Object oriented model for container terminal
distributed simulation

Maurizio Bielli a, Azedine Boulmakoul b,*, Mohamed Rida b

a Institute of Systems Analysis and Informatics, “Antonio Ruberti”—National Research Council, Viale Manzoni 30, 00185 Rome, Italy
b LIST Laboratory, Department of Computer Science, Mohammedia Faculty of Sciences and Technology (FSTM),
B.P. 146, Mohammedia 20650, Morocco

Available online 14 April 2005

Abstract

In shipping port sector, containers are the most dynamic and complex to manage. To provide adequate strategy for
the increasing traffic, ports must either expand facilities or improve efficiency of operations. In investigating ways in
which ports can improve efficiency, this paper outlines a container terminal simulation model and gives components
architecture that are implemented with Java. Simulator calibration and validation are also presented in the paper.
The main goal of the present work is to provide a help tool in a port decision support system. The object oriented soft-
ware design using UML diagrams is deployed in this project.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Object-oriented modeling; Discrete-event simulation; Container terminal; Simulator calibration and validation

1. Introduction

Every day containers arrive to a terminal by multiple means of transport and are stored in the
terminal area. Then, containers leave the terminal
by the same means to reach their final destinations
[2,5,6,21].

A container port, which provides the interface
between railroads, ocean-going ships, and over
the road trucks, represents a critical link in the
intermodal chain. The maritime container termi-
nals show a high integration level of the various
information systems and control engineering
applications in an overall port information system
architecture that include vessel traffic services, sea/
yard and freight station planning operations,
administrative/financial management, manage-
ment and control of handling activities, equipment
maintenance, etc. [1].
The amount of work a container port deals with depends on the quantity of containers in transit. The management of a container port is a complex process, which involves a high number of decisions. A valuable simulation tool for a port manager, then, would be one that predicts profit out of each decision. To develop a decision support system for container terminals, this paper presents an architecture that simulates container terminal operations in order to improve efficiency in container terminal management.

The main goals of our simulation software are:

- evaluation of alternative ships loading and unloading operations in term of time and cost,
- evaluation of several storage policies,
- evaluation of different resource allocation procedures.

The rest of this paper is organized as follows: In the following section we will provide an overview of the operations within container terminals. Section 3 presents the most known simulation techniques. Section 4, then, presents our object oriented modeling for container terminal described with several UML diagrams. Section 5 presents a description of container terminal simulator that we have developed. Section 6, then, gives the calibration and validation of this simulator. Section 7 concludes and incorporates suggestions to continue research on container terminal productivity.

2. Overview of container port operations

A container port is a terminal where containers enter and leave by different means of transport, as trucks, trains, and ships (input/output transport means) [2,5,7]. It provides the interface between railroad, ocean-going ships, and over road trucks, and presents the critical link in the intermodal chain. Containers arrive at terminal by trains, ships or trucks and are stored in the terminal yard. Then, they leave the terminal by the same means to reach their final destinations [9,15]. In this section we present an overview of container terminal operations.

2.1. Quay crane operations

A quay crane is a crane that services container-ship by shifting on a rail to reach the assigned stowage within the same ship and also to move from one ship to a successive ship once the first one has been completed (see Fig. 1). It provides, arguably, the single most important operation (called move) associated with loading and unloading a ship and represents the only means of moving containers to or from a ship [3]. To unload a ship, one or several (even up to 5 or 6 for largest containerships of latest generation) quay cranes pick up containers from the ship and put them on shuttle trucks that move them to the assigned yard positions within the terminal storage area. To load a ship the quay crane unloads a container from a shuttle and puts it on the ship. For both types of move (loading and unloading operation) a very restricted area for buffering some containers is naturally provided at the basis of each one of the on rail crane. Whenever a crane breaks down, its work is interrupted until it is repaired or until another crane is positioned in place of the former one, if possible. Depending on both the size of the ship (small feeder to large oceanic vessel) and the instantaneous availability at the terminal, a different number of cranes are, typically, working in parallel on the same ship at the same instant, provided that a minimal length in between is guaranteed to avoid arm collision.

Access into the ship is provided by a cable suspended carriage, shown in Fig. 1, specifically
designed to pick up and release containers from top corner castings. The carriage expands to accept both 20- and 40-ft containers. Containers of greater length, such as 48 and 52 ft, can be moved by most cranes, though older cranes may be limited by the clearance between the crane’s legs. The expansion or contraction of the container carriage can be done, with negligible delays, while the carriage is in motion. The container carriage is also used to move specialty containers or oversized cargo.

