KNOWLEDGE-BASED CONTROL OF REACTIVE SYSTEMS WITH MULTI-LAYER ARCHITECTURE

P. MATYASIK, G. J. NALEPA
AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY, POLAND

KEYWORDS: Intelligent Control, Embedded Systems, Prolog, XTT, Rule-Based-Systems, Real-Time-Systems, Hexor, EPP, GNU/Linux

ABSTRACT: This paper presents a concept of an integrated development platform for fast and error-free implementation of embedded intelligent systems. The main idea reside in dividing the control program into separate logic layers. The lowest-level layer uses embedded real-time operating system. The high-level control is knowledge-based and provides an intelligent system behavior. The paper presents some implementation details of the low-level layer and the concept of the intelligent control layer. The concept is then applied to the *HexorII* mobile robot platform.

INTRODUCTION

An intelligent autonomous mobile robot, such as *Hexor*, can operate in an ever changing environment. This intelligent behavior can be programmed using procedural languages such as C or Java. Standard development environment for *Hexor* uses *Bascom* [5] programming suite and *Basic* as the main programming language. *PonyProg* [4] programmer is used for writing code to the processor. Bascom suite is able to use the programmer as an internal tool which makes them a flexible tandem. It's very easy to use them in the common development cycle: write code – program – run – rewrite code. Bascom also supports a debugger connected with the *JTAG* (Joint Test Action Group, the IEEE 1149.1 standard "Standard Test Access Port and Boundary-Scan Architecture") interface which is very helpful when hard to find errors occur.
However, this does have some serious limitations such as the Basic language itself. It is a great language for teaching programming, but is not efficient while writing more sophisticated software. Moreover, when testing new programs the robot has to be connected to the host computer with a programming cable. It allows for running code in processor, but makes testing intelligent behaviors of Hexor hard. The solution presented in this paper allows to evade those problems. The approach presented in this paper offers an alternative to a direct code writing. Instead of that it presents a hierarchical, top-down, rule-based method to the reactive control logic design and implementation.
This paper extends the approach presented in detail in [9, 8]. In this approach the design of the rule-base of the control system is supported by an integrated design, analysis and implementation process. The process benefits from a visual knowledge representation method for rule-based systems called XTT [7]. It allows for fast prototyping of rule-based systems using Prolog. The designed rulebase is automatically translated into a predefined Prolog form, which serves as an executable system prototype.
The presented approach benefits from the *Embedded Prolog Platform* [12]. It allows for practical deployment of a XTT-based control logic into an embedded system. The platform is composed of a Prolog interpreter embedded into a GNU/Linux environment communicating with controlled hardware. The communication can be either local, which is comfortable for the development stage, or wireless – then it is truly autonomous robot. While the approach has been applied to the *Hexor* robot, it could be used for a wide class of event-driven control applications.
The paper is composed as follows: in Sect. 1 the *Hexor* robot is presented, including hardware design, internal and external software overview. Also some limitations of the original design are presented with proposed solutions.
Sect. 2 shows new *Multi-layer control architecture* designed to control the Hexor robot. Sect. 3 presents an idea of knowledge-based control. In Sect. 4 *Embedded Prolog Platform* and method of integrating it with presented system is introduced. Sect. 5 shows real-time capabilities of the presented system. Sect. 6 briefly presents of a new integrated knowledge-based environment with graphical interface. Sect. 7 Shows possible directions of extending presented software architecture.

THE HEXOR PLATFORM

HexorII is an autonomous 6-legged intelligent robot device. It was developed by *Stenzel* company as a didactic platform. It has modular construction and can be easily extended by additional components (compass, laser sensors, etc.). The company provides the hardware and some simple development software environment.

