IMPLEMENTATION OF A LINUX-BASED CNC OPEN CONTROL SYSTEM

Tomislav Staroveški, Danko Brezak, Toma Udiljak, Dubravko Majetić

T. Staroveski, dip.ing., University of Zagreb, FSB, I. Lucica 5, 10000 Zagreb
Dr.sc. D. Brezak, University of Zagreb, FSB, I. Lucica 5, 10000 Zagreb
Prof.dr.sc. T. Udiljak, University of Zagreb, FSB, I. Lucica 5, 10000 Zagreb
Prof.dr.sc. D. Majetic, University of Zagreb, FSB, I. Lucica 5, 10000 Zagreb

Keywords: Open Control Systems, Enhanced Machine Control, Real-Time Linux

Abstract
This paper presents a utilization of a Linux based open architecture control system (OAC) as a CNC solution for the mini milling testbed platform. The characteristics of the chosen OAC solution, testbed structure, and implementation details are briefly depicted. Finally, main conclusions and future research work are summarized.

1. INTRODUCTION

In the last two decades, a lot of efforts have been made in the development of open control systems for machine tools. They were recognized as a solution to machine tool and control manufacturers endeavors to elaborate common concepts, develop basic technologies and to produce basic components together, in order to fulfill continuous demands for higher machine tool functionality and flexibility, product quality and costs reduction [1]. According to the IEEE, "an open system provides capabilities that enabled properly implemented applications to run on a variety of platforms from multiple vendors, interoperate with other system applications and present a consistent style of interaction with the user " (IEEE 1003.0). This means that, Open Architecture Controller (OAC) has to be flexible in hardware and software, for all control levels, i.e. must be standardized in the sense of integration with other control systems and permit the integration of independent application program modules, control algorithms, sensors and computer hardware developed by different manufacturers [2]. With the ability of implementation and integration of customer-specific controls by means of open interfaces and configuration methods, their implementation is particularly important in the development of reconfigurable manufacturing systems [3].

The first OAC solution was proposed by National Institute of Standards and Technology (NIST). This project has evolved over time, and is currently focused on the development of open architecture control system named as *Enhanced Motion Controller* [4]. After this first initiative, several other projects have started in the Europe, USA and Japan, among which the most important are [5]:

- OSACA (Open System Architecture for Controls within Automation System),
- OMAC (Open Modular Architecture Controllers),
- OSEC (Open System Environment for Controller),
- JOP (Japanese Open Promotion Group).

These projects were initiated and supported by different machine tool makers, control and software vendors, system integrators, end users and academics. Beside aforementioned, other university research activities in hardware and software area of open architecture CNC systems were also conducted and resulted in systems such as Open Real-Time Operating System (ORTS) [6] or Soft-CNC system based on OSACA principle. However, despite all of these efforts, a universal open CNC architecture still remains undefined. Moreover, according to [5], open control systems that are available today mostly offer the possibility for modifications in the non-real-time environment in a fixed software topology. They also lack the necessary flexibility and are not based on vendor-neutral standards.

In this paper, a concise description of activities regarding the implementation of open architecture controller for the 3-axis bench-top mini milling machine is presented. This project was motivated by the necessity for open controlled machine tool testbed platform, intended to be used in the analysis of different control algorithms and process monitoring techniques, as well as educational purposes.

Among several proposed OAC solutions, *Enhanced Machine Controller* (EMC) was implemented. EMC runs on Linux based operating systems with real-time extensions. Characteristics of such software platforms, which primarily include stability and performance, were main reasons for

choosing this type of OAC. Up to this time, EMC has been successfully used in a several CNC retrofitting projects, including applications with complex kinematic chains [7].

2. EMC OVERVIEW

First version of EMC was originally developed by the Intelligent Systems Division at the (NIST), an agency of the Commerce Department of the United States government [8]. Resulting work from efforts to produce motion control package as a test platform for concepts and standards remained in the public domain and EMC quickly gained attention among open source community.

Current EMC version (EMC2) [9] is actively developed and community maintained software package, presenting an effort to simplify, organize and continuously extent the original work.

Most significant advances in the later versions of EMC, discussed in this paper, feature introduction of Hardware Abstraction Layer (HAL) greatly simplifying interface to the control hardware, major code optimizations as well as extended support for implementation on variety of different machines with complex kinematic structures.

