A DEVS component library for simulation-based design of automated container terminals

Michele Fumarola
Delft University of Technology
Jaffalaan 5
2628 BX Delft
The Netherlands
m.fumarola@tudelft.nl

Mamadou Seck
Delft University of Technology
Jaffalaan 5
2628 BX Delft
The Netherlands
m.d.seck@tudelft.nl

Alexander Verbraeck
Delft University of Technology
Jaffalaan 5
2628 BX Delft
The Netherlands
a.verbraeck@tudelft.nl

ABSTRACT

Modeling and simulation in design processes is traditionally used during the analysis of the future system. However, simulation-based design allows the use of modeling and simulation throughout the synthesis phase which offers greater flexibility to quickly compare alternative designs. In case of container terminals, these alternatives are based on different aspects such as layouts, terminal operating systems, and equipments. Container terminals are characterized by a large number of entity types with multiple instances, interacting in various non-trivial ways. A component-based approach makes the construction of a suitable model easier: the designer can focus on the relevant constructs instead of lower level details. However, much effort is needed to achieve compatibility and modularity between the components. DEVS provides the higher level constructs to conceptualize a complex system independently from the underlying implementation. We present a DEVS component library for container terminal design wherein much attention has been put into the conceptual distinction between control and mechanics. This results in a library that can actively support the design process of containers terminals.

Categories and Subject Descriptors

I.6 [**Simulation and Modeling**]: Model Development

General Terms

Algorithms, Management, Design

Keywords

simulation-based design, component-based modeling, DEVS, container terminal

SIMUTools 2010 March 15–19, Torremolinos, Malaga, Spain.

1. INTRODUCTION

1.1 Simulation-based design and container terminals

The traditional role of simulation in a design study is in the analysis of the designed system. In case of simulations of container terminal, the scope of the model is to do performance analysis (mostly expressed in terms of twenty-foot equivalents per year, TEUs/year). However, in simulation-based design, simulations are used throughout the whole design process: this allows for immediate response to design decisions well before the classical analysis phase in the design study.

Different studies have been conducted in various fields where a simulation-based design approach has been used to design a complex system. Diaz-Calderon *et al.* [1] introduced a design environment wherein this is made possible in the domain of mechatronic systems. To achieve this, they introduced a reconfigurable and port-based modeling paradigm which allows defining a system with models that vary from abstract to concrete, to allow iterative refinement during the design process. Their port-based paradigm allows component-based modeling by connecting the different components through ports. The inner workings of the components are handled by differential equations, focusing on continuous systems. Medeiros *et al.* [6] discuss their Quest-implementation to support simulation-based design for a shipyard manufacturing process. In this research, a modular approach is used for which each module is constructed per simulation model.

In container terminal simulation, modeling and simulation has primarily been used for the analysis phase in the design process. Extensive literature surveys [2, 10, 11, 12] suggest a lack of simulation-based design approaches towards the design of container terminals. The design of container terminals could however benefit from such an approach to quicken design processes. Container terminals are characterized by a large number of entity types with multiple instances, interacting in various non-trivial ways. As with the similar studies on simulation-based design in other domains, a component-based approach makes the construction of a suitable model easier: the designer can focus on the relevant constructs instead of lower level details. A suitable framework is however needed to define the components and the interactions between the components.

In Hu *et al.* [3], the authors discuss how DEVS [14] can be used for component-based modeling and simulation. This is

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8670
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8670

achieved through implementing variable structure in DEVS for which components as well as couplings can be added and removed. Three reasons are given for using variable structure in DEVS: (1) to model systems that exhibit structure and behavior changes, (2) to design and analyze a system under development, and (3) to be able to load only a subset of the system's component when simulating very large models. DEVS, which is suitable for component-based modeling, provides a sound formal framework that can be used for simulation-based design if the flexibility provided by variable structure, is present. Hu *et al.* [4] also demonstrate how variable structure in DEVS can be used in a simulation-based design approach for real-time systems. In the analysis made by Shephard *et al.* [9], who discuss the technologies needed to support the application of simulation-based design in the CAD/CAE community, a similar approach is discussed.

