COMBINING LOGISTIC CONTAINER TERMINAL SIMULATION AND DEVICE EMULATION USING AN OPEN-SOURCE JAVA FRAMEWORK

Dipl.-Inf. P. Joschko[a], Dipl.-Wirt.-Inf. C. Brandt[b], Prof. Dr.-Ing. B. Page[c]

[a], [c] University of Hamburg, Department of Informatics
[b]Hamburger Hafen und Logistik AG

[a]joschko@informatik.uni-hamburg.de, [b]brandt-c@hhla.de, [c]page@informatik.uni-hamburg.de

ABSTRACT
In this paper we intend to show that the open-source simulation framework DESMO-J can be used not only for logistic investigations - as it was being used by many during the last years - but also for evaluating and testing control systems. Therefore we propose to implement model components of container terminals in such a way that they can be used in logistic simulation as well as in device emulation test beds. This paper outlines our experience in extending DESMO-J for a broad range of applications in the context of container terminals and our experience in developing reusable model components for simulation and emulation. While Java-based DESMO-J is a product of the University of Hamburg, the functional container terminal extension, called COCoS, we present in this paper has been developed by the Hamburger Hafen and Logistik AG.

Keywords: discrete simulation framework, logistic simulation, device emulation, container terminal

1. INTRODUCTION

1.1. Hamburg Container Terminals

With a container handling rate of 9.7 million TEU in 2008, Hamburg accommodates one of the ten largest container ports worldwide and accordingly the second largest within Europe, right behind Rotterdam. This capacity is to be expanded to 18 million TEU in the next couple of years.

Hamburger Hafen und Logistik AG (HHLA) is one of the leading port logistics groups in the European North Range. With its Container, Intermodal and Logistics segments, HHLA is positioned vertically along the transport chain. Efficient container terminals, high-capacity transport systems and a full range of logistics services form a complete network between the overseas port and its European hinterland.

HHLA operates three container terminals which were responsible for more than two thirds of the overall container handling in Hamburg in 2008. One of them is the HHLA Container Terminal Altenwerder (CTA) which is considered to be a state of the art terminal worldwide. The northern section of the CTA is shown in Figure 1. Since the container dispatching processes at the CTA are almost fully automated, a particularly significant relevance is attached to the integration of container handling technology and IT-based control systems.

Increasing handling rates require highly efficient dispatching strategies for container vessels and hinterland traffic. Not only that every deceleration would bring financial damage to terminal carriers and forwarding companies but additionally, an interruption in container handling process would immediately cause congestions in metropolitan area traffic because of nearby autobahns and city center.

These requirements stress the importance of using efficient methods for improving and validating the evolution of terminal layout, logistic handling strategies or control systems. The University of Hamburg and HHLA have been collaborating on developing simulation techniques to solve such terminal-specific problems for many years.

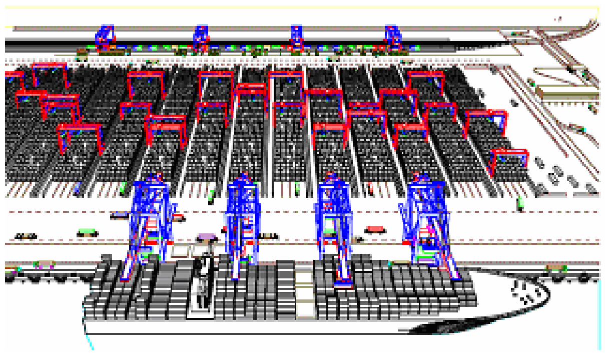


Figure 1: Northern section of Container Terminal Altenwerder (Brandt 2007)

1.2. Container Terminal Simulation

According to (Steenken et al. 2004) a multi modal container terminal can be considered as an open system. The waterside (ships) and the landside (trains, trucks) have always been the two main interfaces to its environment. But distinct container terminals may significantly vary in types and quantity of handling equipment, terminal layout or grade of automation of handling processes. Besides these physical aspects there are also many individual types of supporting and surrounding IT-Systems as well as planning and control strategies integrated therein. The fact that these and further aspects need to comply with various specific requirements of terminal operations finally leads to a considerable number of planning and optimization problems that emerge and evolve throughout a container terminal's life cycle. According to (Saanen 2001) this life cycle may be divided into three main phases:

1. Conceptual, functional and technical design
2. Implementation and Realization
3. Commissioning and Operation

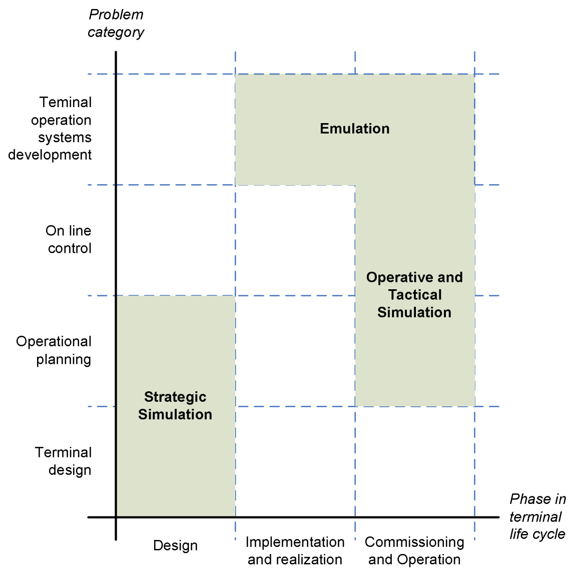
In practice, the phases may overlap significantly and may be interconnected by iterative feedback loops, for instance if an existing terminal receives a redesign due to new requirements.

Simulation is to be used if experimentation with the real system is too expensive, complex or dangerous, which applies to container terminals in particular. Therefore simulation has become an important method in analysis and optimization of container terminal operations in the recent past (Steenken et al. 2004). Just like the planning problems mentioned above, the application of simulation may be attributed to the phases of a container terminal's life cycle and may be differentiated accordingly as illustrated in Figure 2.