HAL provides both means of transferring real-time data from EMC to control hardware or other lower-level software modules, and the framework for development of hardware drivers and software modules for execution in real-time [10].

Code optimizations include replacement of Real-time Control Systems Library (RCSLib) [11] with optimized version, the Neutral Message Language Library (NMLLib) [12]. Both of these libraries provide support for implementation of Neutral Messaging Language (NML) [13], which is communication method implemented within different EMC modules.

Real-time extensions to Linux kernel are required for normal EMC operation, although it is possible to run EMC in a non real-time environment for simulation purposes, without interfacing actual hardware.

Support currently exists for Linux kernel versions 2.4 and 2.6 with real-time extensions applied by either RTAI [14] or RT-Linux [15] patches. Experimental versions of EMC are also built for 64-bit kernels.

2.1 Architecture of EMC

EMC is composed from four components:

- Motion Controller (EMCMOT),
- Discrete I/O Controller (EMCIO),
- Task coordinating module (EMCTASK),
- Text-based and graphical user interfaces (GUI).

In the four modules of EMC, only EMCMOT is a realtime module. The communications between non-realtime modules are implemented by NML channels, and the communications between the realtime module (EMCMOT) and the non-realtime module (EMCTASK) is implemented etither by shared memory or RT-Linux FIFO mechanisms [16]. Figure 1 illustrates architecture of EMC.

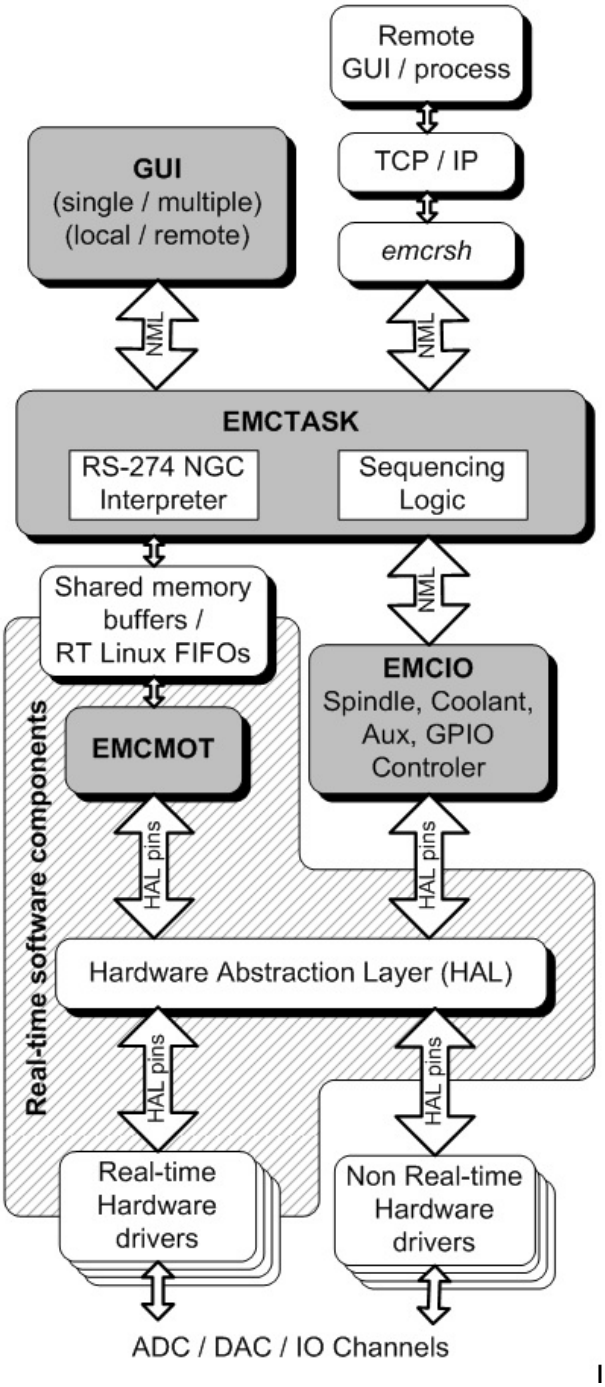

Figure 1. EMC Architecture

2.2 Motion controller (EMCMOT)

EMCMOT is executed cyclically in real-time and performs trajectory planning, direct and inverse kinematic calculations and computation of desired output to motor control subsystems, as shown in figures 1, 2 and 3. This process includes sampling of controlled axis positions, computation of next trajectory point and interpolation between these

trajectory points. Interpolation is done by means of cubic interpolation routines, and a trapezoidal velocity profile generator is used during computation of desired position references [17]. Programmable software limits are also supported, as well as interfaces to hardware limit and home switches. Figure 2 shows EMCMOT structure in greather detail.