The Hexor Robot

HexorII is a metal-based construction. It looks and moves like a scorpion. The robot is moved by three servos: one for tilt, two for forward and backward leg moving. With this kind of mechanical construction *HexorII* executes *fast insect movement algorithm* [14]. Two other servos controls camera which is fitted on the top of the tail and can move +-90 degrees providing wide scanning area. Video and sound from the camera are transmitted to the

receiver by radio. Video and sound signal is converted to a digital form using a common TV-card. Sonar mounted with the camera can measure distance to objects from 3 to 300 cm. Hexor is also fitted with four infrared sensors; three mounted at the back, and one mounted at the front, where two touch sensors (tentacles) are also mounted.
The electronic board fitted on Hexor's hull is equipped with three microcontrolers. The main controlling microcontroler is ATmega128 [1]. Tasks of these chips are:

- scanning sensors (sonar, tentacles, infrared),
- generating signals for servos (PWM),
- executing the movement algorithm,
- communicating with the control panel,
- executing higher level algorithm (obstacle avoidance, obstacle search, etc.)

The second microcontroler is ATmega8 [2]. It is a helper controller used mainly for extensions (speech synthesis, speech recognition, etc.). The third microcontroler is a part of the radio communication module. It is responsible for maintaining reliable bidirectional communication with the host computer. Every microcontroller can be easily reprogrammed using special connectors mounted on Hexor's main board. A proper code execution is guaranteed by the hardware watchdog chip also fitted on board.

Hexor Control Software

Originally, Hexor is controlled by a Basic program running on ATmega128 microcontroler. It executes in a loop the following algorithm:

1. setup hardware,
2. check radio for commands,
3. if moving, make step,
4. if moving, check sensors,
5. if moving, and obstacle change direction,
6. goto 2.

The program is written with the Bascom [5] development suite. The *HexorII* software architecture is schematically presented in Fig. 1.
The robot can be controlled from the host computer using the *Control Panel*. The Control Panel is an application written in C++. It has the following functionality:

- switch between programmed in robot behavior models: search, avoid obstacles, manual,
- control robot movement: forward, backward, turn left and right,
- control camera operation and position,
- watch video from camera,
- see sonar and thermometer readings,

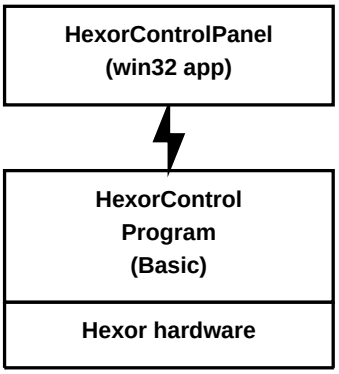

Fig. 1. HexorII system architecture.

- send custom control frame to the robot.

Control Panel allows to control many robots by selecting unique unit address. Commands are transmitted to the robot by a radio module connected to a USB port. Transmission protocol implemented for Hexor allows sending commands from a host computer to the specified machine. For every command send to the robot it responses with an appropriate *ACK* frame containing the state of sensors. Control Panel is available only for the MS Windows platform.

Hexor platform Limitations

While *HexorII* is a good robotic platform for didactic needs, it lacks some features needed for advanced control algorithm development.
The transmission protocol implemented in Hexor does not allow the robot to send data independently. Environmental information can by only pooled from the robot by the host software. This stays in conflict with the mentioned earlier *event driven* behavior.
Writing microcontroler software in Basic with a single control loop is easy. Such a program can be understood even by a user not familiar with microprocessor technology. Unfortunately that approach results in performance loss and *domino effect* while modifying subsystems (changes made to one subsystem results in crashing others).
Last, but not least, the software for host computer is available only for the *Win32* platform. Component for controlling Hexor from user program is distributed only as an *ActiveX* component. Because of this it is impossible to use a PDA as a higher-level control device.

Motivation

Because of the software platform limitations presented in previous subsection, a new Hexor's internal controlling software architecture is proposed. Simple Basic program with one control loop was replaced by a real-time embedded operating system. Every subsystem of the *Hexor* robot is managed by a specified task. Due to this modifications, the source code became clear and much more easy to understand. This way, improving Hexor's subsystems is simpler.
Another advantage of decomposition of this system, is the opportunity to regain processor computing power wasted

for delay loops. The main advantage of changes made is the improvement in response time from the Hexor robot. It's very important in context of an event-driven intelligence layer. New development libraries are provided for the low-level communication with the Hexor. This will allow for fast prototyping of knowledge-based software using common PC as a host computer and placing final version in the target device.

