ELSEVIER

Simulation Practice and Theory 6 (1998) 703–719

SIMULATION PRACTICE AND THEORY

Extension software for real-time control system design and implementation with MATLAB-SIMULINK

Borut Zupančič *

Faculty of Electrical Engineering, University of Ljubljana, Tržaška 25, 1000 Ljubljana, Slovenia

Received 29 September 1997; revised 14 April 1998

Abstract

The paper deals with a unified environment for the design and implementation of control schemes. The widely used MATLAB-SIMULINK is used for control scheme description. The implementation hardware is from Mitsubishi PLC. The SIMULINK library was extended with target hardware blocks. After an off-line design procedure, which can be combined with hardware-in-the-loop experiments, the control scheme is translated into PLC code in three steps. The results show that the autocoding of small, low-cost industrial controllers can be efficiently realized by integrating widely used, inexpensive and commercially available tools.

Keywords: Real-time simulation; Real-time system; Hardware-in-the-loop; Control system; Computer-aided control system design

1. Introduction

The programming of industrial regulators and programmable logical controllers is currently implemented by very inflexible and closed program packages which usually run on personal computers or on special programming hardware. There are no standardization efforts to obtain at least a small degree of portability of regulator configurations. These packages are, in comparison with modern software equipment, much less user-friendly and much more limited in terms of possible expansions with appropriate libraries of blocks. As such, they do not enable the efficient and creative solution of engineering problems. Also such packages do not enable the simulation of control systems before real implementation. For this purpose a simulation or a computer-aided design tool (MATRIXx-System_Build, MATLAB-SIMULINK [1,2], ACSL, etc.) for the verification and validation of a developed control scheme must be used. Although the simulation produces acceptable results it does not

* Corresponding author. Fax: +386 61 1264631; e-mail: borut.zupancic@fe.uni-lj.si

0928-4869/98/$ – see front matter
PII: S0928-4869(98)00012-3

guarantee that the real implementation will produce comparable results. Everyone who has taken part in the implementation of PID schemes knows that controllers with the same gain, integral and derivative constants can behave quite differently. Why? Because control as well as other blocks can be (and usually are) implemented in quite different ways in simulation and implementation packages. The blocks can be implemented with continuous (with integration) or discrete algorithms. The arithmetic can be different. Off-line packages have double precision arithmetic, the arithmetic on low-cost controllers can be very poor (fixed point, integer, etc.). Simulation does not take into account the discretization effects of conversion units (e.g. A/D and D/A converters). Finally, off-line simulation produces results which do not take into account the delay between input and output controller signals. This delay can have a significant influence when the scheme is implemented in real time. All these problems above intensify difficulties in education because at least at the beginning students must learn things which are much more fundamental than those mentioned above.

Powerful CACE packages should cover the whole design cycle from specifications to implementation. However, the packages that enable the automatic generation of a code for target hardware are very rare. Sometimes it seems that this feature is successfully implemented (e.g. MATRIXx-System_Build, MATLAB-SIMULINK). In the MATLAB-SIMULINK environment one can generate an optimized C code for continuous-time, discrete-time and hybrid systems (event-driven behavior which is typical in complex control systems) automatically from SIMULINK and STATEFLOW models with the aid of REAL-TIME WORKSHOP and STATE FLOW CODER. The generated code can be used in a variety of applications, including rapid prototyping, H-I-L simulations, and real-time testing on embedded hardware. The efficiency for H-I-L implementation is also improved by so-called blocksets in SIMULINK. The Fixed-Point Blockset enables a user to model and simulate digital control systems and digital filters within SIMULINK with special blocks having fixed point computation behavior. DSP Blockset brings real-time DSP acceleration to SIMULINK modelling providing access to a wide variety of DSP platforms. For prototyping solutions one can take advantage of solutions available from different companies. dSPACE is a development environment for rapid control prototyping and hardware-in-the-loop simulation. WinCon allows one to run Real-Time Workshop code independently on a PC Real-Time Toolbox SIMULINK Extension. World Up enables the creation and control of 3-D interaction worlds for real-time visualization. RealLink/32 allows one to run Real-Time Workshop code on Intel based computers operating under Windows NT. ADI Real-Time Station is a complete system solution for H-I-L simulation and prototyping.

However, when analyzing education as well as real implementation needs concerning the process industry which is our target area (above all smaller automation projects in chemical engineering), it was concluded that the high performance prototyping solutions cited above are not what is really needed. What would be applicable for our needs is the use of an efficient and widely accepted modelling, simulation and design environment, which would perhaps overcome the gap between university and industry if combined with low-cost but real industrial hardware—industrial regulators and programmable logical controllers which are used extensively

in process automation. This conclusion represented the motivation behind our attempt to realize a unified environment for the design and implementation of control systems. For control system description, off-line simulation and design, a widespread MATLAB-SIMULINK on a personal computer was used. For the implementation hardware, a Mitsubishi programmable logical controller (series A1N and A1S) was chosen due to our experience with it in industrial applications. This controller enables, in addition to sequential functions (traditional for PLCs), regulator functions using the package IDR (Interpretative digital regulator [3]), previously developed by the Slovene Mitsubishi representative INEA. In addition to traditional control loops, IDR also enables some feed forward solutions, ratio control and some other multi-loop schemes. Also the possibility of adaptive and fuzzy control algorithm inclusion is under investigation [4].