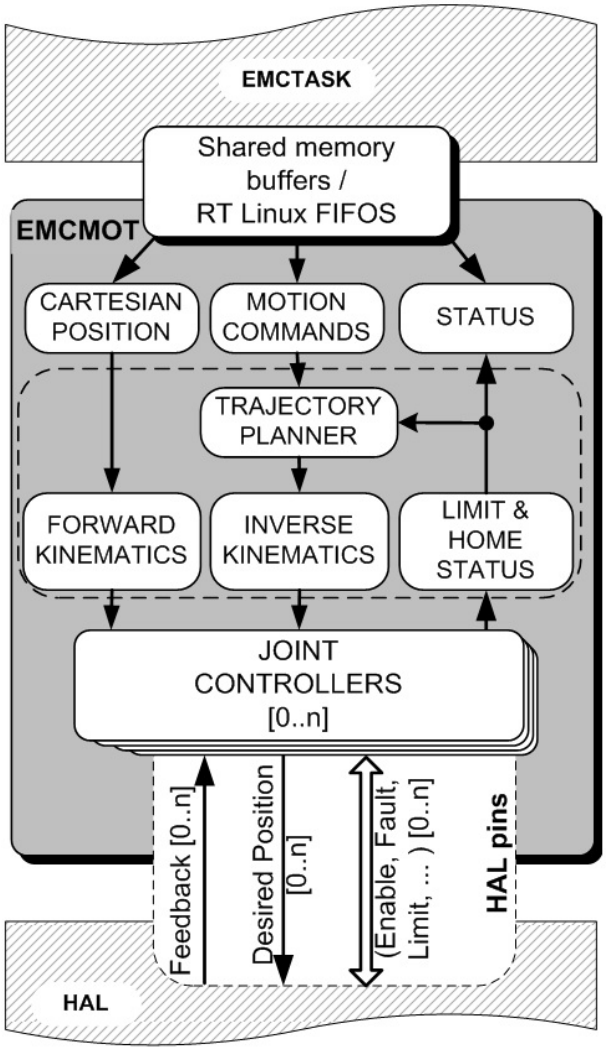

Figure 2. EMCMOT Structure

Supported modes of operation are individual axis jogging (continuous, incremental, absolute), queued blended moves for linear and generalized circular motion, as well as programmable forward and inverse kinematics.

Complex kinematics for robots can be coded in C language, according to a prescribed function interface and linked in to replace the default 3-axis Cartesian machine kinematics routines.

Parameters, such as number and type of axes (e.g., linear or rotary), scale factors between feedback devices (e.g., encoder counts), maximal velocity and acceleration values, axis units (e.g., millimeters) and trajectory planning cycle times are obtained from configuration file during system initialization phase.

Shared memory or FIFO mechanisms are used to receive commands or send status, error, or logging information to user space modules. NML is not used directly by the motion controller since NML requires C++ and the motion controller coding was limited to C in order to provide generic structure and maximize portability to other real-time operating systems.

EMCMOT interacts with the subordinate realtime modules, such as PID compensation algorithms and other hardware drivers using HAL signals. Figures 1, 2 and 3 also show HAL layer, situated between the EMCMOT, EMCIO and the actual machine hardware.

Using provided application programming interface (API) for EMCMOT, specific hardware support can be integrated into the EMC in the form of HAL modules, without modifying any of the core control code.

For servo systems, the output is typically based on a PID compensation algorithm, provided in the form of separate HAL module. PID controller structures include zero, first, and second order feedforward gains as well as maximum following error output. In a typical machine configuration, number of PID controllers usually corresponds to the number of controlled axis. For stepper systems, the calculations typically run open-loop, and step generator, also written as HAL module, is used to sent pulses to the stepper motor drives.

