

Prototype Prolog API for Mindstorms NXT^{*}

Grzegorz J. Nalepa¹

Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
gjn@agh.edu.pl

Abstract. The paper presents a pure Prolog-based API for controlling the LEGO Mindstorms NXT robot platform. In the paper the design as well the implementation is described, with an example algorithm presented. The API uses a multilayer architecture, composed of a behavioral, sensorimotoric, and connection layer. This platform can be used as a generic solution for programming the NXT in Prolog. It also serves as a foundation for a higher-level visual rule-based programming within the XTT knowledge representation.

1 Introduction and Motivation

Building intelligent robots [1] has always been one of the most important areas of both pursuit and research in Artificial Intelligence [2], and applied engineering. Creating such robots requires skills from different domains, including deep knowledge of materials and mechanics [3], as well as control theory, artificial intelligence, computer science, and even psychology and linguistics, when we take human-machine communication into account. However, these days the field became much more accessible to non-experts, thanks to number of ready robotics solutions. These include some ready robots, that can be bought and trained, e.g. SONY Aibo, or iRobot Roomba. Recently, a new release from the LEGO company improved this situation even further.

LEGO Mindstorms NXT is a universal robotics platform, that offers advanced robot construction possibilities, as well as sophisticated programming solutions [4]. It is composed of an embedded computer called *the brick*, including Bluetooth wireless communication and USB port. The NXT has a set of three actuators (servo motors), and a set of sensors, including the touch, sound, light, and ultrasonic sensor. The whole mechanical aspect is solved by the well-established LEGO bricks system.

NXT is a platform with an open specification, since LEGO released virtually all of the documentation, including the brick firmware and protocols description. When it comes to programming, LEGO offers a visual programming environment, that allows for algorithm synthesis using simple flowchart-like visual

^{*} The paper is supported by the Hekate Project funded from 2007–2009 resources for science as a research project.

language. Thanks to the openness of the platform number of open programming solutions emerged, for languages such as C and C++, Java, etc.

While numerous programming solutions exist, they fail to provide a clean high-level *declarative* programming solution for NXT. Programming robots, especially mobile ones, is a complex task, involving some typical AI problems, such as knowledge representation and processing, planning, etc. These areas are much more accessible with the use of a declarative programming solutions, compared to classic, low-level imperative languages.

The paper presents research developed within the *HeKatE* project (hecate.ia.agh.edu.pl) aimed at providing a high-level rule-based programming solution for Mindstorms NXT, based on the Prolog language API for the NXT platform. In the paper the requirements for the API are given in Sect. 3. The design of the API is discussed in Sect. 4, with a prototype implementation introduced in Sect. 5. A simple example using this API is presented in Sect. 6 Future work is proposed in Sect. 7.

2 NXT Programming Approaches

Since NXT is a well established open platform, number of programming solutions exist. From the runtime point of view, these solutions can be categorized into solutions that: communicate with the Brick using the LEGO protocol [5], provide a higher level language that compiles to Brick bytecode [6], replace the Brick firmware with a custom one.

The first approach is a simple, clean and straightforward one. The examples of the first group include LeJOS iCommand <http://lejos.sourceforge.net>, or NXT++ <http://nxtpp.sourceforge.net/>. The second approach requires a dedicated compiler, which makes it more complicated. In the second group there exists number of solutions including NQC (<http://bricxcc.sourceforge.net/nqc>). The third solution is the most complicated one, since it requires developing a dedicated embedded operating system. This type of solution is provided by the Java-based LeJOS <http://lejos.sourceforge.net>.

Another flexible approach to robot programming is to use a high-level *declarative* language such as Prolog instead of low-level C-like, or Java-based programming. Some early attempts are within the LegoLog Project (<http://www.cs.toronto.edu/cogrobo/LegoLog>) Unfortunately the project did not offer a general API, and supported only the older Mindstorms RCX version.

