INTEGRATED EMBEDDED PROLOG PLATFORM FOR RULE-BASED CONTROL SYSTEMS

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

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

ABSTRACT: The paper presents the idea of using an integrated, Prolog-based platform for an efficient implementation of embedded control systems. A principal idea consists in separating an embedded control system into two layers. The first one is the logic control layer, where the control procedures are implemented in Prolog. The second one provides means for communication with particular devices. The paper describes the concept of an integrated platform allowing for implementation of Prolog-based control routines. The platform itself is built on top of a GNU/Linux based embedded runtime, suited for several hardware architectures. In the paper key architectural and implementation issues of the platform are discussed. They are presented using a control system example. The paper also discusses practical hardware implementation on the Palm III platform based on the Motorola/Freescale DragonBall CPU.

INTRODUCTION

The rule-based programming paradigm [2, 3] plays an important role in number of engineering domains, including intelligent and real-time control. However, an efficient design of rule-based systems, including the knowledge engineering process, encounters number of problems [2, 3]. In recent years rule-based systems design benefited from advanced design and implementation tools, that use number of techniques taken from the software engineering [12]. In this paper an integrated approach to the design and implementation of such systems, suitable for embedded applications is put forward.

This paper follows the approach presented in detail in [7, 6]. 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 is centered around a new visual knowledge representation method for rule-based systems called XTT. It combines selected important features of decision tables and decision trees in order to support the design and implementation process efficiently. The XTT-based design framework, the Mirella environment, provides a formal Prolog-based means to analyze the designed rulebase [9, 10]; some critical properties such as determinism or completeness can be automatically verified. A rule-based system designed with XTT is implementation agnostic. The design tools provided with Mirella allow for translation to different rule-based implementations. They also allow for fast prototyping of rule-based systems using Prolog [1]. The designed rulebase is automatically translated into a predefined Prolog form, which serves as an executable rule-based system prototype.

This paper extends these ideas by presenting a new approach to practical implementation of embedded rule-based systems. A principal idea is to provide an integrated platform for embedding Prolog into a hardware control system. This would allow 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 controlling the hardware. The complete rule-based system development cycle proposed in this approach is shown in Fig. 1. It includes the following phases:

1. Rule-based control logic design and analysis using a high level knowledge representation, the XTT.
2. Automatic translation of the XTT rulebase into an executable Prolog-based prototype.
3. The deployment of the prototype into an embedded control system, using the integrated Embedded Prolog Platform (the EPP).

This paper mainly focuses on the third phase of the above cycle. It presents the architecture and implementation of the platform. Results of some practical experiments, including the deployment an the Palm III on Motorola DragonBall CPU platform are also provided.

PLATFORM ARCHITECTURE

In the approach presented in this paper the control system logic is written in Prolog. The language is a good choice thanks to its advanced features such as high expressiveness and conceptual logic-based knowledge representation. However, there are some serious obstacles in using Prolog for practical implementation of real-life control systems. The main problem is, that Prolog is a high level language, and generic implementations do not provide robust means for hardware communication and control. This is why an integrated platform combining Prolog-based control layer with lower level runtime is proposed.

In this approach the platform (see Fig. 2) contains three main layers:
Vague system description
business rules
Formal conceptualization
ARD
Formal logical design
XTT analysis
Formal logical
Prolog
Verification
Refinement
Reformulation
prototype
Prolog
emProlog
Linux runtime

Logical design

Physical design

Embedded hardware

The Embedded Prolog Platform
Prolog prototype
emProlog
Linux runtime

Fig. 1. The system development cycle

- Prolog layer, including an advanced Prolog interpreter (and compiler), that executes the control logic code,
- a supervising middleware layer, used for integration of the Prolog interpreter with the operating system environment,
- an embedded multiplatform operating system with strong hardware support via number of drivers and real-time extensions, providing the main runtime environment.