Strategic simulation mainly points at problems of terminal design and is applied in phase (1) correspondingly. Different logistic concepts, decision rules or optimization algorithms that address problems of operational terminal planning within phase (2) and (3) deserve close comparison by means of operative simulation studies before they can be put into production on an existing terminal. Tactical simulation stands for an integration of simulation models within the terminal's IT-systems in order to generate solutions for operational and on line planning problems in phase (2) and (3). This integration is also referred to as simulation optimization or optimization via simulation. While strategic, operational and tactical simulation are directly focused on, mainly logistical, planning problems within a terminal's life cycle, phase (2) particularly requires simulation techniques for the purpose of development and validation of terminal operation systems by emulating terminal equipment, IT-systems or human behavior.

1.3. Outline

Section 2 details the comparison between simulation and emulation in order to extract differences and similarities. Section 3 concentrates on the simulation framework DESMO-J and explains its main functionality, its software architecture, how it can be extended and how it has been used in harbor simulation so far. Section 4 concentrates on COCoS, the extension HHLA uses in order to develop simulation models based on DESMO-J. Section 5 describes how DESMO-J and COCoS fit the requirements presumed in section 2 by referring to concrete model implementations. Section 6 finally discusses the experiences made in combining logistic simulation and device emulation by presenting benefits and limitations.



Problem category
Teminal operation systems development
On line control
Operational planning
Terminal design
Emulation
Operative and Tactical Simulation
Strategic Simulation
Design
Implementation and realization
Commissioning and Operation
Phase in terminal life cycle

Figure 2: Correlation of simulation and emulation with container terminal planning problems and life cycle

2. SIMULATION VS. EMULATION

In principle there are two different application scenarios for using simulation in context of container terminals. The traditional one is executing logistic simulation experiments to compare different handling strategies or to investigate the terminal layout. A completely different approach is to use simulation technique for testing terminal operating systems (TOS). We speak of emulation in this context. We depict differences of simulation and emulation in this section.

2.1. Definition

The main difference between logistic simulation and device emulation is that not the modelled system – the terminal – is subject of the investigation, but a control system, in this case a real TOS which is linked to the model. Emulation is used to imitate a real system; it offers an interface so that the control system cannot distinguish between a model and reality. This intends to already examine operability of a TOS before the employment at a real terminal. We denote emulation to be a subset of simulation.

(Auinger et al. 1999) proposed using simulated systems in combination with real systems in different ways. Figure 3 illustrates the different possibilities to combine simulation models with reality. Arrow 1 describes testing all components in live respectively prototype mode, not using simulation at all. This is the most realistic way of testing, but also the most expensive, furthermore dysfunctions can cause maximum damage. Arrow 2 is near to classical simulation view, without interaction with reality. This can be used for logistical simulation described above. Arrow 3 shows a kind of simulation called Reality-in-the-loop or Real-Time-Control. Here the material terminal hardware is tested with a model of a control system. (Schütt and Hartmann 2000) described how HHLA tested AGVs (Automated Guided Vehicles) with a virtual control system during CTA's planning phase. Arrow 4 describes the way of coupling simulation with reality which we call emulation and this paper is about: A simulated terminal system is piloted by a real TOS, so that the TOS can be tested with help of a terminal model.

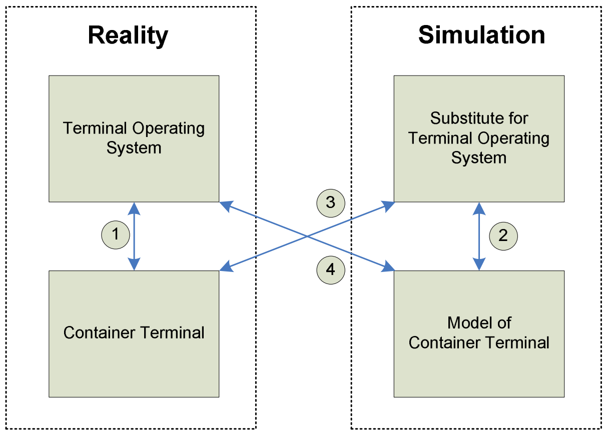

Reality
Simulation
Terminal Operating System
Substitute for Terminal Operating System
Container Terminal
Model of Container Terminal
1
2
3
4

Figure 3: Possible combinations of reality and simulation (based upon Auinger et al. 1999)

2.2. General Requirements

Logistic simulation is often embossed by bird's eye view; we look at the behavior of a terminal in general, so we can abstract from many details and implement the model with a top-down approach. In contrast, device emulation always depends on worm's eye view. We have to implement models with a bottom-up approach that is to say that all activities on the terminal have to be mapped from the view of emulated components, for instance quay cranes or AGVs.

Beyond this, the needed level of detail for emulation requires more time and effort than conventional simulation models. For example, in logistic simulation models, only the time span is relevant, which a quay crane needs for picking up a container on landside and setting it down onto the ship, but when testing a TOS with help of emulation models, a lot of intermediate steps have to be modeled auxiliary. For example, a dual-trolley quay crane owns two trolleys, which get individual briefings by TOS and have to interact for dispatching the container (see section 5.2 for details). The TOS needs information about the exact position of these trolleys, for this reason modeling the kinematic behavior of mobile devices becomes necessary. Therefore, the technical characteristics of the devices have to be investigated and mapped exactly into the model, which is unnecessary in simulation models at all.

Fundamentally in emulation, interfaces for communicating with the Control System (in case of harbor simulation: the TOS) are needed. In general, these are the same interfaces, which are implemented by local device drivers of real terminal components. In most cases, we deal with TCP/IP-based socket communication, and there is a set of telegram types, which have to be implemented. These telegrams may brief the device for doing a selected task or may request the status of the device. In other direction, telegrams which have to be sent from the device to the TOS may contain the status of the device, signal finished jobs or announce error states.