This modular structure allows implementation of different compensation algorithms. HAL concepts are discussed in greather detail in the next section.

2.3 Discrete I/O controller (EMCIO)

All I/O functions, which are not directly related to the actual motion of machine axis, are handled within EMCIO module (shown in the middle right on Figure 1). EMCIO is implemented as a single I/O controller, consisting of hierarchy of subordinate controllers for main spindle, automatic tool change (ATC), coolant, auxiliary functions (e.g., E-STOP chain, lubrication, etc.) and other user-defined subsystems. It is based on a hierarchy controller classes written in C++ using NMLLib.

Since discrete I/O controller design is typically highly machine-specific, customization in general is not intended by single configuration file, used to configure the more generic EMCMOT. Instead, configuration is written in the form of single or multiple HAL files, each describing required application I/O interface, or particular subsystem. These configuration files contain declarations of various HAL modules to be used, as well as HAL signals, by means of which module interconnections are described.

Each instantiated module is visible in HAL as black box, consisting of HAL pins, defined as inputs or outputs. Module interconnections are defined

using signals of same type as corresponding module pins (e.g., binary, float, signed or unsigned integer). References to HAL configuration files are then included in main configuration file to be read during system startup.

Most of previously mentioned subsystems are already pre-configured in EMC, but can easily be adopted or left unutilized, leaving only a subset that fits application needs. In addition, support for writing custom user-defined subsystems is provided either by means of integrated Programmable Logic Controller module (PLC) [18], or by utilizing specialized tool for writing HAL modules called *comp*. PLC is configured during system setup, using ladder logic diagrams within specialized GUI, while *comp* is used to build, compile and install HAL modules from source code.

2.4 Task executor (EMCTASK)

EMCTASK is task level command handler and program interpreter for the RS-274 NGC machine tool programming language [19], commonly referred as *G code*. As coordinating module in the architecture of EMC, EMCTASK is hierarchically placed above EMCMOT and EMCIO, and under GUI, as shown in Figure 1.

EMCTASK monitors the status of subordinate modules (EMCMOT and EMCIO) and coordinates them. It also receives and analyzes the commands, either from the operator through GUI or from another process (locally or remotely in both cases), interprets them into NML messages and dispatches them to EMCMOT, EMCIO or EMCTASK itself at appropriate times. The actual commands, written in the form of *G* and *M code* programs, can be sent to EMCTASK using the Machine Device Interface (MDI) mode or as a file when the machine is in Auto mode.

EMCTASK is coded similarly to the EMCIO using the NMLLib. As interpreter for *G* and *M code* programs, whose coding does not vary significantly between machines, it is less machine specific than the EMCIO.

2.5 User interfaces

Several user interfaces have been developed for EMC: *keystick*, *xemc*, *tkemc*, *mini* and *AXIS*. All of these programs natively run under Linux based operating systems and all run in X11 environment (X Window System), with exception of *keystick*, which is character-based. *AXIS* is the most advanced GUI, featuring interactive *G-code* previewer.

GUI-based programs can be expanded and adopted to match specific application needs by means of virtual control panels (VCP), which is supported with *pyVCP* package. Besides above mentioned user interfaces, telnet based program *emcrsh* is also provided for running remote sessions.

Multiple GUIs can be run simultaneously, either locally or remotely, across multiple networked computers. Several possibilities are presented for remote connection: utilization of X11 protocol, VNC connection, or running GUIs natively on remote computer.

In the first case, GUI runs on the main PC, and the keyboard, mouse and GUI window are forwarded to the remote PC. Installation of X server software on remote computer is required for X11 connection such as Xming for Microsoft Windows™ operating systems [20].

VNC connection is remote desktop system, transfering control of main PC to remote PC. X11 and VNC connections can be tunneled over Secure Sockets Layer (SSL), in order to provide network security during sessions (X11 over SSL, VNC over SSL).

In the third case, GUI programs run nativley on remote PCs and communicate with main PC by exchanging NML messages over the network. This configuration requires modification of configuration files on the main PC (adding appropriate remote IP adresses fields in NML configuration files). All of above mentiond GUIs will work in this configuration under Linux-based operating systems. *Tkemc* and *mini* can be run under Mac OS-X™ or Microsoft Windows™ platforms, if Tcl/Tk programming language, in which these GUIs are coded, has been installed.