Containers stacked in a ship are secured in several ways in order to prevent them from being damaged at sea. Locking corner castings are placed between stacked containers in non-cellularized or ro/ro ships to align the containers and to provide a place to brace them. The cross braces are then secured to the floor of the ship, and, finally, the hatch covers are put back in place. Cellularized ships do not require corner castings or cross braces, since permanent guides and locks (which allow containers to be showed more densely than in non-cellularized cargo vessels) are already on board.

The delays created by bracing the container stacks are usually negligible, since most of the work can be completed while the crane is retrieving the next container. Noticeable delays occur only when corner castings or cross braces must be delivered from the ground to the longshoremen working in the ship.

Another activity that interrupts operations is the movement of the crane from one bay to another bay of a ship. Usually, quay cranes are rail mounted to allow movement laterally along the ship. The time spent moving a quay crane from one bay to the next is on the order of one container move, which ranges from 1 to 3 minutes. Another delay related to crane operations is that of hatch cover placement. Hatch covers are placed over the containers stacked in the holds of the ship. Thus, hatch covers form the decks of container ships, on which containers are stacked three or four high. To gain access to the holds of a ship in service, the supervisor of the operation will have the hatch covers removed and then placed on the ground directly behind the crane. This operation usually takes 5 minutes to complete, and occurs up to 12 or more times per ship, depending on the size of the ship.

Finally, the order or the sequence of the removal of the containers from a ship can occasionally cause delay for the quay cranes for two reasons: First, the quay crane may be required to make one or more container moves within the ship to uncover the desired container. This is known as a restow. The duration of the delay caused by a restow is determined by the number of restows required. Second, the sequence of container moves can have profound effects on the stability of the ship. Ships without the equipment for automatically monitoring displacement, stability, trim and heel pose a difficult problem for the crane operator when placing the carriage on the corner castings of the container. Thus, containers are normally handled sequentially from one side of the ship to the other and from one end to the other. This technique not only simplifies operations for the crane operator, but also minimizes the problem of keeping the ship level while it is being serviced.

2.2. Storage yard operations

Operations in the storage yard are more flexible than quay crane operations, in the sense that there are much more degrees of choice in operations management, due to the combinatorial explosion of the combined possibilities of

(i) identifying a given set of attributes;
(ii) then clustering containers that share some attributes and, finally
(iii) assigning containers to a dedicated sub-area within the yard storage area.

Under the goal of avoiding container’s overflow at selected sub-areas, and also changes of place (overhead moves called “housekeeping”) during the sojourn time of a container within the yard, the complete process by which containers may be moved and stored within the yard area must be represented accurately [4] by adequate tools capable of being adapted to different real frameworks. In a storage yard, yard gantry crane (see Fig. 2), top-pick loader, or straddle carriers are used to stack containers.
In container ports, stacking is the most common container storage method; in particular, straddle carriers are used in those ports where the availability of a “large” yard allows restricting the number of tires to two. More often, containers must be stacked to more than two tires. In this procedure, containers are stacked several levels deep with different types of containers and cargo are placed in specific areas of the storage yard. For example, containers destined for a particular ship are placed together. In the same way, specialty containers, empty containers, and port specific containers are stored in designated areas. Hazardous materials are typically stored away from the general cargo containers, as are flammable materials and refrigerated containers. Finally, with these subsections, 20- and 40-ft containers are separated.

Even with these many subdivisions, the efficiency of storage yard equipment is greatly increased by being able to service only one portion of the yard at a time. This efficiency is particularly desirable when yard cranes are employed as the primary storage method. Stacking requires that close attention be paid to the location, or address, of the container to prevent multiple restows or misplaced containers. Without efficient ways to assign container addresses, multiple restows are likely [15,21].

The container stacking procedure is carried out primarily by yard gantry cranes. The yard gantry cranes operate similarly to the quay cranes, in that a suspended container carriage is used to place and to retract containers. The yard gantry crane allows containers to be stacked three deep, the fourth row being reserved for clearance of another container. The clear span of the yard crane provides space beneath the crane for trucks to be serviced or queued.

There are two types of yard gantry cranes: rubber tire and rail mounted. Rubber tire gantry cranes ensure flexibility and mobility; they are able to move from one container bay to the next in a matter of minutes by traveling to the end of the bay and rotating all four tires in the desired direction. Because of the length of a container bay, it is important to minimize the time required to reach the end of the bay.

A rail mounted gantry crane operates in the same way as the rubber tire gantry crane, with the exception of the rail mounted gantry crane's inability to maneuver quickly from bay to bay. However, the higher stability of the rail mounted crane translates into higher productivity and a denser container stacking.

In a way similar to quay crane operations, containers are assigned specific addresses before entering the storage yard. The address is very important in minimizing the number of restows. Restowing in the storage yard may be slightly faster than in the ship because of the absence of corner castings or cross braces.

