ANNALS OF FACULTY ENGINEERING HUNEDOARA
– International Journal of Engineering
Tome XI (Year 2013) – FASCICULE 2 (ISSN 1584 – 2665)

[1.] Goran FERENC, [2.] Maja LUTOVAC, [3.] Zoran DIMIĆ,
[4.] Jelena VIDAKOVIĆ, [5.] Vladimir KVRGIĆ

DEVELOPMENT OF A REAL-TIME SYSTEM BASED ON THE MODULAR FSM IN DISTRIBUTED SYSTEM FOR ROBOT CONTROL

[1-5.] LOLA INSTITUTE, KNEZA VIŠESLAVA 70A, BELGRADE, SERBIA

ABSTRACT: *In this paper, a real-time robot control system developed at Lola Institute has been presented. The system is a part of a distributed one, in which control logic is implemented using a finite state machine (FSM). The real-time system is based on the Linux platform, where the Linux kernel is patched with the real-time framework Xenomai. OROCOS (Open RObot COntrol Software) is used to provide the real-time environment to implement desired robot control software in C++. The system consists of various structured components with different roles. It is designed to be modular and flexible, which provides easy connection and disconnection of components depending on the requirements. The usage of the real-time FSM enables supervisory control and monitoring. A system modeling has been also presented, using UML graphics tool.*
KEYWORDS: *FSM, real-time, robot control, distributed system*

INTRODUCTION

In the field of development of robot control systems, it is of great importance that the system is modular as much as possible, easy to use, flexible and not expensive. Unlike dedicating machining systems, modular and flexible systems offer these requirements [4, 12]. Engineers from Lola Institute aim to follow that approach. The goal of this research is to find an easy way to supervise and monitor complex robot control systems. The main aim is to create a solution for easy interaction between one control module and other components, as well as to find a way to include required configuration for desired application. The solution is found in a real-time FSM (finite state machine), which provides high level overview in controlling of software components [1, 2]. The developed system is based on the real-time Linux platform. The robot control software consists of appropriate real-time tasks, which are implemented through the OROCOS components [10]. For system modeling UML tool is used. A sequence diagram for one of its states is presented in this paper.

Proposed distributed system for robot motion control consists of two parts: the offline system and the real-time system. A program written in L-IRL programming language [11] is the input of the offline system. The offline system compiles the program and generates appropriate output in a form of an object code. The offline system [9] sends generated object code to the real-time system, where it is interpreted in real-time. Interpretation is performed under the control of the FSM. Communication between the offline and real-time parts of the system is implemented using CORBA protocol [5, 8].

MODULAR REAL-TIME FSM

The FSM for robot control logic is the real-time state machine used in the OROCOS software [10]. The FSM contains a collection of states and each state defines a program on entry of the state, when it is run and on exit. It also defines all transitions to a next state. There are two modes that FSM can run. These are automatic and reactive mode which can be swapped during the run-time. The automatic mode is used in this development.

In automatic mode, after the run program of the current state finishes, the transition conditions to other states are evaluated. If the transition condition is met, the transition program is executed and then the exit program of the current state is called. After that the entry program of the next state is executed. If no transition evaluated to true, the handle program of the current state is called.

A running FSM is always in exactly one of its states. One time per period, it checks whether it can transition from that state to another state, and if so makes that transition. By default, only one transition can be made in one step.

A FSM can have any number of states. It needs to have exactly one initial state, which is the state that will be entered when the FSM is activated for the first time. There is also exactly one final state, which is automatically entered when the FSM is stopped. It means that the transition from any state to the final state must always be meaningful.

The FSM used for the developed system control is shown in Figure 1. It has five different states: Start Robot, Calibrate Offsets, Initialization Sequence, Move Robot and Stop Robot.

Start Robot is initial state and Stop Robot is finite state. Transition from the Start Robot state to some of the next states depends on whether calibration and initialization sequence are done or not. The FSM can make transition from Calibrate Offsets to the Initialization Sequence state or to Move Robot, while Initialization Sequence can make transition only to Move Robot. Stop Robot is selected as the next state when the FSM is in the Move Robot state.

