Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

HYBRID SIMULATION MODELING FORMALISM VIA O²DES FRAMEWORK FOR MEGA CONTAINER TERMINALS

Haobin Li
Xinhu Cao
Ek Peng Chew
Kok Choon Tan
Kaustav Kundu
Hongdan Chen

Centre of Excellence in Modelling and Simulation for Next Generation Ports
Department of Industrial Systems Engineering and Management
National University of Singapore
Innovation 4.0 Building 03-01, 3 Research Link
Singapore, 117602, SINGAPORE

ABSTRACT

This paper briefly introduces the hybrid simulation modeling formalism via O²DES Framework (object-oriented discrete event simulation), and its application in the modeling of mega container ports. From an object-oriented perspective, this paper first lists the entities involved in container ports and the relationships among them; based on it, the "event-based" modeling method is illustrated for its necessity in accurately describing the rules of discrete-event systems, with the case of quay cranes and AGVs interacting with handshakes; then, the hierarchical structure of the model is segmented with a "state-based" modular approach to simplify the process of modeling, as well as the maintenance and reusability of model library; finally, through the "activity-based" perspective, it provides an macro-level overview on the dynamics of the flowing entities in the system. This paper elaborates the connection and cooperation between multiple methods in the hybrid formalism, hoping to provide guidance for future modeling effort.

1 INTRODUCTION

In general, discrete event simulation is a basic perspective for modeling and analyzing dynamic systems. This perspective allows us to describe the dynamics of a system as the cumulative effect of a sequence of discrete events. These events are causally linked to each other by triggering relationships, but are spread out on a timeline with no fixed time intervals between them. Compared to other modeling perspectives, such as continuous-time and discrete-time frameworks, modeling man-made industrial systems using the perspective of discrete events is particularly effective. This is because these systems are essentially coordinated and operated based on discrete operating rules and protocols, and a discrete event framework is able to characterize these key operating rules. Besides, such a framework greatly improves the computational efficiency of simulation models by passing over redundant system state updates, which advance at a fixed increment time in the other two frameworks. Container terminals are a typical example of this "rule-based system". Therefore, for container terminals, discrete event simulation is the most appropriate simulation modeling method.

However, the modeling perspective of discrete events can be subdivided into multiple branches, which we call "modeling formalisms". For example, the "event-based" formalism, which starts with the identification of events, describes the relationship between these events and the impact on state variables; the "state-based"

formalism takes the unit structure and hierarchical relationship of state variables as the focus to describe the numerical changes within the subsystem and the interaction relationship between the subsystems; and the "activity-based" formalism takes the flowing entity as the main perspective and presents the whole system by tracking the activity flow of the entities in the system. Each "modeling formalism" can be expressed using one of the specifications, which generally include graphical specification, algebraic specification, and tabular specification. In particular, graphical specification, being the most intuitive one, is widely adopted in the practice of simulation and modelling.

Each of the above "modelling formalisms" can be used to describe the same discrete event system. However, each formalism may exhibit different pros and cons in different systems based on the system nature of the system. For example:

- The "event-based" formalism can accurately describe the state changes at different points in time in discrete event systems, but for large-scale systems with a large number of events, to model every event could be cumbersome.
- The "state-based" formalism can divide large-scale system structure in layers, reveal the static organizational structure inside, and modularize each subsystem model for reuse and replacement purpose, improving the efficiency of modeling process. However, each partition of a module generates several new event interfaces, duplicates the event in the partitioned modules, or creates redundant state variables in the modules if to reduce the number of interface events that are triggered. In this way, the run efficiency of the model is reduced in both time and space.
- The "activity-based" formalism, which takes the most intuitive approach, illustrates changes in the dynamic system from the perspective of the flowing entity, and through describing the time intervals between the different stages that the entity goes through in the system. This method is easiest for observation. Therefore, in some literature discussions, people often interpret "discrete event simulation" directly as an "activity-based" form of simulation modeling, and thus derive many. However, the "activity" or "process" described is a time interval rather than an accurate point in time, thus there might be ambiguity in the process understanding. In addition, different "activity-based" methods will adopt different methodologies for the differentiation between "flowing entities" and "resource entities", and for handling the changes in the type of "entities" in systems such as splitting, assembly, loading and unloading. In all, there is a lack of consistency.

When solving the problem of simulation modeling of mega container terminals, we found that the accuracy of the model, the modeling efficiency and the run efficiency, as well as the intuitiveness of the model for the modeler, port manager and operators, are all substantial considerations. We wanted to find a way to bring together the advantages of the above formalisms of discrete event simulation modeling, complement each other, and avoid their shortcomings.

Therefore, we chose the object-oriented discrete event simulation framework to implement such a hybrid modeling form to meet our needs for the simulation modeling of mega container terminals.

In simple terms, we use object-oriented entity relationships to parse the internal structure of a complex system, which includes static configuration parameters and dynamic state variables. On top of this, we use an "event-based" approach as the core of the dynamic advance of the simulation system, so as to accurately and unambiguously describe the operating rules of the system. At the same time, in view of the major hierarchical structure of port management and operations, as well as the multiplexing relationship of simulation elements, we use the "state-based" formalism to modularize these events, configuration parameters, and state variables, and clearly define the event interface criteria in between. In the process, we introduce an "activity-based" perspective, distinguishing the interaction between modules into two types, "material flow" and "information flow" of the entities, so the integrated modules can show the full flow of each, which include that of ships, vehicles, containers, yard cranes, and quay cranes, etc., as well as the interaction between them triggered by information flow.

In this article, we will illustrate the above four aspects of the hybrid modeling formalism, using specific examples from mega container terminal operations. Due to space limitations and intellectual property concerns, we are unable to provide full details of the model. However, we hope that through this explanation, we can bring some inspiration to the simulation modeling work in related fields.

2 METHODOLOGIES

2.1 Entity Relationship Structure

For simulation modeling process, we first need to define the scope of the system. Then we need to answer these questions accurately: which components need to be included in the scope of the model, how these components are connected, and what properties do these components and their connections have. To answer these questions, we introduce "entity-relationship diagrams" as the primary tool for analyzing and defining the structure of simulation systems.

"Entity Relationship Diagram" is a graphical specification first used in software engineering and object-oriented programming methods to represent various object entities involved in a software program and the associations between them. This is a very efficient expression language. Various elements in the real system do not exist independently, and the connections between them are rarely in a flat or multi-dimensional grid. Such a structure, using purely simple algebraic language, such as variables, sets, subscripts, or simple data structures, such as arrays, lists, matrices, is difficult to express intuitively. However, in the entity-relationship diagram, simple graphic symbols can be used to enumerate components as object entities, and display the associations between them in a flexible and intuitive way, and on this basis, it defines the attributes of each object entity.

