MODELING, IDENTIFICATION AND CONTROL, 1986, VOL. 6, NO. 4, 257–273
doi:10.4173/mic.1985.4.6

Interactive modeling and simulation using distributed computers and graphical aids†

STEINAR SAELID‡

Keywords: Modeling, computer aided configuration, simulation, distributed data base.

This paper describes a concept for high level configuration and parametrization of process simulators. The simulator is an integral part of a distributed process control system. This system can be switched from a control mode to a simulation mode where i/o signals are generated by/supplied to a process simulator when the system is set in simulation mode. The system is supported by a set of configuration tools.

The system is implemented on distributed processors and is intended to be used for process control-, training-, and simulation applications. The paper describes the system hardware architecture, the distributed data base structure and the mechanisms for configuration and running of the system.

A simple application example is given and performance limits for the system are discussed.

1. Introduction

Simulation and simulators are important tools in control system design and operation. The following application areas are typical:

Testing of control system designs as a tool for structure and parameter selection.

Factory test of turn key control systems.

Operational simulators used by the system operator in order to test planned operations or as an exploratory aid in testing different operational strategies.

Training simulators for training of plant operators.

All of these simulator applications are closely related to a control system and its operation. Kongsberg Albatross a.s. are presently developing an integrated process control and simulator system, where the system can be switched from a control mode to a simulation mode. In the latter mode the process I/O signals are substituted by signals generated by a process simulator.

The simulator operation is based on a modular simulation technique. Colour graphics interactive tools are used for configuration and operation of the simulator. An editor is available for construction of new simulator modules and another editor is used for configuration of a touch-sensitive soft-key system. The soft-key system is used for operator interaction with the simulator.

The hardware design of this combined simulation/control system is outlined in § 2, the main design principles are explained in § 3. The basic operational principles are discussed in § 4 and an overview of the editors used for creating new system

Received 8 October 1985.

† This paper was presented at the International Seminar on Modern Methods in Dynamic Simulation of Industrial Processes, Trondheim, Norway, May 1985.

‡ Kongsberg Albatross a.s., N3601, Kongsberg, Norway.

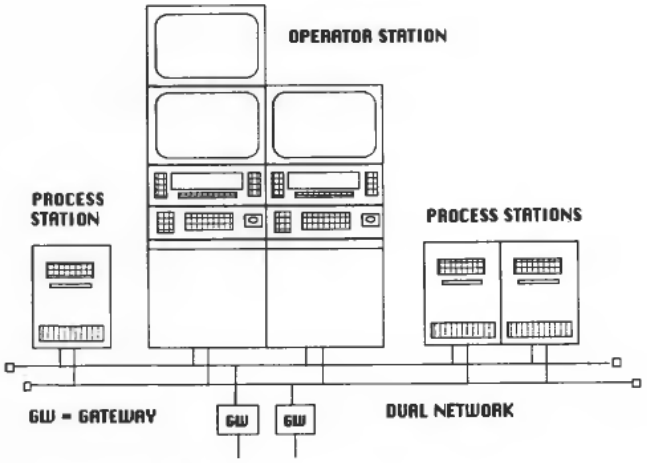


Figure 1. An example of a simulator/control system configuration.

modules and new man–machine interfaces (MMIf) are given in § 5. In § 6 results for a simple simulator implementation are shown.

2. Hardware design

The system is a distributed multi-processor system. The basic units are Intel based single board computers which communicate via a 10 Mbit/sec local area network (LAN) of the Ethernet type. The single board computers (named SBC1000) are designed by Kongsberg Albatross. The SBC1000 is based on Intel iAPX 186/87 processors and has a memory of 784 kbyte. A Winchester disc or a 1 Mbyte bubble memory can be added as a mass storage device.

A typical system layout is shown in Fig. 1.

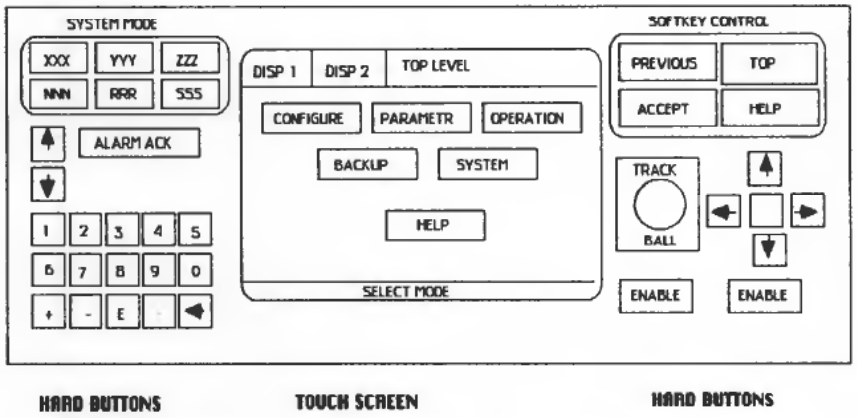


Figure 2. Example of touch screen console part.