This leads us to another main difference between simulation and emulation. In simulation experiments, time spans that do not include any events will be skipped; duration of a simulation run depends on CPU-speed. By contrast in emulation, the advancement of simulation time has to be adjusted to real-time. Since the TOS is a real-time system, which assumes that the processing of jobs consumes time, and since we do not want to change this behavior for testing the TOS, we need to slow down simulation. The challenge of doing this is to find an approach, which delays the advancement of execution time, and nevertheless allows the insertion of new processes which were commissioned by the TOS over the communication interface; i.e., if the next scheduled process is estimated to start in a selected time span, the emulation model may not advise the thread to sleep until just before the end of this time span, because it is possible that a new upcoming task will have to be started prior to the process the scheduler is currently waiting for.

2.3. Manual- or automatic-driven experiments

The way of executing emulation experiments, differs fundamentally from logistic simulation experiments. Because of real-time conditions, the number of executed experiments is substantially smaller, and the analysis of results is extended by methods that are inappropriate in logistic simulation. In the following, we distinguish two types of experiments: manually driven and automatically driven tests.

During the course of the manual experiments, a human tester takes the role of the dock workers and operators of package items. He achieves certain tasks, and examines whether the TOS is reacting correctly to his interactions. Therefore, the model has to provide another interface, which is used for user interaction. All the actions, which dock workers and operators may accomplish on the terminal and thus are not induced by the TOS, like lashing a container or entering a danger area, have to be mapped to this interface. Additionally, all information about component states, which could be in interest for the tester, has to be accessible via this interface. We call it the manual interaction interface.

The tester should get access to this interface in scope of a Graphical User Interface (GUI). This can be realized via buttons, dropdown boxes and input fields, which should give feedback to user action. In this context, the visualization of temporary state of model becomes substantial. This can either be table-based summing-ups of model components. Or, which is quite more ergonomic, via 2d- or 3d-animations enriched by additional information boxes. Since the status of emulated devices has to appear just in time, a concurrent animation is urgently needed, in contrast to logistical simulation experiments, where a post-process animation is adequate.

Beside these manual tests, also automatic tests are to be possible. In case of testing a new version of a TOS, each time a lot of standardized workflows have to be validated. This needs not to be done by human testers. Therefore, another component has to be added, which automatically acts on the manual interaction interface. The behavior of this component has to be described by an easy changeable set of rules. We do not need any animation here, but we have to insert logging mechanisms, which outruns the trace logging of conventional simulation loggings in the level of detail, so that dysfunctions of the TOS can be found via an additional analysis component.

2.4. Combination

There are many intersections of the requirements of logistic simulation and device emulation, so a reuse of model components can help saving time and money. (Vorderwinkler et al. 1999) already proposed reusing simulation components from planning phases for device emulation. But some fundamental differences should not be disregarded.

According to section 2.3 device emulation and logistic simulation differ in their requirements. Table 1 summarizes the most important differences. However simulation and emulation models may share many aspects of model structure and implementation and accordingly requirements on underlying simulation frameworks. Note, that the behavior and the tasks of terminal components like quay cranes are basically the same in simulation and emulation. The aim of our investigations was to investigate whether a reuse of model components in logistic simulation as well as in device emulation is advantageous particularly in container terminal domain.

Table 1: differences between emulation and simulation

Logistic Simulation	**Device Emulation**
bird's eye or worm's eye perspective	worm's eye perspective
keep model as simple as possible	high level of detail
closed system	needs interfaces for interacting with other systems
simulation time advancement depends on CPU speed	simulation time advancement depends on real time
huge number of experiments	smaller number of experiments
automatic-driven experiments	manual- or automatic-driven experiments
analysis depends on reports, post-process animation	concurrent animation and/or exhaustive logging

3. DESMO-J

3.1. Main Functionality

DESMO-J (Discrete Event Simulaton Modeling in Java) is a simulation framework targeted at developers of discrete-event simulation models (Page and Kreutzer 2005). It offers an exhaustive environment to ease implementation of event-oriented, process-oriented or transaction-oriented models using the object-oriented high level language Java. As specialty, a combination of these different simulation world-views is possible, too. DESMO-J was developed by Department of Informatics at University of Hamburg and published under APL 2.0, which is a common open-source license. Binaries, source code, API and a tutorial can be found on http://www.desmo-j.de.

DESMO-J clearly separates between model and experiment. Experiments are realized via several black box elements, which are ready-for-use for executing simulation runs, like the Experiment class which encapsulates experimentation functionality, the Simulation Scheduler or the Simulation Clock. Therefore the modeler only has to implement the model itself and does not have to care about implementing the technical framework. DESMO-J also allows creating hierarchical model composition, since models can be embedded into other models as so-called sub-models.

Models are implemented via deriving several white-box components. The main task the modeler has to solve at this, is mapping model activity to events (in event-oriented worldview) or to processes (in process-oriented worldview). Events and processes are to describe activity of model components, and can be scheduled on simulation scheduler. Entities describe dynamic model components. In event-oriented worldview, an entity's state can be changed by events. In process-oriented worldview, processes are derived from entities, which means a process is an active entity itself, which can interact with other entities, and change its state on its own.

Additionally, there are a lot of helpful components, which can be used in the model building process without derivation, e.g. there are different kinds of queues for waiting entities, or a set of stochastic distributions, which can be used to generate pseudo-random numbers. Since the user can specify a seed for random-number generation, a replay even of stochastic-driven simulation runs is possible. There are several objects, which can be used for reporting and analysis of experimentation runs, e.g. there are statistic classes like counter, time series or histogram, and report classes which generate reports in a selected format (e.g. HTML or XML) for model components, which are of interest for the user (e.g. entities or queues). The user also may implement statistic or reporting-classes on its own.

