AN OBJECT-ORIENTED ENVIRONMENT FOR INTELLIGENT CONTROL ENGINEERING

Wagner C. Amaral[+] , R. M. Nazzetta[+] and Lúcia Valéria R. Arruda[++]

+ School of Electrical and Computer Engineering - FEEC / DCA
State University of Campinas - UNICAMP
P. O. Box 6101
13081-970, Campinas SP, Brazil
tel: 55-92-397074
fax: 55-92-391395
e-mail: wagner@dca.fee.unicamp.br

++ Federal Center of Education in Technology - CEFET - PR
Av. 7 de Setembro, 3165
80230 - 901, Curitiba PR, Brazil
tel: 55-41-3224544
fax: 55-412-2245170
E-mail: arruda@cpgei.cefetpr.br

Abstract Most industrial processes have several particular characteristics, system and performance requirements, which may not be obtained by classical control methodologies. In this case advanced control strategies must be considered. In face of different options, simulation using software packages is a common practice. In this paper, a simulation environment aimed at consistently and uniformly integrating different software resources is proposed. It is based on object-oriented approach implemented into a knowledge-based shell. Full potential of knowledge based systems can be used for intelligent decision. An application example is described to illustrate the abilities of the proposed environment. This example is a prototype for simulation and control of a multivariable process: a heavy oil fractionary column.

Keywords Object-oriented approach, knowledge-based system, hybrid knowledge representation, expert CAD, intelligent control design, system identification, industrial process.

Resumo - Alguns processos industriais são caracterizados por requisitos de ambiente e desempenho que não podem ser tratados através de metodologias clássicas de controle. Nestes casos, é usual a utilização de pacotes de simulação, a fim de se testar estratégias avançadas de controle, antes de aplicá-las em tais processos.

Neste artigo, propõe-se um ambiente de simulação e projeto que integra de maneira consistente e uniforme vários recursos de software que facilitam o desenvolvimento de sistemas avançados de controle. Este ambiente é baseado numa abordagem orientada a objeto e implementado dentro de um núcleo baseado em conhecimento, suportando por isso decisões inteligentes. Um exemplo aplicativo (protótipo para simulação e controle de uma coluna fracionadora de óleo pesado) é descrito, a fim de ilustrar as potencialidades do ambiente proposto.

Palavras-chave: Orientação a objeto, sistema baseado em conhecimento, CAD, controle inteligente, controle adaptativo, identificação, processo industrial.

Artigo submetido em 02/09/97
1a. Revisão em 26/02/98; 2a. Revisão em 03/06/98;
Aceito sob recomendação do Ed. Cons. Prof.Dr Ricardo Tanscheit

1 INTRODUCTION

Industrial process control commonly requires the implementation of complex systems in which controllers must satisfy requirements such as stability and robustness without system performance degradation. In this case advanced control methodologies must be considered and the tasks of problem formulation, model validation, design methodology choice, closed-loop execution and supervision should be performed. Intelligent control is coming up as promising scheme. The hybrid approaches including procedural and heuristic control actions are considered.

In face of different options and a large number of computational algorithms, testing in simulation environments is a necessity. Simulation can be used to validate a specific approach before application to a process. Computational support, such as CAD systems, is an essential tool for control engineering tasks. Today, intelligent CAD systems are also available. Among them, a diversity of system architectures is noted (Taylor *et alii*,1990; Amaral *et alii*,1992; Pang, 1992; Baker *et alii*,1993). These systems are primarily aimed at environments for system development. The capability to adapt

to the user is fundamental. In many systems, the overall package functionalities are always independently present if, for instance, the user needs only a small set of the available tools. This is a serious drawback because the resulting software is often large, inefficient, and difficult to use.

Moreover, architectures used by environments for control systems design are typically composed by three basic parts:

- a set of tools implementing the tasks of *simulation*, *control*, *analysis*;
- an interface to make task specification more suitable to the end-user;
- a main program to coordinate the tools and interface execution.

In this case, a set of tools is usually implemented by programs with a structure that maps system tasks as a sequence of user actions and tool calls. This information is embedded in main program code lines. Every future modification (inclusion of new tools, new user options or system reduction) modifies the main program structure. If modifications are not properly handled, the resulting software is inefficient.

In this paper a system architecture that structures the main program in a more suitable way for modification and evolution is proposed. This architecture is based on the object-oriented approach, with a graphical representation of the information flow derived from the main program structure. A knowledge-based system is used to manage the software resources and make user-interaction friendly. The final environment supporting intelligent design and supervision of control system is developed to satisfy the following requirements:

- a modular and flexible environment;
- interaction with available control tools;
- easy and uniform user interaction;
- knowledge representation capability.

But why object-oriented approach?

Object-oriented is a class of computing scheme in which data and associated procedures are encapsulated to form an *object* (Peterson, 1990). An object is a computational structure similar to a frame that contains both data and related procedures (Dechampeaux *et alli*, 1993). Therefore, objects provide a coarser level of granularity for program decomposition than is available by using data or procedures. The key concepts, collections of data and related operations, should be treated as a single entity rather than separate things.

This approach has gained considerable interest in the last few years, due to the following characteristics (Peterson, 1990):

- In traditional software development, procedures and data structures are considered as independent entities while in object-oriented systems a single type of entity, the object, represents both;
- Unlike data, an object can act. It can determine what to do when a message is received and different objects can cause different actions for the same message;
- Many different objects may be nearly identical in their capabilities. These objects can be collectively unified by defining a class. Moreover, classes may be organized as a tree structure, with the parent of each class being its *superclass*. Each class inherits the capabilities from its superclass (and all ancestors of this superclass).