MULTI-LAYER ARCHITECTURE

Several possibilities where taken into consideration while developing the new Hexor low-level software. The first one was a dedicated application runing directly on the hardware. This one was abandoned because of problems with controlling whole hardware with a complicated state machine for each task, running off an interrupt timer. Thus an embedded operating system became an obvious choice. The requirements where as follows:

- portability (possible changes of Hexor's electronic board architecture),
- source code availability,
- programmers API (task control, task communication, timers),
- licence (possibly free),
- price (possibly nothing).

Search for an OS resulted in final two: FreeRTOS and AvrX [3]. FreeRTOS was chosen because of the pure C language implementation, better portability and a better programming interface (in authors opinion).
New Hexor controlling software is written on top of FreeRTOS [16]. It is an open source portable hard real-time operating system. It can run with a small memory footprint (about 700 bytes for OS, depending on the configuration). It provides task controlling functions, IPC procedures. FreeRTOS is written in the C language and ported to many hardware architectures and many C compilers [16].
HexorNG software architecture features:

- a multi-layer, easily extensible design,
- an intelligent knowledge-based control, based on the XTT rule-based representation,
- ability to distribute computations between many processing units,
- reliable real-time operating system based hardware control.

The new Hexor software architecture is presented in Fig. 2. HexorNG software is connected to the *Embedded Prolog Platform* with Hexor-EPP-Protocol [1] (*HEP*) as shown in Fig 2. *HEP* allows for unification of Prolog clauses between Hexor robot and machine running the higher level, intelligent control software. While *HEP* gets its name

[1] HEP is being developed with Piotr Zięcik, kosmo@agh.edu.pl.

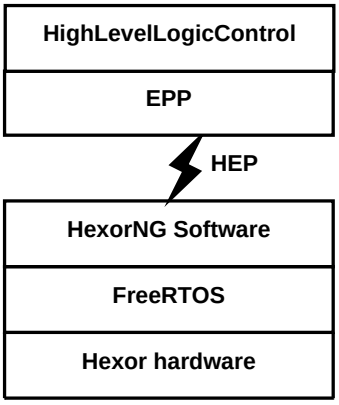

Fig. 2. HexorNG system architecture.

from the first application, it can be used in all cases with similar architecture.
Connecting Hexor robot to the knowledge-based control layer with *HEP* allows for an easy development of the second mentioned layer. In the testing scenario *Intelligence Layer* is connected to robot with wireless link allowing for easy improving of the Intelligence Layer by using a common PC as the host machine.
Final version can be deployed to a mobile device (PDA, single board computer, etc.) and connected to the controlled device directly by cable. That turns Hexor into a true autonomous robot, self-powered and self-controlled. Simple behavioral control is done on HexorNG level (sensor reading, stopping when robot hits something). Intelligent actions are taken under control of the Knowledge-based software (obstacle avoidance, proceeding to specified location, etc.).
Every subsystem of the *HexorNG* layer is implemented as a single task (see Fig. 3). The *MainControl* task is responsible for setting up hardware and interchanging data between other tasks. The *ServoControl* task sets data for the timer interrupt routine which generates signals for servos. Communication services are implemented in the *RadioComm* task. This task receives data frames from upper control layer. HexorNG layer as a whole implements a kind of "instinct" for the robot.

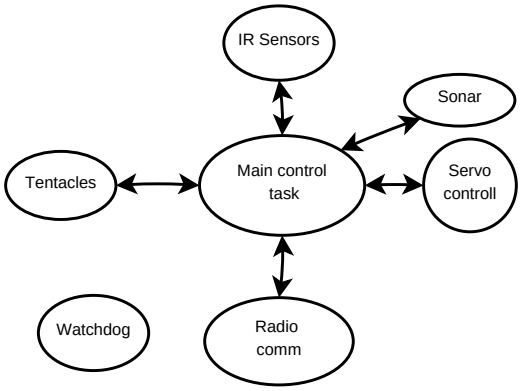

Fig. 3. HexorNG controlling software schema.

