THE ANNALS OF "DUNĂREA DE JOS" UNIVERSITY OF GALATI
FASCICLE III, 2012, VOL. 35, NO. 1, ISSN 1221-454X
ELECTROTECHNICS, ELECTRONICS, AUTOMATIC CONTROL, INFORMATICS

DESIGNING A MODULAR SIMULATOR FOR A NAVIGATION LOCK

Lucian-Florentin Barbulescu[1], Viorel Minzu[2]

[1]University of Craiova, Faculty of Automation, Computers and Electronics, Department of Computers and Information Technology, Craiova, Romania,
e-mail: lucian.barbulescu@cs.ucv.ro
[2]"Dunarea de Jos" University of Galati, Faculty of Automatic Control, Computers, Electrical Engineering and Electronics, Department of Automatic Control and Electrical Engineering, e-mail: viorel.minzu@ugal.ro

Abstract: When building a simulator for an industrial installation, a divide and conquer approach can have very good results. This means that the original installation is divided into smaller components which are simulated independently and then interconnected to obtain the final product. This paper presents the steps required for designing such a modular simulator for a navigation lock that allows the operators to view and control all phases of operation.

Keywords: distributed systems, navigation lock, modular simulators, dynamic systems, discrete event simulations

INTRODUCTION

The evolution of hardware systems and computer networks has led to a rapid development of distributed applications, in general, and parallel/distributed simulators, in particular. It is now very easy and relatively cheap to build a fast network of computers and use their combined power in order to obtain a system on which a parallel or distributed application can run. Today there are a lot of types of applications which are implemented to run in this environment, from databases to web applications. A special type of application that can run in this environment is the parallel/distributed simulator (Fujimoto, 2000)

One such application was presented by Barbulescu (2009) and can be used to build simulators for large industrial installations by means of a divide-and-conquer-like method. In other words the initial industrial installation, which can be viewed as a dynamic system, is "split" into smaller components, each being also interpreted as a dynamic system (Kulakowski, 2007).

For each component there can either be a new split into smaller entities or a software module will be implemented. This module can be derived from the abstract elementary simulator (Barbulescu, 2012), in which case the implementation is very simple and follows only the required functionality, or it must at least implement the interface "ElementarySymulator" described by Barbulescu (2012), Independently on the solution chosen those module must act as dynamic systems, that is they must accept external values as input variables and generate values as output variables. Those values are received from and sent to other modules from within the system.

Based on the specific of the simulated installation there can be other modules required, for example clock signal generators, graphical user interface etc. The task of selecting those additional modules is assigned to the designer of the simulator. He, based on his experience, must decide the required components in such a way that the system will work properly.

This paper was recommended for publication by Adrian Filipescu

When all modules are designed they must be interconnected by the means of the communication framework (Barbulescu, 2010). The simulator designer must also choose he number of communication channels and can apply the algorithm described by Barbulescu (2011).

When the architecture is defined the final stage consists on the implementation of each module, if they are not reused, and the deployment on the target computer or computers.

As a conclusion of the previous presentation, the development of a simulator based on the distributed system described by Barbulescu (2009) consists of four phases:

- **Phase 1**: The analysis of the installation that must be simulated and the definition of the required modules.
- **Phase 2**: The definition of the communication channels required to interconnect the modules in order to obtain the simulator
- **Phase 3**: The implementation of the modules that are not already present within the system
- **Phase 4**: The deployment of the simulator on the target computer or computers, based on the desired distribution.

In this paper there will be presented the first two phases needed when building a simulator for a navigation lock using the distributed simulation system.

THE NAVIGATION LOCK

A lock is a device for raising and lowering boats between stretches of water of different levels on river and canal waterways[1]. Locks are used to make a river more easily navigable, or to allow a canal to take a reasonably direct line across land that is not level.

The distinguishing feature of a lock is a fixed chamber in which the water level can be varied. At the ends of the navigation lock there are big gates that can block the water from entering or exiting the chamber. With both gates closed, the water level within the lock can be adjusted to match the canal water level on either side. Thus, a vessel entering the lock can be raised or lowered in order to enter the next level canal section.