For example, in Figure 1, we illustrate the object entities that may be involved in the container yard model, using a simplified entity-relationship diagram as an example. In order to make our description short and digestible, several complex entities and relationships are deliberately omitted here, and the detailed attributes of each entity are also hidden in this figure to allow our presentation to be more focused. Furthermore, this diagram demonstrates only one way of understanding the container terminal system out of the many. Here, we use this diagram to focus on explaining how Entity Relationship Diagrams are used. In practice, necessary changes can be made to this diagram according to actual needs or a deeper understanding of the container terminal.

In this diagram, we can observe that the entire container terminal entity is related to the quay side, the yard area and the transportation network, which are three major entities. Each of them is associated with several smaller entities, including port equipment, such as quay cranes, yard cranes, AGVs, as well as geographical areas with certain functions, such as wharf line, AGV path, ground slot, apron parking lot, yard working lot, etc. Besides, Entity-relationship diagrams also contain some flowing entities, such as ships, containers, jobs, and so on.

The cardinality of the relationship between entities can be clearly indicated in the entity-relationship diagram. For example, the relationship between quay side and wharf line is "Compulsory one to many". This means that the quay side may contain multiple and at least one wharf line, and the quay side of the entire terminal is only considered a single entity. Another example: the cardinality of the relationship between vessel and quay crane is "Optional-One to Optional Many". For a single vessel, multiple quay cranes can be allocated to handle the loading and unloading of containers at the same time; however, one quay crane can only be assigned to one vessel for operation at one time point. These specify the multiplicity aspect "one" and "many". Furthermore, before a ship enters the port, there is no allocated quay crane; correspondingly, after the quay crane completes the task of the previous vessel, it does not need to be allocated to a new vessel immediately. These constitute the "optional" aspect.

Based on the characteristics of discrete event simulation, we also made some modifications to the representation of entity-relationship diagram. Different from general object-oriented programming, in simulation, it is necessary to distinguish between those "dynamic entities" that can be generated or terminated

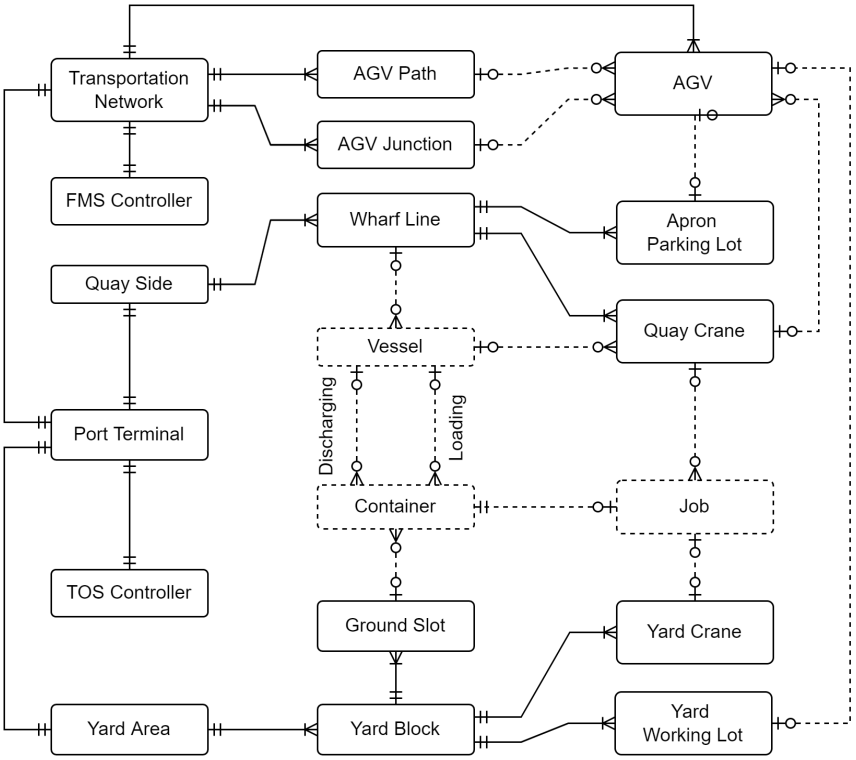

Figure 1: Entity-Relationship Diagram for a Container Terminal.

with the simulation clock, and those "dynamic relationships" that can change with the simulation clock. Thus, in the diagram, we use dashed lines to outline dynamic entities, as well as dynamic relationships. For example, vessels and loading containers are generated in the simulation model according to certain rules and removed from the system after the job is completed. What's more, we can intuitively find that the relationship with dynamic entities must also be dynamic. However, there can also be dynamic relationships between static entities. For example, AGVs always exist in the system, but their positions will keep changing.

Here we need to emphasize one point: the "entity" and "simulation module" in the entity-relationship structure are not completely equivalent. We will explain the concept of "simulation module" in detail below, but in simple terms, a module is an encapsulated part of the model with certain functions. The "simulation module" of the discrete event system needs to contain internal events and clearly defined "interfaces", so that any changes to the module properties from outside of the module can only be done through the pre-defined interfaces. The "entity" defined here is only a data structure or object so far. Whether it requires to be expanded or integrated with related events, or whether it needs to be expanded and encapsulated as a "simulation module", is to be discussed in detail later.

Also, for the above reasons, the "relationship" in the entity-relationship diagram does not specifically refer to the containment and subordination relationships in the module. For example, if the transportation network is treated as a module, the AGV does not necessarily belong to it, but can also travel to the quay side module and the yard area module. However, we can also choose to use different module partition methods, such as treating all AGVs and areas where AGVs can stay as part of the transportation network module. In this way, the jobs at the quay side and the yard area need to be seen as the interaction of these components with the "transportation network module", rather than the interaction with the AGVs. In conclusion, the "relationship" in the entity-relationship diagram has nothing to do with the "subordination"

of the module. Therefore, modularization, no matter what the partition method is, will not affect the general "relationship" defined in the entity-relationship diagram.

2.2 Event-based Operational Rules

After defining the scope of the model and the entities and relationships involved, the next step is to describe how the attributes and relationships of these entities change dynamically as time advances. For discrete event systems, the most accurate way to depict it is to directly describe and define the "events" that affect the state change of the system, that is, to use the "event-based" formalism. Because the dynamic changes of the entire container terminal are too complex, in this article, we only select one of the simple sections to illustrate. We use a graphical specification with "event-based" formalism, i.e. "event graph", to describe how the AGV and the quay crane equipment interact and "shake hands".

Through the example, we will find that the difference in the interaction between components in the discrete event system may affect the operating efficiency of the system more significantly than any of the decision-making algorithms. In other words, the choice of the strategy optimization critically depends on the type of protocol used to coordinate the operation between the equipment. The "event-based" modeling formalism can accurately and effectively describe the differences between different interaction protocols.

In an Event Graph, circles are used to represent events that occurred at a certain point in time. Events themselves have no duration. When an event is executed, the system simulation does three steps. The first is to update state variables according to the pre-defined rules and algorithms, the second is to collect and update the statistical data of the system, and the third is to plan and trigger new events. In the Event Graph, arrows are used to indicate the triggering relationship between events. Besides, the equal sign "=" on the arrow indicates a certain time delay between the two events, and the tilde " " is used to indicate that the triggering of the event needs to meet certain conditions.