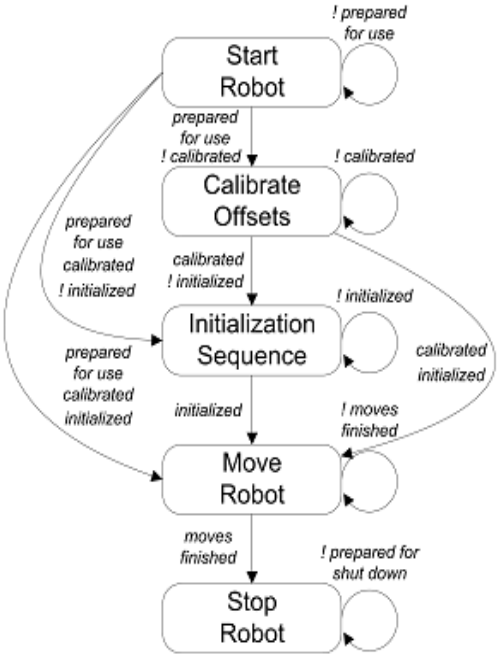

Figure 1. The real-time FSM used in the development of new control system

CONTROL SYSTEM ARCHITECTURE

The system architecture is based on the network of components designed by the developer. Each component presents independent functional block.

Implemented components can be configured using XML files, accessible over a network and listening to a scripting interface, which allows components to be controlled using text commands. OROCOS Device Interface (DI) defines how to interact with analog and digital I/O and encoders. A component which accesses I/O devices can use the OROCOS DI. Components can make use of external libraries as well.

The OROCOS Real-Time Toolkit (RTT) is one of the OROCOS libraries used to provide the real-time environment to build desired robotics applications in C++. OROCOS components which only use the Real-Time Toolkit are portable over different processor architectures and Operating Systems (OS). OROCOS has an internal OS abstraction which allows the components to run on any supported architecture. When some component uses an external library, portability depends on these libraries.

There are four policies for the control of component activities [10]:

- *NonPeriodicActivity;*
- *PeriodicActivity;*
- *SequentialActivity;*
- *SlaveActivity.*

Each component inherits a public interface from its base class (TaskContext class), which defines the following primitives for component interactions (as shown in Figure 2):

- *Events;*
- *Methods;*
- *Commands;*
- *Properties;*
- *Data Port.*

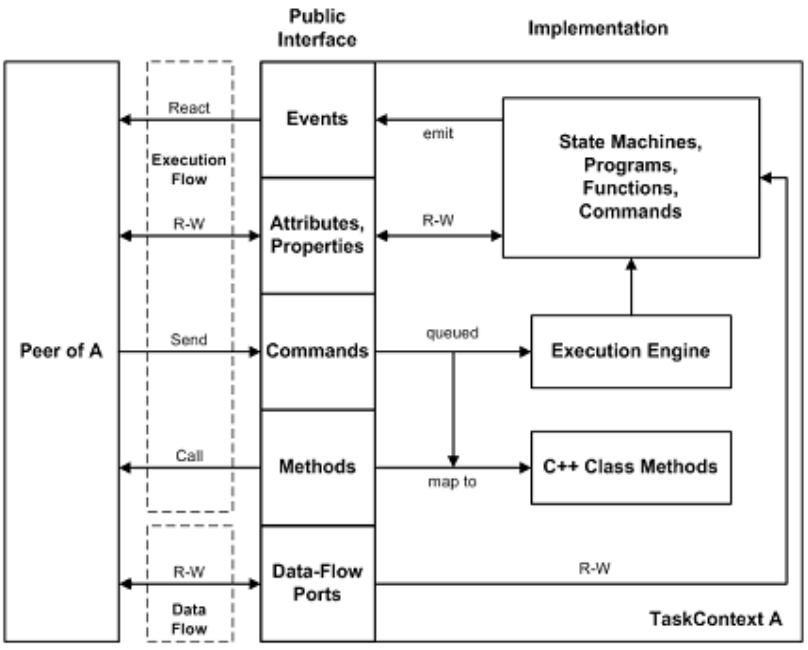

Figure 2. The OROCOS component interface

The basic block model of the control system is shown on Figure 3 [6].

The Viewer component is used to load the FSM and set period of checking whether the FSM can transition from one state to another. The other components are peers of this component. In that way, the FSM has ability to access interfaces of other components. The Viewer component is executed periodically with a 1 ms period.

The Trajectory Generator component is the most important component in the system and presents a generator of trajectory positions. Trajectory Generator can be decomposed into four

submodules: interpreter, path planner, interpolator and kinematics module. It calculates the reference position values that have to be sent to the Servo Controller component. Every 5 ms new values are written in desi_pos (Data Port). Trajectory Generator implements algorithms developed at Lola Institute [7].