The main control logic is contained in a declarative Prolog program. It is executed by the Prolog compiler (all advanced Prolog „interpreters” actually precompile the code). There are number of advanced Prolog implementations available, namely the SWI Prolog and GNU Prolog, with different features (e.g. Constraint Logic Programming). This is why an extra Prolog abstraction layer has been provided (see next section) in the Prolog layer. The Prolog layer is controlled in by the real-time supervisor middleware, using Prolog-to-C interface. It is composed of several elements, including event management, and hardware drivers communication and control. This layer provides unified driver interface for hardware device drivers provided by the operating systems. The supervising layer is implemented in pure ANSI C language for speed and efficiency.

The foundation for this architecture is a multiplatform embedded GNU/Linux-based environment with possible real-time extensions, providing low-level hardware drivers. The idea of using Prolog as the high-level knowledge specification language for control systems is not completely new though. In [11] an idea of combining Prolog knowledge representation with real-time features of the Ada language have been presented. However, this approach differed largely to the one presented in this paper. In [11] the proposal of developing the RT Prolog interpreter as a Task in Ada has been put forward. In order to do this a small Prolog interpreter was developed in Ada and integrated with a larger Ada system. This approach does not address the practical construction of an embedded environment. It also relies on the low-level Ada runtime implementation. In contrast, the approach presented in this paper offers a complete solution, from the high level logic design, to the actual hardware implementation.

PLATFORM IMPLEMENTATION

The approach discussed in this paper is based on the idea of controlling Prolog-based logical core, using a ANSI-C based real-time supervisor. This is made possible due to advanced Prolog-to-C interfaces found in multiple Prolog compilers. However, there are some implementation problems, related to different Prolog-to-C APIs.

One has to decide whether to choose a single specific Prolog implementation, or try to support several of them. The former approach seems to be more desired, since in practice different Prolog implementations have some important features; e.g. SWI Prolog has rich external procedure library and a clean API, GNU Prolog provides CLP (Constraint Logic Programming) capabilities and good performance, YAP Prolog has CLP and SWI interface. So, a decision has been made to provide means to work with different Prolog platforms.

In order to do this, a Prolog abstraction layer has to be provided. The SWI Prolog API has been chosen as the reference one. It seems to be the most flexible and well established. In fact some other Prolog compilers, namely YAP Prolog provide SWI compatibility API. Currently in our approach both SWI and YAP Prolog are supported. GNU Prolog compatibility is in the works, and is planned as a future extension.
GNU/Linux Runtime

The following software components constitute the GNU/Linux runtime (e.g. the RT Linux).

- The Linux kernel provides number of real-time related features and extensions such as low-latency control, kernel preemption and real-time extension (e.g. the RT Linux).
- A comprehensive Real Time Application Interface (RTAI) is used as interpreter for the Init script and few core utilities (such as mount, expand) needed for the system setup. Software upload is provided by Base64 encoder/decoder and other shell scripts. BusyBox combines tiny versions of many common Unix utilities into a single small executable. It provides replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc. BusyBox itself requires a MMU or XIP (Execution In Place) support, which is not present on all platforms. However, msh itself does not have such requirements. Using the components described above a prototype GNU/Linux runtime has been built. It has only about 100kB in size, stored in compressed form in ROM or Flash memory. It is sufficient for a complete system boot and setup, as well as executing the Prolog runtime.

REAL-TIME CONTROL ASPECTS

When it comes to real-life control systems, the real-time aspect of the control has to be considered. Prolog language itself does not provide any explicit time-related facilities. However, the solution presented in the paper introduces a possibility of Prolog-based control in real-time. It is based on the idea of extending the Supervisor with a real-time features.

The Linux Kernel used comes from the uClinux Embedded Linux/Microcontroller Project (uClinux.org), which provides an operating system including Linux kernel releases for 2.0, 2.4 and 2.6 as well as a collection of user applications, libraries and tool chains for number of processor architectures, including chips with no Memory Management Units (MMUs).