Advantages of using object-oriented computing include (Coard e Yourdon, 1991):

- *information hiding* and *data abstraction* increase reliability and help decoupling procedural and representational specification from implementation;
- *dynamic binding* increases flexibility by permitting the addition of new classes of objects without modification of existing code;
- *inheritance* coupled with dynamic binding permit code to be reused, therefore it enhances code "factoring". Code factoring means that a code to perform a particular task is found in only one place and this simplifies software maintenance.

A disadvantage of object-oriented approach, often debated, is the run-time cost of the dynamic binding mechanism. A message-send takes more time than a straight function call.

The architecture proposed herein is undergoing implementation in a Sun SPARC station based network using G2, a real-time expert system shell, and MATLAB package. It is an evolution of those proposed in (Arruda *et alii*,1994a; Arruda *et alii*,1994b). In fact, the environment described in this paper is based on the same ideas proposed in these former papers. The difference is a new module for control supervision with some special toolboxes for analysis, help and knowledge acquisition. With these toolboxes a non-expert user can build complex applications in adaptive control design and develop intelligent solutions for them. The final environment is aimed at education, research and development purposes.

The paper is organized as follows. Section two introduces a formal description of system architecture. Section three describes an implementation of system architecture using G2. A presentation of toolboxes for control engineering available in the environment is in section four. The methodology to include new toolboxes is described in section five. Section six gives an application example using MATLAB as an engine machine. Finally, section seven presents some conclusions.

2 SYSTEM ARCHITECTURE

The proposed architecture models the main program through a set of *objects*, a *connection diagram* and a *scheduler* as shown in figure 1. Objects encapsulate *algorithm* models while the connection diagram represents the flow information between objects. The scheduler translates the connection diagram associated with an object definition into a suitable sequence of execution using various software resources. These resources are part of the object definition to which they return their results. The user acts on the system through an user interface, which is a part of the object definition and the connection diagram. It provides user interaction with the software resources and can be used to modify the information flow among objects. The system also has a logical integer *clock* representing the sampling time interval. At each time interval, the scheduler searches for an algorithm execution.

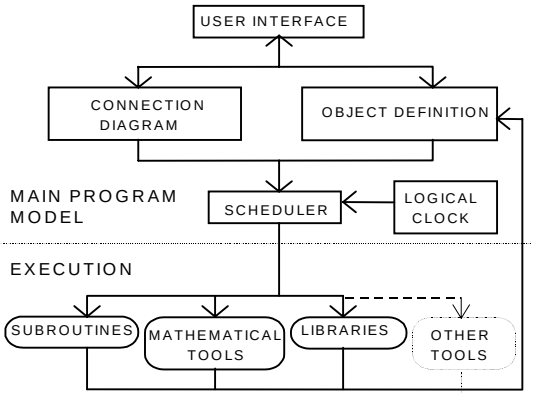

Figure 1 - System architecture.

Object definition

An object models an algorithm with the following attributes:

- *status* : algorithm status (see scheduler);
- *time_increment* : time interval for algorithm execution;
- *last_update* : last time that the algorithm was executed;
- *input* : algorithm input data;
- *output* : algorithm output data;
- *parameters* : parameters to adjust algorithm behavior;
- *user_interface_resources:* algorithm interface objects to user interaction.

Each object is associated with a behavior, implemented by various integrated software resources. For example, an object may have a procedural behavior implemented by means of a specific subroutine, a mathematical tool or a library function. During an execution, if an algorithm must be executed, its associated object receives an executing-message from scheduler. Then the data related in *input* attribute are collected, the algorithm code runs and the variables in *output* attribute are updated.

Connection diagram

The connection diagram establishes the information flow between algorithms. It can represent processes (plant), controllers, or both as shown in figure 2. The connection diagram is a graphical representation of the dependencies among algorithm procedures, i. e., a visual representation of how the algorithm procedures depend on each other. This representation can be easily and consistently manipulated to create, modify or destroy relations between objects.

Scheduler

The scheduler provides control mechanisms to translate the information contained in the connection diagram and the object definition into a suitable sequence of software resource calls. These control mechanisms are designed to provide complete independence from algorithm details. At the scheduler level, an algorithm is viewed as a processing cell that collects input data, returns output data, and is inserted in a context of dependencies, i.e. it depends on the other algorithm results.

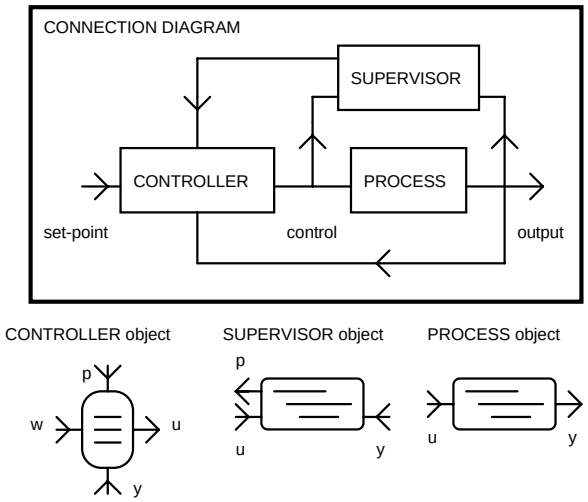

Figure 2 - Connection diagrams and their objects.

This level of abstraction allows the use of several algorithms in a uniform and distributed way. The scheduler sets different algorithm *status* and uses a set of rules for status transition, as shown in figure 3.

