2003 Winter Simulation Conference,* S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds., Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Ervin, E. C. and Harris, R. C. 2004. "Simulation analysis of truck driver scheduling rules", In *Proceedings of the 2004 Winter Simulation Conference*, R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds., Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Hamber, B. 2003. "TLOADS treatment of assigning an filling orders", in *Proceedings of the 2003 Winter Simulation Conference*, S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds., Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Ghiani, G., Laporte, G., and Musmanno, R. 1994. *Introduction to Logistics Systems Planning and Control*, Wiley Interscience Series in Systems and Optimization.

Manivannan, M. S. 1998. "Simulation of Logistics and Transportation Systems", Chapter 16 in the *Handbook of Simulation,* J. Banks ed., John-Wiley & Sons, Inc.

Petre, M. 2000. "Synergies between truckload and intermodal transportation", Master's Thesis, Department of Industrial Engineering, The University of Arkansas, Fayetteville, Arkansas.

Ratliff, H.D. and Nulty, W. G. 1996. "Logistics composite modeling", Technical White Paper Series of The Logistics Institute at Georgia Tech, http://tli.isye.gatech.edu/downloads/lcmwpaper.pdf [accessed March 3, 2007].

Rossetti, M. D. 2007. "JSL: An open source object-oriented framework for discrete-event simulation in Java", in preparation.

Rossetti, M., Miman, M., Varghese, V., and Xiang, Y. 2006. "An object-oriented framework for simulating multi-echelon inventory systems", In *Proceedings of the 2006 Winter Simulation Conference*, L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds., Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Taha, T. and Taylor, G. D. 1994. " An integrated modeling framework for evaluating hub-spoke networks in truckload trucking operations", *Logistics and Transportation Review*, 30, (2), 141-166.

Taha, T. 1992. *Simulation Modeling and SIMNET,* Prentice-Hall.

Taylor, D., Kutanoglu, E., and Tjokroamidjojo, D. 2004. "Efficient Dispatching in a Terminal City Network", Mack-Blackwell Rural Transportation Center Report, http://www.mackblackwell.org/web/research/all-projects.htm [accessed March 21, 2007]

YoungBlood, A. 2000. "Dispatching methodologies between terminal cities in a truckload trucking environment", Master's Thesis, Department of Industrial Engineering, University of Arkansas, Fayetteville, Arkansas.

AUTHOR BIOGRAPHIES

MANUEL D. ROSSETTI, Ph. D., P. E. 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. Dr. Rossetti has published over thirty-five journal and conference articles in the areas of transportation, manufacturing, health care and simulation and he has obtained over $1.5 million dollars in extra-mural research funding. His research interests include the design, analysis, and optimization of manufacturing, health care, and transportation systems using stochastic modeling, computer simulation, and artificial intelligence techniques. He was selected as a Lilly Teaching Fellow in 1997/98 and has been nominated three times for outstanding teaching awards. He is currently serving as Departmental ABET Coordinator. He serves as an Associate Editor for the International Journal of Modeling and Simulation and is active in IIE, INFORMS, and ASEE. He served as co-editor for the WSC 2004 conference. His email and web addresses are `<rossetti@uark.edu>` and `<www.uark.edu/~rossetti>`.

SHIKHA NANGIA, M.S.I.E. is currently working as a Functional Analyst for Manhattan Associates on their Wal-Mart project. She received her M.S. in Industrial Engineering from the University of Arkansas and a B.E. in Mechanical Engineering from Pune University, India. Her areas of interest include computer simulation, trucking transportation, and the design and analysis of supply chain systems.