1.2 Towards the component library

In this paper, we present a component library for simulation-based design of container terminals. Based on variable structure in DEVS, the components are able to dynamically add and remove new components where needed. Moreover, an increase in flexibility is achieved by exploiting abstraction as found in object oriented design. This allows a decoupling of simulation model from operating logic. Separation of concerns plays a role in this choice as our goal is to decouple a generic component library from an instantiation of the component library (i.e. having a component library that can be deployed to design various container terminals with each their own specific operating logic).

The paper is structured based on the design process that we went through to develop the component library. Different phases have been identified in this process: conceptualization, implementation in DEVS, implementation in Java, and finally a reference implementation to construct an instantiation. We will conclude the paper with conclusions and future work.

2. DEVS AND DSOL ES-DEVS

DEVS[14] is a modeling and simulation formalism for the discrete-event specification of systems. It allows for rigorous formal representation of complex dynamical systems and provides the operational semantics to simulate them. Its system theoretical roots endow it with sound constructs for behavior specification. The formalism allows for composition of network models from basic atomic components. The atomic DEVS formalism consists of sets (input values, output values, and states), and mappings on the latter sets (internal transition, external transition, output, and time advance) allowing complete and unambiguous specification of systems according to the discrete event abstraction. The coupled formalism consists of input, output, components, and coupling relation sets. Modularity is guaranteed by only allowing message based communication between components through input and output ports. The property of closure under coupling makes it possible to view network models and atomic models as equivalent, hence providing an elegant way of specifying hierarchical systems.

DSOL ES-DEVS[1][8] implements the parallel DEVS formalism on top of the Distributed Simulation Object Library (DSOL) [5]. Special attention has been put on strictly following the formalism and respecting the separation of concerns between models and simulators. The DSOL ES-DEVS simulation protocol is based on the event-scheduling worldview wherein executions of the internal transition function are scheduled according to the specified time advance function and unscheduled at the reception of external events. Within DSOL, this DEVS implementation benefits from the numerous services provided by an open, service oriented and web enabled open source modeling and simulation library.

[1]Distributed under a BSD-style license and available at http://simulation.tudelft.nl

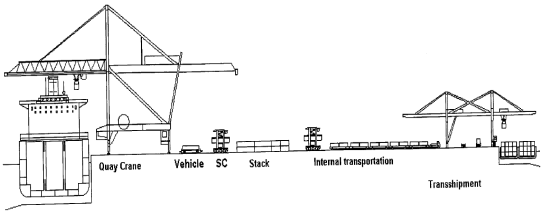

Figure 1: The high level processes taking place at a container terminal as schematized by Vis [12]

For a more intuitive understanding of the models discussed in the remainder of this paper, a graphical notation of DEVS models is adopted. Figure 5 and 7 show the graphical depiction of coupled and atomic DEVS models respectively. A coupled model is represented as a named box with incoming and outgoing arrows representing inputs and output ports. The included components are represented as black-boxes within the coupled model and lines between the ports represent the coupling relations. An atomic model is represented as a white-box containing a digraph, with input and output ports drawn as incoming and outgoing arrows. Nodes of the digraph represent phases with lifetime written inside. Passive phases (i.e. phases with infinite lifetime) are drawn with a continuous line and active phases (with a finite lifetime) are drawn with a dashed line. The edges of the digraph represent transitions between phases. An internal transition is represented as a dashed line with the associated output value. An external transition between two phases is denoted by a continuous line, with the associated input event.

3. TERMINAL CONCEPTUAL DESIGN

Container terminals are physical facilities composed of mainly material handling equipment needed to tranship containers to and from ships, barges, trains and trucks. To identify the various types of equipment, the analysis of the different processes present at a typical container terminal, can help. Vis *et al.* [12] schematized the different processes in Figure 1. Quay cranes are responsible of loading and unloading containers to and from the vessels. Between the quay cranes and the stacks, dedicated vehicles are active to get the containers from the lanes near the quay cranes to the stacks. These vehicles can be either manned or unmanned, depending on the mode of operation. In the stacks, cranes, such as rail mounted gantry cranes, stack the containers in the stacking areas. At the land-side of the stacking area, dedicated vehicles are again responsible to bring the containers from the stacks to the rails, barges or trucks.