3.2. Implementation Language

DESMO-J bases on the object-oriented language Java, which addresses a large community of programmers. Firstly, all of our students have a Java-programming expertise, so teaching simulation with help of DESMO J is in line with their curriculum. Furthermore, many companies focus on the usage of Java, too, because advantages of Java are e.g. the portability across diverse platforms, the huge amount on reusable libraries (e.g. the Java API, GUI-libraries like SWING or even third party libraries like DESMO-J) or the robustness since there are no dangling pointers or memory leaks like e.g. in C++. Furthermore, because of its handy syntax and the large amount of developing tools, implementing simulation models using Java-language reach a satisfactory tradeoff between ergonomics and flexibility. For example there is the Javadocs-tool for easy creation of documentations or there are open-source, extendable Integrated Development Environments like Eclipse, which offer more features for manipulating code than most code-editors of commercial simulation tools do.

Although, the Java Virtual Machine cannot deliver the performance as models compiled to machine code for the underlying platform, note that commercial simulation tools often use scripting languages, which are definitely slower than the pre-compiled java-byte code of DESMO-J.

The current version 2.1.4b of DESMO-J bases on the newest Java version 1.6, so it includes state-of-the-art programming methods like Generics (e.g. Queues can act as specialized queues for selected entities, which primarily induces type safety) or other advanced concepts.

3.3. Expandability

A very important aspect in using DESMO-J is its expandability. It is possible to integrate DESMO-J into other software frameworks, as well as to extend it by additional class libraries. In order to ease modeling and simulation in special application domains, it makes sense to develop domain-specific extensions, which provide objects for that particular domain. These can be quite general, more technical extensions like FAMOS, which permits Multi-Agent-Based simulation with DESMO-J (Knaak 2006), or very specialized approaches, for example to use a DESMO derivate as simulation component to simulate business production processes and calculate ecological input/output-balances (Wohlgemuth 2005). There are several DESMO-J extensions, which are beyond the scope of this paper, however. Because of the flexibility of ASL 2.0, DESMO-J is licensed under, it is possible to implement extensions or changes without licensing it under an open-source license. Therefore, companies like HHLA have the possibility to develop their own extensions, which have not to be published obligatorily, just like the extension presented in this paper.

3.4. Harbor simulation with DESMO-J

At HMS 2003, we presented our first DESMO-J harbor extensions for our research projects in harbour logistics (Page and Neufeld 2003). This class library offers three types of objects: There are dynamic, mobile, temporary objects like ships, trucks and trains; dynamic, mobile, permanent objects like cranes and van carriers; and stationary, permanent objects like holding areas, gates, jetties and yards. While the stationary objects are mostly implemented as derivation of DESMO-J's queues, the mobile objects are derived from DESMO J's processes. The framework distinguishes between the material flow, which deals with transportation and turnover over material goods, and the information flow, which deals with handling of job briefings. We examined the practicability of these early concepts in an empirical investigation within Containerterminal Burchardkai in Hamburg in co-operation with HHLA. Because these extensions are good for logistic investigations but too abstract for the purposes of emulating container terminals devices, we present a new harbor extension for DESMO-J called COCoS in this paper. However since HMS 2003, DESMO-J was used to implement harbour models by several authors. We give a summary on selected articles of foreign parties in the following.

(Asperen et al. 2004) investigated the impact of arrival processes, using the example of a chemical plant's jetty in Rotterdam (Netherlands). They assumed that a company can affect the ship arrival times with help of tactical sales plans and compared four different arrival processes. Initially they used a commercial simulation tool for their investigations, later on in the project they changed to DESMO J. They assigned the following reasons: Limitations in scripting language of the commercial tool made it very hard to implement complex models, while Java-based DESMO J offered much more flexibility. In addition, communication with other components did not work satisfyingly, while DESMO J did not have any problems in that, which was very important for our intention of connecting a TOS to an emulation model, too. Last but not least they stressed the better runtime performance of DESMO-J (Aspeeren et al. 2004).

A team at the Christian-Albrecht-University in Kiel (Germany) compared different dispatching strategies for AGVs at CTA in co-operation with HHLA. With help of DESMO J, they built up a "simple model with short run times which can easily be configured and produces realistic results" (Briskorn and Hartmann 2005).The subject of their investigations were two variants of job assignment. In the first case, only AGVs were considered which were currently idle; in the second case, they also considered AGVs which would finish their task within a certain time. These strategies were tested within five different scenarios, with 100 simulation runs per scenario and strategy. (Briskorn and Hartmann 2005; Briskorn et al 2007)

A working-group at the Bleking Institute of Technology (Sweden) was very experienced with simulating logistic aspects of horizontal transport of container terminals. For example they identified the number of cassette-based AGVs that were required to achieve an optimal workload for quay crane for a given terminal (Henesey et al 2006a); they evaluated dispatching strategies for AGVs (Kosowski and Persson 2006); and compared cassette-based to traditional AGV-Systems (Henesey et al 2006b). They declared that they preferred DESMO-J, since it allowed the suitable approach to implement terminal components as process entities, which had their own life-cycles, properties and behavior, and which coordinated their tasks using Contract Net protocol. Furthermore, simulation experiments were able to be executed via command prompt and reports were created automatically (Kosowski and Persson 2006, p. 31; Henesey et al 2006a).

All these authors used DESMO-J for logistic simulation; we did not find any indications that DESMO-J had been used in emulation context so far. Therefore, the intention in our paper is to show, that using DESMO-J as emulation engine is definitely possible.

4. COCOS

As well as the examples for a usage of DESMO-J in harbour-simulation mentioned above, the COCoS-framework developed in (Brandt 2007) has originally been designed for logistic container terminal-simulation. This relationship and correlating application-domains are illustrated in Figure 4.

As an extension of DESMO-J for the development of integrated container terminal simulation models, COCoS can be used to compare different terminal-layouts or to evaluate the impact of different handling equipment or control strategies on key figures of terminal productivity. COCoS provides black and white box concepts for modelling and implementing such models.

