Advanced Materials Research *Online: 2011-11-29*
ISSN: 1662-8985, Vols. 403-408, pp 4689-4695
doi:10.4028/www.scientific.net/AMR.403-408.4689

Developing Applications On-Board of Robots with Becerik

Bora İ. Kumova and Savaş Takan

İzmir Institute of Technology Department of Computer Engineering

borakumova@iyte.edu.tr,savastakan@iyte.edu.tr

Keywords: On-board application development; robot operating systems; embedded applications

Abstract. Robot applications are mostly first developed on a computer and thereafter loaded onto the robot. However, in many situations, developing applications directly on the robot may be more effective. For instance, children who have not learned using a computer yet and who develop their robot applications while playing. Or for instance in the robots' operating environment, where there is no computer available. In this contribution we present the properties of the software tool becerik, for developing applications on-board a robot and for running them in multi-tasking mode concurrently. Furthermore, we introduce the programming language of the applications that has the same name becerik, which consists of only 6 commands.

Introduction

State-of-the-art in robot programming is that robot applications are first developed on a computer, thereafter loaded to the robot, where they cannot be changed any more. If an application needs to be changed, then it must be developed again on a computer. However, as robot utilisation areas continuously expand and their usage as consumer products become widespread, one of the most important properties of consumer products, which is the requirement of consumers to adapt a product to own needs, directly on the product, will become inevitable for robots as well. In other words, consumers will be able to adapt the hardware, as well as the software of their robots according their needs. We can observe this trend already today with the robotics kits that become increasingly popular. Since most middle-sized and large-sized robots have a built-in computer, all application development can be performed there. Consumers however, would require simpler interaction interfaces.

The requirement for developing applications directly on the consumer product, the robot, will increase in future, as consumers will continue asking for simpler interaction languages. Future robots will behave more autonomous, with smarter applications and for robots to be able to adapt to their environment, their applications will modify themselves, in order to adapt to the environment. In that stage, it will be further necessary to manipulate the applications via natural languages. In fact, even development-oriented interactions with robots will be carried out in natural languages. With these expectations in mind, we can see that on-board application development environments will be needed in the near future.

Robotics kits fascinate people from pre-school to late adult ages. It has been shown that robotics kits were successful in developing cognitive skills and manual abilities in active learning [4]. In the process of developing the hardware of a robotics kit, manual abilities are improved, while cognitive skills are improved, when developing the software. The kit approach can push the child to develop its robot even further, by inspiring the child further with every modification on the robot. Developing applications on-board a robot, without separately using a computer, including modifying, saving versions and saving different applications are all required for facilitating an interactive robot development.

A further advanced requirement for the robot is to autonomously develop its behaviour. That can be achieved by the robot, by inductively learning from its environment. For instance, collecting statistical data, as it perceives from and actuates in the environment. Hence, the robot can increase its autonomy. As soon as a child starts discovering that the robot can learn, it will wonder about what and

how the robot can learn and then will try establishing stronger cognitive relationships with the robot. As a result of this process, while a child tries developing even more complex robots, it will develop own cognitive skills as well [17].

Most toy robots belong to the class of small-sized robots, because of their hardware and software capabilities. Small- sized robots have relative restricted processors and memories, since their design objective is to use the sensors and actuators in real-time, but only reactively, without any cognitive processes. Mostly controllers that can also run like processors are used, instead of general purpose processors. Their operating systems (OS) are relatively shrank-down versions of common computer OSs [1]. Therefore, processing high-level cognitive processes that require more memory and processing power and multi-tasking, is restricted. On-board application development environments are rare and existing ones have many restrictions, such as no support for all sensors or actuators, restricted number of application steps, loading only a single application, no multi- tasking, no learning capabilities. In order to remedy these drawbacks, we have developed an on-board application development environment that can operate even on such restricted small-sized robots in real-time [6], [11].

Since most middle-sized and large-sized robots have either a computer on-board or comparable computing capabilities, common computer OSs run on them. becerik may run on them as well, as a user interface for the consumer or end-user.

In this contribution, we will present the latest version of becerik and the command language becerik, in which user- developed applications are coded.