Another way to stack containers in the storage yard is through the use of straddle carriers. As the name implies, straddle carriers carry containers between their legs to the appropriate place in a storage yard bay. Containers are stacked three high so that there will be clearance for one loaded straddle carrier. The only space between the single container width bays is the space for the legs of the straddle carrier.

A fourth way to store containers in the storage yard is through the use of top-pick loaders. The top-pick loaders operate like a large fork lift and have been modified to pick up containers by the top corner castings. An additional modification is that the loaders are able to reach over one row of containers to place or to retrieve blocked containers. Bays are three containers wide so they can be serviced from either side. Note that more space is required between the bays for the operations of loaders than the operations of gantry cranes. This results in lower density container storage. The advantages of the top-pick loader over other stacking techniques include increased speed and maneuverability.

Finally, containers can be stacked with simple fork lifts. Typically used for empty containers, the fork lift provides excellent maneuverability,
but the fork lift cannot place one container behind another. For stability reasons, fork lifts are only able to stack containers three high. Often, fork lifts operate in storage yards as an accessory unit, retrieving empty containers or occasionally moving cargo into a ro/ro vessel.

It is important to note that storage yard delays can be caused by commercial vehicles. Because the storage yard is the interface of ocean and over the road carriers, the stacking equipment must service both commercial vehicles and yard vehicles. Port managers usually detail stacking machinery to servicing either the yard vehicles or commercial vehicles, but not both simultaneously. However, there are circumstances whereby stacking equipment is required to load or to unload both type of vehicles.

2.3. Shuttle truck operations

The third element of port operations presented in this paper is the movement of containers between quay cranes and storage yard. Quay crane unloads ships and places containers on shuttle trucks, which move them to storage locations in the yard. This operation forms a closed loop that is traveled by shuttles servicing a ship (see Fig. 3). Containers, which are stored in the storage yard, leave the terminal by input/output trucks to reach their final destinations (see Fig. 4). Productivity of containership’s journey depends on efficiency of transport operations between storage yard and quay crane. For example, too many trucks in the system cause long files at the cranes and long waiting times for service. Conversely, few trucks in the system will result in idle stacking equipment. Hence, a good management of container locations and stacking operations at yard storage area together with an optimal assignment of the right number of vehicles to quay cranes (and, in large ports, routes to vehicles) are crucial to avoid both phenomena of crane starvation and, at the opposite side, congestion of vehicles upon their arrival to both quay and/or yard cranes.

A collection of shuttle trucks, called gang, services each ship in the cyclic fashion described above. Each has 6–8 members, depending on several operating characteristics such as the distance that containers are carried from quay crane, and the type of yard storage method employed. Because of the high cost of keeping a ship in port, it is important to keep the quay crane operating without delay in order to turn the ship around as quickly as possible. This is normally done by keeping enough shuttle trucks in the gang so that at
least one vehicle is ready for service at the quay crane. But one must pay a lot of care in some form of dynamic reassignment of shuttles to cranes in order to get a satisfactory compromise between the above-mentioned phenomena of cranes starvation and vehicles congestion. One gang is assigned to each quay crane servicing the ship. If yard cranes are employed in the storage yard, the same gang will be assigned to one or two yard cranes. Thus, the gang operates to ensure the movement of containers between the yard and the quay crane. If containers are stacked by top-pick loaders, or if chassis storage exists, the gang members will be required to drive to the appropriate storage location, not necessarily in the same area of the storage yard.

Occasionally, the productivity of shuttling containers from the quay crane to the storage yard can be increased in several ways: First, truck may be used to move two 20-ft containers at the same time. At the yard or quay crane, the first container is placed at the front of the chassis, and the second is placed on the back of the same chassis. While the service time underneath the crane is lengthened, productivity is increased significantly. Double moves are possible for 20-ft containers. Because a ship may carry a limited number of 20-ft containers, double moves can be sustained for only a short period of time. The second form of double moves occurs when a quay crane, nearing completion of the removal of import containers from a hold, prepares to reverse the process by loading export containers. During that short interval, a truck can transport the imported container into the storage yard, pick up an export container, and deliver it back to the quay crane. Again, productivity increases temporarily, though this type of double move is rare.

Delays caused by the movement of containers are usually negligible, because most delays are rooted at a crane or stacking vehicle. Exceptions include mechanical breakdowns and traveling to the wrong place in the storage yard. Another delay is caused by port congestion, owing to the large number of trucks present. Port congestion occurs frequently when several ships are in port or when two cranes are simultaneously servicing the same ship.