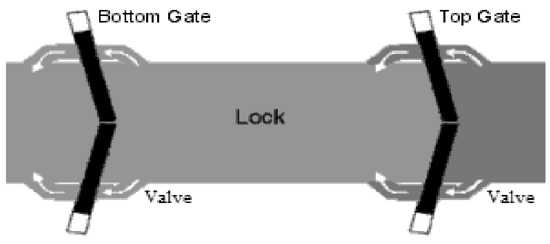

Fig.1. Navigation lock

Today there are several types of navigation lock built[2]:

- Simple locks, composed of a single chamber, as presented in Fig 1;
- Stepped locks, composed of several chamber placed one after another
- Twin locks, composed of two or more chambers disposed one near another.

The passing of a ship through a lock is called lockage or sluicing. Based on the direction of the ship the operation is named upstream-downstream lockage or downstream-upstream lockage if the ship passes from the upper side to the lower side or from the lower side to the upper side respectively.

In Fig 2 the upstream-downstream lockage of a ship is presented.

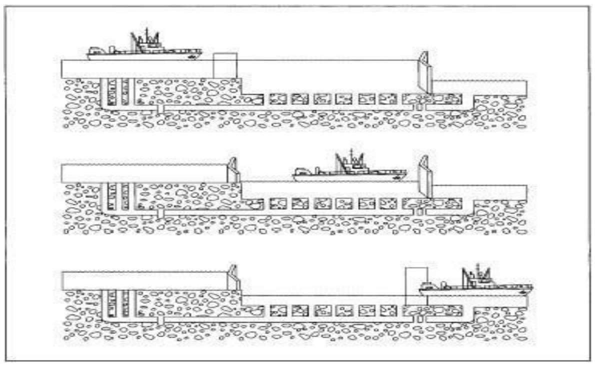

Fig.2. Upstream-downstream lockage operation

The steps required for this operation are:

- The upper valves are opened and the chamber starts filling with water
- When the level inside the chamber is equal to the level upstream, the upper gate is opened
- The ship enters the chamber
- The upper gate is closed
- The upper valves are closed
- The lower valves are opened and the chamber begins to empty

[1] *Lock* (Online) Available at: http://en.wikipedia.org/wiki/Lock_(water_transport)

[2] Ecluza (Online), Available at: http://ro.wikipedia.org/wiki/Ecluz%C4%83

– When the level inside the chamber is equal to the level downstream, the lower gate is opened
– The ship exists the chamber
– The lower gate is closed
– The lower valves are closed.

In Fig 3 the upstream-downstream lockage of a ship is presented.

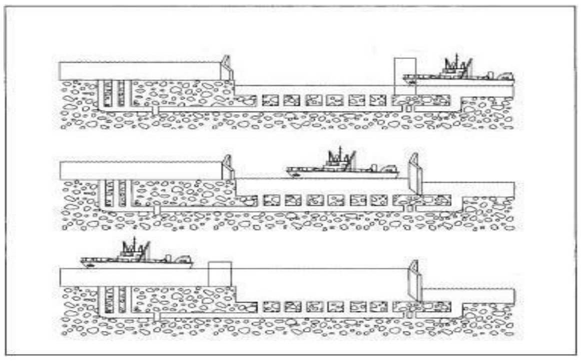

Fig.3. Downstream-upstream lockage operation

The steps required for this operation are:

– The lower valves are opened and the chamber begins to empty
– When the level inside the chamber is equal to the level downstream, the lower gate is opened
– The ship enters the chamber
– The lower gate is closed
– The lower valves are closed
– The upper valves are opened and the chamber begins to fill
– When the level inside the chamber is equal to the level upstream, the upper gate is opened
– The ship exists the chamber
– The upper gate is closed
– The upper valves are closed.

The simulator for this installation will consider all the above steps except the ship movement. This is not part of the current simulation and may be taken into consideration in a future version.

NAVIGATION LOCK SIMULATOR DESIGN

In this paper it is presented the design for a simulator for a navigation lock similar to the Romanian one existing at Iron Gates II. The Romanian navigation lock[3] at Iron Gates II is a simple lock with plane gates upstream and rotating gates downstream. The chamber is 310 m long and 34 m wide and it is filled and emptied by means of four engine-controlled plain vanes situated two upstream and two downstream.