However, using the concrete design environment and the concrete target hardware, the possibility for portability of our proposed solution seemed to be questionable. That is to say, there are no widely recognized standards supporting CACE packages (at least in the sense of model description data base) and the programming of industrial controllers as well. The only possibility was to implement a suitable controllers' configuration description (intermediate description) between the CACE environment and the target hardware programming environment. This part of the proposed environment means the general solution. Interface parts, between CACE and controller environments are specific. All scalability problems are, with such a concept, focused more or less on the target hardware limitations.

With such an environment several benefits were expected:

- efficient and creative students' work during exercises,
- decreased number of tools students have to be acquainted with for the whole design cycle,
- conviction of industrial engineers that tools such as MATLAB-SIMULINK can also be appropriate for industrial problems and not just for university education.

So our attempt does not represent any form of competition with some of the MATLAB-SIMULINK solutions, and especially not with high performance computing solutions. It simply reflects many years of experience in our control group with education for process industry automation in which one basic question appeared: Is an environment such as MATLAB-SIMULINK, with all add-ons beside off-line simulation and design, really appropriate only for high sophisticated and high-performance computing prototyping solutions or can it also be used for design and implementation (programming) of industrial controllers and PLCs?

2. Description of the environment

The concept of the proposed system is presented in Figs. 1 and 2 [5]. The graphic icon oriented editor (SIMULINK in our case) is used for control system definition. The control system can be off-line simulated (or optimized, linearized, etc.) in MATLAB-SIMULINK (1). When such experiments give satisfactory results, the control part of the overall control system can be validated on the real process using MATLAB-SIMULINK real-time facilities with appropriate PC hardware (Burr

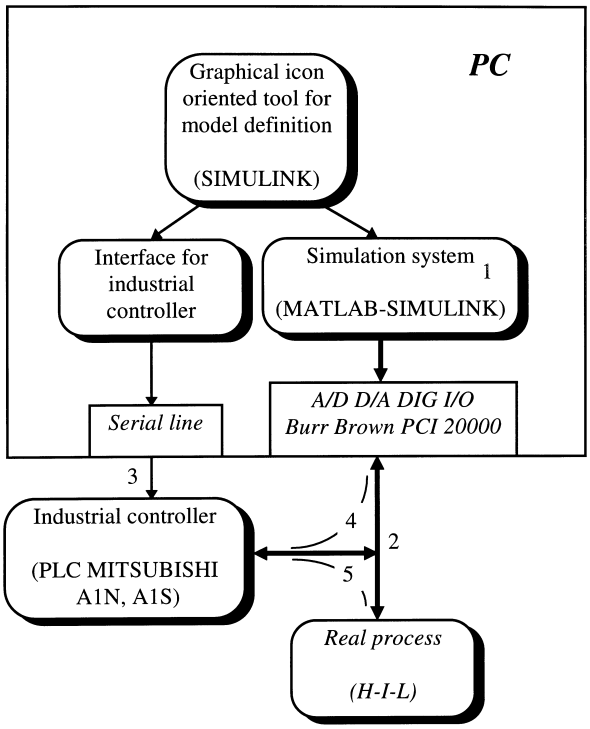

Fig. 1. The concept of the environment.

Brown PCI 20000) (2). When these experiments produce satisfactory results, the control part of the overall simulation scheme can be further processed and downloaded to the industrial hardware (3). When the industrial controller is configured, its functioning can be tested on the model of the process which runs in real-time in MATLAB-SIMULINK (4) or on the real process (5). The implementation of the proposed concept demanded the following activities:

- extension of the MATLAB-SIMULINK with real-time possibilities,
- extension of the SIMULINK library with target hardware blocks,
- translation of the SIMULINK control scheme into intermediate target hardware independent description,
- translation of intermediate control scheme description into the source program of the target hardware (IDR program for programmable logical controller in our case).

The activities mentioned are described in the sections which follow.

3. Extension of the MATLAB-SIMULINK with real time possibilities

A low-cost, real-time stimulation environment was built with the use of a Burr–Brown PCI-20000 process interface. A new block named PCI-20000, which