3. Simulation techniques

3.1. Discrete-event driven simulation

A discrete-event driven simulation [3,8,11] is a popular simulation technique, which is applicable to a large variety of problems in the real world. Discrete-event simulation has been done in a sequential manner. A variable clock holds time up to which the system has been simulated. A data structure, called the event list, maintains a set of messages, with their associated times of transmission, that are scheduled for the future. Each of these messages can be sent at associated time in the system. At each step, the message with the smallest associated future time is removed from
the event list and the transmission of the corresponding message in the system is simulated. Sending this message may, in turn, cause other messages to be sent in the future (which are added to the event list) or cause previously scheduled messages to be canceled (which are removed from the event list).

3.2. Distributed discrete-event simulation

This type of simulation was proposed as an alternative when the number of events to be simulated is high [11]. It offers a radically different approach to simulation, and consists of partitioning the simulation problem on a set of processes executed in parallel, with each process carried out on a machine. An algorithm was suggested to be implemented in each machine in order to simulate a single physical process; messages in the physical system were simulated by message transmission among the machines.

Each process operates autonomously to change its state and interacts with other processes in the system. The interaction is made by messages. Contents of a message sent by a process depend on the characteristics of the process (its initial state, its rules of operation) and the messages that the process has received so far. Our approach consists to apply this type of simulation to container terminal using the multithreaded programming in Java.

In this section we introduce a model of distributed computation and show how a simulation may be carried out by a set of communicating processes. We limit our discussion here to a basic scheme, one which can result in deadlock. More sophisticated schemes that resolve deadlock are discussed in [11].

3.2.1. Basic scheme for distributed simulation

A distributed system consists of a finite number of processes and directed channels connecting some pairs of processes called Logical Processes (LP) [8,11]. Each LP may execute sequential code and two special commands: receive and send. In a send, an LP names an outgoing channel and a message that is to be sent along that channel. Execution of the send results in the message being deposited on the named outgoing channel; the sender then proceeds with the execution of its code. Messages sent along a channel are delivered in the sequence in which they are sent. In a receive command, an LP names one or more incoming channels from anyone of which it wishes to receive a message. An LP wishing to receive may have to wait until a message arrives along one of the incoming channels. Note that the communication protocol is simple and can be implemented on many existing machine architectures.

A set of Logical Processes D is deadlocked at some point in the computation if all of the following conditions hold:

(1) Every LP in D is either waiting to receive or is terminated.
(2) At least one LP in D is waiting to receive.
(3) For any LP_a in D that is waiting to receive from some LP_b, LP_b is also in D, and there is no message in transit from LP_b to LP_a.

It follows then that none of the Logical Processes in D will carry out any further computation since they will remain waiting for each other.

To simulate any given physical system, one constructs a distributed logical system as follows. One will associate one Logical Process with each Physical Process (PP), LP_a will simulate the actions of PP_a. If PP_a can send messages to PP_b, there is a channel from LP_a to LP_b.

An LP can simulate the actions of a PP up to time t if the LP knows the initial state and all messages that the corresponding PP receives up to time t. This is because no future message can affect the PP’s behavior at t. One notes that an LP may be able to simulate a PP beyond time t, even though it knows its input messages only up to t [11].

We make a chronology requirement: if an LP sends a sequence of messages \{..., (t_i, m_i), (t_{i+1}, m_{i+1}), ...\} to another LP, then \( t_i < t_{i+1} \). We note \( T_i = \min_i(t_i) \). The basic distributed simulation algorithm for an LP_t is given as follows [11]:

**Initialize:** \( T_i = 0 \) {All messages received by PP_i up to \( T_i \) are now known to LP_i}
While simulation completion criterion is not met do

{Simulate PP, up to } T_i by doing the following:

for each outgoing channel, compute the sequence of messages \((t_1, m_1), (t_2, m_2), \ldots, (t_r, m_r)\) where \(t_1 < t_2 < \cdots < t_r\) and, PP, ends \(m_i\) at time \(t_i\) along this channel; send each message in sequence along the appropriate channel;

{Note: All messages sent by PP, up to } T_i can be deduced by LP, and sent; also some messages to be sent beyond } T_i may be predicted by LP, and sent. Only new messages that have not been sent before are sent. Also note that some or all of these message sequences may be empty

{Receive messages and update } T_i until } T_i changes value: \(T'_i := T_i\);

While \(T'_i := T_i\) do

Wait to receive messages along all incoming channels;

Upon receipt of a message, update LP,’s internal state and recomputed \(T_i\), the minimum over all incoming channel clock values

End While

End While

4. Container terminal logical view

Recent developments such as object-oriented modeling and Petri-net based simulation were reported by researchers in the field of construction simulation [20]. In this section we present our analysis of container terminal system, based on object-oriented paradigm. Fundamental classes are identified and different diagrams describing the system are presented. Such diagrams are denoted using the UML notations.