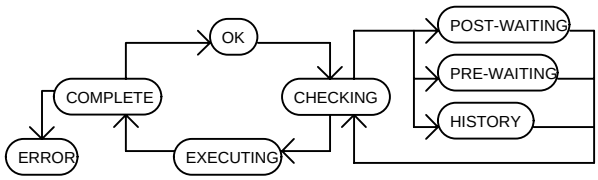

Figure 3 - Algorithm status and status transitions.

The status definition is:

- OK: the input and output algorithm data have valid values until the next algorithm execution time;
- CHECKING: the algorithm must be executed in the actual time interval and the scheduler is checking for dependencies on other algorithms;
- PRE(POST)-WAITING: there are not enough conditions to execute an algorithm and it must wait;
- HISTORY: all algorithms are waiting for executing conditions but any event can change this situation. This deadlock situation can occur in a connection diagram with feedback;
- EXECUTING: an algorithm has collected input data, started execution of the associated code, and is waiting for results;
- COMPLETE: the algorithm has received the results;
- ERROR: the algorithm did not run in the same time interval (this must indicate inconsistent information flow).

The set of rules defining status transition is derived (for each algorithm *A* contained in the connection diagram) by analysis of the necessary conditions for status change. For instance:

- OK→CHECKING: *time_increment* and *last_update* of *A* define the execution of *A* in actual time interval;
- CHECKING→PRE-WAITING: an algorithm connected to the input of *A* must be executed before;

- PRE(POST)-WAITING→CHECKING: the status of an algorithm connected to either input or output of *A* was set to OK;
- COMPLETE→OK: *A* has run successfully in the same time interval.

3 G2 SYSTEM IMPLEMENTATION

The Real-Time Expert System G2 (Gensym, 1992) is a shell for knowledge-based system design. It was chosen as design tool for implementing the architecture presented in last section due to the following features:

- hybrid knowledge representation: object-oriented, rule-based and procedural formalisms;
- communication interface with external programs;
- powerful user interface and debugging facilities;
- incremental and modular design;
- system documentation facilities.

The architecture implementation in G2 is shown in figure 4. The main program model was entirely implemented in G2 using its inference machine to construct the scheduler block and to provide global coordination. Algorithm codes are distributed through many software running at the same or remote G2 machine in a network. The GSI package (G2 Standard Interface) is used to access external software and to perform suitable routing of G2 sends and requests.

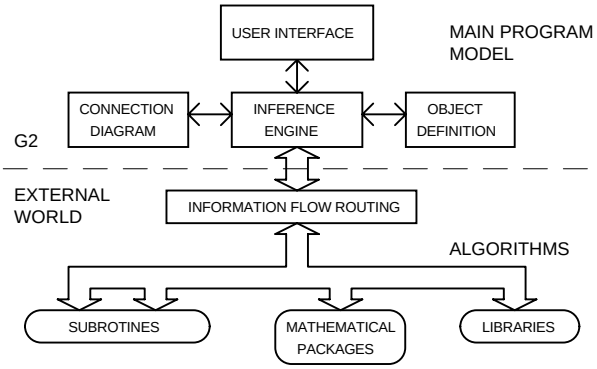

Figure 4 - Architecture implementation in G2.

The implementation of this system is under execution in a modular fashion. This is useful in incremental system design, because developers can easily introduce or reduce system features. The modular architecture is shown in figure 5. In this figure TOP LEVEL module represents the final application, i.e. the system itself. TOOL modules are specified by the developer and contain specific tools for tasks as simulation, control or analysis. Dotted lines means that certain modules may or may not be present in the final application. INTERFACE module provides the elements to support consistent user interface development. Finally, the SCHEDULER module defines the *algorithm class* and contains knowledge to implement the scheduler policy.

Algorithm class definition

The algorithm class attributes for corresponding objects are *status*, *time_increment* and *last_update*. Every algorithm object defined by the developer is a sub-class of an algorithm class and may have particular capabilities (as *input, output, parameters* and *user_interface_resources*). This warrants proper scheduler functioning, providing the flexibility to accommodate particular algorithm features.

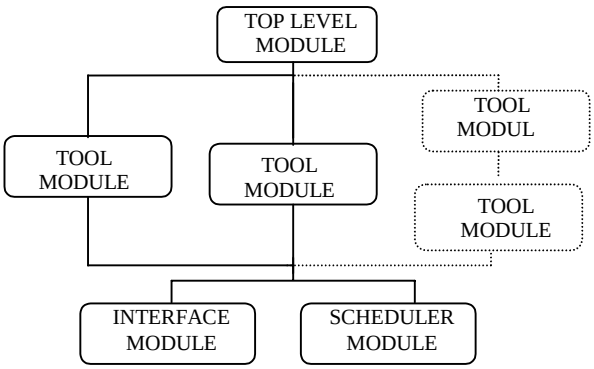

Figure 5 - Modular system implementation

Scheduler rules

The scheduler policy was implemented by four rules; one for each main status, and four procedures - as shown in figure 6. Rules deal with status transitions while procedures take actions such as check for attribute condition and set status. Rules are fired by forward chaining after algorithm status change. The last rule and procedure have a special role: when the scheduler sets algorithm status to EXECUTING it gives control to the algorithm, enabling the execution of actions programmed at algorithm level. The end of execution is signed by setting the algorithm status to COMPLETE, returning control to the scheduler. While an algorithm is running, the inference engine maintains the scheduler policy over other objects.