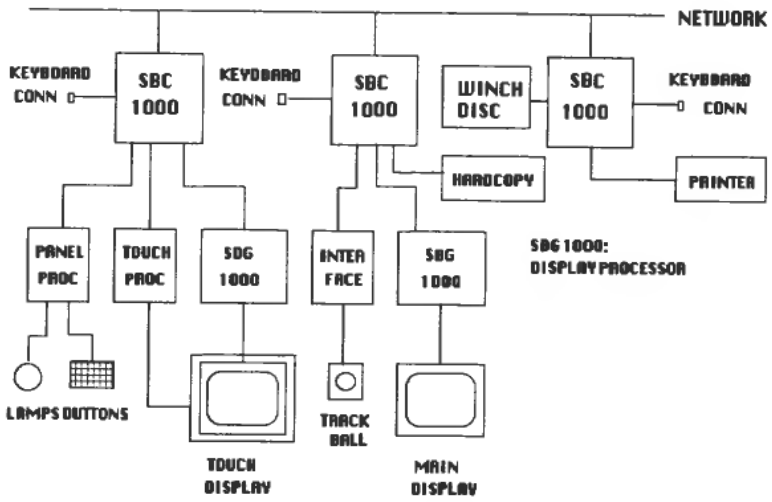


Figure 3. Operator station hardware structure.

2.1. The operator station

The operator station (hereafter termed OS) is used for operator/process or operator/simulator interaction. The OS has a 20 in colour graphics screen used as an information display, a track ball for cursor pointing at displayed elements, a touch sensitive 10 in colour graphics screen for implementation of soft keys, and a number of hard buttons for activation of important and frequently used functions. An example of the touch screen display is shown in Fig. 2.

The internal hardware structure of the operator station is shown in Fig. 3. One computer handles the man–machine interface (MMIf), the second computer handles the display system. The third computer is optional and may take care of report generation, long term historic data storing, printing of reports, etc.

2.2. The process/simulator station

A typical process/simulator station (hereafter termed PS) is shown in Fig. 4. Each of the SBCs can execute control and/or simulation tasks. In control mode the process I/O is multiplexed by the processor via the process bus.

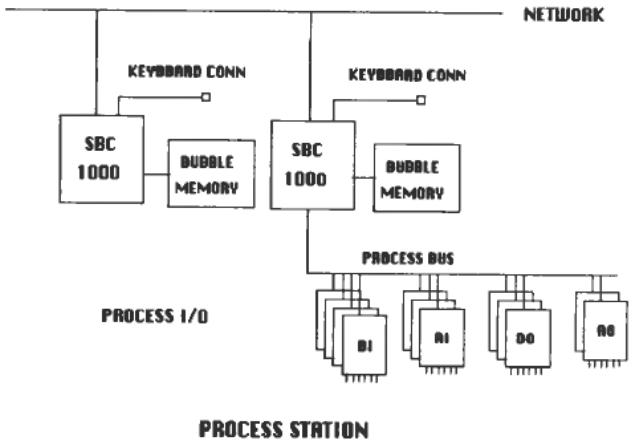


Figure 4. A typical process/simulator station.

3. Design principles

3.1. Scaleability

In a scaleable technology, the system price as a function of capacity is approximately linear over a considerable capacity range. This requires a distributed technology where processing capability can be added in relatively small increments. In all distributed control systems, the operator interaction takes place via an operator station. In most (maybe all) presently available systems, the operator station stores the graphics data used for presentation of pictures, curves, diagrams, etc. This represents a non-scaleable technology. If the number of process variables increases beyond a given limit, the OS can no longer handle the increasing number of flowsheets, trend displays, etc. and a new OS must be added.

In the Albatross system, this type of graphics data is totally distributed to the data gathering and computing modules. The guiding principle is that all information is stored where it is generated and naturally belongs. Hence, also graphics information is stored in a distributed fashion. As an example a trend curve generated by a level transmitter is stored in the process station where the module handling the level signal is located. This applies both to the data values and the formatting data describing the scaling, colour, texts etc. for the trend curve. This makes the OS a general device which contains no configuration dependent information. If a specific picture is needed, this is requested via the communication network, and the appropriate data are returned to, and interpreted by, the OS software.

3.2. Single storage of data elements

A problem in many systems is data duplication. This arises when data are stored both in the process stations and in one or more of the operator stations. If a parameter or connection is changed in one place, and not in the others, data inconsistencies arise with the accompanying problems. This kind of data duplication is prevented in the present system because process related data are permanently stored only at one place.

3.3. Hardware independence

The C language is used for implementation, because this is one of the most processor independent languages available today. The graphics interface is based on a GKS standard. The network communication interface is a simple send/receive interface which is easily adapted to different types of communication hardware.

3.4. Ease of operation

A key principle has been to design the system so that configuration and operation of the system can be done by someone who is not a programmer.