Noting that in the entity-relationship diagram in Figure 1, to simplify the demonstration, we did not indicate the detailed attributes of each entity. Therefore, in the dynamic description of the model below, we will also skip the formulas and algorithms used to update these attributes. However, in the actual modeling process, whether in "Entity Relationship Diagram" or "Event Diagram", the definition of entity attributes and the algorithm for updating them are crucial. Modelers also prefer to use "parametric event graphs" rather than simple "event graphs" to describe system dynamics. Moreover, in actual modeling, the time delay and triggering conditions between events also need to be clearly described using mathematical symbols. However, for easier understanding, this article has made simplifications for the above aspects.

Let's first describe the case where the quay crane and the AGV operate independently without interaction. As shown in Figure 2, part (a) uses 8 different events connected in sequence, to describe the four phases from the individual perspective of the quay crane in moving a container; similarly, Part (b) shows the events that the AGV needs to go through to operate independently in moving the container. Among them, it is not difficult to see that the "Dropping" observed from the perspective of the quay crane is the same job that needs to be coordinated with the "Mounting" from the AGV perspective, which we call as "handshake". However, how to initiate this handshake can be done in several ways.

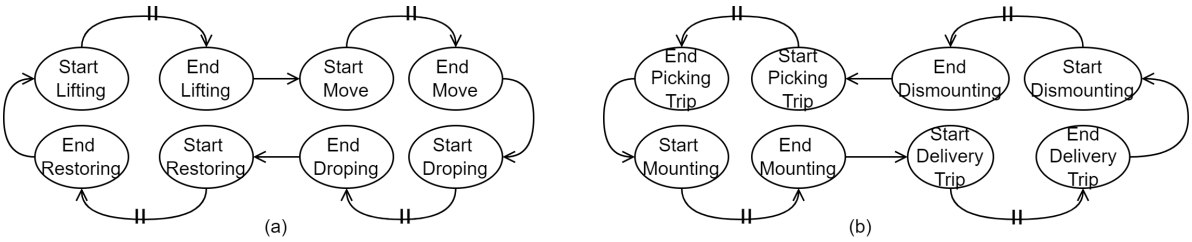

Figure 2: Event Graph for Independent Quay Crane and AGV Cycles.

For example, as shown in Figure 3, after the quay crane and AGV independently go through the other three phases of the workflow, they will trigger the same event, called "Attempt to Start Dropping / Mounting". This event will check the conditions for initiating the handshake, that is, when both devices are ready. The handshake will start only when the conditions are met. The completion of the handshake will trigger the respective independent phases in the operation cycles of the two devices simultaneously.

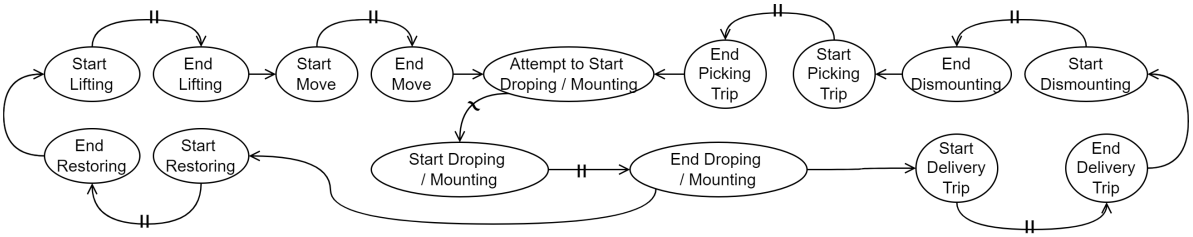

Figure 3: Event Graph for Synchronous Quay Crane and AGV Cycles.

In contrast, Figure 4 uses a different handshake approach. Here, we assume that the AGV has more resource capacity than the quay crane, so the initiation of the handshake depends entirely on the device which has fewer resources. According to the logic shown in the diagram, the independent operation cycle of the AGV is disrupted, and the picking and mounting operations are embedded in the operation cycle of the quay crane. Similarly, the logic expressed in Figure 5 is just the opposite. In the case of AGV resources being scarcer, the operation cycle of the quay crane is broken, and some operations are included in the AGV cycle. However, in real operation, the scarcity of quay cranes may be a more realistic assumption.

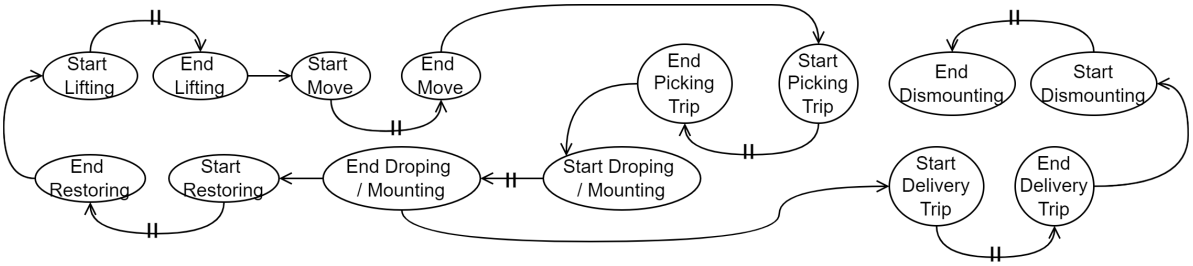

Figure 4: Event Graph for AGV Movement Driven by Quay Crane Cycles.

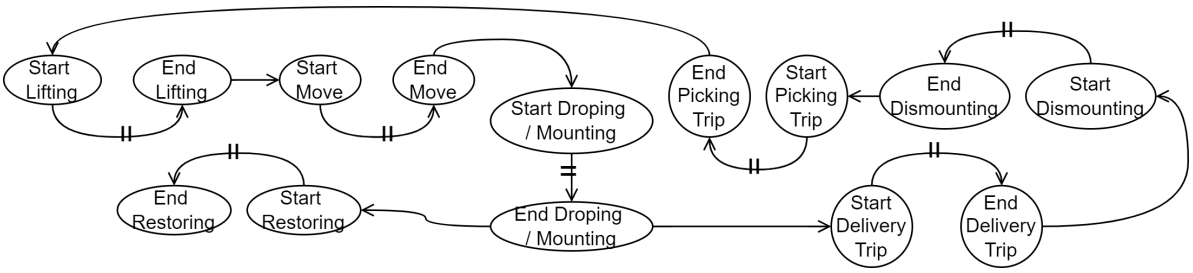

Figure 5: Event Graph for Quay Crane Movement Driven by AGV Cycles.

Nonetheless, the handshake shown in Figure 4 also has its downsides. This is because the operation cycle of the quay crane is expanded with part of the operation of AGV through an "asynchronous method", which lengthens its own workflow, and thus will greatly reduce the operational efficiency in practice. Therefore, Figure 6 proposes a potential improvement so that the quay crane can trigger the AGV to "start picking trip" as soon as the previous operation phase starts. Then, the handshake is initiated if the same conditions as shown in Figure 3 are met, which is both devices are ready. As a result, the scarcer quay

cranes are given control over the operation progression, and the operation time is shortened through the "coordinated method".