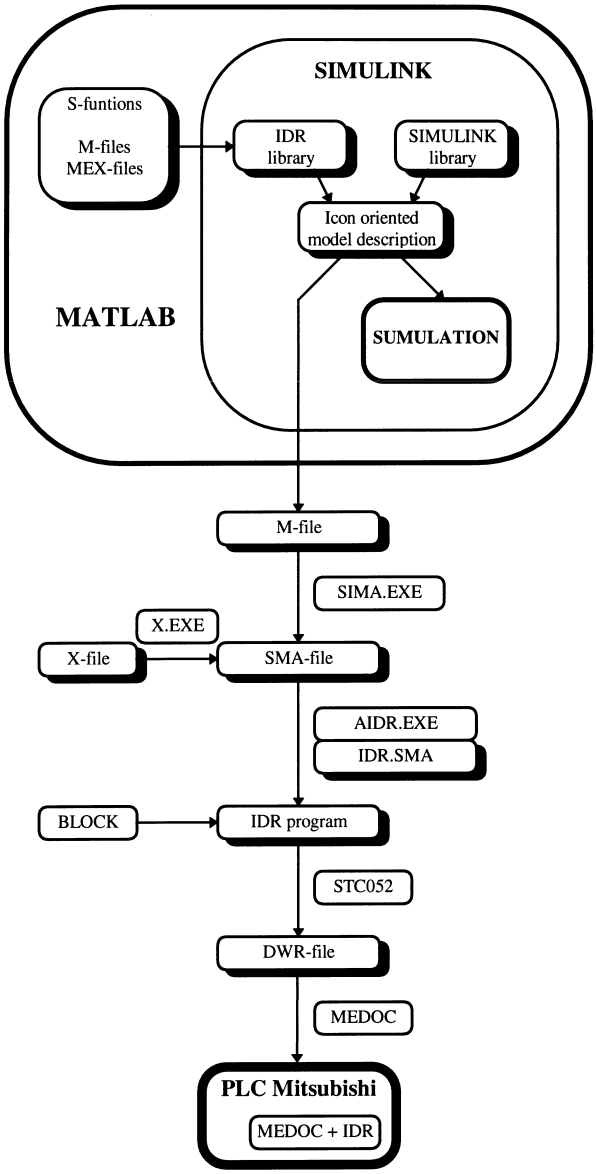

Fig. 2. Flowchart of control schemes processing.

can be considered as real hardware, was added to the SIMULINK block library. The MATLAB-SIMULINK real-time concept is shown in Fig. 3. The block incorporates all functions necessary to access the process hardware and also a mechanism to force the simulation scheme to run in real-time. The input/output functions of the block include the digital to analog conversion (D/A) of the vector of the block

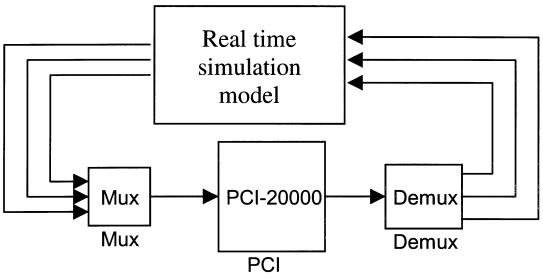

Fig. 3. The concept of real-time implementation in MATLAB-SIMULINK.

input signals and analog to digital conversion (A/D) with resulting values mapped to the vector of block outputs.

The timing of calculations during simulation is illustrated in Fig. 4. The real-time synchronization is obtained by hardware pulse generators which are programmed to produce pulses of the desired sampling frequency. After the simulation begins, some model blocks whose inputs do not depend on the external hardware (PCI) signals, are calculated first. After that the control is transferred to the PCI-20000 block. Here D/A conversion is performed first in every iteration and then the program enters a waiting loop until the synchronization pulse occurs. At this point the A/D conversion

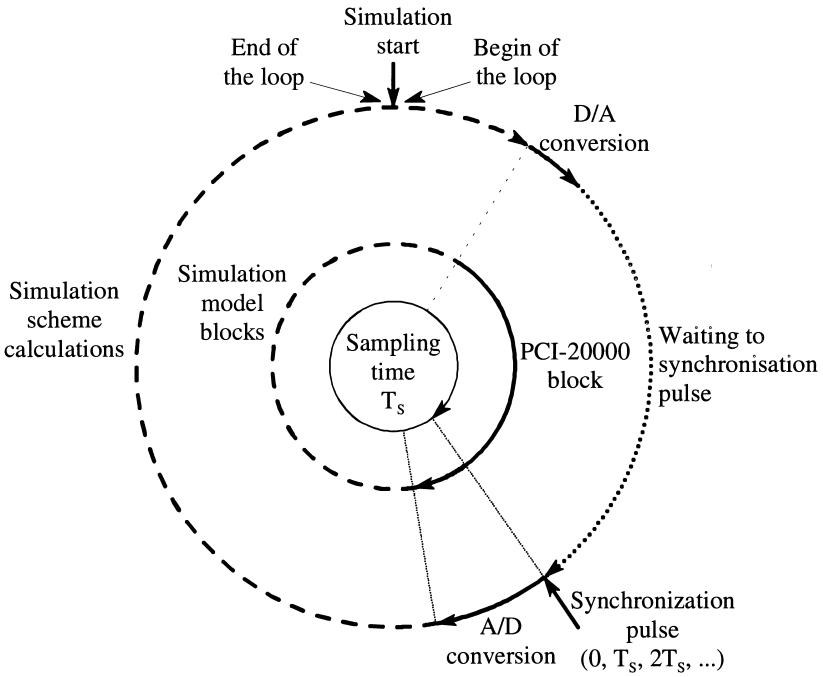

Fig. 4. Timing of calculations during simulation.

starts ($t=0$, Ts, 2Ts, etc.) and after the end of conversion, which is detected through a certain flag setting, the converted values are read and the program control is continued by executing other blocks of the real-time simulation model. The appropriate sequence is obtained as the PCI-20000 block is installed as a delayed block (dirFeedThru = 0). So the sorting algorithm places this block prior to those model blocks whose inputs depend on PCI-20000 (or real hardware) outputs.