4.1. Class diagram

The first step of our conception process consists of identifying classes of the container terminal model. We want to describe these classes in a formal manner and implement them with Java. UML allows, with the help of class diagram, to model container terminal system components and their relationships. In Fig. 5, we report a part of the hierarchy of these components. The class TerminalModel, which represents container terminal in our system modeling, aggregates multiple objects of classes: Quay, TransMean, Crane, Queue, ArrivalsGenerator, and Area, since all these classes represent terminal components. Each of these classes is a super class of the others in the system. In Table 1, we give an explanation about each class.

4.2. Multi-process

Objects of the real world carry out their operations in an independent and a simultaneous manner. In the implementation of our software we used Java [17,18], which is a multithreading language that makes simultaneous activities implementation easy. UML provides also tools destined to the conception of simultaneous models. In this section, we analyze profits that extricate our simulation of the multi-process.

4.2.1. Threads and active classes

Java uses threads to represent simultaneous and independent activities. The UML provides in this context active class notion to represent a thread. In our case, all generators, transport means, and cranes, are considered as active classes, since their objects must work simultaneously and independently. For example, Shuttles move between a storage area and a quay while a QuayCrane loads a container from a ship. We report, in Fig. 6, the main classes that perform simulation of loading and unloading process. The class ShipArrivalsGenerator is associated to class Ship. The association is named “Create”; it indicates that ShipArrivalsGenerator generates ships arrivals. Each of classes Ship, and Shuttle, is associated to a subclass of Queue (ShipQueue, QuayQueue, or YardQueue), which means all these means of transport may wait in a queue for service. Yard crane and quay crane load and unload shuttle trucks and corre-
sponding objects are therefore related to class Shuttle. Finally, classes that possess a thick are active classes, since their objects must run in an independent and simultaneous manner.

4.3. State and activity diagrams

To model a discrete-event dynamic system we need to take into account its states and events.
leading to the state evolutions [3,9,14]. To model the dynamics of the container terminal system, we used state and activity diagrams. Figs. 7–9 describe states and activities of ships, cranes, and shuttle trucks in the terminal. In these networks, scripts concerning all states and activities are designed.

4.4. Collaboration diagrams

UML collaboration diagrams are used to explore the dynamic nature of our software. Collaboration diagrams show message flow between objects in an object oriented application, and also imply basic associations (relationships) between classes. Fig. 10 shows a collaboration diagram that represents interactions between QuayCrane and QuayQueue objects while objects of the class Shuttle arrive for QuayCrane service.

It is a common practice to indicate concurrent threads in an UML collaboration diagram by preceding sequence number of messages by letters. In our case, in Figs. 10 and 11 some messages are followed by letters indicating that those messages are being processed concurrently and asynchronously. For example, a and b messages share the same sequence number 1. A ship that arrives to terminal must wait in a queue (which is modeled by ShipQueue) when berth is not available. When a quay is free, the first ship in queue occupies this quay. Table 2 reports all messages of the collaboration diagram presented in Figs. 10 and 11.

4.5. Events management

As usual, objects do not execute their operations in spontaneous way. A specific operation is, generally, invoked when a sender object (a client object) sends a message to a receiver object (a ser-
ver object) asking it to execute a specific operation. We began system modeling with state-transition and activity diagrams, as well as collaboration diagrams. In this section, we analyze interactions between objects of the system.

4.5.1. Events

In Fig. 10, we presented the example of a QuayCrane object that loads a Shuttle object by sending to it a message LoadShuttle. It means that the QuayCrane object calls LoadShuttle method of

![Activity diagram for a QuayCrane object.](image)

![Activity diagram for a Shuttle object.](image)
the Shuttle object. This message describes an action that occurs currently (crane load a shuttle). Generally, the name structure of a message is formed by a verb that precedes a name. Thus, the message name LoadShuttle is constituted by the verb “Load” followed by the name “Shuttle”.

An event is an object notification of an action occurrence. For example, a Ship sends an event ShipArrived to a ShipQueue when the Ship arrives at terminal. Listening to an event ShipArrived permits ShipQueue to determine actions that must be accomplished when the Ship arrives.

The name structure of an event is the inverse of the first structure. The event name is formed by a name, followed by a verb. For example, the event name ShipArrived is constituted by a name “Ship” followed by “Arrived” (see Table 2).