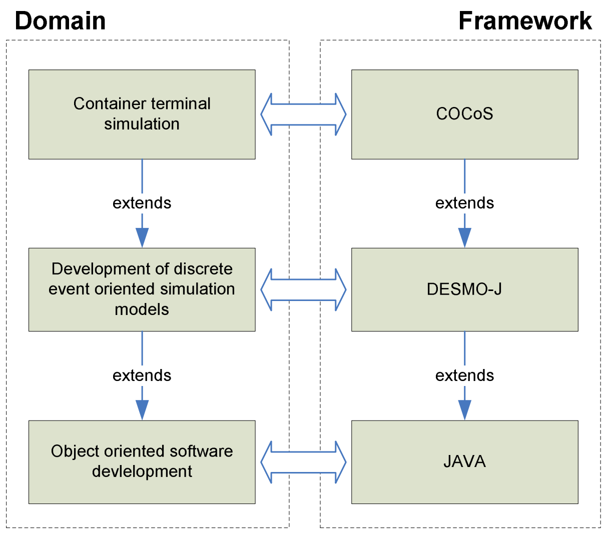

Domain
Framework
Container terminal simulation
COCoS
extends
extends
Development of discrete event oriented simulation models
DESMO-J
extends
extends
Object oriented software devlelopment
JAVA

Figure 4: COCoS and Application-Domains (Brandt 2007)

4.1. Framework-architecture

A container terminal can be conceptually reduced to a collection of enclosed subsystems that are connected by an information and a material flow. Accordingly container terminal simulation models should be composed of enclosed model-components representing theses subsystems (Simulation Building Blocks, see Verbraeck (2004)). To achieve this aim COCoS provides an integration architecture for the combination, connection and communication between reusable model components as well as a basic architecture for the components themselves.

Basic relationships within the framework-architecture are shown in Figure 5. Components representing enclosed terminal subsystems are characterized as functional components (see section 4.2). Unlike these, the framework's technical components do not represent parts of the simulated system but provide services for pure technical aspects of a simulation model such as visualization or statistics. These reusable functional and technical components have been built upon the framework's core which contains all basic framework elements and upon several system libraries which form the framework's technological foundation.

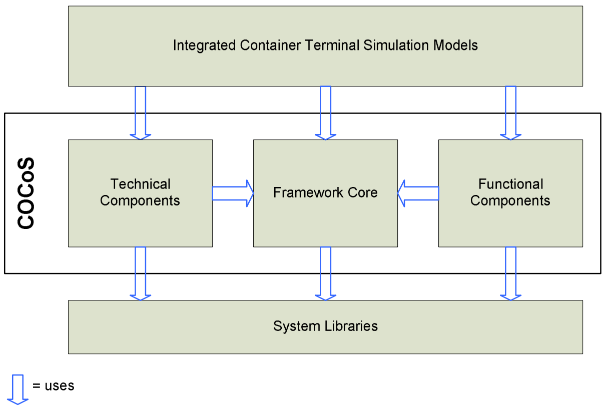


Figure 5: Framework architecture of COCoS (Brandt 2007)

4.2. Functional components

Functional components represent the container terminal subsystems such as quay-cranes, gantry cranes or horizontal transport systems. COCoS defines the architecture of functional components to assure their integrability and reuse in the context of different container terminal simulation models.

Multi-tier architectures can lead to advantages in loose coupling and reuse of software components. COCoS's functional components follow this important principle by implementing a three-layered architecture in order to transfer the mentioned advantages to the context of container terminal simulation. Hence each layer encapsulates a prominent scope typically existing in a container terminal simulation model, especially to refine a model wide separation of logical and physical aspects as described in (Pidd and Castro 1998).

The logical layer is responsible for planning decisions, resource management and task-handling mechanisms such as task-splitting, task-optimization or task-allocation. Through this the logical layer connects a functional component to a model's information flow.

In contrast, the material flow layer implements the physical aspects of a functional component. In terms of container terminal simulation, the material flow layer is mainly responsible for the handling and transportation of containers, thereby consuming simulation time for these physical operations. The communication layer serves as an extension point for adding additional interfaces to a functional component that are not supported by the standardized model architecture. This feature becomes especially important when embedding a functional component within a device emulator as described in section 5.2. But a functional component doesn't require all of the layers described above, given that pure information flow components that only provide the coordination of functionality, that may also be required in constructing a container terminal simulation model.

According to (Pidd and Castro 1998) the fast and secure development of complex simulation models through combination of model components especially requires a coupling scheme which specifies the communication between model components as well as the way they are connected. Therefore, COCoS contains mechanisms for the loose coupling of of functional components within an integrated container terminal simulation model or more precisely within the model's information and material flow. These aspects are to be described within the next two sections.

4.3. Information flow modeling

More than simply providing the basic elements needed for the development of container terminal simulation models, COCoS also supports the conceptual modelling process itself by providing a generic task-handling concept that standardizes the model's information flow as a flow of generic task-objects carrying sets of task-attributes.

Functional model components serve as task-handlers and primarily communicate by assigning tasks to each other accordingly. The underlying hierarchical model structure is defined by a model-wide XML-based task-model which can be considered as an executable coupling scheme within the model's information flow. More precisely a task-model specifies how super-tasks are split into sub-tasks and how task-attributes have to be passed on or generated thereby. The task-model also specifies targets for the generated sub-tasks as well as intermediate sequential relations. During their processing, the start and the completion of tasks are reported to their original sender. In order to execute a task model, COCoS provides black-box components which process the XML-based task-model-definition during experimentation.

By that task-models can be used to conceptually model the main information flow within an integrated container terminal simulation in an early phase of model development, which has also been proposed in (Robinson 2006).