The scope of this research is restricted to automated container terminals: a selection of the large variety of mate-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8670
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8670

rial handling equipment can therefore be made. This selection assumes container terminals wherein quay cranes are active at the sea side, automated guided vehicles take care of the transporation between sea side and stacks, and between stacks and land side, and finally rail mounted gantry cranes handle the containers in the stacks. In automated container terminals, the terminal operating system (TOS) controls the equipment and is therefore a crucial part for the performance of the terminal.

In Figure 2, a conceptual model of an automated container terminal has been schematized in a System Entity Structure (SES) [13]. An SES is a specification of structural and specialization relations for a model family. The diagram shows the decomposition of an automated container terminal into different types of equipment and a TOS at different abstraction levels. Further, the TOS is functionally decomposed to control each type of equipment. Finally, each type of equipment is decomposed both on a physical as on a functional level of each type of equipment. The function of each component will be discussed in greater detail in section 4.

4. DEVS SPECIFICATION OF THE TERMINAL

4.1 The container terminal coupled model

There is a direct mapping of the entities identified in the SES on the model components expressed in the DEVS formalism, these are modeled as either atomic or coupled DEVS-models. We chose to implement the lowest abstraction level by pruning the tree to keep the high definition entities. We will first start by presenting the overall model and continue by discussing the detailed models. The overall structure of the model is presented in Figure 3: this figure shows the coupled models. The overarching coupled model contains the model for the actual container terminal and the experimental frame. The model of the container terminal holds the model of the TOS which sends orders to the models of the different equipments.

Both TOS and the equipment models are coupled models. Although this diagram shows only a subset of possible equipment, other model for equipments can be added as well. This can for instance be done for barge cranes, terminal trucks, etc. We will later on see how the TOS takes care of handling new equipment and how the operating logic is abstracted out of the model itself.

Similar to Saanen's approach [7], the individual components can be grouped into managerial, controlling and physical models. The managerial models are part of the TOS and manage the orders. The controllers receive the orders and translate these orders into specific movement commands that are sent to the physical equipment, that actually perform the commands. As the models pertaining to each group share the same behavior, we will present each group with the models that manage, control and represent the quay cranes.

4.2 The TOS coupled model

The TOS contains a general order manager and equipment-specific order managers, as shown in Figure 4. The order manager receives generic orders ("take the container from place X and put in on place Y"), which we call complex orders, and sending it through to the assigned equipment to perform the specific order, which we call simple orders. The

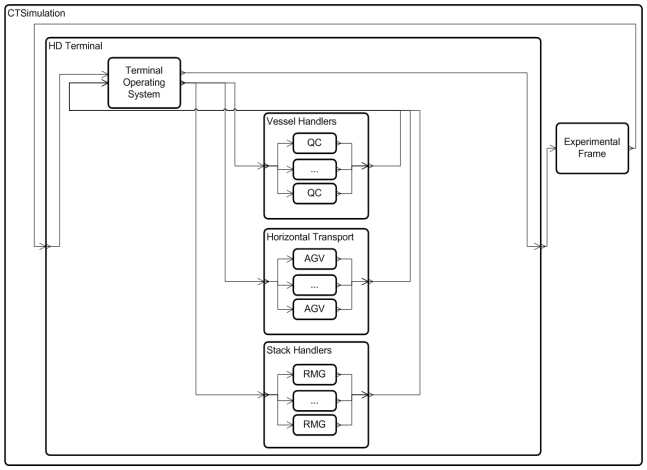

Figure 3: The structure of the model. The coupled models are shown with their in- and outputports.

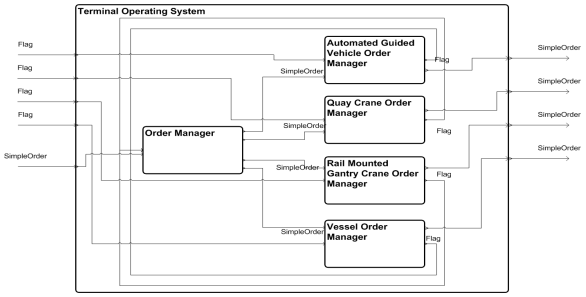

Figure 4: The model of the terminal operating system with a general order manager that sends order to the equipment-specific order managers.

order manager of specific equipment is modeled as a DEVS atomic model that has two phases: an idle and a processing phase. When the model is in phase idle and receives an event, it makes a transition to phase processing, sends a flag out to the relevant port, and makes an internal transition back to phase idle. Flags are received through the input ports connected to the controlled equipment. The order manager of the quay crane is schematized in Figure 5.