Small-Sized Robot Operating Systems and Their Application Development Environments

In an earlier study on small-sized robots we have found out that on-board application development environments are rare and those few existing ones are very restricted [11]. In this section, we briefly discuss OSs and on-board application development environments for small-sized robots.

Common Operating Systems Properties. Since the hardware capabilities of this class of robots are restricted, traditional computer OSs cannot be used. Accordingly, small-sized OSs have been developed that can operate with small memories and slow processors in real- time.

Following common properties can be observed for small- sized OSs:

- very small-scaled: at most a few 100 kb in size
- embedded: covers the whole memory
- real-time: sensors and actuators operated in speeds that are comparable with human motions
- single solution: when loaded into memory, erases any other OS
- processor support: widely spread processors are supported
- device support: plugging-in drivers is supported for any sensor or actuator

Common capabilities are relative large, as most of the robots are supported by all OSs. In other words, since the OSs support most robots, the common properties they share is large.

Small-sized robot OSs may have some similarities with micro kernels, but they are not micro kernels, as their objective was not to build a micro kernel, but rather a mini real-time OS. In fact, all major operating system tasks are implemented somehow within small-sized OSs, just like in monolithic kernels.

Individual Operating Systems Properties. We will briefly discuss some properties of widely used small-sized OSs. For a detailed discussion and comparison of the below OSs, the reader is referred to [11].

FreeRTOS is an open source software, written in C [5]. Since only the kernel is distributed, any device driver needs to be adapted to the kernel explicitly. Applications are first compiled and linked with the OS on a computer, thereafter loaded onto the robot. Similar properties apply to RobotC [16] and pbLua [10].

NXT OS [7] is developed in a C derivative, interpreted language and is adapted solely to NXT robots. Not like the above OSs, applications are not linked with the OS. After the OS is loaded to the robot, multiple applications can be transferred to the file area at any time. To the best of our knowledge, this is the only example that provides an on- board application development environment.

leJOS NXJ [2] is an OS with a built-in Java Virtual Machine (JVM), but itself is written in C. It was primarily developed for NXT robots. Like NXT OS, it is separated from applications and any multiple applications may be transferred at any time. As the JVM supports multiple threads, applications may be executed in multi-tasking mode.

Application Development Environments. Since most OSs do not provide an on-board application development environment or the provided one is too restricted, a separate computer software is required, for developing applications.

Computer software distributions for developing robot applications have following common abilities:

- loading a robot OS onto a robot
- developing applications and loading them onto the robot
- simulation of an application: virtual and visual execution

Loading the OS separately from applications as well as first linking the OS with the application is also supported, for those OSs that require it.

For instance, the NXT application developments software for the computer [9] provides symbolic building blocks that can be related to each other, in order to specify a programme, which is simpler for children to learn. The NXT robot is discussed more detailed below. Urbi [3] supports Java, C++ and Matlab, which is more difficult for children to programme in. Similarly, Webots [14] supports C and C++. These development environment provide a simulation environment, for testing applications, before loading onto the robot. The simulator can visually show the selected robot, within a specified environment and execute the application virtually, for verification purposes.

The NXT Robot

We have developed the first versions of becerik for NXT, since only NXT robots provide hardware capabilities for on-board application development, among small-sized robots. In this section, we briefly discuss the features of the NXT hardware, NXT OS and its original on-board application development environment.

Hardware Properties of NXT. NXT robots are developed by joining various parts of the kit that can be attached to each other. Among them is the NXT control unit that has mechanical and electronic sockets for attaching and controlling all possible sensors and actuators. The NXT controller has following capabilities:

- micro processor 32 bit ARM7TDMI; flash memory 256 kB; RAM 64 kB
- controller 8 bit Atmega48; flash memory 4 kB; RAM512 byte
- 4 sensor sockets: colour; grey scale; infra-red; ultrasonic; gyroscope; compass; touch; acceleration
- 3 actuator sockets: motor; lamp
- Bluetooth: NXT to computer communication or between two NXTs
- display 18x8 LED: for on-board application development or displaying some application output
- buttons: orange input; dark grey exit; light grey left and right, one for each
- USB
- charging

For developing robots with more sensors and actuators, up to 3 NXT controllers may be connected with each other.