Because the natural communication medium for the process engineer and the control system designer, is the process flowsheet and the P & I drawings, a special kind of process flowsheet/P & Id is chosen as the basic configuration tool in the system. We shall call this special flowsheet for the system flowsheet (Fig. 5). At configuration time, the operator is simply putting the system together by drawing the system flowsheet on a graphics screen. The total system is defined by a large

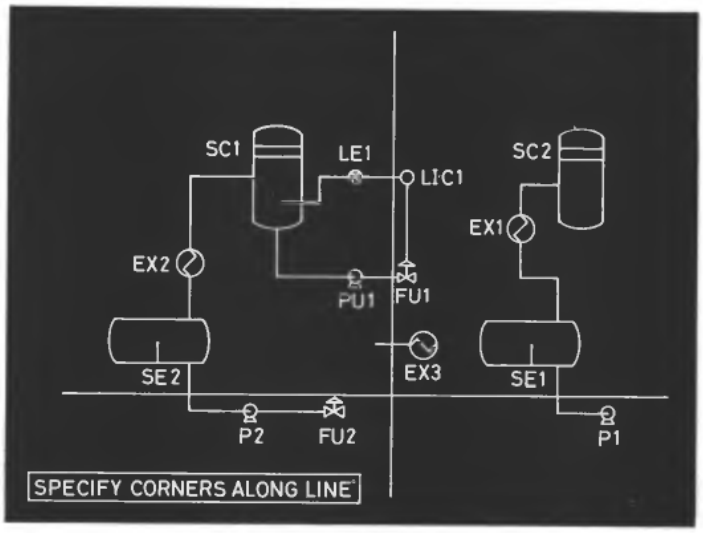


Figure 5. System flowsheet building.

virtual system flowsheet. The operator can at any time view parts of this system flowsheet on the screen, and address specific elements in it by pointing at them with a graphics pointing device.

4. Basic operational principles

4.1. The module concept

The basic computing and data handling element of the control/simulation system is the module. In a given system a library of different modules are defined. Examples of modules are a control valve handler, a pid controller, an algorithm for handling a level measurement signal, a pump handler/simulator, a simulator module of a heat exchanger, a simulator module for a tank etc. Each type of module is represented by a specific algorithm which defines the operation of that module, and an associated data structure. Hence, a module is a kind of type-definition or a template.

One or more instances or realizations of a module can be created by the operator during system configuration time. Ideally we should distinguish between the term module (which is a type definition) and a realization of that module. However, in order to simplify the language, we will in the following use the term module when we actually mean a module realization.

4.2. Creating modules

When a user wants to configure a new simulator, he will select the appropriate modules by using the soft key system and position the symbol for each module in the system flowsheet. This is done by using the track ball. When a module is created in a user selected process station, data space for the data structure representing that module is allocated.

4.5. Module data structure

The layout of the data structure for one module is a hierarchy of sub-structures. Data for all modules in a single PS is organized as a doubly-linked list. This is indicated in Fig. 6. The following is a short description of the most important data structures associated with a module.

Module data. This sub-structure contains the module type and identification, the symbol type for representation of the module in the system flowsheet, the position of the module symbol in the system flowsheet, the number of input-, output- and process terminals for this module, the position in the system flowsheet of the module tag, the tag name (alpha-numeric module identification), the parameter vector size for this module, a status vector for this module, the pointers to sub-data structures and a pointer to the module algorithm.

Parameter data. This sub-structure contains the type dependent parameters of the module. The size of this structure depends of course on the module type.

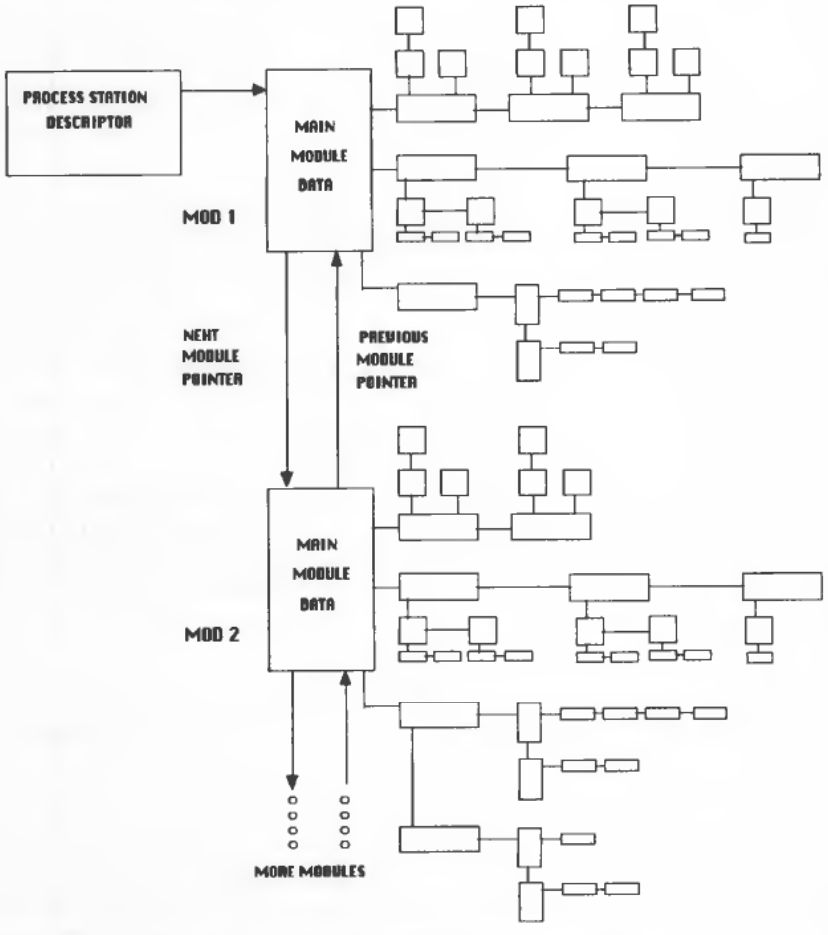


Figure 6. Module data structure layout.