[3] Hidroconstructia (Online). Available at: http://www.hidroconstructia.com/rom/proiecte.php?cmbBranch=5

1.1. Phase 1

The first step in building the simulator for the navigation lock presented above consists in splitting the installation into smaller components for which software modules based on the abstract elementary simulator (Barbulescu, 2012) can be designed and implemented. The components identified at this moment are:

– The plane gate (**PP**) situated upstream
– The rotating gate (**PB**), situated downstream
– The navigation lock chamber (**SAS**)
– The two upper vanes (**VU1** and **VU2**) situated upstream
– The two lower vanes (**VG1** and **VG2**) situated downstream

Of course, some of the above components can also be divided into smaller components like motors, pipes, levers etc., but this is not applied in this case because the result will be a very complicated simulation schema with very few advantages. It will result that, from the initial analysis of the installation, there are seven modules and four elementary simulators required, as the valve simulator can be reused within the schema.

The second step of this phase consists of identifying other modules that are required for the system to work properly. To do this the lockage procedures described in the previous chapter must be analyzed.

It can be observed that, during the lockage operation, the upstream and downstream water levels are required. Those values are needed when deciding that a gate can or cannot be opened and also are used to compute the amount of water that passes through the valves in order to fill or empty the lock chamber. Because the amount of water upstream or downstream is much bigger that the quantity that is at any moment within the chamber, it can be considered that, no matter how much water is lost or gained, the level will remain unchanged, thus those values can be defined as constants. However, since in real life, the water level can have some variations (like when a flood is happening) it was decided that two modules that simulate the water level upstream (**NAM**) and downstream (**NAV**) should be added. The two modules will be represented by instances of the same component that, at this moment, generates a constant value for the water level. If, in the future, it is required to simulate variations of this level, the existing component can be replaced with a new one without affecting the rest of the simulator.

Another observation that can be made is that, during the lockage operation, there are some time depending processes like the opening and closing of the gates, the opening and closing of the valves and the filling and emptying of the lock chamber. In order for those operation to be properly simulated by this system it is required the usage of the existing component named clock generator (**CLK**). This component is already implemented and, once started, it will emit periodically a message that contains the elapsed time interval to the interested modules.

The distributed simulation system makes a separation between the functional part and the user interface. Because of this and in order to be able to offer an interaction with the users, it is required to have the user interface (**UI**) component. This component is already implemented and uses module-defined panels that can be placed together in order to build elaborate and user-friendly interfaces.

The last observation that can be made by analyzing the lockage operation is that some equipment depends on the state of others. For example, in order to be allowed to open the upper gate, the water level inside the chamber must be equal to the upstream level and the upper valves must be opened or to be able to operate the upper valves, both gates and the both lower valves must be closed etc. This can of course be implemented within each component, but it will lead to some awkward restrictions. For example, the valve simulator must be aware of the status of two gates and two other valves, which can limit its reusability. It will be very hard to use the same component within a schema for an installation that requires a similar valve but that doesn't have the same restriction. Also, this component will not be usable for a similar navigation lock simulator where there are more than two upper or lower valves. For these considerations it was decided that it will be more appropriate to add an enable/disable variable as input for the modules that depend on others and to generate those commands within a separate component named control and command (**CC**). This module contains a state machine that can decide, based on the others statuses and on the water level within the chamber or at upstream and downstream level, which entity is active and can be maneuvered and which is not. This solution has the advantage of keeping the other modules focused on their specific functionalities. Also, if a simulator for a navigation lock that contains more valves or more gates must be implemented, only this component has to be updated, the others remaining unchanged.

1.2. Phase 2

After the first phase of the development of the simulator for the navigation lock it was decided that the number of components required is twelve. Each of those components will have input and/or output variables and their values will be exchanged in order to obtain the desired functionality.

In phase two there must be decided what are the variables exchanged by the components and how are the interconnections realized.

In the twelve tables below there are presented the input and output variables for each component.