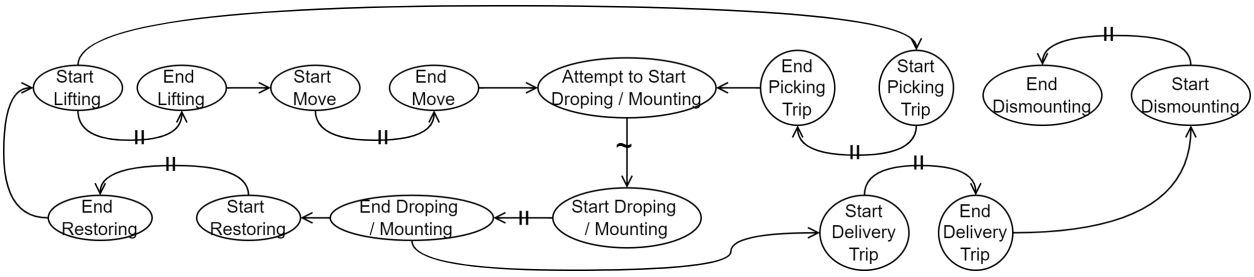

Figure 6: Event Graph for Coordinated Quay Crane and AGV Cycles.

Take note that the handshake in actual port operations may be more complex than the example illustrated here. But this does not affect our findings that, using the "event-based" modeling formalism, the complex operation process in the industrial system can be described clearly and accurately from the underlying logic. Therefore, in our hybrid modeling formalism, the "event-based" formalism, and the "event graph" modeling language, is used as the essential core representation. The "state-based" and "activity-based" formalisms introduced in the later part of the article, are also interpreted and executed using the "event graph" language. As such, in the actual modeling process, we can mix and match the three modeling formalisms based on different needs.

2.3 State-based Hierarchical Modules

The "event-based" formalism of modeling also has pressing shortcomings. First, the use of event graphs to describe the entire industrial system makes the modeling process sophisticated and tedious, and long-winded. Although event graphs precisely describe the dynamic details of the system, they do not provide a holistic understanding of the system, thus it is easy to lose the big picture. Especially when the dynamic rules of the system change and the model needs to be changed, the event graph can easily make the modeler get lost in the tedious details, and it takes a lot of time and energy to find the relevant event logic to make changes. This brings great difficulty to the maintenance and continuous expansion of the model. Similarly, "event-based" models have low reusability. We usually need to build an event graph from scratch for each new system, and it is difficult to achieve the accumulation of system knowledge. Furthermore, we must realize that a complex system in reality is usually maintained and operated by an entire professional team. The same goes for simulation systems, where it is hard to expect a single modeler to understand and build all the details of a complex system on his own. Therefore, we need a new way to realize the sustainable maintenance of simulation model systems, the reusability of model components, and the modeling process which allows team collaboration.

It is not difficult to find that the modularization of simulation models can effectively solve the above problems. However, to construct a modular simulation model scientifically and effectively, we need to use the "state-based" formalism, i.e. focusing on the interrelationships between state variables in the simulated system. When we find that a certain set of state variables are largely affected by the same set of events, but are rarely affected by other events, we can encapsulate this set of states and events together as a "module". This encapsulation method is very similar to object-oriented programming. In the modeling process, we can also use the "Entity Relationship Diagram" introduced above to quickly find these state variables. However, unlike entity objects, "simulation modules" must be encapsulations of state variables and corresponding collections of events. If it does not contain events, we can only call the encapsulated simulation model component an entity "object", not a "module". In some "state-based" modeling methods (such as DEVS), "events" are also called "transitions" of states. What's more, unlike other "state-based" modeling methods, in order to realize the mixed-use of the two modeling formalisms, in the method introduced in this article,

we use "event graph" (event graph) instead of "state diagram" (state diagram) to represent the logic inside the module.

It must be realized that in general, it is impossible for us to completely segment the collection of events and state variables using modularization. In other words, there does not exist a module in which no single state is affected by external events, and no event in the affect external state variables. This is because, if such a situation exists, it means that this module is an independent system completely isolated from other parts. It is also for this reason that it is impossible to have a standard answer on how to partition modules. How to partition the modules depends largely on the mode of collaborative effort in the model building and the possibility of reusability.

For example, for the handshake system between the crane and AGV shown in Figure 3, Figure 7 shows one of the partition methods, i.e. the events related to the handshake process are encapsulated in the AGV module. In comparison, Figure 8 adopts another partition method and encapsulates the handshake events in the quay crane module. Figure 7 and Figure 8 demonstrate the graphical method of drawing boundaries and defining input and output events on the basis of event diagrams, which we call the "internal-view diagram" of the module.

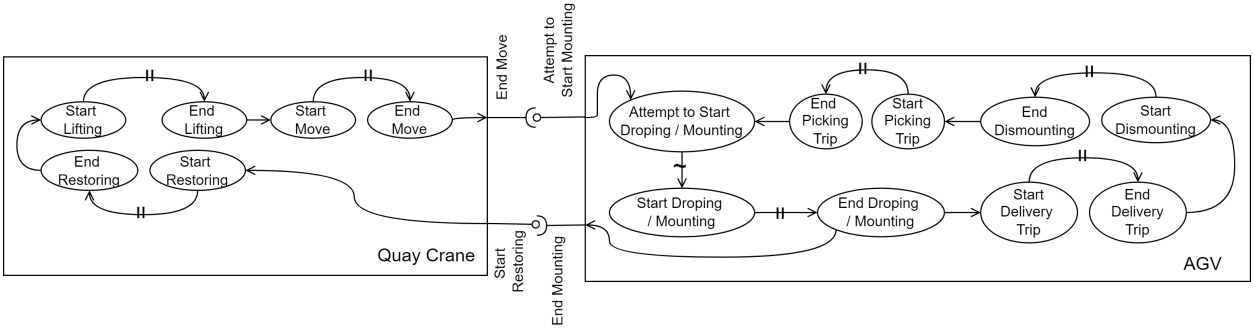

Figure 7: Modular Event Graph for Quay Crane and AGV Handshake (Type I) - The Intern View.

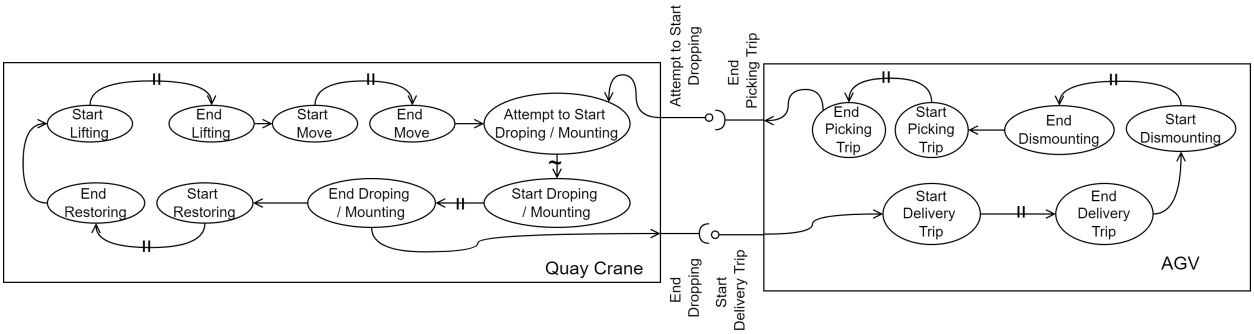

Figure 8: Modular Event Graph for Quay Crane and AGV Handshake (Type II) - The Intern View.