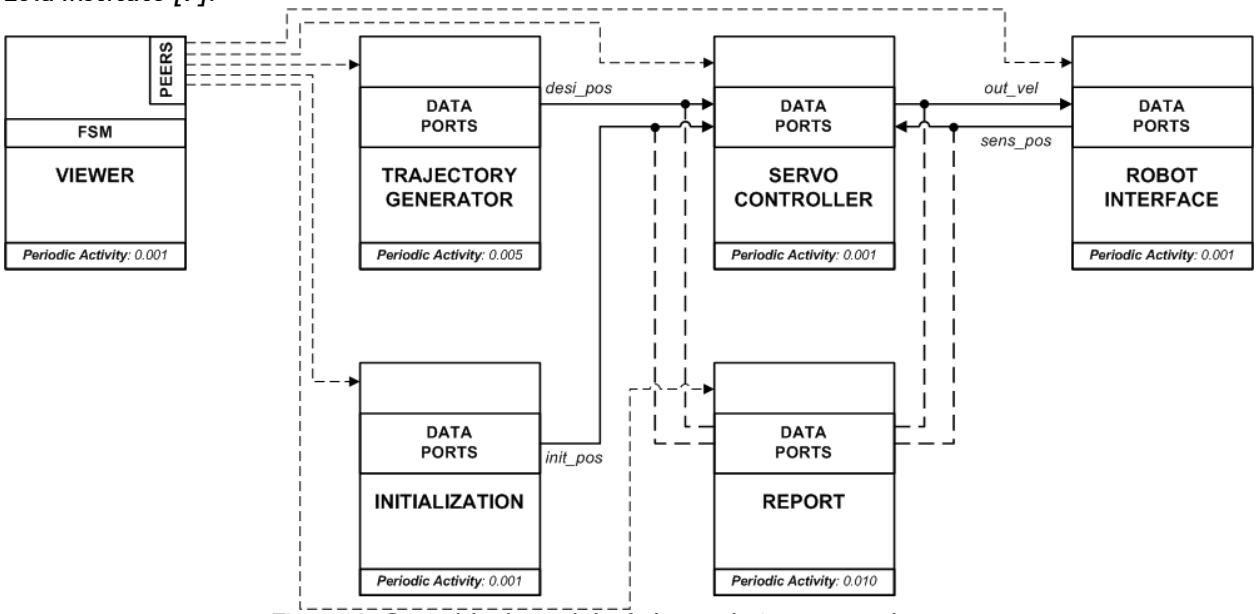

Figure 3. Basic block model of the real-time control system

The Initialization component generates values of positions during initialization sequence. Similar to Trajectory Generator, it writes desired values to init_pos (every 5 ms).

The Servo Controller component implements servo-loop mechanism. Every 1 ms, this component sends reference velocity values that have to be reached. It writes velocities into its Data Port named out_vel. Since the Trajectory Generator and the Initialization components write desired values to desi_pos and init_pos on every 5 ms, Servo Controller has five chances to drive the Robot Interface component to the position sent by Trajectory Generator.

The Robot Interface component abstracts sensors and actuators of the system. This component communicates with the hardware of robot joints and encoders. It has Data Port sens_pos where it writes the values which are read from the incremental encoders. This port is represented by a vector which size depends on the number of robot joints. Also it has a read-only Data Port out_vel where the component reads values (every 1 ms) sent by Servo Controller. Those values should be applied to the robot joints.

The Report component is set to monitor and capture data exchanged by other components. Its activity is periodical as well as the activities of other components. The Report component is executed with a 10 ms period.

FSM AND SYSTEM COMPONENTS RELATION

The real-time FSM allows easy control of OROCOS components. It is possible to configure, start and stop execution of components, call their methods and commands, etc.

Further, these features enable both static and dynamic control system reconfiguration depending on the requirements. The system integrator can modify and configure each component during the static configuration state and define more than one configuration. If more than one configuration is defined, the system integrator has to add new state (Robot Reconfiguration for instance) in the existing FSM in order to provide dynamic reconfiguration. The application user then can change configuration entering the Robot Reconfiguration state and choosing desired configuration at run time (dynamic reconfiguration state).

The real-time FSM and control system components communicate bidirectional through their interfaces. That relation is presented in this section. Initial conditions, before starting the FSM, are: the Trajectory Generator, the Servo Controller, the Robot Interface components are configured, the Viewer, the Robot Interface components are started and FSM is activated and started in automatic mode.

