ADVANCED CONTROL ALGORITHMS + SIMULINK COMPATIBILITY + REAL-TIME OS = REX

Pavel Balda, Miloš Schlegel, Milan Štětina

University of West Bohemia in Pilsen,
Department of Cybernetics,
Univerzitní 8, 306 14 Plzeň, Czech Republic.
e-mail: pbalda@kky.zcu.cz

Abstract: This paper describes the structure and features of a new industrial control system called REX. During the system design, special attention was paid to the simulation facility of the control algorithms. The equivalent behavior of simulation and real-time control is guaranteed by a large function blocks library which is implemented for both Simulink and each target platform. REX is suitable for building real-time control systems of real, virtual and remote laboratories. REX does not utilize Real Time Workshop by The Mathworks.

Keywords: industrial control system, embedded control, Simulink, OPC, real-time control system.

1. INTRODUCTION

There is a rather wide gap between control engineering theory and practice. The way from the development of a new algorithm to its practical implementation is usually very long and laborious because both areas use different tools. Currently, Matlab-Simulink (The Mathworks, 2001) is perhaps the most widely spread program system used for the development and testing of new algorithms in industrial process control. Simulink is more frequently used not only in academic institutions but also in engineering practice. Practical use of standard Simulink blocks is, however, not without obstacles, because these blocks have not been developed for industrial applications.

To bridge the above mentioned gap, a new control system, REX, has been developed. The main design requirements have been as follows:

- Compatibility with Matlab-Simulink
- Development of industrial control block library (blockset)
- Openness of the system, easy creation of new algorithms
- Suitability for the teaching of control engineering
- Support of industrial standards and internet technologies

The objective of the paper is to describe briefly the creation of a REX application and to emphasize the possibility of simulating it in Simulink before putting it into operation. The REX control system is an open and scalable system, which is suitable for embedded control. REX can be easily ported to different platforms with C and C++ language compilers, from dedicated control cards and simple real-time executives to process stations equipped with standard operating systems. At present, REX supports Windows NT/2000/XP, Windows CE .NET and a hard real-time operating system Phar Lap ETS (Embedded Tools Suite).

REX has been used for the building of the remote and virtual laboratory in the department of cybernetics (`lab.fav.zcu.cz`). Below is the list of sections following this Introduction:

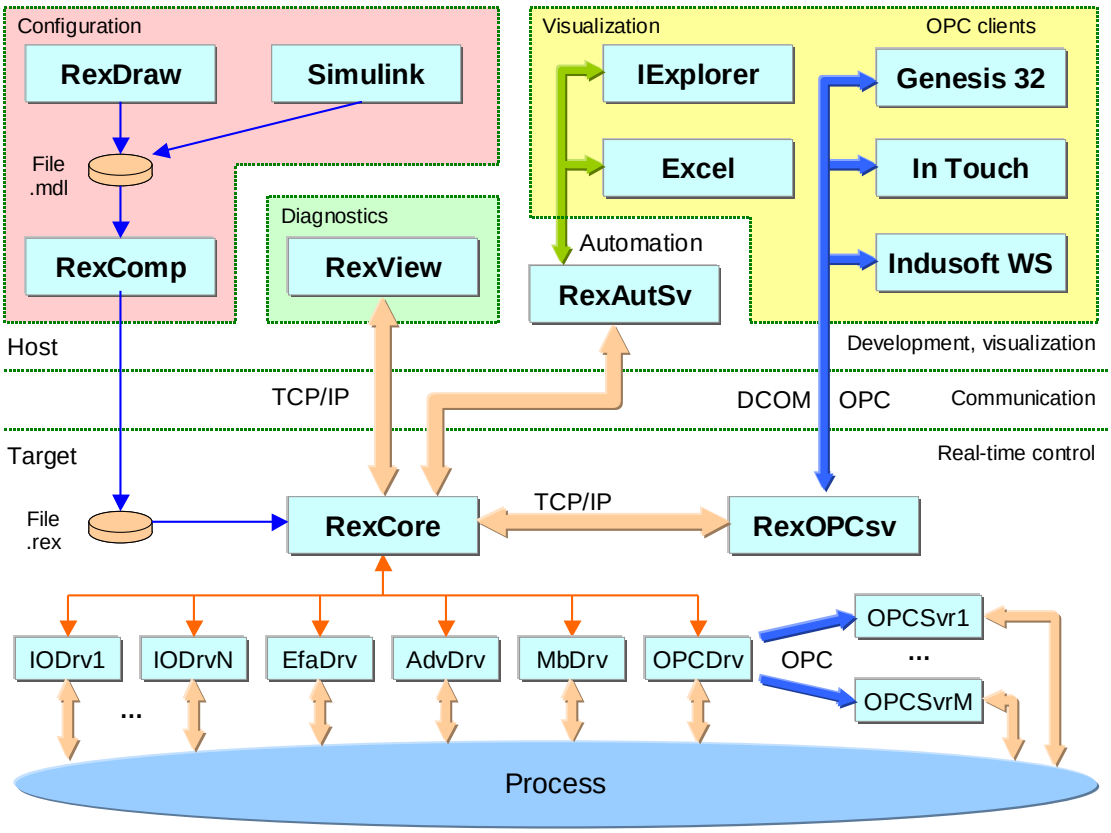

Fig. 1. Structure of REX control system

3 Function block library RexLib
4 REX runtime and diagnostics
5 Configuration of REX
6 OPC (OLE for Process Control) utilization

2. STRUCTURE OF REX

The structure of the REX control system and its relations to an environment are shown in fig. 1.

The top part of the figure contains components of a host development environment which is (simultaneously) the visualization and operator management environment. All current development tools of REX are created for the Windows operating systems (95/98/ME/NT/2000/XP). Standard visualization tools are used (some of them are shown in the figure). Visualization tools are connected to the control system through an OPC interface, which uses COM (Component Object Model) and DCOM (Distributed COM) services, or through Automation (OLE Automation) and various scripts (VBScript, JScript, Visual Basic).

The bottom part of the figure presents the structure of the target enviroment, which implements real time control algorithms. In addition to the Windows operating systems mentioned above, also e.g. Windows CE or Phar Lap ETS can be chosen for the target environment. In the case of the Windows operating systems, the target and host environments can be the same, even on a single computer.

The links between host and target environments are carried out by a communication layer (in the middle of fig. 1). The most frequently used communication protocol is the TCP/IP standard, which is used as a basis of our own diagnostic protocol of REX.

3. FUNCTION BLOCK LIBRARY

The control algorithms performed by REX are developed using only a special function block library (block set) called RexLib, (Schlegel *et al.*, 2001). At present, the library contains more than 120 function blocks, which are compiled for Simulink and each target platform (Windows, Windows CE and Phar Lap ETS). The library contains both simple function blocks (similar to the corresponding Simulink blocks) and advanced industrial control blocks. The most sophisticated function blocks are depicted in fig. 2.

PIDMA is the full ISA PID controller supplemented with the built-in autotuning procedure. After start of the autotuner, the rectangle pulse is applied to the input of the process and the first three process moments are computed from the pulse response. Subsequently, the controller parameters are determined by a model based design procedure (Schlegel *et al.*, 2002).