4.4. Material flow modeling

The model's material flow is finally responsible for physical execution of tasks generated by the information flow. COCoS complements the described principles of information flow modeling by providing black- and white-box components for modeling and implementing these physical processes within a functional component's material flow layer. COCoS also contains basic resource classes like horizontal transporters routed on a graph based path net or crane components that can be combined to complex crane systems including a precise kinematic simulation. All these components follow DESMO-J's process oriented modeling style as it is set as default for implementation of functional components in COCoS. Material flow resources and their operational states are managed by the component's material flow layer making them available to the logical layer which then assigns tasks to these resources. Besides this connection to the information flow, material flow layers are interconnected by linking physical component layouts within an integrated terminal layout in order to facilitate container transportation in between components. This is realized by synchronous and asynchronous handover mechanisms that serve as the only coupling mechanism between functional components or their material flow layers respectively.

4.5. Technical components and model integration

Technical components are not part of the model logic and therefore do not follow the described architecture of functional components. Nevertheless COCoS defines architectural principles for important technological model aspects as well as for the support of the development of reusable and exchangeable technical components. The most important aspects are model visualization, model statistics, graphical user interfaces (GUI) or logging. In addition COCoS makes use of aspect oriented programming techniques (AOP) to separate technical and logical concerns as far as possible during the development process.

Functional and technical components are integrated to a container terminal simulation model by extending a centric COCoS model class. This model class is a direct descendant of the DESMO-J model class and thus provides the connection to DESMO-J's experimentation environment and simulation engine. It also serves as a model context for included functional components by offering common technical and functional services such as visualization, object and layout management or the input of scenario data.

5. COCOS APPLIED

At first this section briefly summarizes how COCoS has been used for the development of logistic simulation models and then proceeds with a more detailed description of the framework's application in developing device emulators.

5.1. Logistic Simulation with COCoS

As mentioned above COCoS has originally been developed for modeling and implementing integrated container terminal simulation models for the investigation of logistic problems. The process of developing such models can be summed up as follows:

1. Modeling the information flow in the form of a model-wide task-model
2. Selection or development of appropriate functional components for the task model's execution
3. Selection or development of technical components for statistic analysis, visualization, user-interface et cetera.
4. Integration of functional and technical components within a COCoS-Model that is controlled by the task-model.

By iterating through these steps and varying model components and model configuration accordingly, different options of terminal design or control strategies can be explored.

Figure 6 shows a screenshot of a simple proof-of-concept-model during eperimentaion. The model features a simple pathnet based horizontal transport component, a single trolley quaycrane component and a storage-crane component as well as technical components that provide services for animation, logging and collecting task-based model statistics in a relational database. While the COCoS-Control-Center serves as graphical user interface to the model, DESMO-J's simulation engine is driving model execution inside.

Although most of these components are kept very simple, the model is able to show that COCoS's integration concepts work excellent for modeling, assembling and execution of logistic simulation models.

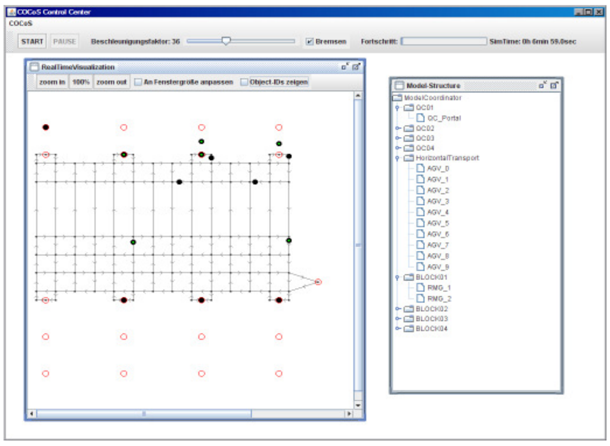

Beschleunigungsfaktor: 36 · ✓ Bremsen · Fortschritt: · SimTime: 0h 6min 59.0sec

Figure 6: Logistic simulation model implemented with COCoS

5.2. Emulation with COCoS

Beyond this usage on logistic problems, functional terminal-components such as quay cranes or gantry cranes can be extended to device emulators by increasing detail level and adding specific communication- or user-interfaces. Related concepts have been proved in practice and will be described in this section.

5.2.1. General concepts

Different general, mainly technical features needed for device emulation had to be integrated directly into COCoS. Slowing down simulation time is a prominent example, since implementation of real-time functionality into DESMO-J has been in progress, but is not yet part of the official published DESMO-J release. To bridge this, COCoS was extended by a simple but adequate real-time control process that synchronizes real-time and simulation time by means of an adjustable time-synchronization cycle. Derived from DESMO-J's white-box simulation process, the simulation time control process basically linearises the discrete event oriented simulation time variation by constantly (re-)scheduling real-time consuming sleep-events. These sleep events are completely independent from the remaining functional model logic, rather their only purpose is to fill up the models discretionary time gaps. Within each cycle the advances of simulation time and real time are compared measuring the time divergence that is generated by computing time for model execution which is not reflected by the simulation time at all. This divergence is factored into the calculation of the next sleep delay in order to prevent simulation time and real time from continuously diverging. Cycle time and accuracy of this process as well as an arbitrary acceleration factor can be configured through model parameters. These parameters indicate the maximum response-time for the emulated devices for messages on their communication layer. For logistic simulation experiments real-time functionality can be turned off, but it can also be used to observe model behaviour by enabling real time visualization.