Start Robot is initial state where the Robot Interface component plays the main role. Robot is prepared for use and the breaks of the axes are released. As long as all these actions are in progress, the FSM stays in the Start Robot state. When the robot is prepared for use, the next state is determined depending on the previous use of the robot. If calibration was not done Calibrate Offsets is selected as a next state, otherwise Initialization Sequence or the Move Robot is selected depending on whether the initialization was done or not.

Calibrate Offsets is a state where the calibration is executed. In this state the Servo Controller component is started as well as the Report component that monitors and captures data exchanged by other components. As long as the calibration lasts, the FSM stays in the Calibrate Offsets state. After calibration is done, transition to the next state depends on whether the initialization sequence was previously done or not. In the first case, Move Robot is the next state and in the latter case, the next state is Initialization Sequence.

Initialization Sequence is a state where robot is driven to the starting position and reset of the encoders is done. At the beginning, the Initialization component is configured and started and it is stopped at the end, in the Exit Program of this state. First, it aims to find inductive sensors and after that Z points of the incremental encoders for each joint. Considering known distances from Z points to the initial positions, robot goes to the starting position after Z points are found. The FSM is in this state, while the initialization is in progress. It makes transition to the Move Robot state at the end of initialization.

Move Robot is a state where robot motions are executed. When entering the state, the Trajectory Generator component is starting. At this stage, all components are started and executed except the Initialization component. When all motions are finished, the FSM makes transition from this state to the finite Stop Robot state. In the Exit Program of the Move Robot state, the Trajectory Generator component is stopped as well as the Report component.

Stop Robot is finite state where robot is prepared for shut down and the breaks of the axes are pressed. When all is done, the Servo Controller, the Sensor, the Robot and the Viewer components are stopped.

USING UML IN REAL-TIME SYSTEM DEVELOPMENT

The UML is a graphical language that is based on the assumption that each system can be composed of a group of interactive entities and different aspects of these entities. Their communication can be described using five diagrams: use-case, sequence, collaboration, state chart and activity diagram, while the rest is considered architectural or static aspects.