Input terminal data. This sub-structure holds data related to the signal input terminals. If a module has N signal input terminals, the data for these are organized as a linked list of N data structures of this type, terminal position relative to the module graphics symbol, geometry of the line connecting the input terminal to an output terminal and a data fetch address. This address specifies the process station, the module and the output terminal for the data fetch operation.

Output terminal data. The signal output terminal data for a given module are also organized as a linked list with one list element for each output terminal. A list element holds data similar to the input terminals, except for the fetch address data which are lacking. In addition alarm and instrument limit data are included as well as pointers to optional sub-structures. These data structures contain data related to the graphics system. If f.ex the output signal is included in a data curve display, stored data and formatting information for this purpose is connected to the corresponding output terminal data structure.

Process terminal data. These data structures contain data related to process equipment terminals and are also organized as linked lists with one link element for each process terminal. In addition to identification, status and topology data (connecting lines and addresses), a link element contains a data vector for the process terminal variables. A process terminal data vector can f.ex. hold pressure, temperature and flow for that process connection.

Picture data. Several types of picture data structures can be attached to the module data structure. A total picture can then be composed of a number of picture fragments. A picture fragment related to a specific module or output terminal is then locally stored together with the corresponding module or output terminal data. A picture fragment may be a trend curve, a data table, a bar graph, a symbol, a numeric expression, a text element or entries in preformatted pictures such as overview displays, group displays, loop displays, alarm displays etc.

4.4. The process/simulator station

The functional structure of the PS is shown in Fig. 7. The data base is already described in the previous sub-section. It contains a PS descriptor and a linked list of module data structures.

The module scanner traverses the module list and executes the module algorithms at each sampling interval. During the module scan, the needed input variables for the different modules are fetched from other modules or from process input tables according to the fetch addresses stored in the input/process terminal data structures. When the system is in simulation mode, the simulator parts of the modules are activated during the module scanning, and the simulated process variables are exchanged between process terminals. Operator communication can be performed during a simulation run, or the simulation can be stopped while changing parameters, connections etc. In the former case, the operator messages are buffered during the module scanning part of the sampling interval, and are processed in the idle time slot at the end of the sampling interval.

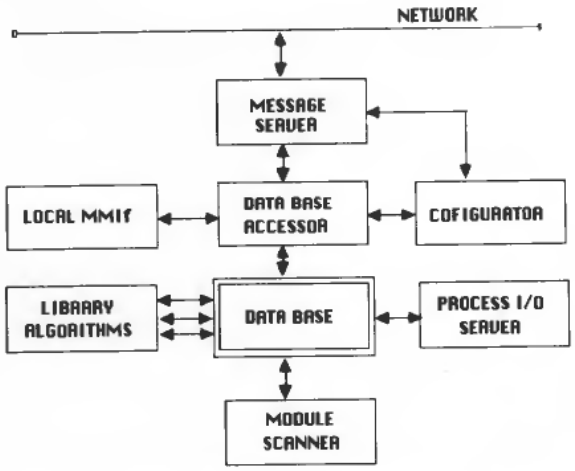


Figure 7. The functional structure of a process station.

The message server receives messages from the network and stores the messages for later action, or takes immediate action, depending on the type of message.

The data base accessor represents the access mechanism to the data base. Module data elements can be addressed by specifying the module tag name, the module position in the system flowsheet, the module number and the PS number, the terminal identificator etc. Requests based on set membership can also be used. The operator station can f.ex. issue a request for all flow valve modules, or all modules within a specified system flowsheet window. Another example will be that all necessary information to activate a predefined trend picture is requested. In that case the data base accessors will search each process station data base for this type of information and return all requested information to the requesting operator station.

The configurator reacts to configuration messages. When a new module is requested, the configurator allocates memory space for the data structures and links the main module data and its sub-data structures into the module list. When a module needs to be deleted, the configurator asks the data base accessor to find the module. When the module is identified, the configurator de-allocates the occupied data space and restores the module list.

The local MMIf system is a service interface to the process station which can be accessed by connecting an alpha-numeric terminal to the corresponding RS232 port on the PS.

The process I/O server is a driver for reading and writing data from/to the process I/O system.

4.5. The operator station

The functional structure of the operator station is shown in Fig. 8. The operator station has two main parts, the MMIf handler and the display system. Actually, an