4.3 The equipment coupled models

The order goes from the TOS to an equipment. A type of equipment is decomposed into controlling and mechanical models. The behavioral models are called controllers, whereas the physical models depend on the different physical parts of the equipment. The controllers of equipment receive the order and decompose this to control the different physical models. This happens according to different movements that the equipment has to do in order to fulfill the order. To model this, the choice has been made to decompose the controller into subcontrollers that are responsible for the different movements. For a quay crane, this can be a preparing movement ("put the crane into a predefined position which makes it able to freely move"), positioning movement ("put the crane to the exact place where the container has to be picked up"), and finally the transporting

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8670
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8670

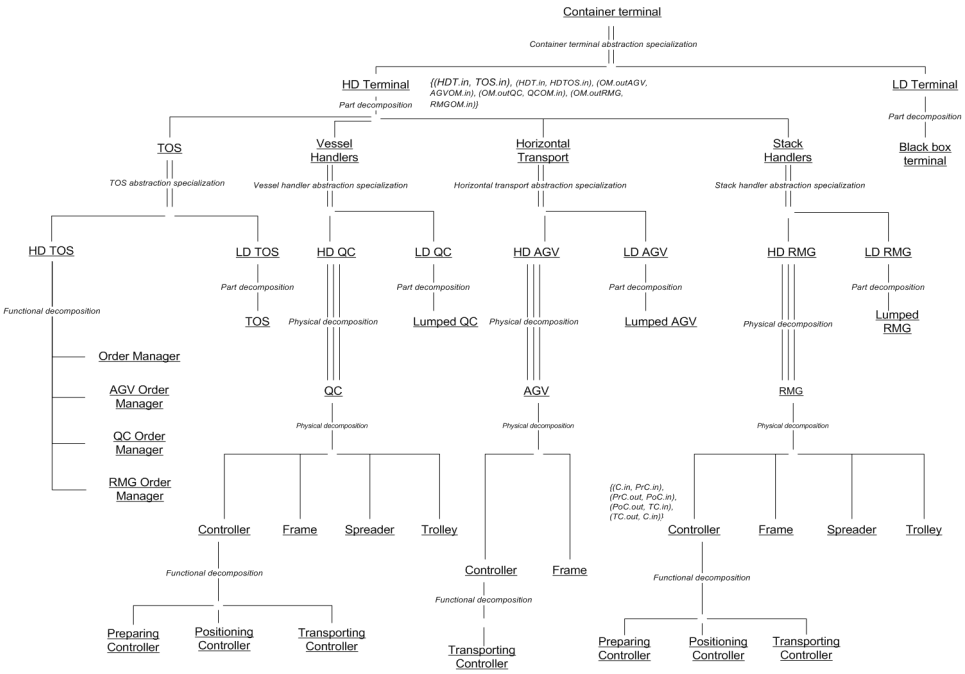

Figure 2: The System Entity Structure of the conceptualized automated container terminal. For clarity reasons, only major couplings have been specified. (Abbreviations: QC for quay crane, AGV for automated guided vehicle, RMG for rail mounted gantry crane, HD for high definition, LD for low definition, TOS for terminal operating system)

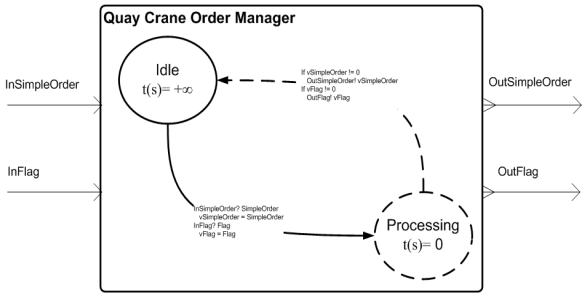

Figure 5: The model of the order manager of the quay crane.