Decomposition of the controlling software to set of tasks allowed for a better code maintenance, simpler modifications and faster execution. Some independence given for every subsystem of the robot allow to connect it to

the higher level intelligent layer. Hexor can now interchange data with the upper level software while making steps which is important due to event-oriented controlling schema. Use of multitasking operating system allows Hexor to simultaneously walk, communicate and immediately react for new sensor readings.
HexorNG software substitutes all of the functions of the original one (low-level hardware control, movement execution, sensors reading, communication). The most important feature of the presented redesign is the improvement in response time of the robot. Communication window in the new design is not correlated with the robot movement. New orders, in the original design can be executed only when a robot step is finished. In the new design orders can be executed regardless of the step phase. This is crucial for the controlling schema presented in this paper. *HexorNG* software provides all necessary features to equip the Hexor robot with basic low-level "behavioral intelligence".

KNOWLEDGE-BASED CONTROL

The intelligent behavior of the controlled system can be described using a high-level, rule-based model. Following assumption are taken into consideration in constructing the knowledge base model:

- a top down approach,
- visual form,
- hierarchical structure,
- ability to verify system properties,
- high-level of abstraction form the hardware,
- distinguish *time* as special attribute for time-oriented verification and time related behavior.

The knowledge base is described using the *XTT* method presented in [6, 7, 11]. This method provides implementation agnostic approach for rule-based systems. *XTT* allows for fast prototyping of the knowledge-based models with Prolog. Very important feature of the *XTT* method is the graphical, hierarchical form of representation of the given knowledge. It features well defined logical form of presenting data, high level of expression and high density of the presented information.
An XTT-based development environment allows for a fast implementation and testing of new, high level control algorithms. Moreover, with its ability to verify knowledge during editing phase, allowing for avoiding problems related to rule system development stage, like completeness, or to optimize edited system. Development of the knowledge-based control system in the XTT-based environment is presented on Fig. 4.
XTT representation allows for generating a Prolog-based control logic prototype. In order to execute this logic, a Prolog interpreting environment must be provided. In this approach the *Embedded Prolog Platform* is used. *EPP* as presented in [12] provides ability to run Prolog-based meta interpreter for rules produced by a visual editor.

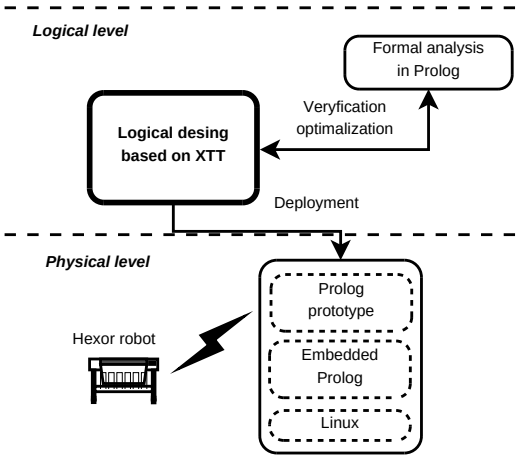

Fig. 4. XTT-based development cycle

EPP INTEGRATION

Prolog was developed as a practical implementation of the *programming in logic* paradigm. As a programming language it offers great level of expressiveness and conceptual logic representation. It's a high level language and lacks low level, hardware related interfacing functions. Commonly, discussion about using Prolog in control applications ends with asserting new orders to memory and somehow transferring it to the hardware. *EPP* presents how to connect Prolog to the controlled device in a reliable way. *EPP* consists of three layers [12]:

- Prolog layer, with Prolog compiler and interpreter for executing control logic code,
- a supervising middleware, for connecting the Prolog interpreter with operating system,
- an operating system itself, general purpose or embedded for interfacing hardware, possibly with real-time extensions.

The EPP architecture allows for using different Prolog implementations as an inference engine by introducing a *Prolog abstraction layer*. Currently EPP supports two: SWI Prolog and YAP Prolog. Support for GNU Prolog is in development stage. *Real-time supervisor* is a crucial part of the solution. It controls the Prolog core in the real-time. Its main tasks are:

- event management – registration, scheduling and execution according to priorities,
- call management – execute, registered from Prolog level, event handlers,
- device management – registering device drivers, routing events to the appropriate devices,
- virtual timer driver – ability to use a time related functions from the Prolog level.