Properties of the NXT Operating System. With respect to our objectives, following NXT OS properties are significant:

- build-in drivers for all available sensors and actuators
- can file multiple applications that were loaded from the computer
- allows developing applications, by using the on-board provided LED and buttons
- build-in interpreter for executing applications

Since multi-tasking is not supported, only one application can run at a time.

Application Development Environment of NXT. The most important application that is available on the NXT control unit is the default NXT operating system with the built-in application development environment. The user menu choices are of the application development environment are embedded into the remaining operating system code.

Any application that is developed within this environment must consists of exactly 5 steps. Except the 5. step, all other steps may be left empty. All commands that are available for a particular step can be selected from the menu of that step. Unfortunately, all on-board available commands are only a small subset of all commands that the NXT interpreter can execute. For instance, turning commands can execute only 90 degree or non-standard sensors or actuators cannot be selected. In order to utilise the full range of all commands that are available with the NXT interpreter, one must develop applications within a software environment on a computer. There is another drawback of the NXT firmware: although applications that are developed on a computer can be saved on-board of the NXT, on-board developed applications cannot be saved on-board.

In a Turkish localisation project of the NXT user interfaces [15], we have examined the NXT code also with the objective to extend the 5-steps restriction to n-steps. Due to the compact and memory-efficient coding of the NXT software, but its unfriendly maintenance properties, such an upgrade was unreasonable.

The Application Development Environment Becerik

Becerik is an application development software that runs on top of leJOS. User interface languages are currently available in English and Turkish.

Becerik for leJOS NXJ. The driver and file management of leJOS is very similar in functionality to that of NXT, however without any constraints. For instance, all drivers may be loaded or only the most used ones.

When the robot starts, either a default application may be started automatically or the user interfaces of leJOS. If first leJOS has started, some operating options may be selected from the menus. Thereafter, a particular application may be started.

Any application that was developed with becerik, is stored in a separate file under leJOS. Multiple such applications may be selected from within becerik and concurrently run under leJOS.

The Becerik Software. Before becerik is loaded onto the robot, first a language is selected. Whichever language is needed at the user interface, the related language file with extension lng is selected and becerik is compiled. After becerik has been loaded onto the robot, the first menu "Define Application" will be displayed and the application development environment has started (*Figure 1*).

After choosing "New Application", all commands that are supported by NXT can be added step by step under "Command" (*Figure 3*). Since there is no restriction on the number of application steps, commands can be added to an application, until the memory is full.

If multiple applications are required to run, for instance line following and ball following, then these can be chosen under "Load Application" one after the other (*Figure 2*) and leJOS would execute them all, together with becerik, concurrently. The applications would run in the background, whereas becerik would remain active on the display. The user may chose becerik options for displaying currently running applications "Running Applications", start a further application or chose to stop a particular application. Thus complex applications can be defined (*Figure 3*).

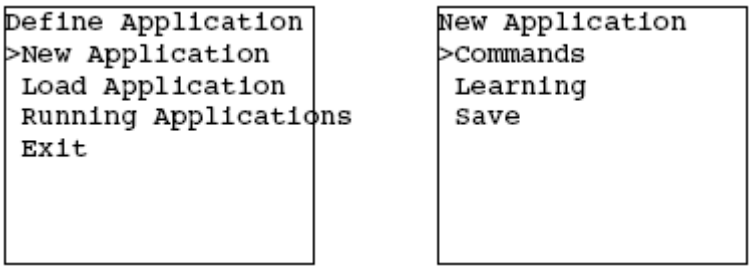

Figure 1. Application Definition and New Application Menus of becerik.

The Command Language Becerik. It has been shown experimentally that pre-school children can perceive a word symbolically, when the association of that symbol to a known object is shown, then grasp the relationship and, to some extend, can use it [13]. In robotics courses that we give in our laboratory [12], we observe that children, who do not know English, are able to use the English menus on the robot properly, even though the English words are explained to them only once experimentally. We have designed the becerik command language according to these experiences. It consists of the following 6 commands (*Appendix*):

- new: defines a new object that will be used
- set: assigns a value to an object
- var: assigns a value to a variable
- get: reads the value of a used object
- compute: defines arithmetic and logical operations for given variables
- jump: jumps to the given specification

Objects can be selected form the available sensors and actuators.