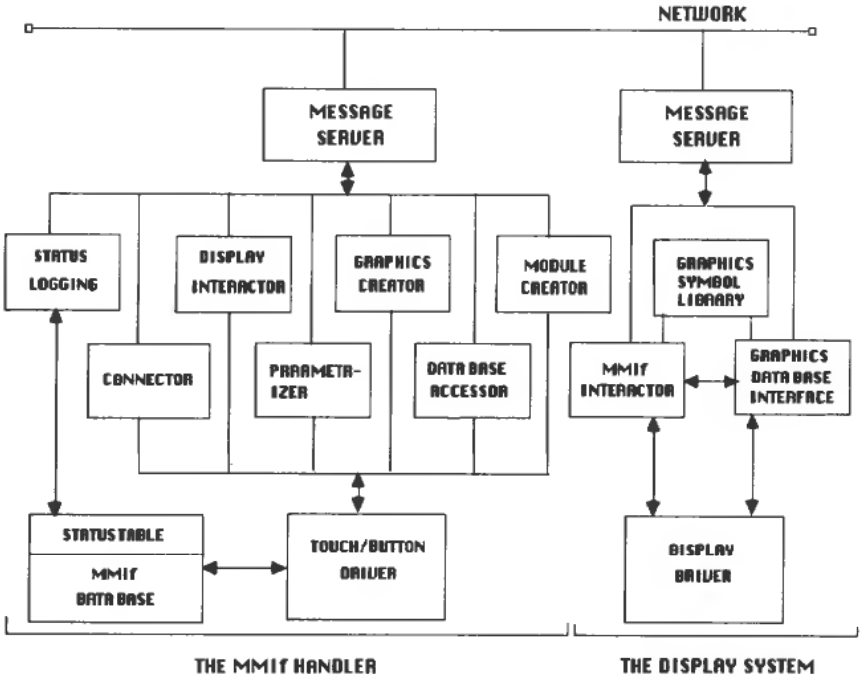


Figure 8. The operator station functional structure.

arbitrary number of display systems can be connected to one MMIf console. Both the MMIf handler and the display system has a message server which receives/sends messages from/to the network and activates the appropriate sub-systems on incoming messages.

The MMIf interactor in the display system and the display interactor in the MMIf handler, are used for graphics interaction between the two systems. Examples of interaction are: positioning of symbols in the system flowsheet, pointing at terminal positions, pointing at parameter values in a parameter picture or definition of curve display formats.

The graphics data base interface in the display system is activated when graphics primitive messages arrive via the network. When f.ex. a flowsheet display is requested from the MMIf handler, a multicast is issued via the network to the process stations. These react by sending module and terminal data for modules in the requested flowsheet part to the display system. At the display system side, the arriving data are organized and displayed as a flowsheet picture.

In addition to the display interactor, the MMIf handler has a couple of other sub-systems. The module creator is used for interaction with a display system and a process station when creating or deleting a module. The graphics creator is used when building or changing graphics pictures and the connector is used for changing the system topology. The parametrizer is used for changing parameters in existing modules. The data base accessor is a sub-system which is used by the other actors for remote access to the PS data bases.

The touch/button driver operates the soft-key system, and is the real interface to the operating person. When a button or a soft key is pushed, the driver identifies the button and activates the appropriate action. Not only the operator can change the status of the button system. The system can be configured so that simulator or process events can influence the button system via the status logging system. This sub-system receives change messages via the network from the process stations and operates on the status table. A change in the status table can trigger a blinking mode for a button or a soft key, a colour change of a soft key, the interlocking of one or more keys or the activation of a new set of soft keys on the touch display.

4.6. The application protocol

The application protocol for communication between different parts of the system consists of about one hundred different types of messages with their associated action patterns. The messages can be divided into database commands, configuration commands, parameter transfer commands, graphic display commands and data commands for exchange/delivery on a sampling time basis. Examples of single messages are:

Report process station descriptor.
Create new module of a given type.
Request module data for a given module.
Send input terminal data for a given interminal.
Request for all modules of a given type.
Send process output value.
Send elements related to specified picture to the specified display.

In order to illustrate the operation of the communication system, we shall show the communication sequence for disconnecting the connection between an input terminal on module LIC1 and an output terminal on module LE1 in Fig. 9.

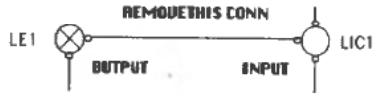
4.7. Main modes of basic operation

The main running modes of the simulator/control system are the configuration mode, the parametrization mode, the operational mode and the system mode.

In the configuration mode, the operator can change the system topology, introduce new modules, delete modules, create new graphics pictures, create new curve displays, reports, etc. One of the operator stations can be in the configuration mode while the system is running.

In the parametrization mode, the operator can call up a module by pointing at it in the system flowsheet, or by giving the tag name of the module. A parameter picture is displayed on a selected display screen, and the operator can address and change a specified parameter by cursor pointing. A parameter page for a tank module is shown in Fig. 10.

In the operational mode the simulator or the control system is running on a regular sampling time basis and data are flowing between modules as defined by the system configuration. In the simulation mode, the system acts as an explicit integration



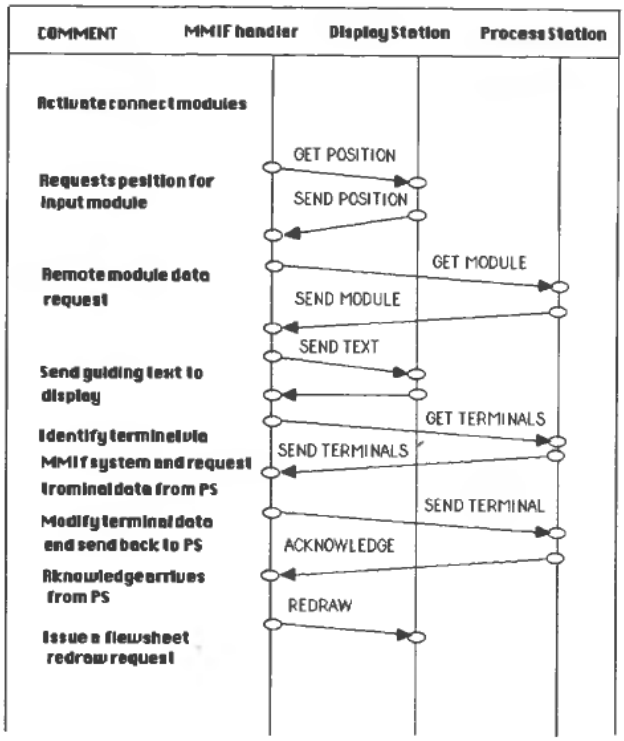


Figure 9. Illustration of communication sequence for removing a connection between two modules.