EPP is based on GNU/Linux components because of availability and high customization capabilities. What is important, is the ability to modify and extend the source code, due to the GNU GPL licence and wide range of the supported hardware.

EPP is a crucial element of the presented *multi-layer knowledge-based* development environment. It allows to use Prolog as a controlling language for hardware device.
However, the control system presented here differs from the original one [12]. In the original design, embedded Prolog runs on a Linux runtime which runs on the controlled hardware. In this design the Linux runtime connects to possibly remote hardware with serial link. It simplifies the development process, allows to use an external computer power for computations, and reduces the cost of the development. Moreover, providing wireless links for more devices constructs of multi-agent testing platform.

REAL-TIME EXTENSIONS

EPP introduces real-time capabilities and hardware drivers for Prolog. In this design, Prolog execution is controlled by an additional layer responsible for event queueing and execution. Using embedded Prolog-based lower-layer allows to control hardware devices on very high level of abstraction as provided by XTT itself. As a whole system provides knowledge-based, event-driven control for devices.
Embedded Prolog Platform consists of Prolog runtime and supervisor. Executing these task in the real-time schedule allows to achieve reliable, possibly hard real-time environment [12]. The connecting of EPP with controlled hardware, doesn't affects real-time controlling capabilities because it's time-deterministic protocol.

HEXOR NG ENVIRONMENT

Presented solution comes from authors earlier experiences in the field of real-rime systems, artificial intelligence, formal methods and knowledge engineering [8, 9, 10, 15, 13]. Project of the integrated development environment is based also on derivatives from the *Mirella Project* [10].
The new *HexorNG* environment consists of:

- a knowledge base editor,
- a communication library,
- new controlling panel.

Knowledge base editor uses an extended version of the XTT diagrams for the graphical representation. A highly development character of presented project forced change of the underlying architecture of integrated editor. Due to the "easy modification" requirement the *Eclipse* development suite is used together with *Eclipse Modelling Framework* (EMF) and *Graphical Modelling Framework* (GMF).
The *EMF* and *GMF* are used for fast prototyping of graphical editor for knowledge-base used for controlling device. Eclipse with EMF and GMF was chosen because models developed with this tools are easy to modify. Moreover, around EMF many converters between models are available. Internally EMF stores domain model in a generally known *XMI*. GMF lays on the *Graphical Editing Framework* which provides a wide range of 2D graphical functions available for programmer to customize presentation layer of the editor.
Using EMF and GMF allows for even significant changes at any stage of a development process of the given model. It's because large parts of the model editor and some parts of the graphical editor can be generated automatically.
Presented design benefits from the EPP which provides ability to directly use Prolog as a language for hardware control. However, Prolog is not directly used for writing controlling program. Instead of that, a meta-interpreter for rules produced by the knowledge editor is written in Prolog. It provides two abstraction mechanisms. One is to allow to transparently change the representation of knowledge in Prolog code. Second is ability to switch underlying hardware without changing base of rules. Moreover using Prolog meta-interpreter provides flexible control over rule execution. This feature is helpful in context of real-time control allowing for supervising time of the execution of given action.
As a whole, presented system will provide easy, direct deployment of the control logic into the robot. Use of *EPP* and Prolog meta interpreter allows for flexible system design. Calculation of the longest decision path in decision tree in rule base together with size of facts database will allow to estimate needed processor power for servicing a particular event.
Simultaneously, a new controlling software for the original *Hexor* code is developed. It will consist of:

- low-level communication library,
- background controlling daemon,
- plugins for controlling daemon.

The *communication library* will provide API for controlling Hexor from third party software.
Controlling daemon will be build on top of the communication library and provide TCP/IP interfaces for the remote steering of the robot. This includes communication procedures for sending and receiving command frames to/from device. Functions for grabbing image frames form robot's camera to use for vision-guided operation.
Controlling daemon will allow for the remote Internet connection and controlling Hexor with telnet-like interface using simple commands like *forward, stop, etc.* Also embedding a Web server is planned. It will allow to control Hexor over the Internet including: movement, sensor readings and video form camera. Described software will run on GNU/Linux platform allowing, with minor modifications, to port to other *NIX-like operating systems.
During early development stage simple testing software was written. Main goal of that work was to check how third party software works with Hexor controlling software. It was written using Perl language and allowed for steering robot with game-pad device. This program confirmed presence of proper USB device drivers for used communicating hardware for *NIX systems.