In our case, we create a super class called TerminalModelEvent that represents an event of our model. The TerminalModelEvent contains a reference of a Location, which represents the place where the event has been generated, and a reference of an Object, which represents the source of the event (see Fig. 12). In the simulation, objects
use instances of `TerminalModelEvent` to send events to the other objects. For example, `Ship` sends a `TerminalModelEvent` to a `QuayQueue` when it arrives to terminal. To describe all actions in the system, we created several subclasses of `TerminalModelEvent`, as it is presented in Fig. 12. Thus, instead of `TerminalModelEvent`, `Ship` sends `ShipEvent` to `ShipQueue` when it arrives to terminal. In Table 3, we present a part of actions that create events in the system. Note that each action is constituted by a “name” followed by a “verb”.

### 4.6. Event management in Java

In Java [12,14], the concept of event management is like collaborations described in Section 4.4. An object sends a message to other objects, which are listening to this type of message [15,19]. The difference is that objects must be registered to receive the message. Therefore, these objects are qualified to be event listeners. To send an event, the sender object calls a particular method of the receiver object and gets to it in parameter the event object wished. In our simulation, this event object is a subclass of `TerminalModelEvent`. In Fig. 13, we propose a modified diagram taking account of the event management. Fig. 14 presents a class diagram, which illustrates the realization between `QuayQueue` and `ShipEventListener`.

#### 4.6.1. Event listeners

We illustrated the management of events between classes in the simulator with the help of modified collaboration diagram presented in Fig. 13. The `Ship` object sends an event `ShipEvent` to the `ShipQueue` object (message 1). Then, we must specify the event object that the `Ship` must transmit to the `ShipQueue`. According to the note presented in the Figure, the `Ship` passes a `TerminalModelEvent`
object to \textit{ShipArrived} method, which is in fact a \textit{ShipEvent} object. The \textit{ShipQueue} implements an interface \textit{ShipEventListener} that “listens” \textit{ShipEvent} occurrences, what makes \textit{ShipQueue} an \textit{EventListener} (see Fig. 14). The interface \textit{ShipEventListener} provides, among others, a method \textit{ShipArrived} that permits to the \textit{Ship} to notify \textit{ShipEventListener} when it arrives to the terminal.

A ship arriving to the terminal calls only methods declared in the interface \textit{ShipEventListener}, but only when \textit{ShipQueue} is registered to this \textit{Ship} to receive \textit{ShipEvent} events [11]. In Fig. 15, we illustrate the elided class diagram representing several realizations in our simulation model. The class \textit{TerminalModel} implements all methods of the interface \textit{TerminalModelListener}; we touch up this diagram to take into account the fact that the class \textit{TerminalModel} implements all interfaces by inheriting \textit{TerminalModelListener} from these interfaces. Thus, the class \textit{TerminalModel} can receive all events of the system.

4.7. Model-view-controller

Design patterns define efficient strategy for building reliable object oriented software systems

<table>
<thead>
<tr>
<th>Event</th>
<th>Sent when (action)</th>
<th>Sent by an object of the class</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{ShipEvent}</td>
<td>a ship arrives to a point in the terminal</td>
<td>\textit{Ship}</td>
</tr>
<tr>
<td></td>
<td>a ship leaves the terminal</td>
<td>\textit{Ship}</td>
</tr>
<tr>
<td>\textit{ShuttleEvent}</td>
<td>a shuttle arrives to a point in the terminal</td>
<td>\textit{Shuttle}</td>
</tr>
<tr>
<td>\textit{QuayEvent}</td>
<td>a quay is free</td>
<td>\textit{Quay}</td>
</tr>
<tr>
<td></td>
<td>a quay is busy</td>
<td>\textit{Quay}</td>
</tr>
<tr>
<td>\textit{QuayCraneEvent}</td>
<td>a quay crane is idle</td>
<td>\textit{QuayCrane}</td>
</tr>
<tr>
<td></td>
<td>a quay crane is busy</td>
<td>\textit{QuayCrane}</td>
</tr>
</tbody>
</table>

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
Event & Sent when (action) & Sent by an object of the class \\
\hline
\textit{ShipEvent} & a ship arrives to a point in the terminal & \textit{Ship} \\
& a ship leaves the terminal & \textit{Ship} \\
\textit{ShuttleEvent} & a shuttle arrives to a point in the terminal & \textit{Shuttle} \\
\textit{QuayEvent} & a quay is free & \textit{Quay} \\
& a quay is busy & \textit{Quay} \\
\textit{QuayCraneEvent} & a quay crane is idle & \textit{QuayCrane} \\
& a quay crane is busy & \textit{QuayCrane} \\
\hline
\end{tabular}
\caption{Representation of actions that create events in the system}
\end{table}
Our simulator supports Model-View-Controller (MVC) architecture, which uses several design patterns. MVC architecture divides the system into three parts:

- the model, which contains the data and logic of our program;
- the view, which provides a visualization of our model;
- the controller, which defines the system behavior and integrates the inputs to the model.