Figure 10. A parameter page display for a tank module.

system. This implies that the system, in its present form, is not suitable for simulation of stiff systems. However, if the stiff systems are totally confined to one module, this can be taken care of within that module. Generally, the present simulator requires that the model system is well balanced with respect to time constants as a result of initial model reduction during the writing of the module algorithms. If the system has static computing elements, such as simple valve models, these static computations are performed at the end of each sampling interval. This is organized by the PS configurator by putting these modules at the end of the module list. In the operational mode, the process stations will transmit data requested for display to the appropriate display systems.

In the system mode the operator has access to system logging functions such as network performance measures, system error logs, process station idle times, the system operation functions such as starting and stopping of processors, down- and uploading of processors etc.

5. The system editors

The previous section described the basic operation of the system, which assumes that a man–machine interface exists, and that the necessary module types also exist in a module library. In this section we shall describe how a new MMIf and new modules are generated.

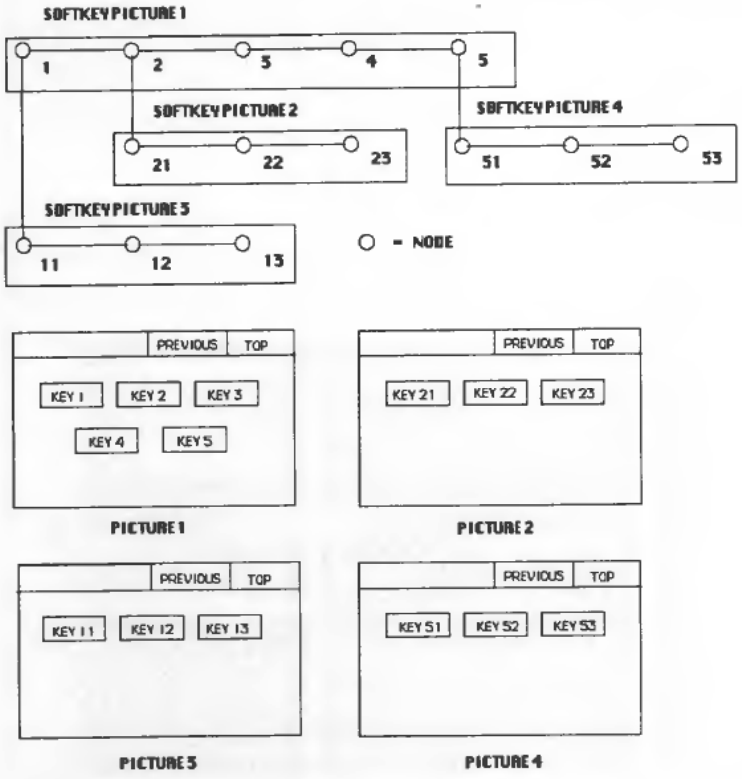


Figure 11. A simple soft key system and its associated binary tree.

5.1. The MMIf editor

The operator communication system (the MMIf) is modeled as a binary tree structure where each node in the tree represents a touch sensitive soft key. Associated with each node is a data structure containing attributes and data related to the corresponding soft key such as colour, size, position, text etc. The node also contains status data and names of action sub-routines and status response sub-routines. These are called when the button is activated by the user or by an appropriate system or process status change.

All right child nodes descending in a sequence from a branching node, represents one screen picture of soft keys. This is illustrated in Fig. 11 where a tree representing 4 soft-key pictures is drawn. When an operator activates a soft key, the nodes in the present picture is traversed in order to identify the soft key. When the correct soft key is found, the action sub-routine for that button is activated, and the operating system traverses the tree to the next soft-key picture by moving to the next lower level along the corresponding left child branch of the tree.

When the user wants to return to a previous command level, he can do so by pressing the previous, or the top level button. These buttons will be implemented as hard buttons in the final system together with other often used or important buttons.

The control/simulator system will be available in a standard implementation with a standard MMIf system. Using this system, the user will be able to configure and run a large variety of different simulator configurations. However, in many cases a tailor made MMIf will be desirable for implementation of special purpose simulators and control systems. For this purpose, an MMIf editor is designed.

This editor is a soft-key-operated system where the user can specify the new MMIf soft-key pages, the layout of each button (position, colour, size, text, blinking capability, etc.), the name of an optional action sub-routine to be called when the button is activated, and the name of a status action sub-routine to be called when an external event occurs.