Several GUIs are also provided as tools for configuration and tuning during machine setup in real-time. These include *ClassicLadder*, used for graphical editing of ladder logic diagrams; *HALConfig,* which is a tool for testing and parameterization of machine configuration; *HALScope,* which is software oscilloscope for monitoring HAL signals and *HALMeter*, which is a simple tool for monitoring individual HAL signals.

Additional GUI packages are also being developed, that will provide support for graphical building of HAL configurations and machine motion visualization.

3. TESTBED SETUP

At present, 3-axis bench-top mini milling machine is used as testbed. Although this machine is based on very simple Cartesian kinematics structure, it is still nevertheless sufficient for testing most of EMC concepts and overall system stability.

Figure 3 illustrates detailed configuration of HAL for presented testbed, as well as essential wiring schematics for feed drives. Each controlled axis with corresponding drive and HAL signals is denoted by *i*, where *i* = [X,Y,Z]. Allocated hardware I/O ports are denoted by *n*.

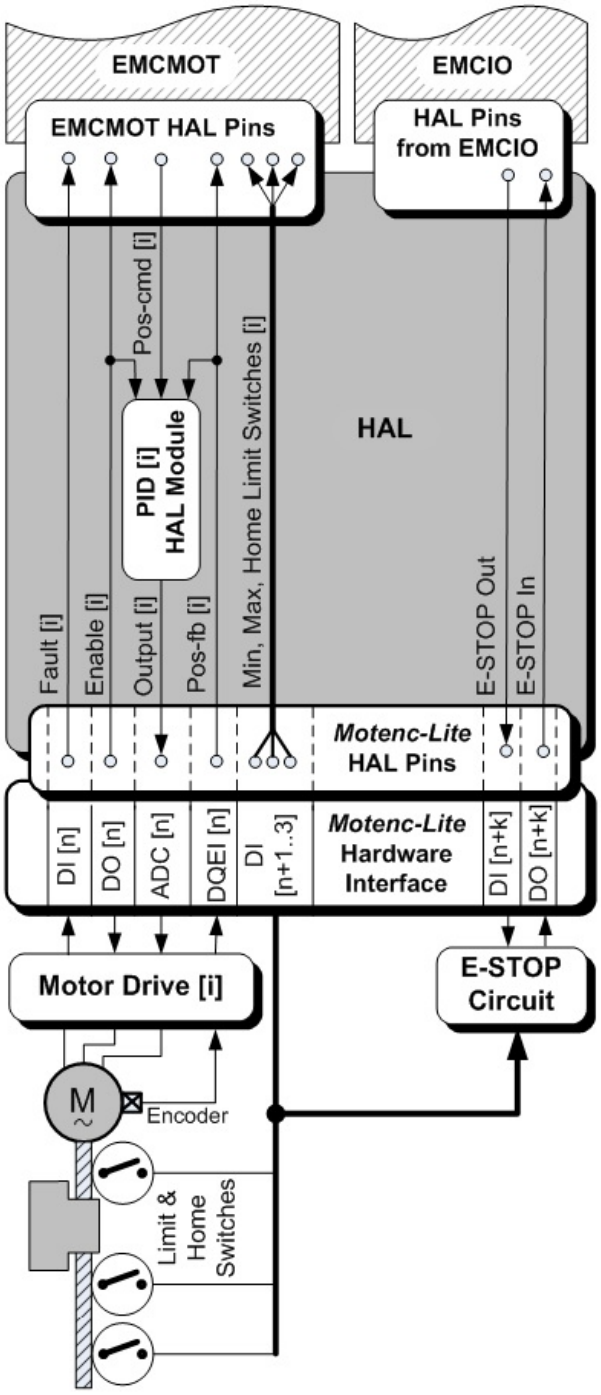

Figure 3. HAL configuration for testbed

Configuration of feed drives, which applies to all axis, is based on closed-loop servo system, by means of Permanent Magnet Synchronous Motors (PMSM) with integrated incremental encoders, corresponding motor controllers and ball screw assemblies. Basic technical information for test bed is presented in Table 1, while detailed technical information for the selected motors, type *SB04A* manufactured by *Mecapion Co*., can be found in [21].