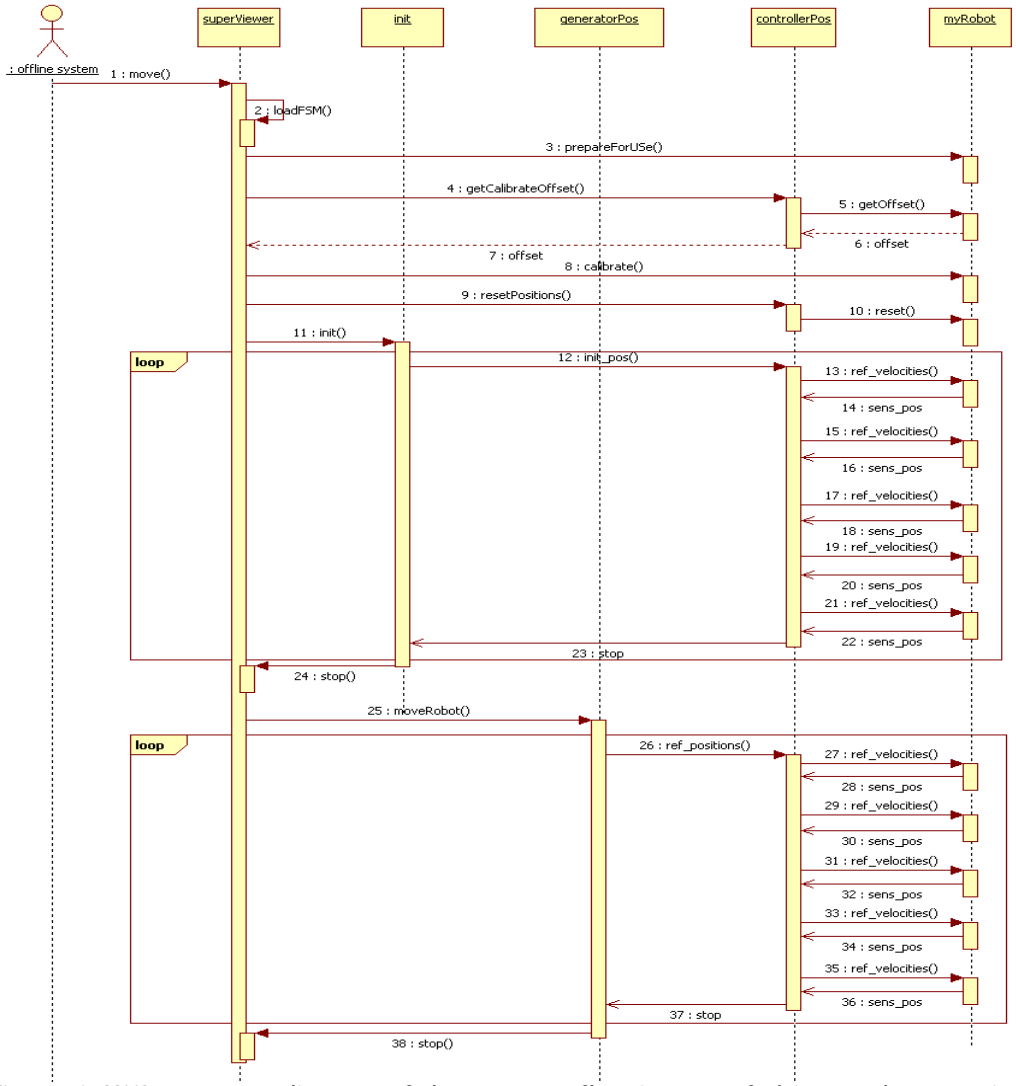

Figure 4. UML sequence diagram of the program flow in case of object code execution

For real-time systems these types of behavioral patterns are of interest. The primary goal when using UML is structural system specification, the organization of the elements that exist at the time of execution, the internal organization and the organization of elements in interaction [3].

Sequence diagram of the program flow is shown in Figure 4. Diagram describes the behavior of the main components of the system in case of execution of an object code.

TESTING OF THE SYSTEM

Real-time system operations are tested on an experimental test station. The parts of the stations are PC, three DC servomotors as well as three inductive sensors and three incremental encoders, required cards and appropriate hardware to connect the PC with servomotors. Servomotors simulate robot joint actuators. The configuration of 6-axis Lola 15 robot manipulator [10] is used to test the system. L-IRL (Lola Industrial Robot Language) is used for programming the tool path. The test code is shown on Figure 5. It is executed in the Move Robot state. Considering the test station has three axes, the other three were set to zero degrees.

```
program test;
system_specification "lola15.spc";
seq
    move ptp joint(mainjoint( 45,   0,-45, 0, 0, 0)) speed_ptp:=0.1 acc_ptp:= 0.1 act_rob:="lola_15";
    move ptp joint(mainjoint(-45,  45, 45, 0, 0, 0)) speed_ptp:=0.1 acc_ptp:= 0.1 act_rob:="lola_15";
    move ptp joint(mainjoint( 45, -45, 45, 0, 0, 0)) speed_ptp:=0.1 acc_ptp:= 0.1 act_rob:="lola_15";
endseq
endprogram;
```

Figure 5. -Test code written in L-IRL

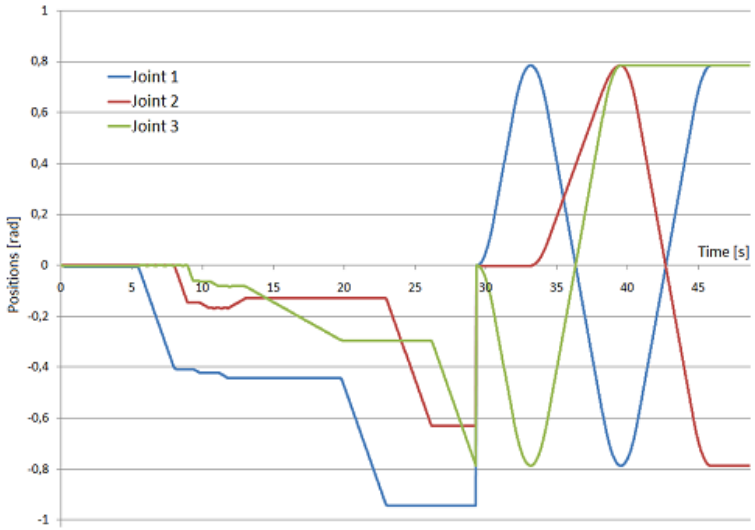

Figure 6. - Positions of servomotors read from encoders

During testing, the positions of servomotors are recorded from the incremental encoders. Figure 6 shows all robot phases (FSM states). In each state the components are controlled by configuring, starting or stopping. They are used by calling appropriate methods and commands. In the Start Robot state robot is prepared for use. After preparation and calibration (the Calibrate Offsets state), the Initialization Sequence state is executed. Joints and their reference points, one by one, are moving in order to find inductive sensors. When all sensors are touched, joint reference points are removed from sensors and Z points of the incremental encoders are being searched. When Z points are found, considering known distances from Z points to the initial positions, robot joints goes to the starting positions. The incremental encoders are reset and then the test code from Figure 6 is executed (the Move Robot state). When all motions are finished, the FSM makes transition to the final Stop Robot state.