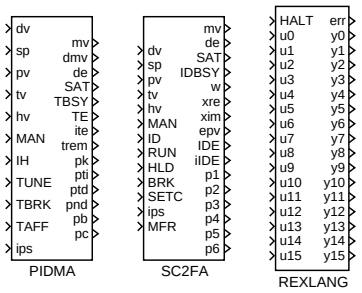

Fig. 2. Examples of advanced function blocks from RexLib

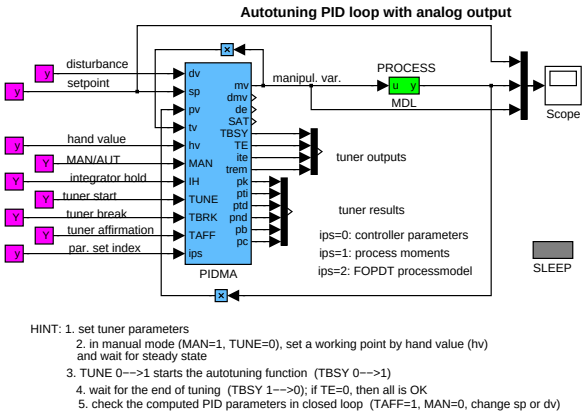

Fig. 3. Simple autotuning control loop in REX

SC2FA is a self-tuning controller for lightly damped processes. A new algorithm of the frequency domain identification and the standard state space approach form the core of the automatic controller design method (**?**).

REXLANG is a special function block which allows the user to write his/her own code in the programming language called RexLang. This language supports all common C language commands (e.g. `if`, `else`, `for`, `while`, `do-while`, `switch`), calling user defined functions, standard math functions, and work with indexed arrays. RexLang is almost a subset of the C language except for commands `input`, `output`, `parameter`, which map block inputs (`u0` to `u15`), outputs (`y0` to `y15`) and parameters (`p0` to `p15`) to user defined C language variables.

Fig. 3 demonstrates the example of a simple control loop with autotuning feature based on the RexLib blocks. Detailed information about these and other blocks can be found in (REX Controls, 2004).

4. REX RUNTIME AND DIAGNOSTICS

The core of REX consists of a program, RexCore, which runs continually on the target platform. Other tools are necessary for watching the control system behavior from the host platform, especially during putting the control system into operation. This aim is partially satisfied by the already mentioned visualization tools, but a detailed view is provided by a program called RexView.

4.1 RexCore – the control system core

RexCore is a rather complicated program performing, simultaneously, different activities which are usual in control systems. Particular activities are executed according to their priorities in the preemptive multitasking mode. The following activities belong to the core subsystems:

Real-time Subsystem takes care of the triggering of tasks and nested fuction blocks, and triggering of I/O drivers. It also collects and renders diagnostic information about the system performance and the timing variables of tasks and drivers.

Input-output Subsystem provides interfaces for the I/O drivers of hardware devices which are used to acquire process inputs and to actuate process outputs.

Algorithmic subsystem includes function block algorithms which are called from real-time subsystem tasks.

Diagnostic subsystem provides runtime diagnostic information and enables application download and debugging.

Archive subsystem serves for the archiving of events, alarms and historical trends of selected variables.

4.2 RexView – the diagnostic tool

The RexView program makes it possible to watch what is going on in the control system core. Therefore, it is a very important tool during the debugging of the control system and also in routine operation when a problem occurs. The program offers detailed, hierarchically ordered information about all core subsystems. The communication protocol based on TCP/IP makes it possible to connect RexView to a running core on the local computer, local area network or remote network (e.g. through Internet).

Fig. 4 demonstrates one type of the RexView screen. A more detailed description of the program is contained in the user manual.

The left part of the figure is filled by the hierarchy of control system subsystems and objects. The selected `PIDMA` block is tenth block of the real-time subsystem task `mtuner` in the tree structure.[1] The `PIDMA` block implements the PID controller equipped with a built-in autotuner (Schlegel *et al.*, 2002).

The `Workspace` property page containing the `PIDMA` block workspace variables is chosen on the right hand part of the screen. The top part of the property page

[1] Blocks of the given task are ordered in the same way as they are executed in the task.