4. Target hardware blocks in SIMULINK

Although SIMULINK possesses some blocks that are functionally compatible with IDR ones, they are in many respects so different that they cannot be used for a reliable simulation/implementation approach. So it was decided to generate a new IDR library which should contain all target hardware blocks (IDR blocks) in the MATLAB-SIMULINK environment. These blocks are programmed as similarly as possible to the real hardware blocks. This means that the code, the inputs, the outputs and the parameters are the same. They are implemented as discrete blocks. The arithmetic used is as close as possible to the arithmetic in IDR. As the blocks in IDR are programmed in C too, the features mentioned can easily be obtained (using S-functions as MEX files). The inputs and outputs of all blocks are integers, as is the case in IDR. The process and the control parts are connected (separated) by ANIN (A/D) and AOUT (D/A) blocks. At the beginning the following blocks were included:

PROG —program block. Each IDR program must possess this block. It also defines the name of the program.
LOOP —one or several blocks in each scheme. Each block defines the identificator number of a loop, the sampling period (frame time) and the priority of the loop.
CON —the generation of a constant.
ANIN —analog input.
DIN —digital input.
AOUT —Analog output.
PID/P —positioned PID controller.
ADD —summation of two signals.
SUB —subtraction of two signals.
MUL —multiplication of two signals.

The following characteristics and problems appeared during the realization of IDR blocks and the simulation/implementation environment in SIMULINK:

- All IDR blocks have the mnemonic idr in the first row inside the icon. The second row defines the type of each IDR block. Only the blocks with the mnemonic idr are later converted into the IDR program.
- In SIMULINK, the sampling time is a feature of a particular block. In the IDR implementation package, the sampling time is a feature of the loop. During simulation each discrete block automatically obtains the sampling time of the appropriate loop (defined in LOOP block) in which it appears.

- Each loop in IDR PLC implementation additionally has a priority. It is not possible to take it into account during simulation.
- It is well known that blocks are automatically sorted during simulation. However, the implementation package IDR does not support this feature. So each IDR block in the SIMULINK library has an additional parameter that denotes its successive number in the loop. If blocks are differently sorted during simulation and implementation respectively, this can be an important source of differences in results.
- SIMULINK enables a block with a maximum of six parameters. As particular blocks in the IDR package have more parameters, some of them were grouped into vectors.

To explain better the procedure of how to develop the IDR library in SIMULINK, three important blocks will be described briefly; PID controller (PID/P), analog input (ANIN) and analog output (AOUT) blocks.

4.1. Realization of PID/P block

The syntax of its definition in IDR language is as follows:

BLOCK:	name	;name of block
!STATUS:	act or noact	;activity
MODE:	sup or cas	:mode (constant local reference, cascaded external reference, tracking)
TYPE:	PID/P	;type of block
PVB:	name	;name of block which gives controlled variable (input)
CSPB:	name	;name of block which gives reference value (input)
!TRACKB:	name	;name of block, to which the output should track in the case TMODEB > 1 (input)
!TMODEB:	name	;name of block which defines tracking (input)
!DEB:	name	;name of block which defines the signal coming to the differential term (input)
!SP:	value	;local reference, −32000 to 32000 (sup mode)
!K:	value	;gain, −100.00 to 1000.00
!TI:	value	;integral constant, 0 to 3000.0 s
!TD:	value	;derivative constant, 0 to 3000.0 s
!BIAS:	value	;initial value of output, 0 to 100.00
!DEAD:	value	;dead zone, 0 to 10000
ENDB		

As seen from the description, the controller is much more complicated than it is usually considered in simulation studies. The SIMULINK implementation must provide an internal (local) or external reference, the possibility for tracking an output variable to an input one and three different possibilities for signals, which are differentiated: the control error, the controlled variable or an arbitrary signal. Fig. 5 shows the implementation of the IDR type PID controller in SIMULINK.

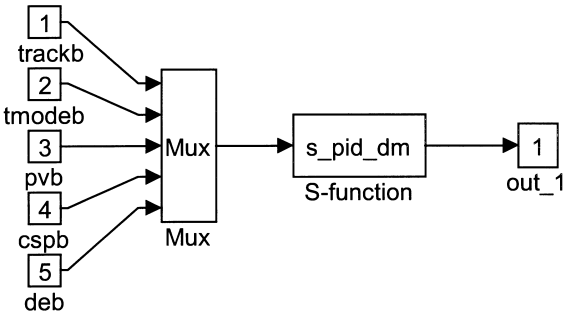

Fig. 5. Implemenation of IDR type PID controller in SIMULINK.