Table 1 The input and output variables for the valves and gates (VU1, VU2, VG1, VG2, PP, PB)

Module	**Input Variables**	**Output Variables**
VU1	cmd_{vu1}: command for VU1 t: time interval p_{sas}: pressure in chamber p_{amonte}: pressure upstream	s_{vu1}: state of VU1 q: water quantity poz_{vu1}: position of VU1
VU2	cmd_{vu2}: command for VU2 t: time interval p_{sas}: pressure in chamber p_{amonte}: pressure upstream	s_{vu2}: state of VU2 q: water quantity poz_{vu2}: position of VU2
VG1	cmd_{vg1}: command for VG1 t: time interval p_{aval}: pressure downstream p_{sas}: pressure in chamber	s_{vg1}: state of VG1 q: water quantity poz_{vg1}: position of VG1
VG2	cmd_{vg2}: command for VG2 t: time interval p_{aval}: pressure downstream p_{sas}: pressure in chamber	s_{vg2}: state of VG2 q: water quantity poz_{vg2}: position of VG2
PP	cmdpp: command for PP t: time interval	spp: state of PP pozms: left engine position pozmd: right engine position
PB	cmdpb: command for PB t: time interval	spb: state of PB pozps: left gate position pozpd: right gate position

Table 2 The input and output variables for the lock chamber, water levels, clock and control and command (SAS, NAM, NAV, CLK, CC,)

Module	**Input Variables**	**Output Variables**
SAS	q: water quantity lost	P_{sas}: pressure in

	or gained	chamber h_{sas}: water height
NAM	q: lost water quantity	p_{amonte}: pressure up. h_{amonte}: water height
NAV	q: gained water quantity	p_{aval}: pressure down. h_{aval}: water height
CLK	S: start clock	t: time interval
CC	h_{amonte}: water height upstream s_{pp}: state of PP s_{vu1}: state of VU1 s_{vu2}: state of VU2 h_{sas}: water height in chamber s_{vg1}: state of VG1 s_{vg2}: state of VG2 s_{pb}: state of PB h_{aval}: water height downstream	cmd_{pp}: cmd for PP cmd_{vu1}: cmd for VU1 cmd_{vu2}: cmd for VU2 cmd_{vg1}: cmd for VG1 cmd_{vg2}: cmd for VG2 cmd_{pb}: cmd for PB

Table 3 The input and output variables for the UI

Module	Input Variables	Output Variables
UI	h_{amonte}: water height upstream s_{pp}: state of PP poz_{ms}: left engine position poz_{md}: right engine position s_{vu1}: state of VU1 poz_{vu1}: pos of VU1 s_{vu2}: state of VU2 poz_{vu2}: pos of VU2 h_{sas}: water height in chamber s_{vg1}: state of VG1 poz_{vg1}: pos of VG1 s_{vg2}: state of VG2 poz_{vg2}: pos of VG2 s_{pb}: state of PB poz_{ps}: left gate pos poz_{pd}: right gate pos h_{aval}: water height downstream	S: start clock cmd_{pp}: cmd for PP cmd_{vu1}: cmd for VU1 cmd_{vu2}: cmd for VU2 cmd_{vg1}: cmd for VG1 cmd_{vg2}: cmd for VG2 cmd_{pb}: cmd for PB

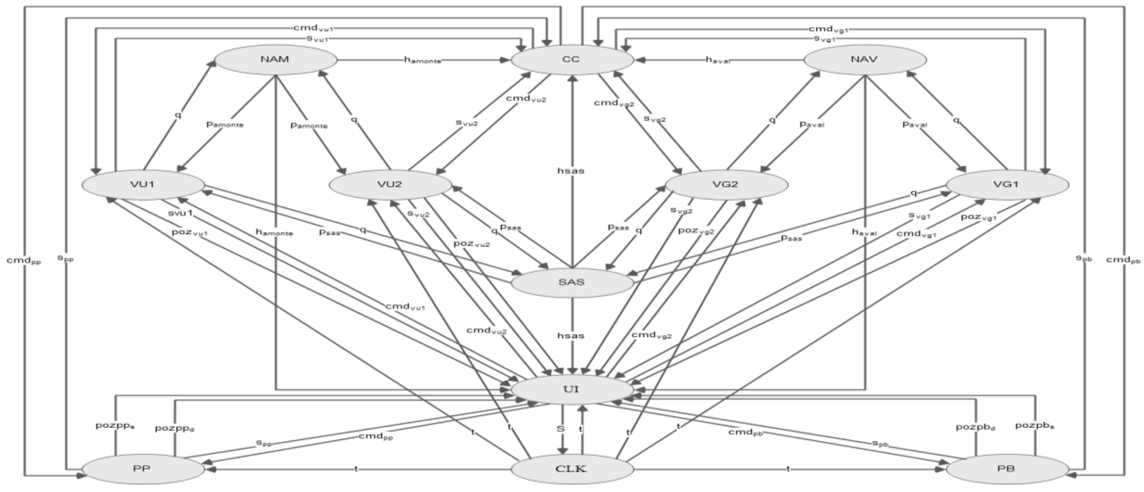

Fig.4. The modules used to build the simulator for the navigation lock. Each inter-module communication is one-to-one.