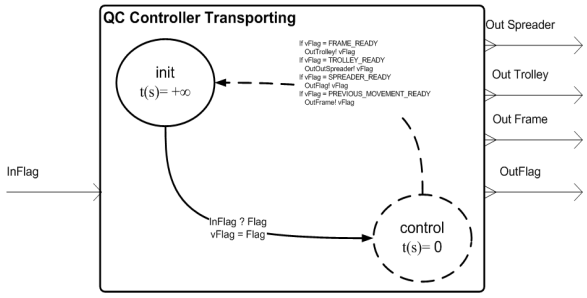

Figure 6: The model of a transporting controller is shown here as a reference implementation to all equipment controllers.

movement ("pick the container and transport it to the final destination"). As shown in Figure 6 (the transporting controller has been taken as an example) , the controllers are fairly simple, having the sole responsibility of sending commands to the physical models, which happens in the output function. It has to be noted that the decomposition of the main controller serves as a reference model, but does not imply a restriction for future uses. The controllers can easily be changed by other controllers with different behavior.

The physical models are in charge of performing the physical activities, which is mostly changing positions. In case of a quay crane, the physical models are a moving frame, a spreader and a trolley. As an example, the moving frame is schematized in Figure 7. It is interesting to note how this model differs from a more traditional way of modeling moving equipment, which mainly happens in discrete time fashion. In this model, a continuous movement is being discretized into just five distinct phases instead of a large amount of time-segments. Figure 8 helps clarifying the advantages of such an approach: in discrete time, we have to calculate the new position and speed at every single time step whereas in discrete event, the calculations can be performed three times: one time at the beginning of every phase relevant to the state. The calculations of the acceleration, cruise and deceleration times (which are presented in Figure 7 as function f) are calculated as follows:

Require: $CurrentSpeed, Rate, Distance$
$QuadraticSolution < T >$
$\leftarrow SolveQuadraticEquation(Rate * T^2 - InitialSpeed * T + Distance)$
$MovementTime \leftarrow Max(QuadraticSolution < T >)$

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8670
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8670

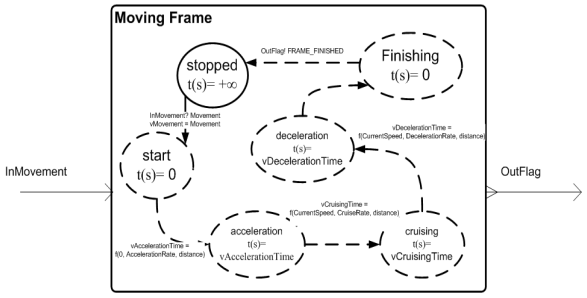

Figure 7: The model of a moving frame is shown here as a reference implementation to all the physical models. (f is the function to calculate times based on speed, rate and distance)

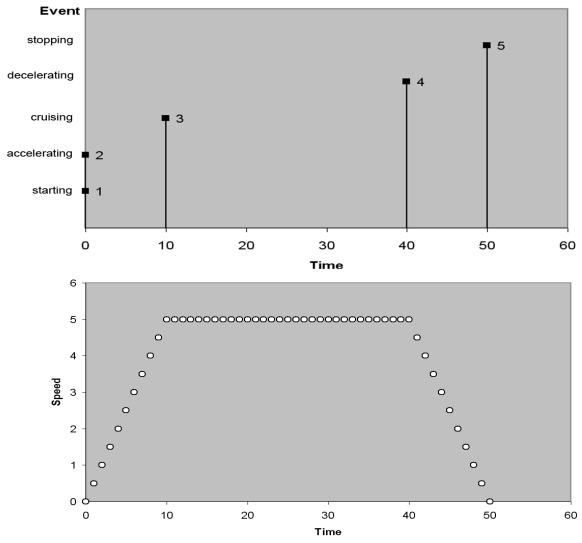

Figure 8: The difference between time based and event based modeling: the model of a moving vehicle can be done in fewer calculations.

return $MovementTime$

These types of calculations are made for each moving equipment and for each subpart of an equipment (for instance a trolley that goes for- and backwards). This spares a large number of discrete time calculations, which in large numbers can be computationally expensive, that can result in a quicker execution of the model without loss in precision.

5. MODEL BUILDING PROCESS

A dynamic build-up mechanism of the container terminal model is able to add and remove equipment from the model and construct and remove the couplings between the TOS and the equipment. The container terminal model does this by receiving events that define the addition and removal of equipment. This results in the creation and destruction of the model instance of the equipment and in the creation and destruction of the coupling from this new model instance to the rest of the container terminal model. This mechanism corresponds to variable structure as described by Hu *et al.* [4].