These two modularization approaches, have no obvious advantages or disadvantages when looked at individually. Only when we start to adjust the mode of handshaking between the two set of equipment, we discovered that the two modular approaches could be used to drive into respective different directions. For example, according to the modularization method in Figure 7, we retain part of the operation logic of the quay crane, and only change the AGV module to realize the quay crane - initiated handshake method shown in Figure 4; according to the partition method in Figure 8, The AGV module can be retained, and only by changing the quay crane module, the AGV initiated handshake method described in Figure 5 can be realized. The above comparison can be summarized by the interface of the two-way handshake shown

in Figure 9 (a) and (b). However, the synchronous operation method proposed in Figure 6 cannot be realized by the reuse of any previous module, but need to redefine the interaction between the two modules, which is described by the three-way handshake interface shown in Figure 9 (c). This kind of graphical specification that only draws model boundaries and interface events, ignoring internal events, is called the "external-view diagram" of the module.

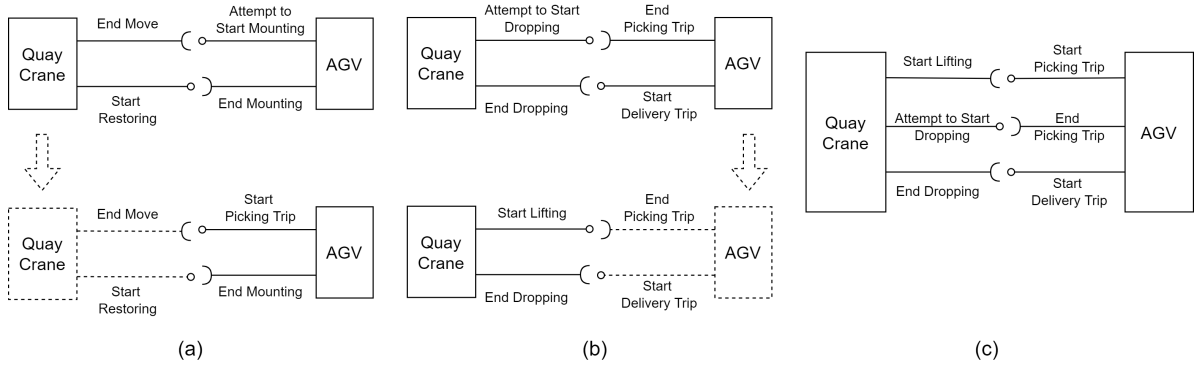

Figure 9: Modular Diagrams for Quay Crane and AGV Handshake - The External Views.

Using the external-view diagram, we can design the general outline of the entire port model and the hierarchical structure between the modules on a macroscopic level, ignoring the detailed logic inside the module. As shown in Figure 10, we can observe the affiliation between all port simulation modules, as well as with entity objects. Those round-cornered rectangles with no interface events represent non-modularized entity objects. Among them, the dotted bordered ones represent the dynamic entity objects. Most of them can appear in multiple modules simultaneously and are constantly generated and removed during simulation. They have a dynamic life cycle with different state stages. It should be noted that for the sake of simplicity, we are not showing the exact definition of interface events, nor providing an "interior view" to show how the internal events of the module interact with submodules. However, in the actual modeling process, accurate definitions and representations of them are very necessary.

2.4 Activity-based Entity Flows

Although there is no standard correct method for the segmentation of simulation modules, there are certain requirements for the definition of module interface events. Inappropriate module division can make it difficult for modules to be reused, unable to connect through interface, or cause redundancy or lack of event logic after connected through interface, resulting in errors in simulation logic. Therefore, we need to propose a method to guide and standardize the division of modules and the definition of interfaces, so that modules that conform to the standardization can be compatible with each other.

In addition, we also found that the "state-based" modularized formalism can well represent the static structure of the system, but when reflecting the global dynamic changes, the "state-based" method is difficult to intuitively reflect how those flowing components moves in the system. The flow of these components, although not the only form of dynamic changes in all systems, is precisely the most intuitive dynamic change to interpret a system. Therefore, we hope that the modularized simulation model can also accurately capture and intuitively reflect these changes.

Based on the above factors, we use an "activity-based" formalism to achieve these goals. In a nutshell, we first identify each component that moves in the system, which we call a flowing entity, or load. For each load, we divide the total time the load spends in the system into stages, based on their resource utilization or effect on other parts of the system. For each stage of each load, we call it "activity". In the modeling process, we can treat each "activity" as a logical module, or we can treat a group of "activities" associated with a certain type of resources as a logical module, or a resource entity module.

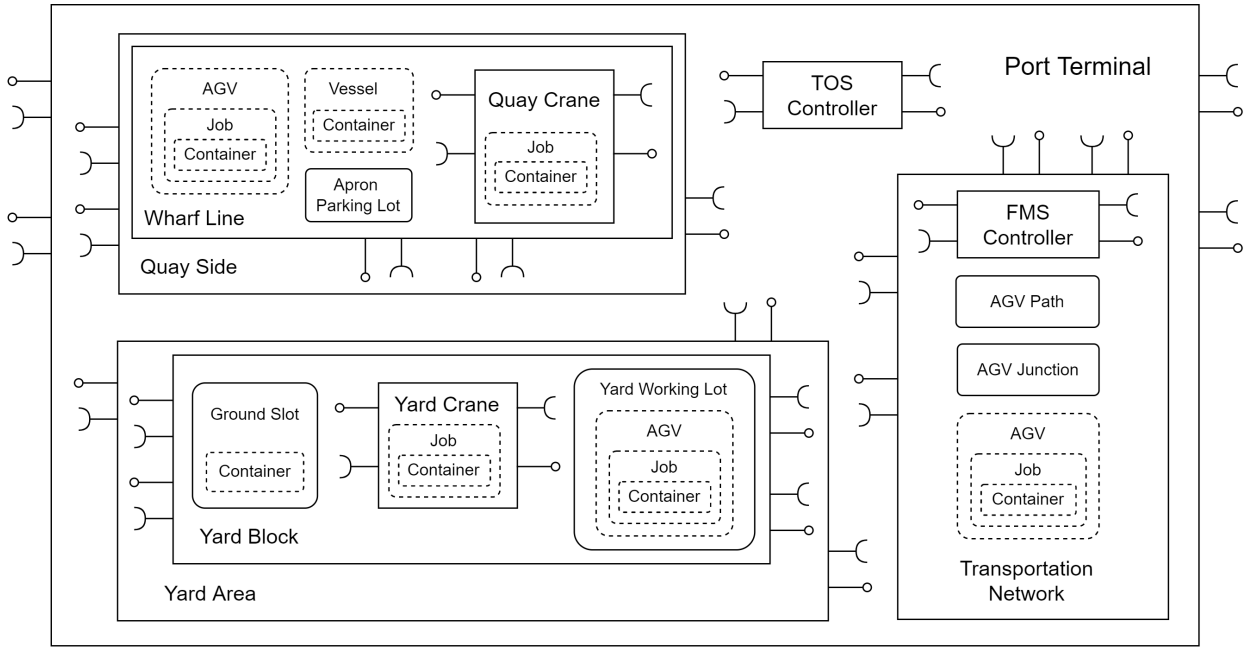

Figure 10: Hierarchical Modular Structure for Mega Container Terminal Simulation Models.