Table 1. Technical characteristics for feed drives

	Rating	Units
Motor rated power	0,4	kW
Motor rated torque	1,27	Nm
Max inst. motor torque	3,822	Nm
Motor rated speed	3000	rpm
Maximum motor speed	5000	rpm
Encoder resolution	3000	p/rev
Ball screw pitch	5	mm/rev
Axis X length	300	mm
Axis Y length	400	mm
Axis Z length	55	mm

Digital servo controllers type *DPCANIE-030A400* are used for driving X and Y axis motors, while *DPCANIE-060A400* is used for driving Z axis motor [22].

Selected drives provide multiple modes of operation (closed loop control of position, velocity or torque, as well as encoder-following for electronic gearing), variety of common industrial interfaces for acquisition of reference signals (±10V, PWM+Direction, STEP+Direction), CAN-bus interface with *CANOpen* protocol for fieldbus connections, as well as multiple general purpose analog and digital I/O ports. Diverse interface possibilities make these drives practical choice for both industrial and research purposes.

Selection of particular types (*DPCANIE-030A400*, *DPCANIE-060A400*) is a temporary solution and only serves for initial testing, as power ratings greatly exceed application needs. Presented drives were originally obtained for other ongoing project and will be shortly replaced with more suitable drives from same manufacturer, regarding adequate power ratings, and with same interface possibilities.

In the current setup, feed drives are configured as closed loop velocity controllers, since position loop for each axis is implemented within EMCMOT module, discussed in previous sections. PID gain factors for both current and velocity loops are obtained manually, using royalty-free *DriveWare* configuration software package [23].

Interface to drives from PC side is done via *Motenc-Lite* PCI card [24]. With all breakout boards installed, single *Motenc-Lite* card features 8 Analog Outputs (Range ±10V, 13-Bit Resolution), 8 Analog Inputs (Range ±5V, 14-bit Resolution), 4 Differential Quadrature Encoder Inputs (DQEI), as well as 16 digital outputs (24VDC) and 32 digital inputs (24VDC). DQEI interface is mapped to internal 32-bit up/down counter with maximum update frequency of 2MHz. Multiple installations (up to four) of this card is possible in order to further expand configuration.

Incremental encoders, which are powered from motor drives and utilize differential quadrature TTL output signals, are used as feedback devices for both drives and EMC. Signals from encoder are first fed to the motor drive as velocity loop

feedback, buffered, and then sent to EMC trough DQEI channels of *Motenc-Lite* interface card. Emulated Hall Effect sensor signals are also provided from encoders, which are used by the drives for commutation.

Signals from DQEI channels provide resolution of less than 0.417 μm in EMC for given ball screw pitch of 5 mm/rev. Velocity reference signal of ±10V is fed to the drives from *Motenc-Lite* DAC channels using a constant of 300 rpm/V.

Minimal hardware IO pin allocation requirement per controlled axis also includes *Fault* input for drive fault detection, *Enable* output for enabling drive power stage, as well as limit & home switch inputs.

Testbed in its current configuration is shown on figure 4. Figures 5 and 6 show more detailed views of servo controllers and mechanical design. *AXIS*, one of EMC GUIs is shown in figure 7. Finally, Figures 8 and 9 show machining results and finished parts of several initial test runs, respectively.