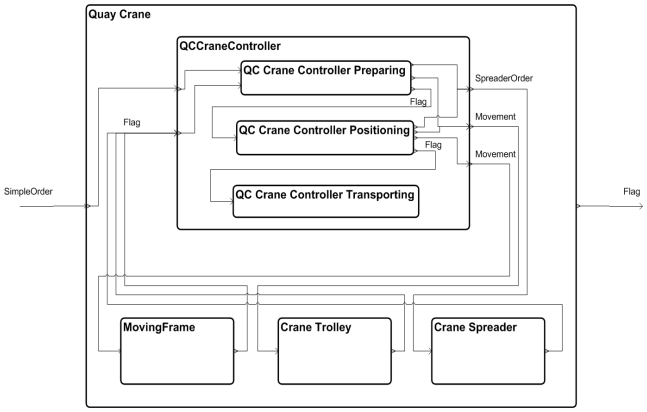

Figure 9: The model of a quay crane contains a controller, a movingframe, a trolley, and a spreader.

We can use the running example of quay cranes to explain this mechanism. The container terminal receives an event to create a new quay crane. The creation of the quay crane model is performed based on the received parameters. This model is directly connected to the TOS which on its turn will secure the connection with the right order manager. In a similar way, the removal of the quay crane takes place, by having an event to remove a certain quay crane which is present in the container terminal.

Although this mechanism will mostly be used in the initial creation of a container terminal, thus sending events at simulation time 0, other possible scenarios can be thought of. During a long term simulation of a container terminal, a decision could be made to add a number of equipments after a couple of (simulated) years. With a static model, this would be a difficult task. However, with the mechanism presented here, an event can be sent for the creation for which the container terminal will just continue to work with the extra equipment.

6. JAVA IMPLEMENTATION

6.1 Object-oriented abstraction

The DEVS model presented here, has been implemented in DSOL-DEVS. Although a one-to-one mapping of the DEVS conceptual model in a class model would have been a possibility, careful consideration is put into exploiting the object oriented environment provided by Java. Abstraction, as found in object oriented programming, is a powerful mechanism that can be exploited in the implementation of DEVS models without conflicting with the formalism. As many atomic models can have some communinalities (for instance a common external transition function), reimplementing the same function multiple times, would become cumbersome and error-prone.

Throughout the discussion of the DEVS model, there was a clear grouping of similar models: managers, controllers and physical equipment all share some similitarities. These groupings can give use the possibilities to abstract different functions on the managers, controllers and physical models. In the class diagram presented in Figure 10 these abstractions are presented through the inheritance relationship. Each ordermanager inherits the deltaExternal method

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8670
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8670

(External transition function) and the timeAdvance function from the abstract class *EquipmentOrderManager*. This is due to the basic behavior that has been presented in Figure 5. In a similar way, the controllers (e.g. *RailMountedGantryControllingPreparing* and *AutomatedGuidedVehicleController*) inherit their behavior from abstract controllers *CraneControllingPreparing* and *VehicleController* respectively. Finally, equipments sharing same controllers and physical models, can be abstracted by using the abstract classes of the containing models.

6.2 The Terminal Operating System

In an automated container terminal, the TOS runs the main algorithms to control the equipment. In the model presented here, the TOS contains an order manager and the various equipment-specific order managers. The order manager keeps a list of orders and always picks the order first in the list. This list is ordered according to an instance specific algorithm which allows the user of the component library to implement a specific order allocation algorithm by re-implementing the ordering method. The same approach was chosen for the equipment allocation, by keeping a list of equipment which will be ordered according to a specific algorithm.

The task of the order manager is to take complex orders and transform them into multiple simple orders that can be sent to the equipment managers that will handle the orders. To make the transformation from complex order to simple order possibile, the order manager has access to a class that implements the *TOSLogic* interface. This makes it possible to assess different types of algorithms (e.g. scheduling of equipment, stacking of containers, etc) without changing anything of the container terminal model.