Memory Utilisation. Every command occupies 2 bytes in memory. Since 3 bits encode the commands, in total 8 commands could be used. The remaining 13 bits are used to code all possible parameters of a command.

After loading the robot with the NXT OS, only 110 kB remain for user applications. After loading leJOS, instead of NXT OS, 210 kB remain for user applications. When becerik is loaded additionally, then 190 kB remain. In other words, although leJOS and becerik provide more functionality than NXT OS, they occupy considerably less amount of memory.

Additional Properties of becerik. All properties of becerik that we have introduced are included in the open source distribution [6]. becerik is still under development and we plan to implement additional properties.

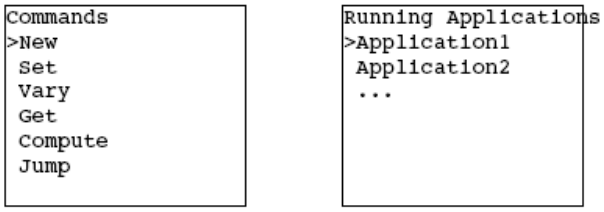

Figure 3. Command Definition and Running Applications Menus of becerik.

One property is, to define a symbolic language and replace the current textual user interface with symbolic choices. The intention is to create an international language as well as to facilitate the programming for children.

To facilitate the utilisation of becerik on middle-sized and large-sized robots, as on-board application development environment, we are currently developing an Eclipse plug-in. The plug-in will enable one to develop becerik applications within Eclipse. The adaptation of the becerik interpreter on middle-sized and large-sized robots would then be easier, since they mostly run computer OSs, where a JVM can easily be installed.

We are planning to implement the learning capability that we have mentioned before, by maintaining statistical data about sensor and actuator data, related to the application steps. One can enable a particular application to learn, not to learn, freeze or forget already learned, by choosing appropriately under "Learning" (*Figure 2*).

Conclusions

We have presented the on-board application development environment becerik and the becerik command language, in which the applications are coded. The language consists of only 6 commands and should be easy to learned, even by pre-school children.

Our objective with becerik is to facilitate the hardware- software-co-design of a robot. When a child realises that while it is creating a robot hardware, it can simultaneously create the software, then it may personalise the robot and thus start experimenting with cognitive skills, of its own as well as of its robot.

We are currently developing another software for developing applications in the becerik language on the computer. This software will be plugged-in to Eclipse, for developing becerik applications on the computer. This would enable more skilled children to develop first most of an advanced becerik application on the computer and then continue developing on the robot.

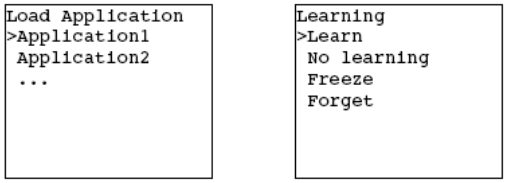

Figure 2. Load Application and Learning Menus of becerik.

One future work is, to develop a learning component, where applications can learn from interactions with the environment.

A further work is, to replace all textual communication at the display with symbols only. That should facilitate the learning as well as provide a single international language for robot-user interaction

References

[1] Andersson, Karl; Andersson, Robert; 2005; "A comparison between FreeRTOS and RTLinux in embedded real-time systems"; Linköping University

[2] Bagnall, Brian; 2002; "Core LEGO MINDSTORMS Programming"; Prentice Hall; lejos.sourceforge.net

[3] Baillie, JC; Akim, D; Hocquet, Q; Nottale, M; Tardieu , S; 2008; "The Urbi Universal Platform for Robotics"; Simulation, Modeling and Programming for Autonomous Robots (SIMPAR); Venice/Italy

[4] Barker, BS; Ansorge, J; 2007; "Robotics as Means to Increase Achievement Scores in an Informal Learning Environment"; Jounal of Research on Technology in Education; International Society for Technology in Education