Two versions of PID S-functions were realized: one as the M-file and one as the MEX-file.In the second example, the original IDR C-coded module with corresponding modifications due to S-function requirements for dynamic and discrete blocks was used. Comparing the results of both implementations, the calculation efficiency as well as the influence of integer arithmetic in MEX realization was studied.

The parameters of original IDR blocks are integers. Nevertheless they are defined in SIMULINK IDR blocks as floating point numbers, which is much more user-friendly. These values are automatically converted to integers by MATLAB in the case of simulation and by program module AIDR.EXE in the case of PLC implementation.

4.2. Analog input (ANIN) and analog output (AOUT) blocks

The implementation of both blocks for the IDR library in SIMULINK is a complex problem. During simulation these blocks include the effects of hardware modules, e.g., the ranges, the calibration and resolution.

Fig. 6 represents the implementation of the ANIN block in the IDR SIMULINK library. The ANIN block usually "converts" the range 0 to 10 V into the integer

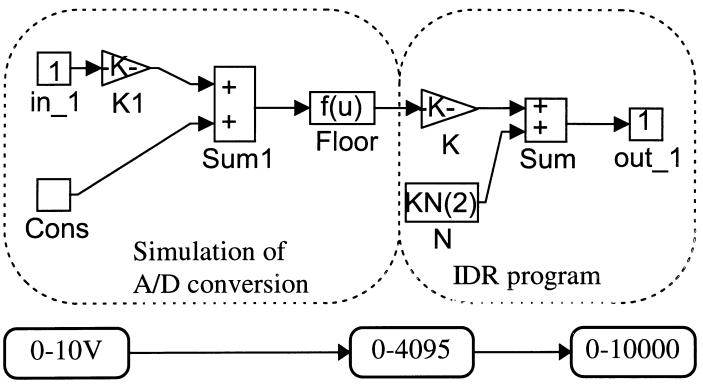

Fig. 6. Implementation of ANIN block in IDR SIMULINK library.

value 0 to 4095 (this is the simulation of what the hardware really does). In reality this code is transformed to the value 0 to 10000 by the IDR program. When the SIMULINK scheme is processed to the IDR program only the second part of the scheme is taken into account.

Fig. 7 represents the implementation of the AOUT block in the IDR SIMULINK library. During simulation the block usually converts the input range 0 to 10000 into a code 0 to 4095 or 0 to 2047. The second part simulates IDR software, which converts this code into current 0–20 mA or into voltage 0–10 V. During implementation processing only the first part of the scheme is used.

5. Translation into the hardware independent model description

After the definition and testing of the overall control system in the SIMULINK off-line simulation environment, the SIMULINK model written in M-file S-function (⟨model⟩.M), which describes the topology of a model and parameters, is translated into target hardware independent and block oriented SMA model file (⟨model⟩.SMA) using program module SIMA (SIMulink into Alternative file). The description of the intermediate file was defined by the influence of some well-known block oriented simulation languages, especially PSI language [6]. The inclusion of the intermediate model description enables the following advantages:

- The format of the IDR implementation program is very stiff, parameters and inputs must be defined in a fixed order.
- The concept becomes much more convenient and open to the use of other model definition (simulation, CACE) environments and other hardware (implementation) configurations respectively.

Each SMA description consists of CONFIGURATION PART and of PARAMETERS

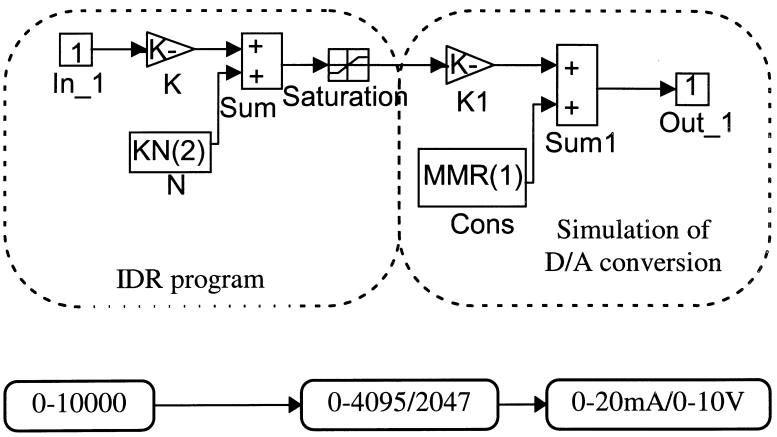

Fig. 7. Implementation of AOUT block in IDR SIMULINK library.

PART. The first part describes the structure of a model and the second one describes its parameters. The rows have the following syntax:

CONFIGURATION PART
Block, Type, $Input_1$, $Input_2$, $Input_3$, ...
.
.
PARAMETER PART
Block, Address, Par_1, Par_2, Par_3, ...
.
.