The Unix LibC library for embedded systems is provided by the uClibc Project (uclibc.org). The uClibc is much smaller than the GNU C Library, but nearly all applications supported by GLibC also work perfectly with uClibc. It currently runs on standard Linux and MMU-less systems with support for number of CPUs.

The whole runtime has been built using widely available GNU/Linux components. GNU/Linux has been chosen because of its availability, high customization capabilities, including the right the modify and extend the source code, thanks to the GNU GPL license, and good hardware support. It is also important to point out that the Linux kernel provides number of real-time related features, such as low-latency, kernel preemption and real-time extension (e.g. the RT Linux).

The following software components constitute the GNU/Linux runtime:

- The Linux kernel configured and optimized for an embedded system,
- The LibC library providing system functions,
- A simple Init facility, providing basic system setup and software upload mechanism,
- A simple system shell for program execution,
- Some extra shell utilities may be needed for other Unix-specific functions.
The current version of the Supervisor and Logic Controller has some basic time-related features, such as: wake-up event, and a simple built-in timer are provided. They allow for simple soft-real-time operation. In the future these will be extended and integrated with advanced features the Linux kernel provides, in order to allow a complete real-time environment, possibly even a hard real-time one.

**CONTROL SYSTEM EXAMPLE**

As a proof of concept a simple elevator control system has been implemented. Some illustrative excerpts of the Prolog implementation are included below. Every control logic begins with event handler registration predicates:

```prolog
/* Event namespace:
   <driver>/<event> OR
   <driver>/<device>/<event>
Driver namespace:
   driver_<driver>_<procedure> */
idle.
system_init :-
   register_event_handler('button_pad/button_pushed',button_pushed),
   register_event_handler('elevator/on_level', elevator_on_level),
   register_event_handler('elevator/stopped', elevator_stopped),
   register_event_handler('doors/opened', doors_opened),
   register_event_handler('doors/closed', doors_closed).
```

Main event handling predicates are as follows:

```prolog
button_pushed(L) :-
   idle,
   assert(level(L)),
   driver_doors_open.
button_pushed(L) :-
   assert(level(L)).
elevator_on_level(L) :-
   level(L),
   retract(level(L)),
   driver_elevator_go_up.
elevator_stopped :-
   driver_doors_open.
doors_opened :-
   driver_timer_create('doors_timer'),
   register_event_handler('timer/doors_timer/alert',
    time_to_close_doors),
   driver_timer_run_once('doors_timer', '+5s').
doors_closed :-
   choose_direction.
```

When the elevator’s door are opened an extra door timer is created. It uses the unified time driver provided by the Supervisor. The choice of direction for the elevator is programmed by an auxiliary Prolog predicate:

```prolog
choose_direction :-
   driver_elevator_get_level(L),
   level(X), X > L,
   retract(idle),
   driver_elevator_go_up.
choose_direction :-
   driver_elevator_get_level(L),
   level(X), X < L,
   retract(idle),
   driver_elevator_go_down.
```

The timer measuring the time to close doors is implemented as follows:

```prolog
time_to_close_doors :-
   driver_doors_are_blocked,
   driver_timer_run_once('doors_timer', '5s').
time_to_close_doors :-
   unregister_event_handler('timer/doors_timer/alert'),
   driver_timer_destroy('doors_timer'),
   driver_doors_close.
```

Thanks to clean and transparent Prolog syntax the above code is quite self-explanatory and easily extensible. On the other hand from this approach point of view it is the fully executable specification.