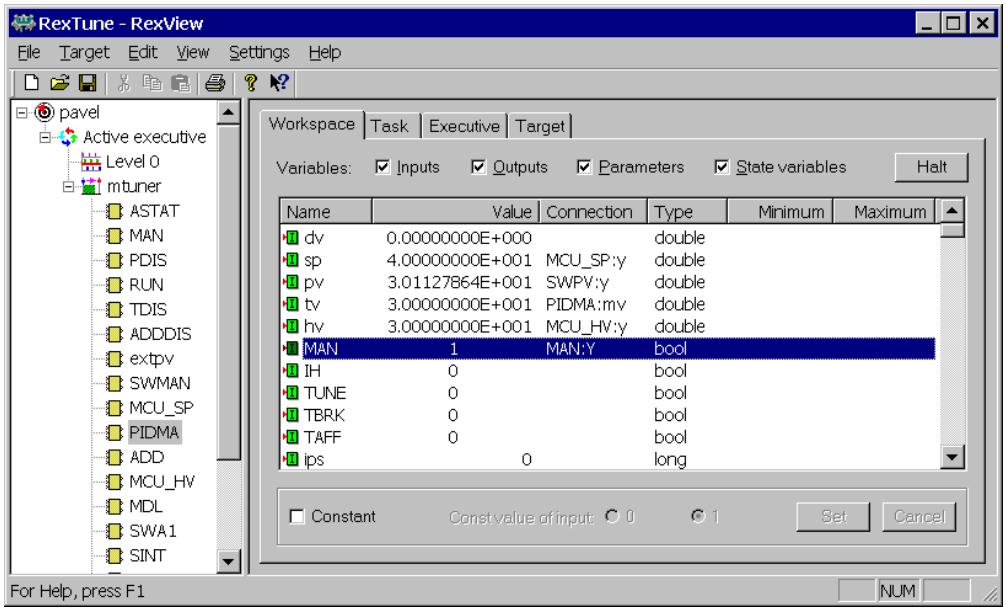

Fig. 4. Sample of a RexView program screen

contains checkboxes for the selection of the kinds of variable (inputs, outputs, parameters and states). The middle part of the page contains the list of selected variables whose values are periodically updated. The connections to adjacent blocks are also visible. The bottom part makes it possible to enter the values of parameters and to simulate the values of selected inputs. In this case the input `MAN` is selected. Its value `1` means that the controller is in the manual mode. If we want to switch the controller to the automatic mode directly (for the debugging purpose), it is sufficient to check the item `Constant`, then to choose the value `0` in the `Const value of input` field and finally to press the button `Set`.

The other property pages in the right hand part of the figure correspond to the properties of the superior objects of the selected block `PIDMA`:

`Task` depicts the diagnostics of the task `mtuner` (e.g. the measurements of current, average, minimum and maximum execution time variables, computing error indications)

`Executive` shows the identification and diagnostic data of the current real-time executive instance `Active executive` (e.g. creation, download and last start dates and times, memory usage)

`Target` identifies the current target device (here `pavel`) which is on-line (e.g. target device type, operating system name and version, REX version and build date)

The program contains also property pages of the other REX control system core objects. The appropriate page is activated when a given type object is selected, e.g. a module, a driver, a trend or an archive.

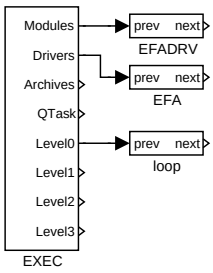

Fig. 5. A main file `exec.mdl` of a simple control loop example

5. CONFIGURATION OF REX

REX is configured by creating function block diagrams consisting of a large function block library (Schlegel *et al.*, 2001). The function block library is available in Simulink and all target platform versions of REX. This fact makes it possible to draw function block diagrams in the Simulink built-in editor. The second possibility is to use REX's own editor, RexDraw.

Both programs, Simulink and RexDraw, store function block diagrams in files with the `.mdl` extenstion (model). These text files are too big to be transferred to some memory limited embedded devices. That is why the `.mdl` files are compiled by the RexComp compiler into the binary format `.rex` (section 5.2).

The configuration of REX is demonstrated on a simple control loop example which is shown in figures 5, 6 and 7.

The rest of this section explains the mutual relations of all the three figures.