The *TOSLogic* interface contains an enumerator of order types, a function to create simple orders, one to create order stacks and finally two functions to check whether an order is compatible with a selected instance of equipment. The most important function is the one responsible of creating a simple order. To achieve this, the implementation of the interface can access the list of available (and unavailable) equipment, and all current orders. The interface supposes also an implementation of a datastructure reflecting the layout of the container terminal. In the reference implementation, a graph has been used where each node restricts access to certain type of equipment so that the paths of equipment can be deduced.

7. EXPLANATORY DESIGN PROCESS

To use the components library for a specific container terminal, an instantiation for that terminal has to be made. To do so, two main components need to be adapted: the TOS and the experimental frame.

As discussed earlier, the order manager in the TOS uses implementations of the Java-interface *TOSLogic* to implement the operating logic of the container terminal. Although reference implementations can offer a choice between predefined components, custom-made implementations are often required. Implementing the operating logic consists of providing order and equipment priority and of providing algorithms for the decomposition of complex orders to simple orders. The latter is especially important to implement specific AGV scheduling and routing algorithms. The current reference implementation decomposes complex orders to simple orders as follows:

for ComplexOrder co: ListOfComplexOrders **do**
  $mp \leftarrow GetBestPossibleMapZone(co)$
  $eq \leftarrow GetBestAvailableEquipment(co, mp)$
  $so \leftarrow ConstructSimpleOrder(so, co, mp, eq)$
  $SendOrder(so)$
end for

A description of an instance of a container terminal together with experimental frame can be stored in an external XML-file. Once the XML is present, all equipment can be added to the model through the dynamic coupling mechanism of the build-up process. Experimenting with different designs throughout a design process therefore means writing several XML-files with different number and types of equipment, that can be loaded into the model. Herewith, comparison and analysis of the results of the different models pertaining to the different XML-files can be performed.

8. CONCLUSIONS AND FUTURE WORK

We have presented a component library for simulation based design of automated container terminals. We chose to use the DEVS-formalism to achieve an elegant, formal and clear model. The SES and strongly typed couplings assured that only compatible models can be coupled, thus constraining the relations between components.

On an implementation level, DEVS provided a clear separation between model and simulator. Further, the abstraction in Java provided a less error prone implementation and a smaller code-base for the model. Finally, by excluding the TOS operational logic from the TOS model, a clear separation of concerns has been achieved which makes us able to connect different implementations of a TOS logic to our model. With the resulting implementation, various designs can be assessed. These designs can differ in number and specification of the equipments, the algorithms used in the TOS and the overall layout of the terminal. Further, phased design of container terminals can be assessed more easily by exploiting the dynamic structure model construction process.

Future work consists of using the model in a full fledged design environment. Visualization plays an important role: to separate it, and other sorts of output, from the model, the publish-subscribe mechanism will be used.

9. REFERENCES

[1] A. Diaz-Calderon, C. Paredis, and P. Khosla. Reconfigurable models: A modeling paradigm to support simulation-based design. In *Proceedings of SCS 2000 Summer Computer Simulation Conference*. The Society for Computer Simulation International, July 2000.

[2] L. Gambardella and A. Rizzoli. The role of simulation and optimisation in intermodal container terminals. In *Proceedings of 10th European Simulation Symposium*. The Society for Computer Simulation International, September 2000.

[3] X. Hu, X. Hu, B. Zeigler, and S. Mittal. Variable structure in DEVS component-based modeling and simulation. *Simulation*, 81(2):91–102, February 2005.

[4] X. Hu and B. Zeigler. Model continuity in the design of dynamic distributed real-time systems. *IEEE Transactions on Systems, Man, and Cybernetics -*

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8670
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8670