Block	...	name of block (e.g. ANIN, PID/P, ...)
Type	...	type of block (e.g. SUM, INTEG, ...)
$Input_i$	...	name of block in which output is connected to the ith input of this block
Address	...	not used in this application
Par_i	...	ith parameter of block

The procedure of translation has the following important steps:

- Initialization of the system and data structures.
- Decoding of particular SIMULINK model description M-file statements; processing of add_block (detection of blocks and sub-blocks which must be ignored, placement of blocks' names into a list), new_system, set_param (position, inputs, orientation, initial, gain, mask type, mask entries (list of all parameters), add_lines (determination of connections).
- Finding connections in the true topological form (many problems have been solved due to different orientation, re-dimensioning of blocks, connection tolerances (signal lines are not precisely connected to blocks or other lines), non-connecting inputs, branching connections; the main problem is that there is no documentation about the topological description in M-file).

6. Translation into the IDR program

As shown in Fig. 2, the SMA model description file is further automatically translated into target dependent IDR program. To achieve maximum flexibility, all target program structures were avoided by the translation code. Instead an external controlling file (IDR.SMA) containing all the necessary data for translation was added. So the following advantages were achieved:

- new IDR blocks can easily be added,
- the format of an IDR block can be changed,
- different controlling files IDR/SMA can be used for different target hardware programs.

The row from the file IDR.SMA describing a block can be generalized as follows:

idrNAME,,LOOP,*,[K*]$PARAMETER_1$|#$INPUT_1$|!IINPUT$_1$,[K*]$PARAMETER_2$|
#$INPUT_2$|!IINPUT$_2$, ...

idrNAME	name of IDR block
LOOP	number of the loop which includes the block
PARAMETER$_i$	ith parameter
INPUT$_i$	name of the ith input of the block
*	successive number of the block in the scheme
K*	constant with which a SIMULINK defined parameter must be multiplied in order to obtain an adequate value for IDR program
#	sign for inputs (to be distinguished from parameters)
!	sign for a non-standard feature—the parameter is the name of a block

The scheme which is used for off-line experimentation in SIMULINK contains, in addition to IDR blocks, some other standard SIMULINK blocks. As all blocks are translated into intermediate SMA description, the translator AIDR.EXE must translate only the appropriate IDR blocks into the IDR program.

7. Example: cascade control of a hydraulic plant

The features of the described environment for the simulation and implementation of control scheme will be presented using the cascade control of a hydraulic plant [7]. As the main goal was to validate or to compare the results obtained by SIMULINK simulation with results obtained by automatically coded PLC Mitsubishi A1 as the controller, it was decided to use the simulated model instead of the real process in both cases. So the possible disagreements due to some modelling discrepancies were eliminated. Simultaneously the implemented MATLAB-SIMULINK real-time facility was tested, as in conjunction with the Mitsubishi controller the hydraulic model had to be simulated in real-time.

Fig. 8 shows the block diagram, which was realized with standard SIMULINK blocks. The hydraulic process consists of three tanks. The levels are measured in the first and in the third tank. The main loop controls the level of the third tank $h(t)$ and the auxiliary loop measures the level of the first tank. The control valve is driven by the auxiliary controller output.

The results obtained with standard SIMULINK simulation with a Mitsubishi

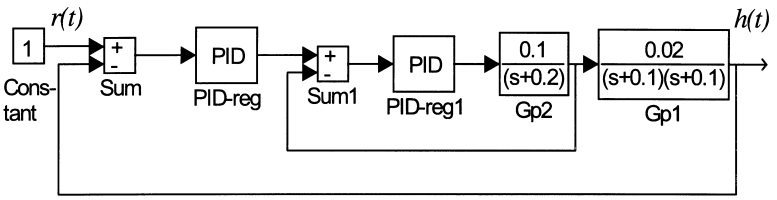

Fig. 8. The cascade control of a hydraulic plant.

implemented controller were quite different for the reasons mentioned in Section 1 and which motivated the implementation described. So the next step was to implement another SIMULINK scheme using IDR blocks. This scheme is shown in Fig. 9.

Two idrLOOP blocks define the sampling times for main (slow) loop and for auxiliary (fast) loop. So the example is a good test for the evaluation of multi-rate sampling schemes' performances. With experimentation in SIMULINK the following controllers' parameters were chosen:

main controller	(PID_REG)	K = 1.8	Ti = 20.6	Td = 5.5	Ts = 1s
auxiliary controller	(PID_REG1)	K = 2	Ti = ∞	Td = 0	Ts = 0.2s

The scheme shown in Fig. 9 was then translated into hardware independent intermediate control system description file and further into IDR program. These files are shown in Tables 1 and 2.

The IDR program was then compiled and downloaded into the PLC Mitsubishi A1S and then tested in the real-time model of the process. The required scheme was obtained from the scheme in Fig. 9. The blocks belonging to the control scheme were eliminated and then the SIMULINK real-time block PCI-20000 representing real hardware was added.

The scheme for the experimentation described is shown in Fig. 10. The sampling time in real-time simulation was 0.2 s. Three D/A (reference,main and auxiliary control variables) and one A/D channel (for control signal) were used.