**HARDWARE SUPPORT**

Number of hardware platforms are considered as targets for the GNU/Linux runtime. These include embedded PC-like systems, PDAs, and mobile phones. The following CPUs will be eventually supported: Motorola/Freescale m68k/Coldfire/DragonBall, Intel x86, and ARM platforms. Some preliminary test of a non-trivial platform have been performed so far. A skeleton Linux runtime system has been successfully bootstrapped on a Palm IIIx PDA. The PDA is based on the Freescale 68EZ328 DragonBallEZ CPU [4, 5]. The Palm IIIx was chosen as a fast prototyping platform because of its multiple advantages; it:

- provides a platform found in many real-life embedded systems,
- is an easily available and cheap solution,
- is based on a classic, flexible and very well documented Motorola/Freescale processor,
- has the Flash EEPROM memory that can be reprogrammed in-circuit via the serial interface provided by the PDA,
- has number of well documented build-in peripherals (such as an LCD touchscreen, the keypad, the PWM audio, an IrDA and RS232 interfaces),
- is extendable using special extension slot.

In order to fully support the Palm IIIx device, the \( \mu \)CLinux kernel has been extended by rewriting a bootstrapping code to boot the system from built-in ROM, improving
ROM generator for Palm devices to provide support for original PalmOS tools, and providing basic power management and keypad drivers. Similar work has been done by LinuxDA project – it is however completely uncooperative towards free software community (it is considered by some as a violator of the GNU GPL).

Below a µClinux kernel booting on the Palm IIIx is shown:

```
68E328 DragonBallEZ support (C) 1998 Rt-Control, Inc
uClinux/MC68EZ328
Flat model support (C) 1998,1999
Kenneth Albanowski, D. Jeff Dionne
Palm IIIx support by
Piotr Zieciêk <piotr.zieciêk@angel.net.pl>
PalmV support by Lineo Inc. <jeff@uclinux.com>
Calibrating delay loop.. ok - 1.28 BogoMIPS
Memory available: 3896k/40796k RAM, 0k/0k ROM
(241k kernel code, 184k data)
Swansea University Computer Society
NET3.035 for Linux 2.0
```

MMU availability All Prolog implementations that have been tested assume MMU availability. It allows for Prolog stacks to grow and shrink dynamically. Without an MMU the memory needs to be preallocated in order to prevent memory fragmentation. It is a workable but not very flexible solution.

Performance The SWI-Prolog has been benchmarked on the Palm IIIx (2MB ROM, 4 MB RAM, MC68EZ328 CPU at 16 MHz). The measured speed is approximately 1340 lips (Logical Inferences Per Second). For a comparison a modern PC (P4 1.7 GHz) achieves 28.6 Mlips and PII 450 MHz achieves 5 Mlips. It is worth noting that modern embedded platforms provide much faster CPUs, e.g. modern Palms run on 200-416 MHz CPUs.

Peripheral Devices On the testing Palm platform the drivers for the available devices including the keypad (as the input device), and the LCD backlight and PWM (as the output) have been implemented. There is a possibility to connect any device to the Palm board because of the µC Bus availability on special socket inside the Palm. These drivers will serve as prototypes for the future drivers on other platforms.

CLOSING REMARKS

In the paper an integrated embedded Prolog platform (EPP) has been presented. The platform plays a key role in practical deployment of rule-based control systems. The development of such systems is supported by an integrated, hierarchical process, including a new logical knowledge representation and analysis method called XTT. The application of this method include number of other fields, where efficient rule-based control systems are used, such as network security solutions [8].

The platform discussed in the paper integrates the high level control logic encoded in Prolog, with a GNU/Linux runtime, suitable for number of embedded platforms. The paper includes results of practical test on the Palm III platform running the Motorola/Freescale DragonBall CPU. The work on the platform should be still considered a work in progress. Future work will include support for other hardware platforms, including the ARM family. Another important issue that will be addressed is the incorporation of a real-time optimized Linux kernel. A full API making real-time extensions available to the Prolog control logic, is in the works too.

THE AUTHORS

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

and in

Institute of Physics, Kielce Pedagogical University
Ul. Świętokrzyska 15, 25-406 Kielce, Poland

Mr. Piotr Zieciêk is a student at AGH UST, he writes his Master’s Thesis under supervision of Dr. G. J. Nalepa
E-mail: piotr.zieciêk@angel.net.pl
REFERENCES