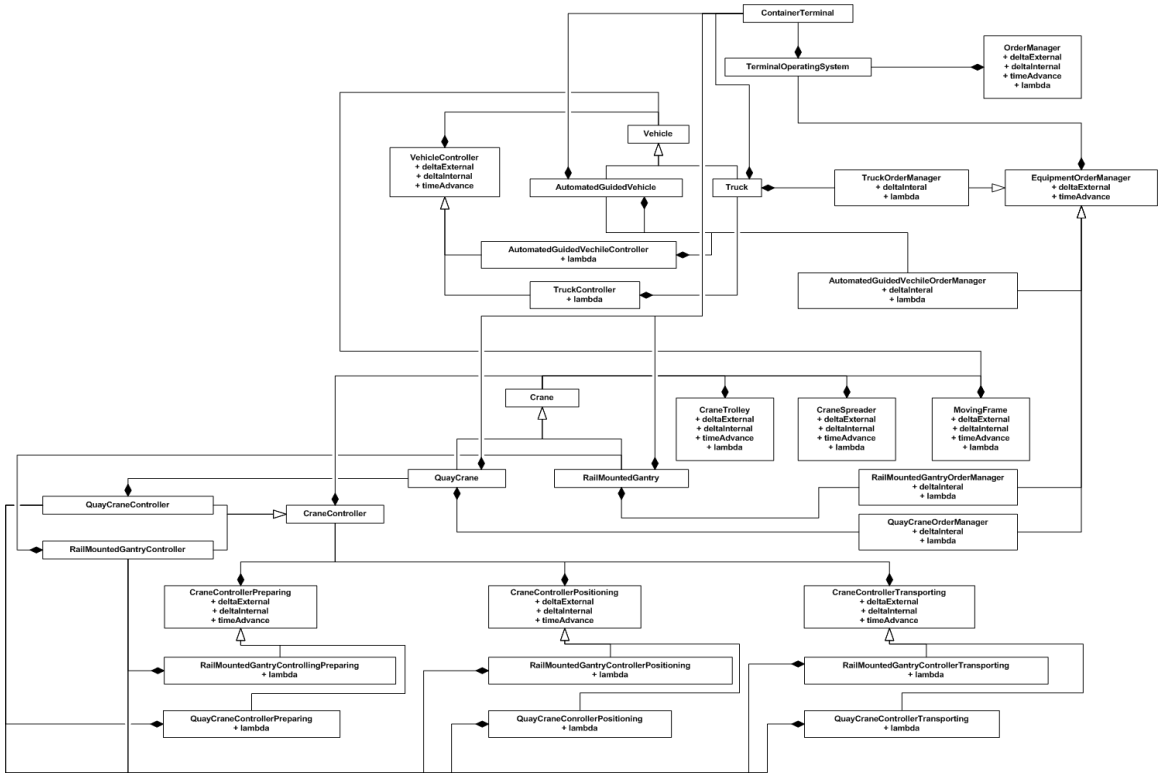

Figure 10: The UML class diagram of the DEVS model. Every class is either an atomic model or a coupled model. The latter can be recognized by the absence of methods.

Part A: Systems and humans, 35(6):867–878, November 2005.

[5] P. Jacobs. *The DSOL simulation suite - Enabling multi-formalism simulation in a distributed context*. PhD thesis, Delft University of Technology, 2005.

[6] D. Medeiros, M. Traband, A. Tribble, R. Lepro, K. Fast, and D. Williams. Simulation based design for a shipyard manufacturing process. In *Proceedings of the 2000 Winter Simulation Conference*, pages 1411–1414. Winter Simulation Conference Board of Directors, December 2000.

[7] Y. Saanen. *An approach for designing robotized marine container terminals*. PhD thesis, Delft University of Technology, 2004.

[8] M. Seck and A. Verbraeck. DEVS in DSOL: Adding DEVS operational semantics to a generic event-scheduling simulation environment. In *Proceedings of SCS 2009 Summer Computer Simulation Conference*. The Society for Computer Simulation International, July 2009.

[9] M. Shephard, M. Beall, R. O'Bara, and B. Webster. Toward simulation-based design. *Finite Elements in Analysis and Design*, 40(12):1575–1598, 2004.

[10] D. Steenken and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, January 2007.

[11] D. Steenken, S. Voß, and R. Stahlbock. Container terminal operation and operations research - a classification and literature review. *OR Spectrum*, 26(1):3–49, January 2004.

[12] I. Vis and R. de Koster. Transshipment of containers at a container terminal: an overview. *European Journal of Operational Research*, 147(1):1–16, May 2003.

[13] B. P. Zeigler and P. E. Hammond. *Modeling & Simulation-based Data Engineering: Introducing Pragmatics Into Ontologies For Net-centric Information Exchange*. Academic Press, 2007.

[14] B. P. Zeigler and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press, 2000.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8670
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8670