The variables described in the above tables must be exchanged between the system modules in order to obtain a smooth simulation. A graphical representation of the twelve modules and the variables exchanged between them is presented in Fig. 4

Based on this representation, the number of communication channels needed is 62, which is a very big number and implies that many hardware resources will be used for the communication framework instead of the actual modules needed for the simulator.

In order to reduce the number of communication channels the algorithm described by Barbulescu (2011) can be applied. This ensures that one component will send the value of any output variable on only one communication channel and it will receive the values for the input variables on as few as possible communication channels.

After the algorithm is applied, the number of communication channels is reduced to only 15.

Table 4 Communication channels used by the simulator

Channel	Variables	Source Modules	Destination Modules
1	s_{vu1}, s_{vu2}, s_{vg1}, s_{vg2}, s_{pp}, s_{pb},	VU1 VU2	UI CC

	h_{amonte}, h_{aval}, h_{sas}	VG1 VG2 PP PB NAM NAV SAS	
2	poz_{vu1}, poz_{vu2}, poz_{vg1}, poz_{vg2}, $pozpp_s$, $pozpp_d$, $pozpb_s$, $pozpb_d$,	VU1 VU2 VG1 VG2 PP PB	UI
3	q	VU1 VU2	NAM SAS
4	q	VG1 VG2	NAV SAS
5	p_{amonte}	NAM	VU1 VU2

Table 5 Communication channels used by the simulator

Channel	**Variables**	**Source Modules**	**Destination Modules**
6	p_{aval}	NAV	VG1 VG2
7	p_{sas}	SAS	VU1 VU2 VG1 VG2
8	cmd_{vu1}	UI CC	VU1
9	cmd_{vu2}	UI CC	VU2
10	cmd_{vg1}	UI CC	VG1
11	cmd_{vg2}	UI CC	VG2
12	cmd_{pp}	UI CC	PP
13	cmd_{pb}	UI CC	PB
14	t	CLK	VU1 VU2 VG1 VG2 PP PB UI
15	S	UI	CLK

After this analysis all steps needed to design the simulator are finished. From this point forward all the components must be implemented and configured to work as described in the tables 4 and 5.

CONCLUSIONS

The distributed simulation system provides a framework and a methodology which can be followed in order to build simulators for a navigable lock. The number of independent components required is equal to 8, although the number of modules used is 12. This difference is because for some components more than one instance will be used.

This analysis proved that the system can be used for designing modular simulators

FUTURE WORK

The implementation of all modules presented in this paper is the next step required for the validation of the distributed simulation system. Also, based on the observed results, the system can suffer modifications to improve the performance and to allow the development of simulators for more complicated installations.

REFERENCES

Barbulescu, L.F., Lungu M. and Andrei D.O. (2009), Distributed system for industrial simulation, *Annals of the University of Craiova*, **Volume 6(33), issue. 1**, pp. 1-5

Barbulescu, L.F. (2010), Functional analysis of a communication framework used in a modular simulator, *14th International Conference on System Theory and Control*, **Proceedings**, pp. 62-66

Barbulescu, L.F. (2011), An algorithm designed to determine the optimum number of communication channels in a modular simulator, *Bulletin of the Polytechnic Institute of Iasi, Automatic Control and Computer Science Section,* **Nr. LVII (LXI), Fasc. 3**, pp. 21-32

Barbulescu, L.F. (2012), The abstract elementary simulator – the base component of a modular simulator, *16th International Conference on System Theory and Control,* **Proceedings**

Fujimoto R.M. (2000), *Parallel and Distributed simulation Systems*, New York, John Wiley & Sons,

Kulakowski, B.T., Gardner J.F. and Shearer J.L., *Dynamic Modeling and Control of Engineering Systems* 3rd ed., New York, Cambridge University Press, 2007, ch. 1.