Logical clock

The dynamics of status transition is taken into account by a logical clock. It is implemented by a pair rule-procedure as described in figure 7. For each clock update, each algorithm in the connection diagram is tracked and status transition started. Clock update is either done automatically by the inference engine in normal mode or manually (step by step) in debug mode.

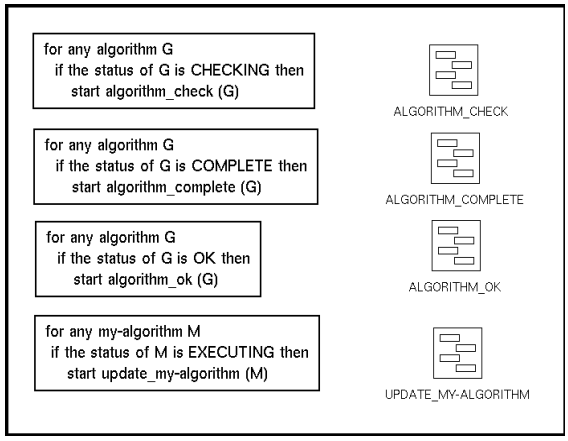

Figure 6 - Scheduler rules and procedures.

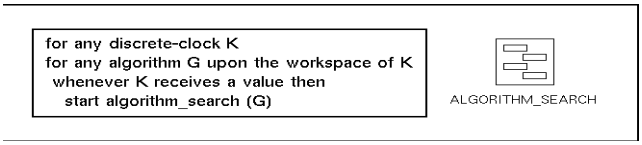

Figure 7 - Status transition rule.

4 TOOL MODULES FOR CONTROL ENGINEERING

Presently there are four general toolboxes in our environment. These toolboxes gather functions related to the main tasks in control engineering: *simulation, identification, control and supervision.*

The **simulation module** gathers all implemented process available to the users. There is also a specific tool to read any process written by the user. These users process can be coded in "C" language or they can be an "m-file" developed in MATLAB or SIMULINK.

The **identification toolbox** is formed by many modules, which implement the tasks related to system modeling. The tools can be a single procedure, a knowledge base, an artificial neural network or a fuzzy method. The identification modules and the corresponding tools included in our environments are:

- **Data processing:** Square root compensation for signals acquired directly from different sensors; graphical visualization and signal editing, spectral analysis via FFT, data filtering, etc.

- **Model structure determination:** The general approach adopted for process identification is based on the ARMAX model. The following procedures are available: determinant ratio; instrumental determinant ratio; Akaike information criterion; F-test.

- **Parameter estimation:** There are several recursive parameter estimation methods available, including the extended least squares, the instrumental variable, Kalman filtering, standard least squares, etc. Some methods to identify a set in the parametric space are also provided.

- **Model validation**: This module contains a set of rulebases performing tests based on the loss function behavior, prediction error analysis, and correlation analysis.

From the environment philosophy, it is worth to note that any procedure or rulebase in a sub-module can use a function in another sub-module. For instance, the F-test procedure in the **Model structure** sub-module is used by Loss-function rules at **Model validation** sub-module.

Like the identification toolbox, the **Control module** is formed by some modules that implement the tasks related to control system. The following tools are available:

- **Controller configuration:** To select the most appropriate controller and its parameters, the configuration tasks consider process model characteristics such as order, poles and zeros provided by identification tools and the control requirements specified by the user. Some available functions are *selection of criteria performance; selection of initial controller parameters; and pre-identification analysis*.

- **Control Methods:** PID controller, generalized minimum variance, generalized predictive controller, dynamic matrix controller, etc.

- **Control Design:** This module contains rulebases and procedures to modify the controller parameters in order to achieve the user specified requirements.

- **Performance criteria determination:** The procedures in this module are charged to compute the criteria to be used by the supervisor module to analyze the performance of closed-loop system: unit-step response, impulse response, sensitivity function, rise time, overshoot, settling time, peak time, root locus, nyquist plot, and others.

The tools performing the supervision of closed-loop system are grouped into the **Supervision Module**. There are two kinds of sub-modules: **help tools** and **analysis tools**. The first ones are the sub-modules **(Help modules)** helping the user to build an experiment by use of all available tools, and the tools to coordinate the final application **(Coordination module)**. The **Help modules** contain procedures and rulebases for the *selection of the identification algorithm, selection of the control algorithm, selection of the criteria performance, selection of input signal and data processing tools,* and *selection of graphics tools.* In the **Coordination module** we can found the scheduler, the user interface and other modules described in sections 2 and 3.

The second kind of sub-module performs the supervision of the closed-loop system. Usually these modules are rulebases to diagnose the state of the complete system. For instance the **Performance detection** module includes rulebases and procedures to detect the presence of model non-stationarity, persistent excitation, estimator convergence and estimated parameters bias. Moreover there are rules to suggest and to run some actions that warrants the specified performance level. Some rules from these modules are illustrated in figure 8.