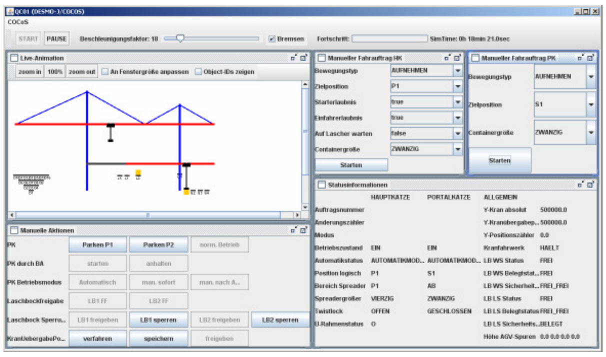
The second fundamental feature of an emulation framework is a concept for communication with an control systems and a human tester who want to manipulate a device emulator during run time. Since, we neither wanted to predefine the technical interface nor message types supported, we aimed for an abstract solution here. In addition to information flow layer and material flow layer, we implemented optional communication layers, which could be freely assigned when instantiating the device emulator. For example, there are communication layers for communicating via TCP-connection or via JMS-messages. These layers interpret incoming messages; depending on that, they choose a method to call on a device-specific task adapter, which creates the tasks according to the message. Logging the traffic on communication interface can be very helpful for debugging both the model and the TOS. The task adapter is also used by manual interaction interface, so that the human tester can directly insert tasks. If an event occurs which possibly postulates an outgoing message, the information flow layer will have to detect if a communication layer is allocated, and in this case call the according method on its interface.

5.2.2. Emulating a dual-trolley quay crane

As an instance, we outline the model-building process of a dual-trolley quay crane in this section. In Hamburg, these crane types are located at CTA. The two trolleys for handling containers are characteristic.

There is a manually driven main-trolley, which can reach every position under the quay crane, i.e. the ship or the quay lanes. Thus, the main-trolley is able to work autonomously without interacting with the second trolley. The driver of the main-trolley is also able to move the whole quay crane along the quay, which allows him to pick up all the containers from a ship in sequence. The second trolley is an automatic-driven portal-trolley, which gets its instructions directly from the TOS. It can only reach quayside transfer lanes and a special platform, called lashing platform, where it can set down the containers, which he picked up from the transfer lanes. From there, the main-trolley can retrieve them to set them down on ship. On the lashing platform, there are working people, who are preparing containers for transport; none of the trolleys is allowed to enter this danger area, if people are left on the platform.

This model component depicts the need of the manual interaction interface (see section 2.3). The behaviour of the lashing platform crew, the driving of the main-trolley and the possibility of deactivating the portal-trolley, everything has to be mapped in scope of this interface. Figure 7 shows the GUI of the COCoS' experimentation environment. Currently, there is a simple concurrent animation window showing a dual-trolley quay crane, one window containing some status information, and several windows providing access to the manual interaction interface. Thus, there are two possibilities to create tasks: Creation by the TOS via communication layer or by the human tester via GUI.



Beschleunigungsfaktor: 18

Fortschritt: SimTime: 0h 18min 21.0sec

Manuelle Aktionen				
PK	Parken P1	Parken P2	norm. Betrieb	
PK durch BA	starten	anhalten		
PK Betriebsmodus	Automatisch	man. sofort	man. nach A...	
Laschbockfreigabe	LB1 FF	LB2 FF		
Laschbock Sperru...	LB1 freigeben	LB1 sperren	LB2 freigeben	LB2 sperren
KranUebergabePo...	verfahren	speichern	freigeben	

Manueller Fahrauftrag HK

Bewegungstyp	AUFNEHMEN
Zielposition	P1
Starterlaubnis	true
Einfahrerlaubnis	true
Auf Lascher warten	false
Containergröße	ZWANZIG

Manueller Fahrauftrag PK

Bewegungstyp	AUFNEHMEN
Zielposition	S1
Containergröße	ZWANZIG

Statusinformationen

	HAUPTKATZE	PORTALKATZE	ALLGEMEIN	
Auftragsnummer			Y-Kran absolut	500000.0
Änderungszähler			Y-Kranübergabep...	500000.0
Modus			Y-Positionszähler	0.0
Betriebszustand	EIN	EIN	Kranfahrwerk	HAELT
Automatikstatus	AUTOMATIKMOD...	AUTOMATIKMOD...	LB WS Status	FREI
Position logisch	P1	S1	LB WS Belegtstat...	FREI
Bereich Spreader	P1	AB	LB WS Sicherheit...	FREI_FREI
Spreadergrößer	VIERZIG	ZWANZIG	LB LS Status	FREI
Twistlock	OFFEN	GESCHLOSSEN	LB LS Belegtstatus	FREI_FREI
U-Rahmenstatus	0		LB LS Sicherheits...	BELEGT
			Höhe AGV-Spuren	0.0 0.0 0.0 0.0 0.0

Figure 7: COCoS User Interface for device emulation)

A device like a dual trolley quay crane consists of several device parts, such as trolleys, lifts or spreaders. These device parts can be found on other terminal components, too. For example a van carrier contains a lift as well as a spreader, but their kinematic characteristics differ from the quay crane's lift and spreader. These detail information often is irrelevant in simulation studies, but in scope of emulation they are of much importance. That is why, we designed our model components of reusable and exchangeable device parts, which can be configured via XML-files. So there is the possibility to use very detailed device parts mapping the exact kinematic behaviour for emulation, or more simple device parts, whose time consumption depends on stochastic distributions for simulation experiments.

6. CONCLUSION

Combining logistic simulation and device emulation without overcharging the COCoS framework and the developed quay crane component was a challenging task. At first, it was important to clearly separate the component's layers for an efficient embedding of different communication interfaces. We supported this approach by developing a generic task adapter that could be used to translate arbitrary messages to COCoS task objects. This task adapter was also used to connect the manual interaction interface that was required for manually driven experiments. To satisfy the real-time requirements of device emulation we implemented an adequate time control process as a part of COCoS.

With real-time functionality a concurrent animation of simulation state became possible. This was urgently needed for manual device testing, but also pointed out to be very helpful for building, debugging and observing logistic simulation models.

Besides these more technical aspects, the main problem in combining logistic simulation model building and device emulation model building was the level of detail to decide on. To combine logistic simulation and device emulation, a precondition was to take worm`s eye perspective and build up model components in process-oriented world view, which is adequate in such cybernetic models at all. Nevertheless, there remain some substantial differences in requirements on detail level. For example, in logistic simulation the movements of components could be approximated via calculating time consumption by means of stochastic distributions, while exact kinematics are required in emulation context. Since devices managed by the material flow layer of a functional model component are exchangeable, it is advisable to use less detailed devices in simulation context, but more detailed devices in emulation context.