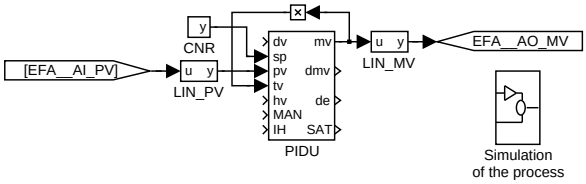

Fig. 6. A particular simple control loop configuration example `loop.mdl`

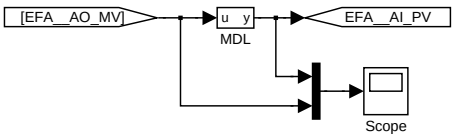

Fig. 7. The simulation subsystem `Simulation of the process` of the configuration `loop.mdl`

5.1 RexDraw – Graphic editor of function block diagrams

The RexDraw program is used for function block diagrams design in a way very similar to the Simulink built-in editor. Both programs generate files with the `.mdl` extension but there are some differences in their options.

First of all, RexDraw allows to compose `.mdl` files using only blocks from the large library of REX blocks (Schlegel *et al.*, 2001). All these blocks work in discrete time although many of them are discretized for a given sampling period. This fact corresponds to the Simulink `Solver options` parameters: `Type: Fixed step` and `discrete (no continuous states)`.

While a whole Simulink configuration consists of the only one file which can contain many subsystems (see the file `loop.mdl` in fig. 6 and the subsystem `Simulation of the process` in fig. 7), any REX configuration consists of at least two files. Only one of them is called *the project main file* (in this case the file `exec.mdl` in fig. 5).

The project main file specifies the configuration of the RexCore particular subsystems, which are described in paragraph 4.1. Our example is very simple; the real-time executive is represented by the block EXEC, the remaining blocks serve the following purposes:

EFADRV (ETS Fast Advantech Driver) is one of the REX control system modules. This particular module [2] implements the fast driver of several PCI boards by Advantech for the hard real-time operating system Phar Lap ETS (Embedded Tools Suite) by VenturCom.

EFA is an I/O driver block. Its first parameter is the name of a module implementing the driver, in this case EFADRV. The next parameter is the name of a configuration file which contains information about the source of control system inputs and destination of outputs in addition to the device parameters. For control system design, the most important parameter is the name of the block (here it is EFA) which determines the prefix of all names of inputs and outputs connected to this driver (see paragraph 5.2 for more details).

`loop` is the file name (the `.mdl` extension is appended automatically) containing the control task configuration. This task is inserted into the control level 0 (see the output `Level0` of the block EXEC). It is presented in fig. 6.

The described blocks possess inputs `prev` (previous) and outputs `next` which allow to chain several blocks representing modules, I/O drivers and control tasks in the configuration. The same rule holds for archives (not used in the example) which are connected to the `Archives` output of the EXEC block and for tasks of ohter control levels (`Level1`, `Level2` and `Level3`). The only exception to the rule is the block `Qtask` (not used here) representing very quick control task; there can be, at most, only one such task.

5.2 RexComp – Compiler of configurations

The RexComp program compiles a project main file in `.mdl` format into a binary configuration file of the REX control system. Note that the main file `exec.mdl` configuration is shown in fig. 5. The compilation process consists of the following steps:

(1) Finding of the real-time executive block EXEC in the project main file, checking of the executive parameters.
(2) Finding and checking of all control system objects which are drawn in the project main file.
(3) Addition of configuration objects: modules, drivers, archives, the quick task and tasks of used computing levels.
(4) Memory allocation and setting of configured blocks parameters.
(5) Checking of control task connections.
(6) Whole configuration validation check.
(7) Storing of the compiled file with the extension `.rex` on the disc.

During its run, the compiler writes out the information about the files being compiled and about possible compilation errors. Each of the described steps can fail fatally, which results in compilation abort and the binary file is not created.