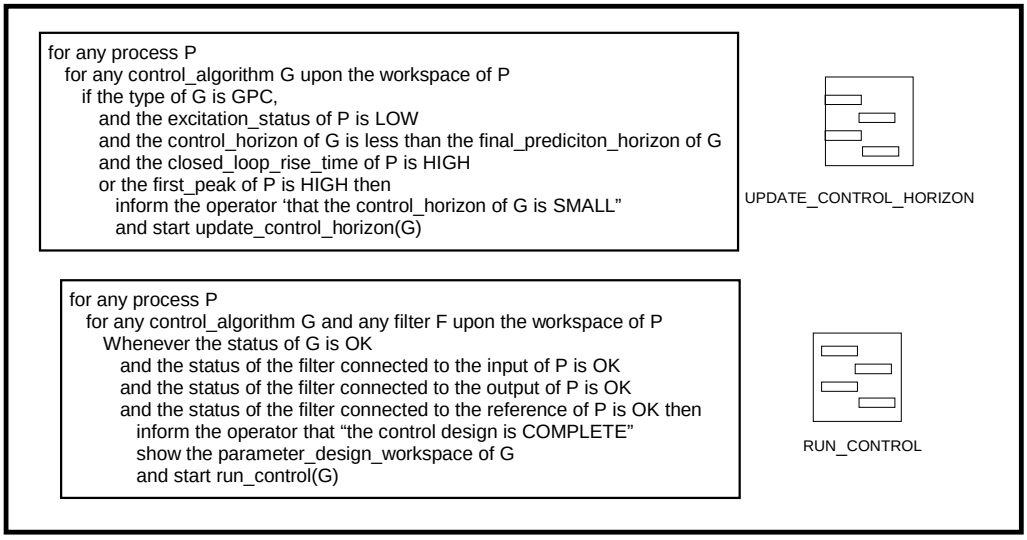

Figure 8 - Rules from supervision module

The third module, named **knowledge acquisition**, is part of the supervision toolbox, but it is usually used before running any control application. The goal of this module is to get from the user or another module, information about the process such as noise characteristic, impulse and step responses, non-minimum phase behavior, etc.

To illustrate the use of tools cited above, we present in figure 9, a connection diagram to implement the adaptive control of a generic process. From this diagram we can explode every module to get the complete connection diagram shown in figure 10. Therefore for sake of clarity we eliminate from this picture all modules forming the TOP-LEVEL of application: the coordination rulebases, the scheduler, the user-interface, the communication interface and so on. This new connection diagram reflects the methodology used in our environment to implement a control design application.

5 TOOL INTEGRATION METHODOLOGY

The integration of new modules (process, controller or any other tool for analysis/design) is based on the object-oriented paradigm. By the use of G2 tools, the user must associate to each new module, two specifically objects. The first one belongs to the *user-interface* class and the other one is an *algorithm* object (for instance, if the new tool is a controller this object belongs to *control-algorithm* sub-class). The main steps involved in module definition are:

i) identification of input and output variables and algorithm parameters;

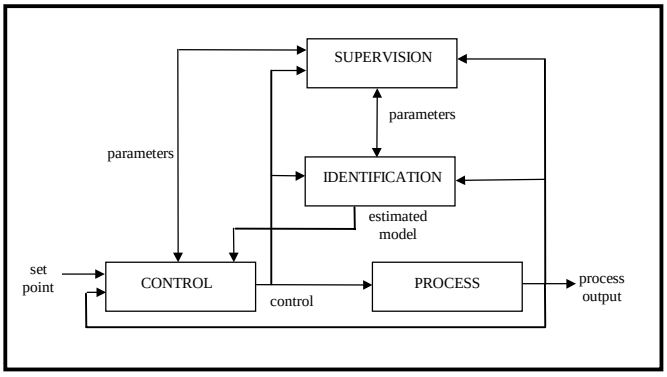

Figure 9 - Connection diagram to adaptive control application.

ii) data association between G2 and algorithm implementation;

iii) user interface building;

iv) algorithm implementation.

The algorithm implementation can be coded in MATLAB or "C" language. Example of data shared between G2 and MATLAB is shown in figure 11. This figure shows input and output variables and their respective element vectors needed by MATLAB. There is a vector "w" whose elements contain values for algorithm inputs. These values are specified in G2 and sent to MATLAB. Also, there is a vector "y" whose elements contain values of the algorithm outputs obtained by application of the algorithm upon the input values. These values are requested from MATLAB by G2. The algorithm code acts on these vectors for each algorithm call. The "routing" have the necessary information to associate G2 requests with proper variables mapped either in G2 or in MATLAB. This bridge-code is automatically generated by GSI package, when the module definition is completed.

User interface is built creating a process image and a pleasing way for parameter adjustment. This is done using G2 interface facilities: icon description, input data, display objects, etc. The end-user has a process schematic in a color graphic multiwindowing display including mouse interaction.

After the new module definition is consistently completed, the tool becomes available to any module in the environment.

6 APPLICATION EXAMPLE

The application example is a prototype for simulation and control of a multivariable process: a distillation column.

Problem Control Description

The process is a heavy oil fractionary column considered in literature as Shell Process. It is based on a model developed by Prett *et alii* (1990) to include all important control problems of a real fractionary column. The process schematic is shown in right side of figure 13.

The column feed flow provides necessary heat to proper column operation. There are three output flows: top, side and