Fig. 11 shows the control variables (control valve input) and the controlled variables (level) of the simulation and implementation experiment. The responses are very similar. Some differences occur in control signals manly because PLC Mitsubishi A1S and SIMULINK real-time implementation have separate real-time synchronization.

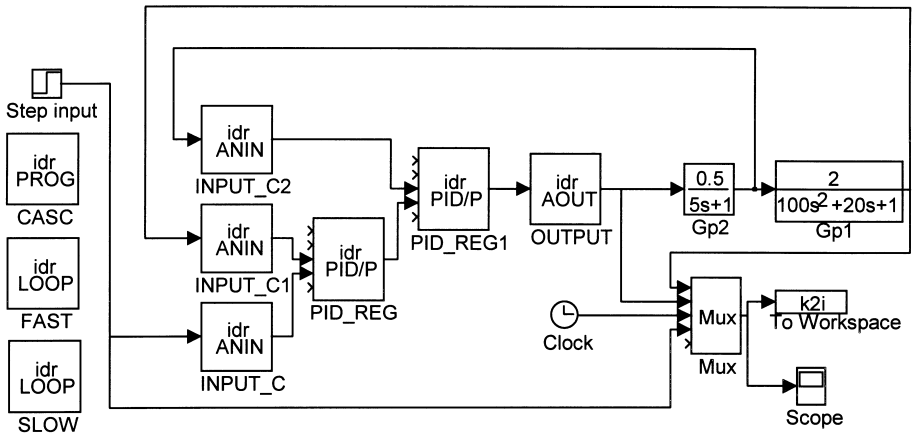

Fig. 9. Cascade control using IDR blocks.

Table 1
Hardware independent control scheme description file

CONFIGURATION PART		
CASC,	idrPROG	
Gp1	Transfer Fcn,	Gp2
Step input,	Step Fcn	
Gp2,	Transfer Fcn,	OUTPUT
Mux	Mux	OUTPUT, Gp1, Clock, Step input
To Workspace	To Workspace,	Mux
Scope,	Scope,	Mux
Clock,	Clock	
FAST,	idrLOOP	
SLOW,	idrLOOP	
INPUT_C2,	idrANIN,	Gp2
INPUT_C1,	idrANIN,	Gp1
PID_REG1,	idrPID/P,	N.C., N.C., INUT_C2, PID_REG
PID_REG,	idrPID/P,	N.C., N.C., INPUT_C1, INPUT_C
INPUT_C	idrANIN,	Step input
OUTPUT,	idrAOUT,	PID_REG1
PARAMETER PART		
CASC,		
Gp1,		
Step input,		
Gp2,		
Mux,		
To Workspace,		
Scope,		
Clock,		
FAST,,	1, 0.2, 1	
SLOW,,	1, 1, 1	
INPUT_C2,,	2, 1, 602, 1, 0, 0, 10, 4096	
INPUT_C1,,	1, 1, 601, 1, 0, 0, 10, 4096	
PID_REG1,,	2, 3, cas, 0, 2, 0, 0, 0, 0, 0	
PID_REG,,	1, 2, cas, 0, 1.8, 20.6, 5.5, 0, 0, 0	
INPUT_C,,	1, 1, 600, 1, 0, 0, 10, 4096	
OUTPUT,,	2, 4, AUT, 630, 0, 100, 0.5, 0, 0, 10, 2048	

8. Conclusions

As stated, our attempt was not to compete with some of the MATLAB-SIMULINK real-time high performance computing solutions but to answer the important question of whether an environment such as MATLAB-SIMULINK can also be used for design and implementation (programming) of process automation solutions with industrial regulators and PLCs. Initial experience with the described environment is quite promising. It seems that significant problems of how to generate the intermediate model description form the SIMULINK model have been solved. However, this processing still needs some testing, particularly in terms of obtaining topological data from graphical data. More information about corresponding

Table 2
IDR program

PROGRAM:	CASC	LOOP:	FAST
LOOP:	SLOW	PERIOD:	0
PERIOD:	10	PRIORITY:	1
PRIORITY:	1	BLOCK:	INPUT_C2
BLOCK:	INPUT_C	TYPE:	ANIN
TYPE:	ANIN	CHANN:	602
CHANN:	600	K:	1000
K:	1000	ENDB	
ENDB		BLOCK:	PID_REG1
BLOCK:	INPUT_C1	MODE:	cas
TYPE:	ANIN	TYPE:	PID/P
CHANN:	601	PVB:	INPUT_C2
K:	1000	CSPB:	PID_REG
ENDB		DEB:	PID_REG1
BLOKC:	PID_REG	K:	200
MODE:	cas	ENDB	
TYPE:	PID/P	BLOCK:	OUTPUT
PVB:	INPUT_C1	MODE:	aut
CSPB:	INPUT_C	TYPE:	AOUT
DEB:	PID_REG	PVB:	PID_REG1
K:	180	CHANN:	630
TI:	206	HILIM:	10000
TD:	55	K:	500
ENDB		ENDB	
ENDL		ENDL	
		ENDP	