On top of this, we designed standardized event interfaces for the connection between activities. Therefore, the activities of the same load, no matter how the internal logic is customized, can be connected end to end, branched in parallel or merged. For each load, there will be a specific generator (generator), a terminator (terminator), and a series of activities connected between them, which constitute the entire life cycle of the load in the system.

Unlike other traditional "activity-based" approaches, in our hybrid modeling formalism, we use "state-based" modules to realize the collection of one or more "activity", and use " Event Graph" to define the discrete event logic within the "activity" and the interaction between the activities.

For example, shown in Figure 2, we can find that for the equipment, whether it is a quay crane or an AGV, its operational cycle can be seen as the alternating rotation of the four activities. However, in a general situation setting, we cannot guarantee that when each activity ends, the next activity can start smoothly. When the subsequent activity cannot be started due to resource constraints (such as road surfaces, tracks, operators, etc.) or other operating conditions (such as job planning, etc.), the resources of the previous activity cannot be released.

Therefore, a more general event graph model is shown in Figure 11. In this graph, each "activity" will first be triggered by the "request" event, but only when the resource and other conditions are met, will it then start and trigger the "start" event. After the "start" event is triggered, a time delay is scheduled to simulate the timespan of the job in the activity, and the job ends with "ready to end" event. However, "ready to end" cannot directly terminate the activity and release resources, but can only trigger the "request" event of the subsequent activity. The preceding activity can be terminated through the "end" event only when the subsequent activity successfully "starts". After the previous activity is terminated and the resources are released, if the conditions are met (for example, there are waiting tasks), the released resources will be used to restart the activity job of the next load. We divide logic modules according to a single activity, so that every two modules interact through a standard two-way handshake.

When an interaction occurs between two independent equipment operation cycles, as proposed in Figure 6 using synchronization method, we extend the original event graph to obtain Figure 12 using an "activity-based" perspective. It can be found that events in activities 1,3,4,9 got certain changes. In

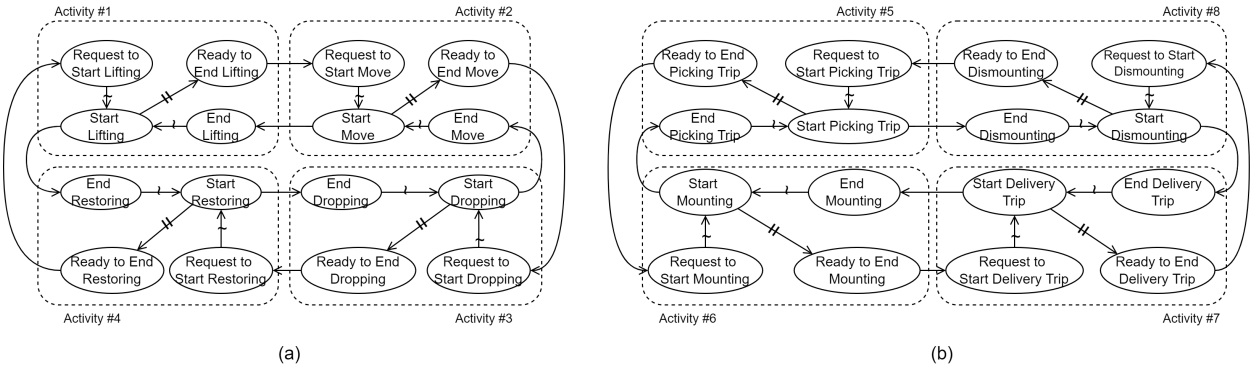

Figure 11: Extension of Event Graph for Individual Equipment Cycles from an Activity-based Perspective.

short, an output and an input event are added to activities 1 and 9 respectively to transmit information; inside activity 9, passive information received is used to end the activity instead of using active time delay; Activities 3 and 4 merged the internal "start" and "ready to end" events, and the simultaneous consideration of the resources and mobile entity tasks on both sides was added when judging the conditions. insert the consideration for resource availability and pending flowing entities in event triggering conditions.

Considering that the modular event graph is not sufficiently intuitive and concise, in the actual modeling process, we can use the method indicated in Figure 13 to simplify the modular external view in (a) to the Entity-flow diagram in (b). Among them, the two-way handshake interface composed of "ready to end", "request", "start", and "end" describes the transfer of material of the flowing entity from one activity to another. Such material flow of the flowing entity is simply represented by the bold arrow. The dashed thin arrows are used to indicate message transfer without material transfer. On top of this, we use a dashed line with a "$\sim$" symbol to synchronize activities that belong to two different flowing entities. Therefore, the event diagram in Figure 12 can be simplified as the entity flow diagram in Figure 14. In this way, the overall dynamics of the system can be seen at a glance.

Thus, using an "activity-based" formalism, we can the more concise "entity flow diagram" to represent the dynamic operation of different entities in the whole terminal. As shown in Figure 15 and 16 , the operation of the entire container terminal can be viewed as the flow and interaction of six different flowing entities, vessel, container, QC Line, QC, YC, and AGV. In the diagram, we marked the boundaries of the Quay Side, Yard Area, and Transportation Network modules with long dashed lines. Each module contains part of the activities of several different flowing entities. Tracking the flow direction of each entity and observing how they cross the module boundary can clearly explain the interface that needs to be defined for each module and how the modules interact with each other. Due to space limitations, we cannot make a more detailed segmentation of the above three modules in this article. However, in principle, we can divide the model further according to the hierarchical structure outlined in Figure 10. Then together with the methods shown in Figure 7 and Figure 8, the internal logic of each module can be accurately described with an event graph.

3 CONCLUSION

In conclusion, we have presented an overview of how the hybrid modeling formalism implemented with O²DES Framework is used to model the simulation of a mega container terminal.

While unable to represent show all the details of our modeling work, it is hoped that this brief introduction will inspire other similar simulation modeling efforts in the future. Our team will also publish some more detailed information, and share more details about the methodologies and techniques with our simulation communities for mutual progress and improvement.