Actual pictures during operation of the MMIf editor, are shown in Figs 12(*a*)

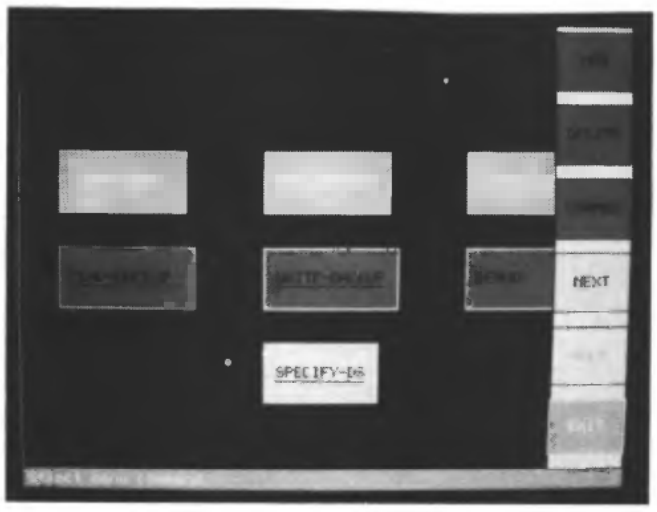


Figure 12(*a*). Touch screen during MMIf generation.

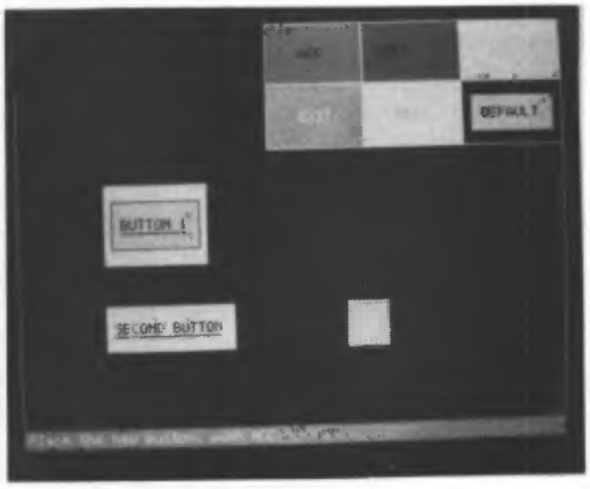


Figure 12(*b*). Touch screen during MMIf generation.

and 12(*b*). The generated soft-key buttons are shown on the screen. The configuration menus used for building the soft keys can be moved around the screen so that they do not hide the new keys being created.

When the user has created a satisfactory MMIf by using the editor, he activates the system generation part which generates C code and a data file for running of the MMIf system. Then the user has to compile and link the code together with the remaining system code and user written action, and status sub-routines if this is necessary. Figure 13 shows a small part of the generated code for an MMIf system.

5.2. The module editor

In order to create new types of modules for inclusion in the module library, the system has a module editor. This editor is also operated by using graphics interactive tools.

The module library or directory consists of four separate sub-directories: A directory of algorithms, a corresponding graphics symbol directory, a default data base directory and a parameter layout directory.

The algorithm for a new module is written as a C function with a standard call layout. A graphics symbol for the module is generated by a graphics symbol editor and stored as a graphics segment.

The data base structure for the module is generated and parameters defining structural properties such as number and type of input, output and process terminals as well as default names and parameter values are defined.

The parameter layout for the module is determined by building a screen layout for a parameter picture. Each parameter can be defined as an integer, a floating or a string.

When all four sub-directory entries are filled in for a new module, a system generator can be started which generates code for data definition, initialization, and a system calling interface for the new module. Now the updated module directory can be downloaded to the OS and PS and the system is ready for running.

```
/*    FILE   : trees.c                    page 1                    */
/*    MODULE : C_0                                                  */

C_0()
    /*                                                              */
    /* Top level command  TOP NODE                                 */
    /*                                                              */
    /*  Commands on this level :                                    */
    /*        CONFIGURE                                             */
    /*       PARAMETRIZE                                            */
    /*        OPERATION                                             */
    /*       READ-BACKUP                                            */
    /*       WRITE-BACKUP                                           */
    /*       SPECIFY-OS                                             */
    /*       DEBUG                                                  */
    /*                                                              */
{
    unsigned In_TS();
    int s_1,s_2,s_3,s_4,s_5,s_6,s_78;

    while(1)
    {
       if(Stat_0(&s_1,&s_2,&s_3,&s_4,&s_5,&s_6,&s_78) == 1)
       {
           BranchPath();
       }

        switch(In_TS())
        {
            case 1 :            /*  CONFIGURE   */
                while((s_1=U_1()) < 0)
                    continue;
                C_1();
                break;
            case 2 :            /* PARAMETRIZE */
                while((s_2=U_2()) < 0)
                    continue;
                C_2();
                break;
            case 3 :            /*  OPERATION   */
                while((s_3=U_3()) < 0)
                    continue;
                C_3();
                break;
            case 4 :            /* READ-BACKUP   */
                while((s_4=U_4()) < 0)
                    continue;
```

```
/*    FILE   : trees.c                    page 2                    */
/*    MODULE : C_0                                                  */

                break;
            case 5 :            /* WRITE-BACKUP */
                while((s_5=U_5()) < 0)
                    continue;
                break;
            case 6 :            /* SPECIFY-OS */
                while((s_6=U_6()) < 0)
                    continue;
                break;
            case 78 :           /* DEBUG        */
                while((s_78=U_78()) < 0)
                    continue;
                C_78();
                break;
            default :
                return;
                break;
        }
    }
}/* end C_0 */
```

Figure 13. Code generated by the MMIf editors code generator.