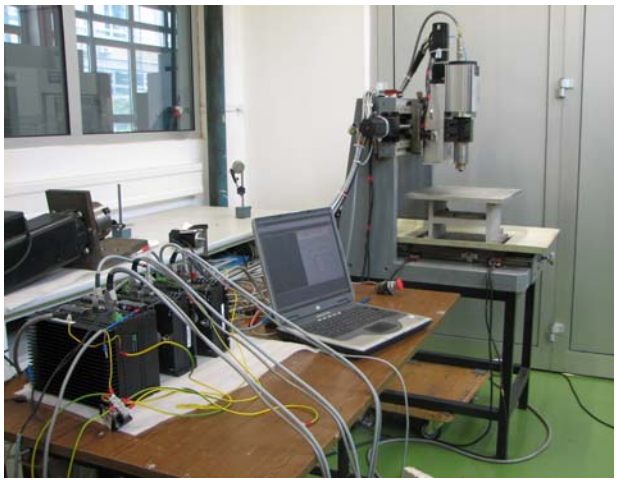

Figure 4. Testbed system

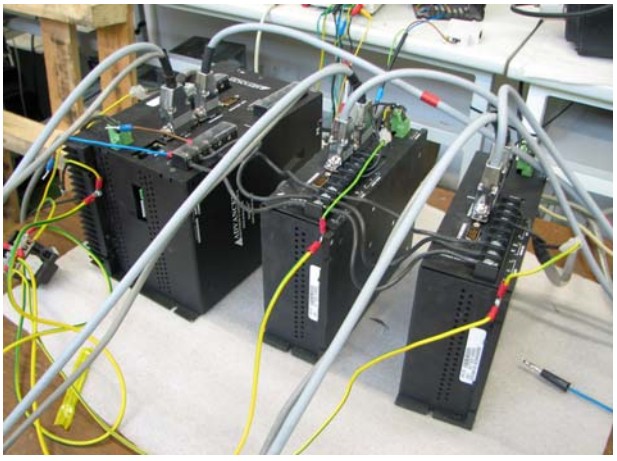

Figure 5. Servo controllers

Figure 6. 3-axis mini milling machine

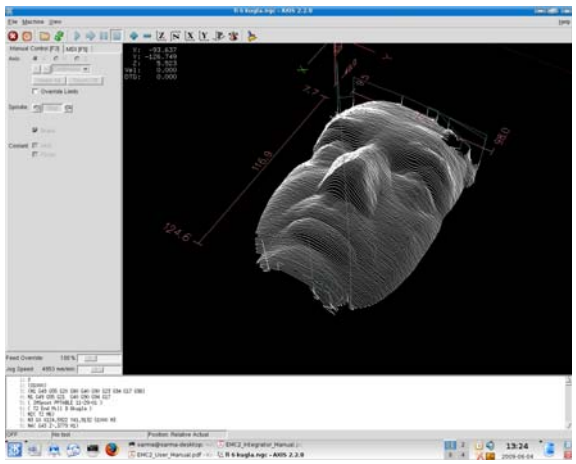

Figure 7. AXIS, one of EMC GUI modules

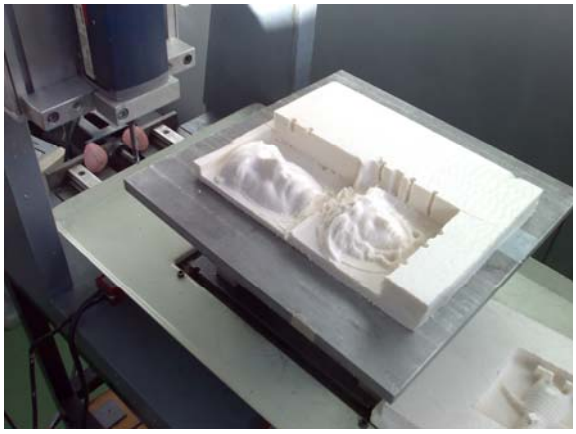

Figure 8. Machining results

Figure 9. Finished parts

4. CONCLUSION

Implementation of EMC, a Linux based OAC system on 3-axis mini milling machine testbed has been presented in this paper. Although current testbed setup in this work cannot be used to present full potential of this approach, some key features can be emphasized.

Due to stability and robustness in general, Linux based operating systems have already proved their potential in many fields, including industrial automation. Combined with real time extensions, they also provide possibilities for mission critical implementations.

EMC can be considered of great academic and educational importance, as it fulfills essential research demands for this field with benefits of robust open source OS, including expansion possibilities with other open-source programs.

Significant potential for commercial applications has allready been proven by several retrofitting projects, including systems with complex kinematics. It is important to note that complete software sollution based on this approach is entirely royalty-free. However, GPL and LGPL licence restrictions must be taken into account prior to any commercial considerations.

Lack of device driver support from most hardware vendors for Linux based operating systems with real time extensions, especially regarding industrial fieldbus communication interfaces, can be considered main disadvantage of this approach. However, compatible hardware interfaces for both servo and stepper systems exist and are actively developed by several manufacturers.

EMC implementation is relatively simple task for common machine configurations, although advanced applications may require extensive multidisciplinary knowledge and approach.

Testbed configuration presented in this paper is the base on which several ongoing projects will continue.