[5] Barry, Richard; 2010; "Using the FreeRTOS Real Time Kernel – a Practical Guide"; Real Time Engineers Ltd

[6] becerik; sourceforge.net/projects/becerik

[7] Ferrari, Mario; Ferrari, Guilio; Astolfo, David; 2007; "Building Robots with LEGO Mindstorms NXT"; Syngress Publishing

[8] Gasperi, Michael; 2008; "LabVIEW for LEGO MINDSTORMS NXT"; NTS Press

[9] Grega, W; Pilat, A; 2008; "Real-time control teaching using LEGO® MINDSTORMS® NXT robot"; International Multiconference on Computer Science and Information Technology; (IMCSIT)

[10] Ierusalimschy, Roberto; 2006; "Programming in Lua"; lua.org; www.hempeldesigngroup.com/lego/pbLua

[11] Kumova, Bİ; Takan, S; Tos, U; Geçer, EC; Aytar, A; 2010; "Türkçeleştirilmiş bir Robot İşletim Yazılımı İle Robot Üzerinde Uygulama Geliştirme" (On-Board Robot Application Development Using A Turkish Localised Robot Operating System); Ulusal Otomatik Kontrol Toplantısı (TOK); GYTE

[12] Kumova, Bİ; arf.iyte.edu.tr/~bkumova/teaching/RobotBehav/School

[13] Mason, JM; 1980; "When Do Children Begin to Read: An Exploration of Four Year Old Children's Letter and Word Reading Competencies"; Reading Research Quarterly
[14] Michel, O; 1998; "Webots: a powerful realistic mobile robots simulator"; Workshop on RoboCup; Springer Verlag
[15] nxtturkish; SourceForge, sourceforge.net/projects/nxtturkish
[16] RobotC; www.robotc.net
[17] Silk, EM; Schunn, CD; Shoop, R; 2009; "Synchronized robot dancing: Motivating efficiency and meaning in problem solving with robotics"; Robot Magazin
[18] Wendell, KB; Connolly, KG; Wright, CG; Jarvin, L; Rogers, C; Barnett, M; Marulcu, I; 2010; "Incorporating engineering design into elementary school science curricula"; American Society for Engineering Education Annual Conference & Exposition, Louisville/USA

- **Appendix: Ebnf Of the Command Language Becerik**

Becerik applications are coded in the command language with the same name becerik. The language that consists of only 6 commands, is specified below in the Extended Backus-Naur Form (EBNF).

```
Program : ( new | set | get | compute | jump)*;
new : 'new' object port;
set : 'set' port methodName VALUE;
get : 'get' port methodName varName;
compute : 'comp' computeType varName varName;
jump : 'jump' varName HEX_DIGIT;
varName : 'a' | 'b' | 'c' | 'd' | 'e' | 'f' |'g' | 'h';
port : inputPort | outputPort | multiPort |
    varName | spacialPort;
computeType : '*' | '/' | '+' | '-' | 'Mod'|
                                        '<' | '<=' | '>' | '>=' | '=' | '!=' |
    'no criterion' | 'if zero';
inputPort : '1' | '2' | '3' | '4';
outputPort : 'A' | 'B' | 'C';
multiPort : 'AB' | 'AC' | 'BC';
spacialPort : 'tone port' | 'wait port' |
    'Bluetooth port';
object : ( 'Variable' varName
            | 'Motor' outputPort
            | 'Robot' multiPort
            | 'Object' inputPort
            | 'Voice' inputPort
            | 'Light' inputPort
            | 'Compass' inputPort
            | 'Touch'   inputPort
            | 'Color' inputPort
            | 'Gyroscopic' inputPort
            | 'Accelerometer' inputPort
            | 'IR Seeker' inputPort
            | 'Tone' 'tone port'
            | 'Wait' 'wait port'
            | 'Bluetooth' 'Bluetooth port');
methodName : ('a'..'z' | 'A'..'Z' | '0'..'9'
    | '.')+;
VALUE :'0'..'8';
HEX_DIGIT : '0'..'9'|'a'..'f'|'A'..'F';
```