6. Simulation example

In this section a simple simulation example will illustrate the working of the system. A simple tank system is simulated. The system flowsheet for the process is shown in Fig. 14. TA1 and TA2 are tanks, LIC1 and LIC2 are pid controllers, LE1 and LE2 are level measuring modules and SOURCE and SETPNT are signal generating modules. The signal generation modules can generate a sum of a sinus function, a square function, a noise signal and a bias value. The same setpoint is used both for LIC1 and LIC2. The setpoint is chosen as a square signal with an amplitude of 2 metres, a period of 100 minutes and a mean value of 5 metres. The input source signal has a mean value of 6 m^3/min and a random noise component with standard deviation of 0.5 m^3/min. A group picture display and a trend picture display from the simulation are shown in Figs. 15 and 16.

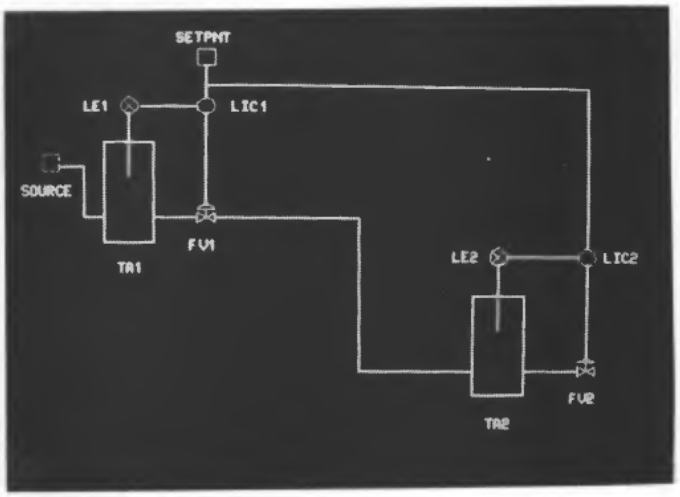


Figure 14. System flowsheet for the tank system simulator.

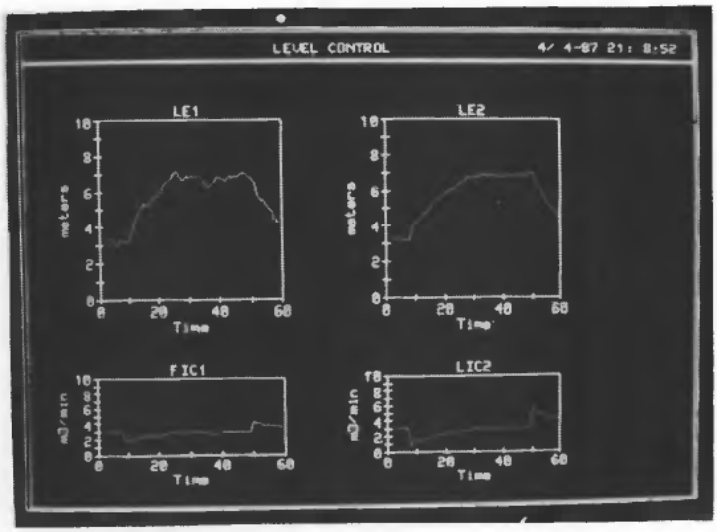


Figure 15. Trend display from the simulation.

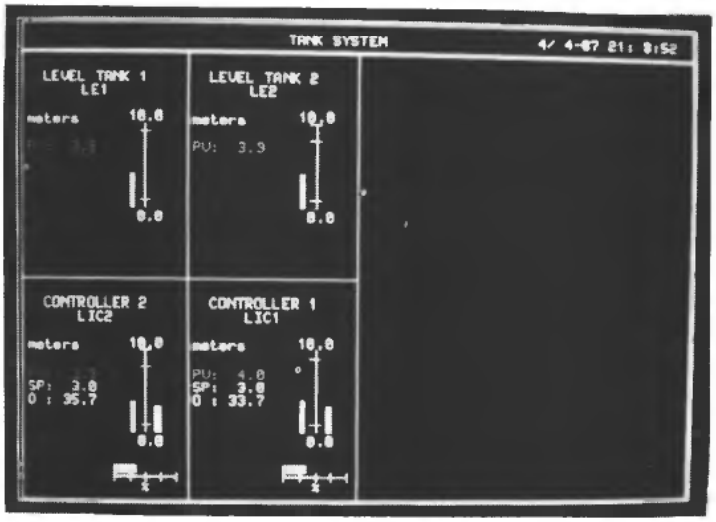


Figure 16. Group display during the simulation.

7. Conclusion

The design and implementation of a combined simulator/control system which is presently being developed at Kongsberg Albatross a.s. has been discussed. At present, a working prototype of the system exists. The system has already turned out to be very easy to operate and changes in the simulator's structure and parameters are very easily implemented. This is also so for the presentation part of the system. The operator can, for example, implement a new trend display in less than one minute, and then continue the simulation.

The system is very easily expanded because a new processor in a process station, or an entirely new process station, or an operator station, can be included without modifying the existing parts of the system.

The system is also very well suited for simulating other control system makes, because other types of man–machine interfaces can be created by using the MMIf editor.

It is the available technology which makes it possible to design the system we have described in this paper, where the basic system software consists of more than 100 000 lines of C source code. We think that our system is one of the first very high level simulation/control system available, and that we will see more of this type of system in the future.