The controller permits a user to modify the data of the model. Then, the model informs the view about changes. The view changes visualization according to inputs of the model. We applied MVC architecture to our container terminal simulator. Fig. 16 presents a more elevated level class diagram of the simulation. The class TerminalSimulator, which is a subclass of Javax.swing.JFrame, includes an object of each of classes TerminalModel, TerminalView, and TerminalController, to create the application TerminalSimulator. The class TerminalView implements TerminalModelListener, which implements all the interfaces of our simulation. Therefore, TerminalView can receive all events of the model for visualizing the simulation. The TerminalSimulator class does not contain any other attribute, its unique behavior is to start the program. Therefore, in Java, the class TerminalSimulator must contain a static method main to call the constructor, in order to create objects TerminalModel, TerminalView, and TerminalController.

5. Container terminal simulator

The simulator is used as a test bench to evaluate management policies produced by the optimization modules [2,5], and it is charged with a realistic reproduction of the activities and flows that occur inside the terminal [10,15,21]. It allows managers and engineers to experiment and compare different policies and techniques before their implementation. It provides also a graphical interface in order
to have easy access to the current state of the simulated terminal and to simulate particular events. Performances of various management policies can be compared according to indicators [3,16,21]. These indicators must take into account the cost of cranes and operators during the various work shifts, penalty to be paid to shipping company if ship departure is delayed and the income generated by each container loaded and unloaded from a ship. Besides these economical indicators, the simulator allows to assess the resource utilization and to measure congestion indicators such as the average queue length of operations on terminal cranes.

An important application of the simulator is related to the problem of estimating cost functions and measures of performance not easily computable. The characteristics of the container terminals imply that the real cost of each container displacement is a complex function of several factors, whose influence has to be accurately investigated. Therefore, a simulation tool able to represent different scenarios is indispensable for a correct modeling of the problem. In our project, partially described in this paper, objectives and finalities fixed in [2,20] are also projected. The software components that we developed allow the deployment of the simulator to reach such objectives. Another important goal is the ability to generate operational indicators of container terminal, such as

- the global productivity
  \[
  \frac{\text{total number of containers moved by crane}}{\text{total elapsed vessel service time}}
  \]

- the net productivity
  \[
  \frac{\text{total number of containers moved by crane}}{\text{total time spent by crane servicing vessel}}
  \]

- quay crane and yard crane utilization index
  \[
  \frac{\text{travel time} + \text{working time}}{\text{travel time} + \text{working time} + \text{waiting time}}
  \]

- shuttle utilization index
  \[
  \frac{\text{travel time}}{\text{travel time} + \text{waiting time}}
  \]

- container yard occupancy rate
  \[
  \frac{\text{occupied space per unit time}}{\text{total capacity}}
  \]

- average ship waiting time
  \[
  \frac{\text{total waiting time of ships}}{\text{number of ships berthed}}
  \]
Starting from current terminal configuration in term of occupancy level, the expected plan of ship arrivals (Fig. 17), predictions from the forecasting system in term of expected import/export flows, and a storage policy based on some reservation/allocation criteria for all the different areas inside the terminal (Fig. 18), we run the simulation in order to produce some possible final terminal states. In this step we can make an evaluation of the policy adopted in the experiment. Our cost function is
based on different factors such as the ratio between import areas and export areas, the violation of the storage criteria and some performance indexes of the terminal equipments.

In Fig. 19, we report a typical screen-shot of the terminal during the simulation. A ship is moored and it is being unloaded by three quay cranes. Containers are to be moved and positioned on a yard area. In Fig. 20, a histogram that is updated on-line is reported. This histogram shows the number of shuttle-trucks that have waited in the quay queue. It is important to reduce the average of queue length under cranes: a better balance in the crane usage reduces the possibility of having lengthy queues. The application of computer generated management policies could improve the terminal performance, making possible the allocation of fewer resources, thanks to a better usage of the cranes. In Fig. 21, we present the quay crane utilization index calculated during unloading operations. The queue length of shuttle trucks underneath a quay crane during loading operations is reported in Fig. 22. Analogous results are produced for yard cranes. In Table 4, some operational indicators, averaged over the number of cranes and worked shifts, are presented.

Clearly, the reproduction of several sample paths of the state variables of our interest to carry out a statistical analysis of the simulation output data is required, to achieve some significant insights upon the system behavior, or, in other words, upon the goodness of some implemented
policies of resources dimensioning and management.

6. Simulator calibration and validation

Calibration and validation play a very important role since the terminal operators must actively co-operate with the simulation specialists in order to gain confidence in the model results. Advanced graphical interfaces and virtual reality can also be useful to provide immediate insight into the model workings even to the operators with scarce technical background [13].