CONCLUSIONS

The paper describes the development of the system which provides all necessary functionalities for robots, machine tools and similar devices in a more efficient way and makes development easier and less expensive. This is achieved using the real-time FSM and the OROCOS components. The usage of OROCOS provides wide range of benefits in development of multi-axis machining systems. Modularity and flexibility of OROCOS libraries enable implementation of complex robotic systems using user defined components as well as some ready to use components from OROCOS Component Library (OCL). Each of these components presents an object that can be added or removed, and offers its services through neutral programming interface CORBA. This brings interoperability between components, even if the components are written in various programming languages or running on separate machines with different operating systems.

The testing of the system was performed with an experimental servo station. The obtained results were presented as the positions of robot joints. The results also showed the FSM ability to make real-time transition from one state to another.

ACKNOWLEDGEMENT

This work was created within the research project that is supported by the Ministry of Science and Technological Development, Republic of Serbia: "Development of the devices for pilots training and dynamic flight simulation of modern combat aircraft: 3 DoF centrifuge and 4 DoF spatial disorientation trainer".

REFERENCES

[1] Brecher, C., Verl, A., Lechler, A. and Servos, M., Open control systems: state of the art, Production Engineering, ISSN 0944-6524, Vol. 4, No. 2-3, pp. 247-254, 2010.

[2] Brogardh, T., Present and future robot control development-An industrial perspective, Annual Reviews in Control, ISSN 1367-5788, Vol. 31, No. 1, pp. 69-79, 2007.

[3] Douglass B. P., Real Time UML: Advances in The UML for Real-Time Systems, Third Edition, 2004.

[4] Endsley E. W., Lucas M. R. and Tilbury D. M., Software tools for verification of modular FSM based logic control for use in reconfigurable machining systems, Proceedings of the 2000 Japan-USA Symposium on Flexible Automation, ISBN 0-7918-0765-8, July 2000.

[5] Ferenc G., Lutovac M., Vidaković J., Dimić Z. and Kvrgić V., "Real-time robot control logic using modular FSM", International Conference Management of Technology - Step to Sustainable Production MOTSP 2012, Bar, Montenegro, pp. 259-265, June 19-21, 2012.

[6] Katholieke Universiteit Leuven, Department of Mechanical Engineering, http://people.mech.kuleuven.be/~orocos/, 2012-12-19.

[7] Kvrgić, V., Development of intelligent systems for industrial robots control and programming, PhD thesis, Faculty of Mechanical Engineering University of Belgrade, 1998.

[8] Lutovac M., Dimić Z., Ferenc G., Vidaković J. and Kvrgić V., "Distributed system for robot control based on the CORBA protocol", (in Serbian), 56th Conference for electronics, telecommunications, computers, automation, and nuclear engineering - ETRAN, , Zlatibor, Serbia, pp. RO1.3-1-4, June 11-14, 2012

[9] Lutovac M., Ferenc G., Kvrgić V., Vidaković J., Dimić Z., "Robot programming sistem based on L-IRL programming language", Acta Technica Corviniensis – Bulletin of Engineering, Fascicule 2. April-June, pp. 27-30, 2012.

[10] OROCOS – Open Robot Control Software, http://www.orocos.org.

[11] Pavlovic M., "High level programming language for multi-robotic operations", (in Serbian), M.S. thesis, University of Belgrade, Scholl of Electrical Engineering, Belgrade, Serbia, 1994.

[12] Pritschow, G., Altintas, Y., Jovane, F., Koren, Y., Mitsuishi, M., Takata, S., et al., Open Controller Architecture – Past, Present and Future, Annals of the CIRP, ISSN 0007-8509, Vol. 50, No. 2, 463-470, 2001.

ANNALS of Faculty Engineering Hunedoara

- International Journal of Engineering