Effective transition from simulation in Simulink to realtime control is supported by the following two rules of RexComp:

(1) All subsystems (in `.mdl` files) the names of which begin with the word `Simulation` are skipped (i.e. they are not compiled)
(2) All blocks `From` and `Goto` whose `Tag` is of the form `<prefix>__<name>` are substituted

[2] The present version of REX supports also other drivers, e.g. Modbus master an slave (also over TCP/IP), standard Advantech driver, OPC driver, etc. (see fig. 1.)

by standard input and output blocks of the REX system. These blocks refer to the I/O driver block with the name <prefix>. Further, the symbolic name <name> of an input or an output is searched in the driver configuration. The two characters _ (underscore) serve as the separator of a driver prefix and an input or an output variable name in the driver.

The following example demonstrates how these rules work. Due to rule 1, the subsystem Simulation of the process is eliminated. Rule 2 determines that the block From with the tag EFA__AI_PV is substituted by the standard REX input block which refers to the signal AI_PV of the driver with the name EFA in the project main file (see fig. 5). Similarly, the block Goto with the tag EFA__AO_MV refers to the signal AO_MV of the same driver.

Keeping the rules above guarantees the possibility of transition from real-time system simulation in Simulink to real-time control in REX without the necessity of any configuration file change.

6. OPC (OLE FOR PROCESS CONTROL) UTILIZATION

OPC is a relatively new but, at present, a very widespread standard for automatic control devices data exchange, especially for communication of process stations and SCADA (Supervisory Control And Data Acquisition) or HMI (Human Machine Interface) levels. The current REX version supports the OPC Data Access Specification (OPC Foundation, 2000) in two ways described below.

6.1 RexOPCsv – OPC server of the REX system

The first way is the OPC Data Access server RexOPCsv, which can be seen in the bottom Target part of fig. 1, was created mainly for the purpose of communication with a supervisory level. The supervisory system can be an arbitrary system which is OPC Data Access version 2.0 client. Today, there are tens of such systems on the market, some of them are presented in fig. 1.

The structure shown in fig. 1 is applicable if the target environment supports the DCOM standard, i.e. mainly in the Windows operating systems. The server can also be launched in the host platform; it is necessary for target platforms supporting TCP/IP only. This structure is suitable for large networks (Internet) where the gateways to local networks are protected against unauthorized access (Firewalls). For this reason, the DCOM packet transfer may be a problem.

6.2 OPCDrv – I/O driver communicating with third party OPC servers

The OPCDrv I/O driver represents the second way of OPC utilization in REX. In contrast to the RexOPCsv server, this driver is an OPC Data Access 2.0 client. Thanks to this fact, REX can read (write) inputs (outputs) of arbitrary devices which are equipped with OPC Data Access servers. There are hundreds of such devices, produced by many companies, on the market.

7. CONCLUSION

The paper introduces a new control system called REX which is compatible with Matlab-Simulink. Under certain circumstances, described in section 5.2, it is even possible to move from Simulink simulation to REX real-time control and vice versa without any configuration modification.

Using REX instead of Real Time Workshop (RTW) or prototyping devices such as dSpace has several advantages:

- Existence of industrial control blockset RexLib.
- No need of software licenses for prototype devices development.
- Low price of prototype and/or final control system application.

The proposed solution is also suitable for the building of real, remote and virtual laboratory control systems.

REFERENCES

OPC Foundation (2000). *OLE for Process Control, Data Access Custom Interface Standard, Version 2.04*. OPC Foundation.

REX Controls (2004). *Function blocks of the system REX (in Czech)*. REX Controls s.r.o. Plzeň.

Schlegel, M., P. Balda and M. Štětina (2001). C MEX blocks industrial control library with application examples. In: *Summaries Volume of the Matlab 2001 Workshop*. Humusoft s.r.o. Praha. pp. 361–369.

Schlegel, M., P. Balda and M. Štětina (2002). PID autotuner for industrial use. In: *Proceedings of the conference Control of power & heating systems*. Zlín, Czech Republic.

The Mathworks (2001). *Using Simulink, Version 4.1*. Using Simulink, Version 4.1. Natick, MA.