Once the simulator is available, it must be calibrated and validated in order to verify its capability of reproducing the real terminal behavior. In this phase, the simulation module evolves using the same inputs and policies used by the terminal management over the calibration and validation periods. In general, calibration means tuning the simulator parameters in order to match as close as possible the simulation outputs with the data measured in the real terminal over a given time interval. The validation phase ensures that the result of the calibration is such that the simulator reproduces the reality under different conditions. In this section, we give a short example of calibration and validation of our simulator. All the data used in the present paper are taken from the database of the Casablanca container terminal in Morocco. The focus is on the period of two weeks, from 5/11/2004 to 5/24/2004. The data describe the activity of Casablanca container terminal in great detail, in such a way that every container movement can be found. The database reports also the resources that move container, the time of the operation and the origin and destination of the movement, described by three coordinates: bay, row, and tier in the terminal yard. The database tracks also the activity of every transport means that enters and leaves the terminal (trucks, ships, end trains). From these data, it is considered the period that start 1 am on the 5/11/2004 and ends after 10 shifts of 6 hours each. The advantage of such a choice is that no ships are at the terminal at the selected starting time, and the system is therefore in a natural initial empty state; finding the initial empty state is often one of the major problems for the simulation practitioner. The four shifts constitute the calibration period and the remaining shifts (from 5 to 10) are used for validation. The calibration parameters are

- the mean time $\mu_{qc}$ needed to move a container from a ship to the shuttle truck for each quay crane and vice versa,
- the mean time $\mu_{yc}$ needed to move a container from a yard position to the shuttle truck waiting for service for each yard crane and vice versa,
- the mean time $\mu_{st}$ of travel of a shuttle truck from point to point.

The arrival dates of ships, trains and trucks are deterministic (trace driven simulation) since they are read from the Casablanca container terminal database. Note that a truck carries at most two containers, while trains bring an average of 50 containers and ships vary from hundreds to thousands of containers. Their identification numbers are read from the database and then simulation evolves loading and unloading these transport means.

**Table 4**

<table>
<thead>
<tr>
<th></th>
<th>The average of utilization index (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shuttle trucks</td>
<td>37</td>
</tr>
<tr>
<td>Yard cranes</td>
<td>82.5</td>
</tr>
<tr>
<td>Quay cranes</td>
<td>73.2</td>
</tr>
</tbody>
</table>

![Fig. 22. Queue length of shuttle trucks waiting for quay crane service, during loading operations.](image-url)
A set of 10 experiments (simulations) was performed. Each experiment is identified by a different combination of the three calibration parameters. Table 5 reports real data with the simulated results for a quay crane (QC1). Only two experiments are presented in the table.

The simulation combinations correspond to the values of the parameters reported in Table 6.

Analogous results are produced for the yard cranes. In order to establish a ranking among different parameters combinations, we calculated the error measure (Err) that is based on the distance between the real number of containers moved by each crane and the simulated number, in each shift. This error is then normalized with respect to the maximum number of containers which can be moved by a crane, and averaged over all the quay cranes and over the number of worked shifts [16]. The best set of parameters is found at the value Err = 0.13, corresponding to experiment “simulation1”. The interpretation of such a value is that, on average, the simulator moves a number of containers (per shift and per quay crane) that is 13% far from the real one, as compared to the maximum possible distance (i.e., 100% error).

The validation of this model returns Err = 0.20. Table 7 reports the results for the validation of an other quay crane (QC2).

The results deserve a few words of comment, since they seem susceptible to improvements. An important remark is that our objective is to obtain a model of the terminal to test alternative scheduling policies. At this stage of research we want to assess the policies where the only bottlenecks can be caused by resource competitions on the yard that is by creation of lengthy queues under the yard or quay cranes.

Up to that point, the focus is on the terminal model which is a reasonable compromise between the ideal situation, where all cranes work in good conditions, and the worst case scenario, where cranes and operators show marked drops in their performances.

7. Conclusion

This paper represented a part of our research work aiming at improving management practices of a container terminal. In this paper we have presented the simulator that we have developed. The proposed model is not exhaustive, other factors and other models are to be integrated in future. In the present phase of the project, the specifications of the simulator are calibrated and validated. The deployment of the simulation on terminal site is foreseen in the project. Using the simulation model of the terminal, we evaluated policies generated by optimization algorithms. Here, the simulation acts as a test bench to assess effectiveness and robustness of policies in such a non-deterministic environment. The web-based information technology has to be considered to improve interoperability between the simulator and all the port.
information system components. Simulation tools, such as the one described in this paper, will surely contribute to the improvement of internal operations of container terminals.

References