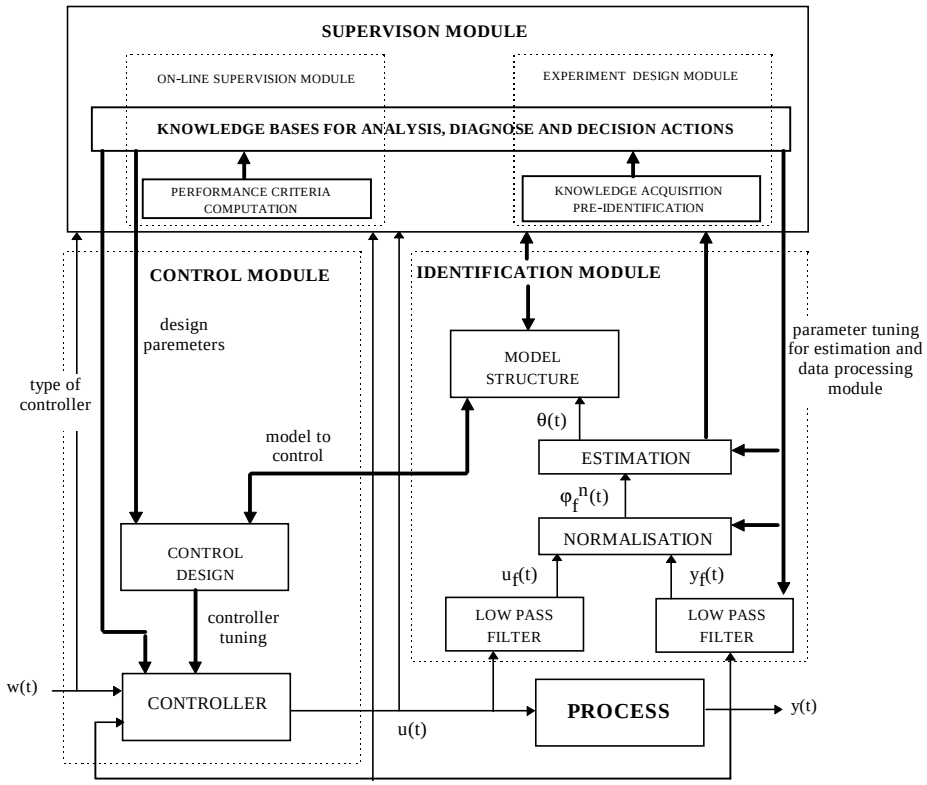

Figure 10 - All modules used into adaptive control application.

bottom draws. Top and side product specifications are fixed by economic and operational goals. Composition values are measured by analyzers in the top and side end. The bottom temperature must be controlled between fixed limit set by operational constraints. There are three control loops to regulate heat flow: the upper, intermediate and bottom reflux duties. The bottom loop has an enthalpy controller to regulate heat removal through the bottom reflux duty. Heat removed from the other parts acts as column disturbances.

The composition and bottom temperature are the controlled variables. The input variables top and side draws are used to control the top and side compositions, respectively. The bottom temperature is controlled by the bottom reflux duty. All manipulated and controlled variables are constrained within the limits of 0.5 and -0.5. The controller must be design so that the following control objectives are satisfied in the presence of input and output constraints and model uncertainty:

- Maintain the top and side draw products end points at specification (0.0 ± 0.005 at specification).
- Maximize steam production in the steam generators in the bottom circulating reflux. This means maximize heat removal.
- Reject the disturbance entering the column from the upper and intermediate refluxes due to changes in the heat duty requirements from other columns.
- Keep the closed loop speed of responses between 0.8 and 1.25 of the open loop process bandwidth.

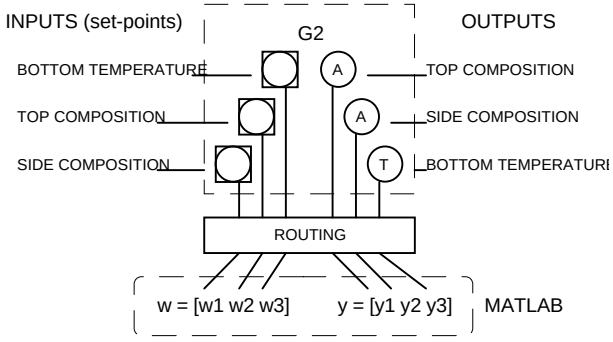

Figure 11 - G2 / MATLAB input and output mapping.

Since this module integrates process simulation and control into a single input-output model, only values for reference and output are available (control variables are internals) as shown in figure 11. The control algorithm implemented is a Generalized Predictive Controller with input constraints described in (Lopez *et alii,* 1995).

Application Example Design

The whole plant is shown in upper left corner of figure 13. There are five objects in the plant (sub-classes of algorithm object): *top draw composition*, *side draw composition* and *bottom temperature set points*; *shell-process* and *supervisor*. Set-point objects provide set-point patterns to test process behavior. They can be a sequence of set-point changes or they can be read from real process data. For every time interval, *set-point* objects provide values for the inputs of *shell-process*. The control algorithms are executed with these values and the results can be displayed for analysis. Supervisor object realizes intelligent plant diagnostics. These tasks are separate modules and they were presented in section 4.

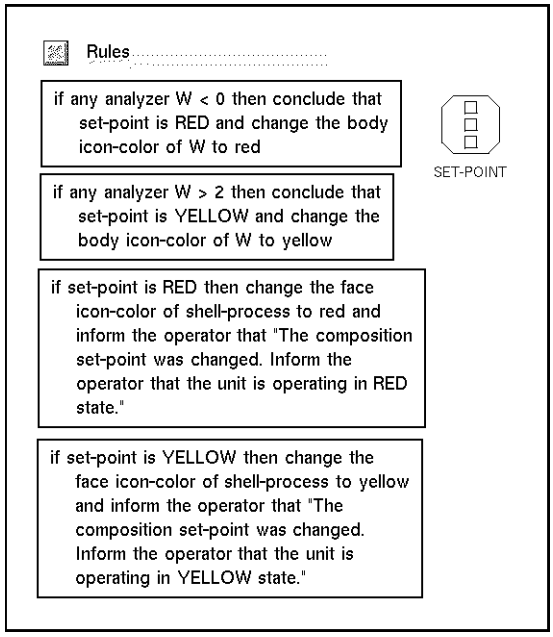

Figure 12 - Rules to supervise the flow composition in an analyzer.