Retrofitting of large planning machine is the first project at FSB in cooperation with *HSTec Co.*, which will result with conversion into 3-axis CNC machining center and will include EMC as open control system. Upgrading current testbed to 4-axis milling machine is also ongoing short-term project.

Future research will be focused on using EMC as platform for integration and analyzes of various process monitoring and control algorithms on both developing testbeds.

5. REFERENCES

[1] Pritschow, G., Daniel, Ch., Junghans, G. Sperling, W., 1993, Open System Controllers – A challenge for the Future of the Machine Tool Industry, CIRP Annals – Manufacturing Technology, 42(1), p. 449-452.

[2] Asato, O.L., Kato, E.R.R., Inamasu, R.Y., Porto, A.J.V., 2002, Analysis of Open CNC Architecuture for Machine Tools, Journal of the Brazilian Society of Mechanical Sciences, p. 208-212.

[3] Koren, Y. et al., 1999, Reconfigurable Manufacturing Systems, CIRP Annals – Manufacturing Technology, 48(2), p. 527-540.

[4] Proctor, F.M., Michaloski J., 1993, Enhanced Machine Controller Architecture Overview, Available from: ftp://ftp.isd.mel.nist.gov/pub/NISTIR_5331.pdf

[5] Pritschow, G. et al., 2001, Open Controller Architecture – Past, Present and Future, CIRP Annals – Manufacturing Technology, 50(2), p. 463-470.

[6] Erol, N.A., Altintas, Y.. Ito, M.R., 2000, Open system architecture modular tool kit for motion and machining process control, IEEE/ASME Transactions on Mechatronics, 5, p. 281 - 291.

[7] EMC applications, Available from: wiki.linuxcnc.org/cgi-bin/emcinfo.pl?Videos; www.linuxcnc.org/component/option,com_weblinks

[8] Proctor, F.M., Shackleford, W.P., 2001, Use of open source distribution for a machine tool controller, Sensors and controls for intelligent manufacturing, Boston MA, pp. 19-30

[9] EMC User Manual, Available from: http://www.linuxcnc.org/docs/EMC2_User Manual.pdf

[10] HAL User Manual, Available from: http://www.linuxcnc.org/docs/HAL_User_Manual.pdf

[11] Real-Time Control Systems Library – Software and Documentation, Available from: http://www.isd.mel.nist.gov/projects/rcslib/

[12] The NML Programmer´s Guide, Available from:http://www.isd.mel.nist.gov/projects/rcslib/NMLcpp.html

[13] Proctor, F.M., Shackleford, W.P., Michaloski J.L., 2000, The Neutral Message Language: A Model and Method for Message Passing in Heterogeneous Environments, Available from: www.isd.mel.nist.gov/documents/shackleford/Neutral_Message_Language.pdf

[14] RTAI - the RealTime Application Interface for Linux, Available from: http://www.rtai.org

[15] RT Linux, Available from: www.rtlinuxfree.com
[16] EMC components, Available from: http://wiki.linuxcnc.org/cgi-bin/emcinfo.pl?EMC_Components
[17] Simple Tp Notes, Available from: http://wiki.linuxcnc.org/cgi-bin/emcinfo.pl?Simple_Tp_Notes
[18] EMC Integrator Manual, Available from: http://www.linuxcnc.org/docs/EMC2_Integrator_Manual.pdf
[19] Marietta, M., 1994, Next generation controller (NGC) specifications for an open system architecture (SOSAS) revision 2.0. Technical report, National Center for Manufacturing Sciences, Available from: ftp://ftp.isd.mel.nist.gov/pub/NGC_document.pdf
[20] Xming X Server, Available from: http://www.straightrunning.com/XmingNotes/
[21] Technical characteristics for servomotors type *SB04A*, Available from: http://mecapion.com
[22] Technical characteristics for digital servo controllers types *DPCANIE-030A400 and DPCANIE-060A400*, Available from: http://www.a-m-c.com
[23] DriveWare software package, Available from: http://www.a-m-c.com
[24] Technical characteristics for *Motenc-Lite* interface card, Available from: http://www.vitalsystem.com

ACKNOWLEDGEMENTS
The authors wish to thank the Ministry of Science, Education and Sport of the Republic of Croatia for funding this project, and also to Mr. Gyorgy David from HUNOR Ltd. Company for providing assistance during testbed design.