By bridging the described differences and combining logistic simulation and emulation it becomes possible to share the advantages of component based software development between the building processes of logistic container terminal simulation and device emulation models. Synergy effects emerging from exchanging and reusing functional and technical model components reduce effort and costs of model development and lead to improved model quality and a consolidated model architecture. It could be shown that DESMO-J and COCoS provide an extendible basis for this synergistic combination.

REFERENCES

Asperen, E. van, R. Dekker, M. Polman, H. de Swaan Arons. 2004. *Arrival processes in port modeling: insights from a case study.* Available from:

http://publishing.eur.nl/ir/repub/asset/1275/ei2004 16.pdf [accessed 31. March 2009]

Auinger, F., Vorderwinkler, M., Buchtela, G., 1999. Interface driven domain-independent modelling architecture for „soft-commissioning“ and „Reality in the Loop“. *Proceedings of the 1999 Winter Simulation Conference*, pp. 798-805, December 5-8, Phoenix (Arizona, USA).

Brandt, C., 2008. *Entwurf und Implementierung eines Frameworks zur Entwicklung von Containerterminal-Gesamtmodellen mit DESMO-J*. Thesis (Diploma), University of Hamburg.

Briskorn, D., Hartmann, S., 2005. Simulating dispatching strategies for automated container terminals. *Proceedings of the Annual International Conference of the German Operations Research Society (GOR).* September 7-9, Bremen (Germany).

Briskorn, D., Drexl, A., Hartmann, S., 2007. Inventory-based dispatching of automated guided vehicles on container terminals. In: Kim, K.H., Günther, H. eds. *Container Terminals and Cargo Systems*. Berlin, Heidelberg, New York: Springer.

Henesey, L., Aslam, K., Khurum, M., 2006a. Task Coordination of Automated Guided Vehicles in a Container Terminal. *Proceedings of 5th International Conference on Computer Applications and Information Technology in the Maritime Industrie*s. 2006, Oud Poelgeest (Netherlands).

Henesey, L., Davidsson, P., Persson, J.A., 2006b. Comparison and Evaluation of Two Automated Guided Vehicle Systems in the Transhipment of Containers at a Container Terminal, In: Henesey, L., 2006. *Multi-Agent-Systems for Containter Terminal Management*. Thesis (PhD), Blekinge Institute for Technology, 204-226.

Knaak, N., Kruse, S., Page, B., 2006. An Agent-Based Simulation tool for modelling sustainable logistic systems. *Proceedings of the iEMSs Third Biennial Meeting 2006*. July 9-13, Burlington (Vermont, USA).

Kosowski, P., Persson, O., 2006. *Development and evaluation of dispatching strategies for the IPSI™ AGV system.* Thesis (Master), Blekinge Institute of Technology.

Page, B., Kreutzer, W., 2005. *Simulating Discrete Event Systems*. Aachen: Shaker Verlag.

Page, B., Neufeld, E., 2003. Extending an object-oriented discrete event simulation framework in Java for harbour logistics. *Proceedings of International Workshop on Harbour, Maritime & Multimodal Logistics Modelling and Simulation*, pp. 79-85. September 18-20, Riga (Latvia).

Pidd, M., Castro, B., 1998. Hierarchical modular modelling in discrete simulation. *Proceedings of the 1998 Winter Simulation Conference*, pp. 383–389. December 13-16, Washington D.C. (USA).

Robinson, S., 2006. Conceptual Modeling For Simulation: Issues And Research Requirements. *Proceedings of the 2006 Winter Simulation Conference*, pp. 792–800. December 3-6, Montery (California, USA).

Schütt, H., Hartmann, S., 2000. Simulation in Planung, Realisierung und Betrieb am Beispiel des Container-Terminals Altenwerder. In: Möller, D. ed. *Simulationstechnik, 14. Symposium.* Hamburg: SCS, 425–430.

Steenken, D., Voss, S., Stahlbock, R, 2004. Container terminal operations and operations research – a classification and literature review. In: *OR Spectrum 26*, pp. 3–49. Berlin, Heidelberg, New York: Springer.

Vorderwinkler, M., Eder, T., Steringer, R., and Schleicher, M., 1999. An Architecture for Soft-Commissioning – Verifying Control Software by Linking Discrete Event Simulators to Real World Control Systems. *Proceedings of 13th European Simulation Multiconference*, pp. 191-198. June 1-4, Warsaw (Poland).

Verbraeck, A., 2004. Component-based distributed simulations: the way forward? *Proceedings of the 18th workshop on Parallel and distributed simulation,* pp. 141–148. May 16-19, Kufstein (Austria).

Wohlgemuth, V. 2005. *Komponentenbasierte Unterstützung von Methoden der Modellbildung und Simulation im Einsatzkontext des betrieblichen Umweltschutzes*. Thesis (PhD). University of Hamburg. Aachen: Shaker.

AUTHORS BIOGRAPHY

Bernd Page holds degrees in Applied Computer Science from the Technical University of Berlin, Germany, and from Stanford University, USA. As professor for Applied Computer Science at the University of Hamburg he researches and teaches in the field of Computer Simulation as well as in Environmental Informatics.

Philip Joschko studied Computer Science and Psychology at the University of Hamburg. Since he holds a diploma (MSc) degree in Computer Science he works as a research assistant and PhD candidate within the Center of Architecture and Design of IT-Systems at the workgroup of Prof. Dr. Page. Research interests are discrete event simulation, cooperative model building processes and the EU emission trading system.

Christopher Brandt studied Information Systems at the University of Hamburg with an emphasis on logistic simulation and distributed information systems. He holds a diploma (MSc) in information systems received for his thesis in which he developed the COCoS-framework for HHLA. Right now he works as a software architect for HHLA in the area of simulation, operations research and business intelligence.