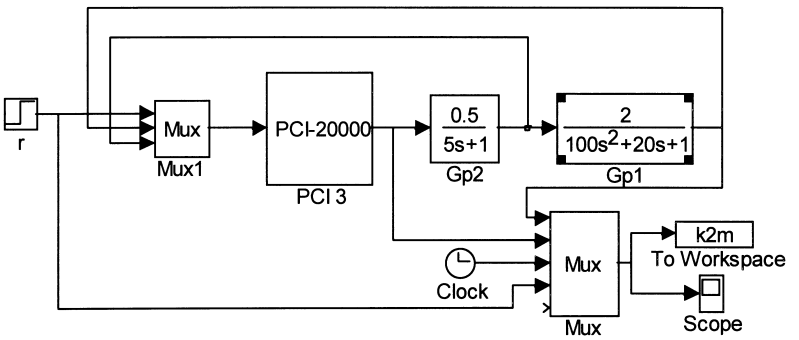

Fig. 10. Experimentation with the real-time model.

SIMULINK features would be helpful. The important advantage of the implemented real-time MATLAB-SIMULINK facilities is that the interactions and presentations of results are attained during simulation. The disadvantage is of course the speed limitation. The true value of the results described lies, in our opinion, in the fact that such efficient environments which can be used for autocoding of low-cost small industrial regulators and controllers are very rare, if indeed they exist at all. Valuable

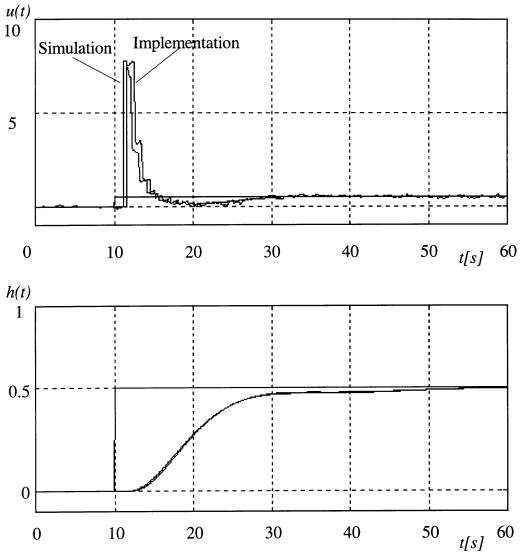

Fig. 11. Comparison of the simulation and implementation experiment.

experience was also obtained in undergraduate education, which introduces new aspects in the proposed environment validation.

The problems concerning the portability of the proposed solution, which are described in Section 1, were tackled by using a suitable controllers' configuration description between the CACE environment and the target hardware programming environment. In this way the manufacturers of CACE packages (e.g., MATLAB-SIMULINK, MATRIXx-System_Build, etc.) could develop the interface of the intermediate description and the manufacturers of industrial controllers and PLCs (e.g., Siemens Sipart DR24, Mitsubishi A1S, etc.) the interface between intermediate control scheme description to a target hardware program (or programming environment). All scalability problems with such a concept are focused more or less on the target hardware limitations.

The stage of implementation of the environment described is, of course, at an academic rather than a professional level. However, we do not see serious problems for companies producing, for example, industrial regulators to develop an appropriate interface for MATLAB-SIMULINK. Can you imagine lucky users being able to develop control schemes and to implement them with a kind of unified and standardized environment, e.g., MATLAB-SIMULINK? But those were the old days when the standards had chances for acceptance. Even for pure simulation tools they came to an end with the CSSL '67 standard. From then on agreements between producers of tools, or even between different working groups preparing standards, were no longer possible. CSSL '81 fell through, what about VHDL or Modelica? Probably these considerations lead to the key problem: If standards cannot be accepted in a pure simulation area, how can they be accepted in areas such as control engineering where the influence and interest of industry is even stronger?

References

[1] MATLAB, High Performance Numeric Computation and Visualization Software, Users Guide, The Math Works, Natick, MA, 1993.
[2] SIMULINK, Dynamic System Simulation Software, Release Notes, Version 1.3, The Math Works, Natick, MA, 1993.
[3] D. Čuk, I. Makovec, B. Kramar, User's Manual for the Program Package Interpretative Digital Controller IDR BLOK, INEA, Domžale, Slovenia, 1993.
[4] D. Matko, B. Zupančič and Z. Šehić, Selftuning PLC, in: Preprints of IFAC Symposium on System Identification SYSYD '94, Copenhagen, Denmark, vol. 3, 1994, pp. 585–590.
[5] B. Zupančič, R. Karber, D. Matko, M. Vrečer, Environment for simulation and implementation of control schemes using MATLAB SIMULINK, in: Proceedings of the 1995 Summer Computer Simulation Conference, Ottawa, Canada, 1995, pp. 722–727.
[6] P.P.J. Van den Bosch. Simulation program PSI, User's Manual, Delft University of Technology, Netherlands, 1987.
[7] D. Matko, B. Zupacic, R. Karba, Simulation and Modelling of Continuous Systems—A Case Study Approach, Prentice Hall, London, 1992.