FUTURE WORK

The paper presents the concept of efficiently applying a high-level knowledge-based control logic to the mobile robot *Hexor.* The research presented in this paper should

be considered a work in progress. Future research will be concentrated on improving the visual representation of the knowledge in editor, and extending real-time control features provided by the HexorNG environment.

THE AUTHORS

Piotr Matyasik, M. Sc. works in Institute of Automatics, AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland
E-mail: ptm@agh.edu.pl
Dr. Grzegorz J. Nalepa works in Institute of Automatics, AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland
E-mail: gjn@agh.edu.pl

REFERENCES

[1] Atmel. *8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash ATmega128 ATmega128L*, rev. 2467g-avr-09/02 edition, 2002.

[2] Atmel. *8-bit AVR Microcontroller with 8K Bytes In-System Programmable Flash ATmega8 ATmega8L*, 2486qs-avr-10/06 edition, 2006.

[3] Larry Barello. Avrx real time kernel. http://barello.net/avrx/index.htm.

[4] Claudio Lanconelli. http://www.lancos.com/prog.html.

[5] MCS Electronics. *BASCOM-AVR*, ver. 1.11.6.3 edition.

[6] Grzegorz J. Nalepa. *Meta-Level Approach to Integrated Process of Design and Implementation of Rule-Based Systems*. PhD thesis, AGH – University of Science and Technology, Institute of Automatics, Cracow, Poland, September 2004.

[7] Grzegorz J. Nalepa and Antoni Ligęza. A graphical tabular model for rule-based logic programming and verification. In Zdzisław Bubnicki and Adam Grzech, editors, *Proceedings of 15th International Conference on Systems Science*, Wrocław, 2004. Wrocław University of Technology, Oficyna Wydawnicza Politechniki Wrocławskiej.

[8] Grzegorz J. Nalepa and Antoni Ligęza. Conceptual modelling and automated implementation of rule-based systems. In Tomasz Szmuc Krzysztof Zieliński, editor, *Software engineering : evolution and emerging technologies*, volume 130 of *Frontiers in Artificial Intelligence and Applications*, pages 330–340. IOS Press, 2005.

[9] Grzegorz J. Nalepa and Antoni Ligęza. A graphical tabular model for rule-based logic programming and verification. *Systems Science*, 31(2):89–95, 2005.

[10] Grzegorz J. Nalepa and Antoni Ligęza. A visual edition tool for design and verification of knowledge in rule-based systems. *Systems Science*, 31(3):103–109, 2005.

[11] Grzegorz J. Nalepa and Antoni Ligęza. Prolog-based analysis of tabular rule-based systems with the xtt approach. In *to be published in: Proceedings of the 19th FLAIRS Conference, Melbourne Beach, Florida, USA*. AAAI Press, 2006.

[12] Grzegorz J. Nalepa and Piotr Zięcik. Integrated embedded prolog platform for rule-based control systems. *XXXX*, XXX(XX):XXy–XXX, 2006.

[13] Paweł Skrzyński, Michał Turek, and Piotr Matyasik. Code generation from uml model - comparison of modern case tools. In Andrzej Wardziński ; Janusz Górski, editor, *Software engeenering: The new challenges*, Warszawa, 2004. Wrocław University of Technology, Wydawnictwa Naukowo-Techniczne.

[14] Stenzel. *HexorII robot manual*, 2006.

[15] Marcin Szpyrka, Piotr Matyasik, and Jacek Piwowarczyk. Rtcp-net approach to verification of embedded systems implemented in ada. In Marian Adamski, editor, *Proceedings volume from the 3rd IFAC workshop on Discrete-Events System Design 2006*, Zielona Góra, 2006. International Federation of Automatic Control, University of Zielona Góra Press.

[16] FreeRTOS team. Freertos documentation. http://www.freertos.org.