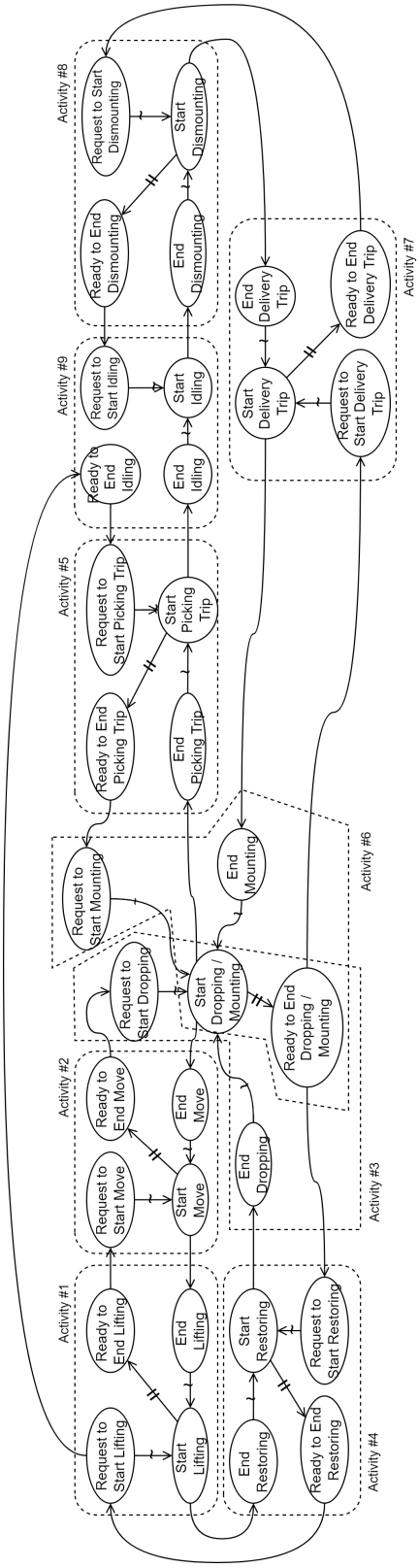

Figure 12: Extended Event Graph for Coordinated Equipment Handshake from an Activity-based Perspective.

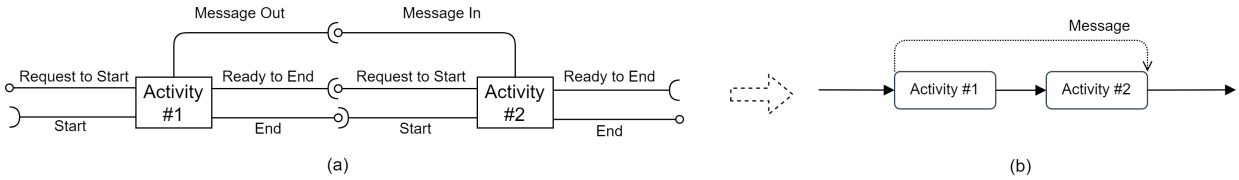

Figure 13: Simplification of Modular Diagrams as Entity-Flow Diagrams.

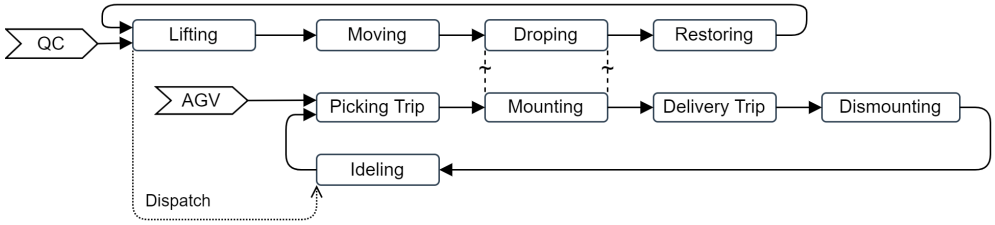

Figure 14: Representation of Coordinated Equipment Handshake using Entity-Flow Diagram.

ACKNOWLEDGMENTS

This research has been made possible by the funding support from the Singapore Maritime Institute. And the authors have been partially supported by Singapore A*STAR IAF-PP.

The co-authors fully recognize our dearest friend, colleague, and mentor, late Professor Lee, Loo Hay, for his strong leadership and contribution in conducting this research work.

AUTHOR BIOGRAPHIES

HAOBIN LI is a Senior Lecturer and the Assistant Head of the Department of Industrial Systems Engineering and Management (ISEM), National University of Singapore (NUS). Dr. Li received his Ph.D. and B.Eng. degree with 1st Class Honors from the same department of NUS. He is the Co-Director of the Centre of Excellence for Simulation and Modelling for Next Generation Ports (C4NGP), and the Academic Director of MSc Program in Maritime Technology and Management (MTM) in NUS. His email address is li_haobin@nus.edu.sg.

XINHU CAO is a Research Assistant Professor at the Centre of Excellence for Simulation and Modelling for Next Generation Port (C4NGP), National University of Singapore. She obtained her Ph.D. degree in Maritime Studies from Nanyang Technological University, Singapore. Her research interests focus on catastrophe risk assessment, disruption management, as well as transportation system simulation and optimization. She is a member of International Association of Maritime Economists. Her email address is isecx@nus.edu.sg.

EK PENG CHEW is a Professor at the Department of Industrial Systems Engineering and Management, National University of Singapore. His research interests include supply chain and simulation optimization. Prof. Chew is the Director of C4NGP, and the Director of Centre for Maritime Studies (CMS) in NUS. His email address is isecep@nus.edu.sg.

KOK CHOON TAN is an Associate Professor and the Deputy Head of the Department of Analytics Operations, NUS Business School, Deputy Head of Centre for Maritime Studies (CMS) in NUS, and a collaborator with C4NGP. His email address is kokchoon@nus.edu.sg.

KAUSTAV KUNDU is a Research Fellow at the Centre of Excellence for Simulation and Modelling for Next Generation Port (C4NGP), National University of Singapore. His email address is isekk@nus.edu.sg.

HONGDAN CHEN is a Teaching Assistant and PhD student at the Department of Industrial Systems Engineering and Management, National University of Singapore. Her email address is chd.ise@nus.edu.sg.