In this application example, the workspace *Shell-Process-Details* is automatically created and shown by clicking on *Shell-Process* object into *Plant-Overview* workspace. Numerical values for any variable can be displayed into a graphics (as shown in figure 13 for *side draw composition*), by choosing the analyzer or another sensor associated with the variable. Moreover to prevent the user when a problem or an important event has occurred, the *supervisor* changes the color of the icon associated with the variable that has changed. For instance, the set of rules used to supervise the flow composition in an analyzer is presented in figure 12. In this figure, an analyzer with *yellow-icon* is under normal operation and *red-state* assigns a fault situation. In summary, the user interface is a front-end for external software that is very similar to the operator panel of the real plant.

Control algorithm description and some results

To implement the Generalized Predictive Control law, we have used the methodology proposed by Lopez *et alli* (1995). In this case, the cost function is defined by

$$\mathbf{J}(\Delta u) = \sum_{j=1}^{NU}\left[y_p(t+j)-\omega_y(t+j)\right]^t R_1\left[y_p(t+j)-\omega_y(t+j\right]+$$
$$\sum_{j=1}^{NU}\left[\Delta u(t+j-1)\right]^t R_2\left[\Delta u(t+j-1)\right]+$$
$$\sum_{j=N1}^{NY}\left[u(t+j-1)-\omega_u(t+j-1)\right]^t R_1\left[u(t+j-1)-\omega_u(t+j-1)\right]$$

where N1 and NY define the output prediction horizon; NU is the control horizon; R_1, R_2 and R_3 are the weighting diagonal-matrices, acting respectively on the output error, the control increments and the control error; $y_p(t)$ represents the prediction of the output signal y(t) and u(t) is the control signal; $w_y(t)$ and $w_u(t)$ are respectively the output and input set points; and $\Delta u(t) = u(t) - u(t-1)$ represents the incremental control signal.

By using the **Knowledge acquisition** module, and considering the particularities of the Shell process, i.e. the regulation of y_1 *(top draw composition)* and y_2 *(side draw composition)* while minimizing u_3 *(bottom reflux duty)*, the tuning parameters for the control algorithm were chosen as: [N1, NY, NU] = [1, 100, 5], diag R_1 = [10 0 0], diag R_2 = [5 5 5] and diag R_3 = [0 0 0.1]. From the supervisor's rule explanations, the reasons for these value choices are:

- the third output isn't a controlled variable, so its weight in the matrix R_1 is chosen equal to zero,
- in the matrix R_3, the only non-zero value (0.1) is the weight associated with the input u_3 we have to minimize,
- the other weights in matrices R_1 and R_2 were selected to achieve the performance requirements specified to the input/output signals.

The *bottom reflux duty* (u_3) minimization is performed by putting the corresponding set point at -0.5, which is the minimum value for all control signal; and all output set points values are null in order to achieve the output regulation. Two disturbances were considered: the first one corresponds to a step, whose magnitude value is -0.5, in the *intermediate reflux duty* (e_1); and the other one corresponds to an unmeasured disturbance of 0.5, in the *upper reflux duty* (e2).

The complete application was runned over 400 minutes, the final values of y_1 *(top draw composition)* and y_2 *(side draw composition)* are -0.0013 and 0.0042. These values satisfy the specifications 0.0 ± 0.005. The *bottom reflux duty* minimization after this time leads to -0.221 as final value for u_3. The *side draw composition* (y_2) graphic is displayed in the right side of figure 13.

7 CONCLUSION

A simulation environment is proposed for intelligent control engineering. The environment has four mainly software requirements:

- object-oriented implementation,
- hybrid knowledge representation,
- procedural knowledge is completely dissociated from intelligent knowledge,
- modularity and flexibility characteristics are prioritized before run-time cost.

By use of object-oriented approach, the environment is designed to assure an easy, uniform and consistently interaction with users. They can manipulate different tools (objects) to create a dedicated software configuration according to their preferences and application. These characteristics avoid the use of a large and inefficient environment and provide rational profit of available software resources.

To illustrate the environment potentialities, a prototype for simulation and control of an oil industry process was described. This prototype has a friendly user interface and its algorithms are implemented in MATLAB. There are also some knowledge bases to help the control design and supervise the closed-loop.

In summary, the environment proposed in this paper has four main characteristics: existing software can be easily integrated