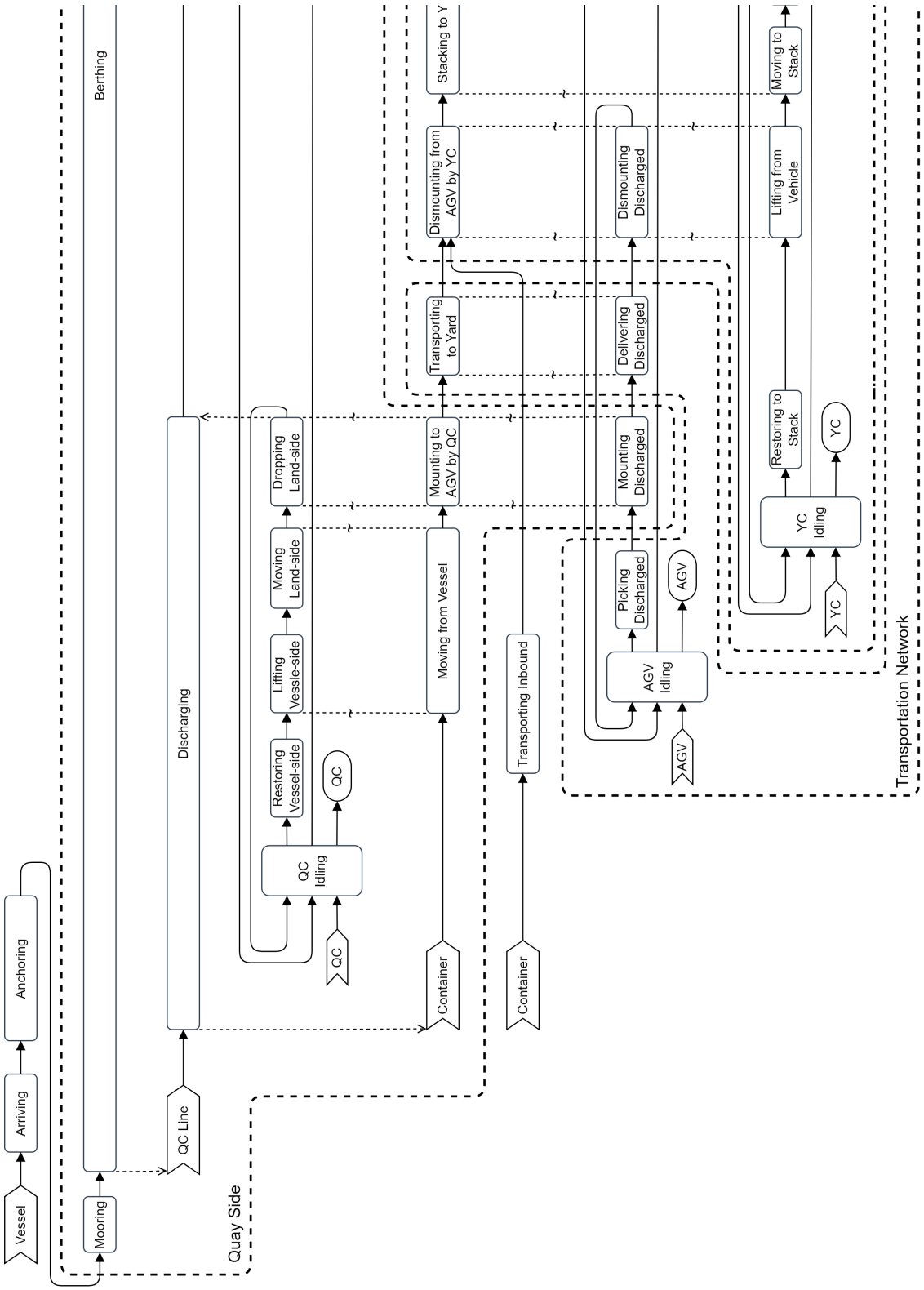

Figure 15: Entity-Flow Diagram for the Entire Mega Container Terminal (Left Portion).

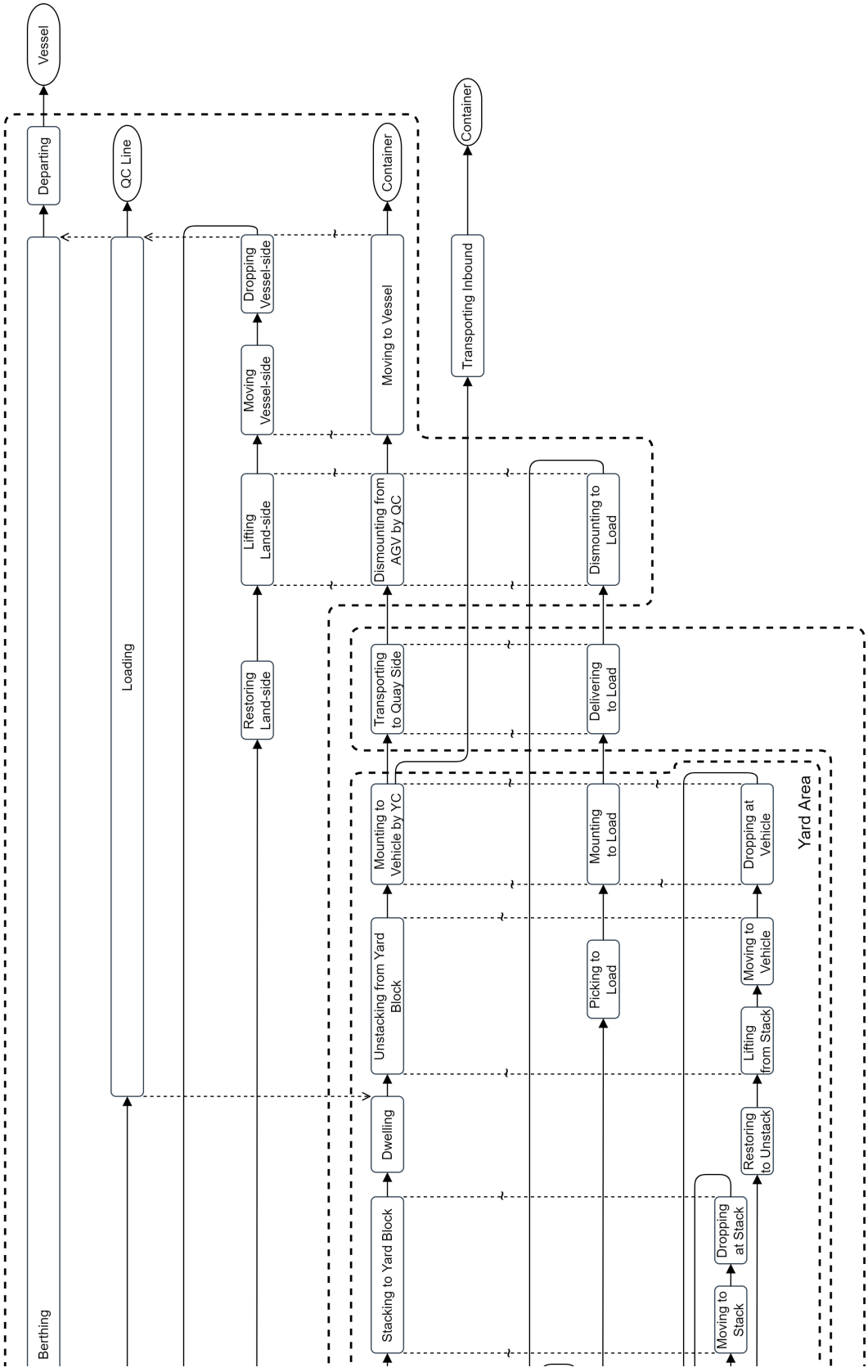

Figure 16: Entity-Flow Diagram for the Entire Mega Container Terminal (Right Portion).