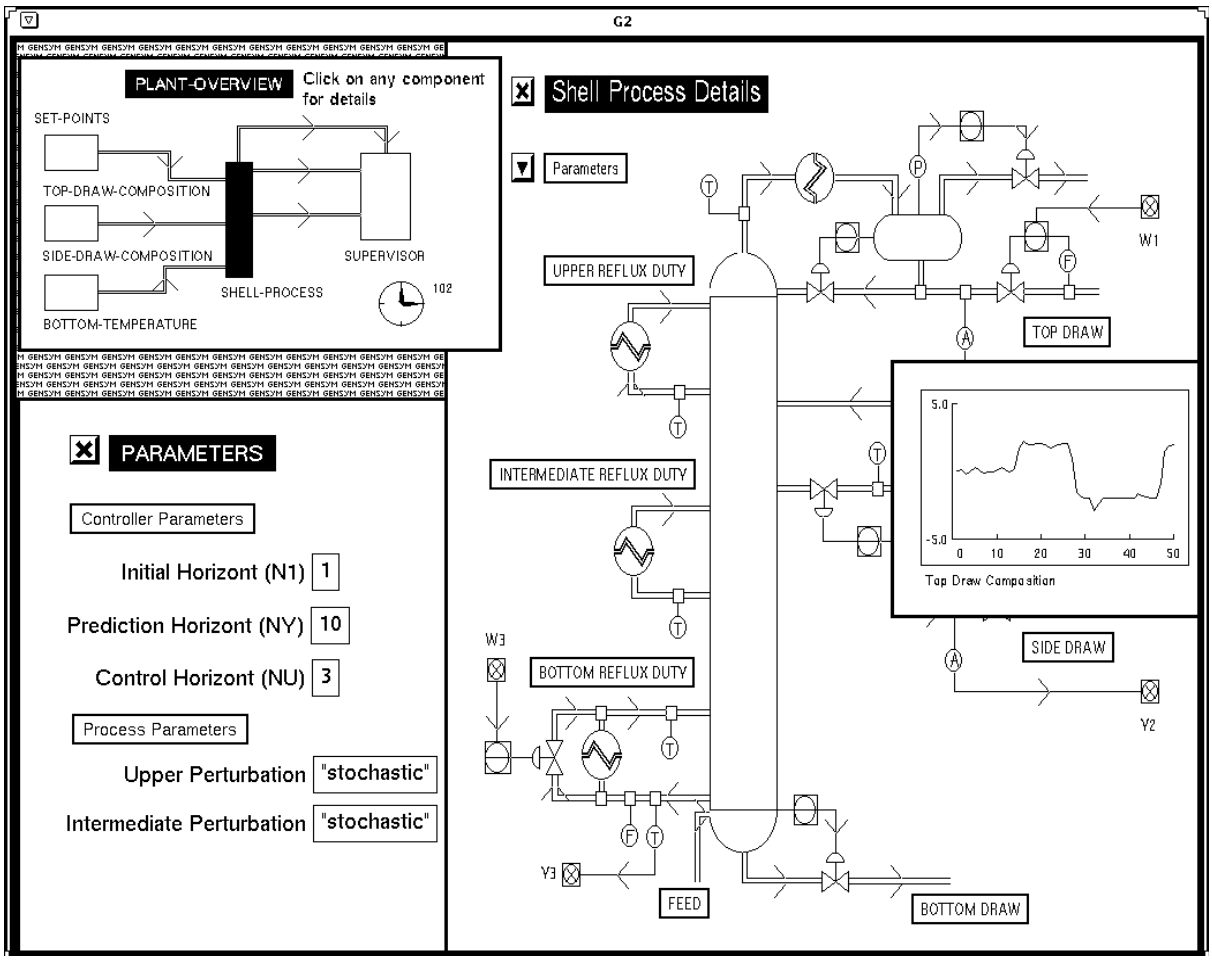

Figure 13 - System Interface.

allowing the use of desirable software properties (access to a large set of solutions to common problems, good numerical properties, efficient implementation, etc.); the end-user interacts with only one interface; modular and object-oriented development allows custom versions to be easily created, and the use of an expert shell allows "intelligent" solutions as a natural option. The potential of the proposed approach is aimed at developing more complex systems integrating control, identification, analysis and design tools.

REFERENCES

Amaral, W C, Gomide, F A C, Nazzetta, R and Oliveira, G H C (1992). Intelligent environment for system identification and adaptive control, in Jamshidi, M and Herget, C J (eds.), *Recent Advances in Computer-Aided Control System Engineering*, Elsevier Science Publishers B. V.

Arruda, L V R, Lüders, R, Amaral W C, Gomide, F A C (1994a). A Knowledge-based Environment for Intelligent Design and Supervision on Control Systems, *IEEE Int. Conf. on System, Man and Cybernetics*, San Antonio, Texas, USA, pp. 2680 - 2685

Arruda, L V R, Lüders, R, Amaral W C, Gomide, F A C (1994b). An Object-oriented Environment for Control Systems in Oil Industry, *3rd IEEE Conf. on Control Applications*, Glasgow, Scotland, pp. 1353 - 1358

Barker, H A, Chen, M, Grant, P W, Jobling, C P Townsend, P (1993). Open architecture for computer-aided control engineering, *IEEE Control System*, vol.13, n° 3, pp. 17-27.

Coad, P, Yourdon, E (1991). *Object-oriented Analysis*, Prentice-Hall, USA

DeChampeaux, D, Lea D, Faure P (1993). *Object-oriented system development*, Addisson Wesley, USA

Gensym Corporation (1992). *G2 Reference Manual - version 3.0*

Lopez, J F, Oliveira, G H C, Amaral, W C, Latre, L G, Favier, G, and Acundeger, E (1995). Multivariable constrained predictive control methods applied to the shell benchmark problem: a comparison. *3rd European Control*, Rome, Italy, pp. 3259 - 3264

Pang, G K H (1992). Knowledge Based Control System Design, in Jamshidi, M and Herget, C J (eds.), *Recent Advances in Computer-Aided Control System Engineering*, Elsevier Science Publishers B. V.

Peterson, G E (1990). *Tutorial: Object-oriented computing*, vol. 1, IEEE Computer Society Press, USA

Prett, D M, García C E, and Ramaker, B L (1990). *The Second Shell Process Control Workshop.* Butterworths, Stoneham, USA

Taylor, J H, Frederick, D K, Rimvall, C M, and Sutherland, H (1990). Computer - aided control engineering environments: architecture, user interface, database management and expert aiding. *Proc. 11th IFAC Congress*, Taillin, USSR, pp. 337-348