Besides basic programming languages, NXT robot programming can be supported on a higher logical level, offering a *visual logic representation*. The prime example is the default LEGO environment. Similar solutions are provided by LabView-based environment, as well as Microsoft Robotic Studio. In these cases the control logic is represented with use of flowcharts representing the control algorithm. However, this is mainly a procedural representation.

A step further in this direction is to use more advance *knowledge representation* methods form the classic AI, such as the decision rules, and/or decision trees. Within the HeKatE Project (see hecate.ia.agh.edu.pl), an advanced

knowledge representation method for forward-chaining rule-based systems is being developed. The eXtended Tabular Trees concept offers a generic rule-based visual programming solution, combining the power of decision tables and decision trees, see [7]. XTT is implemented with the use of a Prolog-based inference engine. Providing a Prolog-based API for Mindstorms NXT would allow to develop control logic for NXT robots with the use of the XTT method. The requirements for such a platform are presented in the next section.

3 New Platform Requirements

Basing on the review of existing solutions presented above, the requirements of a new Prolog API for NXT has been formulated. The main requirements are:

- support for all functions of the standard NXT components, that is sensors and motors,
- crossplatform solution, for both Windows and GNU/Linux environments,
- reuse some of the available solutions, and provide compatibility where possible,
- ultimately integrate with the higher XTT layer.

The whole platform based on this new Prolog API is planned as presented in Fig. 1. In the figure the highest visual rule-based logic design with XTT is also included. The focus of this paper is the design and prototype of the Prolog API layer, indicated in the figure by the dotted line.

In order to reuse and support some of the existing solutions, as well as provide implementation flexibility, it's been decided to provide in the first stage a Prolog library, with the following features: it is executed on a PC, controlling an NXT-based robot, the control is performed with the use of the Bluetooth or USB cable connection, the low-level communication is provided by some well-tested existing communications modules, at the functional level the API is coherent with other available solutions. A more detailed design of this API is presented next.

4 Prolog NXT API Design

Considering the requirements the following API architecture has been designed. It is composed of three main layers as observed in the Fig. 1.

communication layer providing the low-level communication with the robot,
sensomotoric layer allowing the exchange information with sensors and motors,

behavioral layer providing a higher-level functions, e.g. drive.

The behavioral layer exposes to the programmer some high level functions and services. It provides abstract robot control functions, such as *go*, or *turn*. Ultimately a full navigation support for different robots is to be provided. (However, different robot configurations require different control logic)

The sensomotoric layer controls the components of the Mindstorms NXT set motors, all the sensors, as well as Brick functions. This layer can be used to directly read the sensors, as well as program the motors. This is a layer, that can be used by a programmer to enhance high-level behavioral functions.

The goal of the communication layer is to execute the actions of the sensomotoric layer and communicate with the NXT Brick. Currently in this layer several modules are present, providing different means of communication:

- a pure Prolog module, using a serial port communication, and the NXT protocol commands,
- a hybrid solution based on the Java-based iCommand library,
- a hybrid socket-based solution, using the NXT++ library, that communicates with the robot.

All of these actually wrap the *Mindstorms NXT Communication Protocol* [5].

The first solution is the most straight forward one. In this case standard ISO Prolog stream predicates can be used to control the serial port. Prolog terms that wrap the NXT protocol have to be provided. In the second case the Prolog communication module is integrated with iCommand with the use of the SWI Java to Prolog interface called JPL. Prolog calls are mapped to the iCommand methods. In the third case, a simple server written in C++ exposes NXT communication with a TCP socket. The Prolog communication module connects to the server and controls the robot through a TCP connection. This opens up a possibility of a *remote* control, where the controlling logic is run on another machine, or even machines.

Besides some basic send/receive functions the library has to provide certain services. These are event and time-based callback. For example, it should be possible to instruct the robot to e.g. “drive till you hit/approach an obstacle, then make a sound, wait for a response for some time, if you don’t get one, turn and drive on” So the library has to provide *timers* that trigger some callbacks, as well as *event-driven* callbacks. This requires *parallel* execution of certain threads.

5 Prototype Implementation

Currently a prototype implementation of the API is available. The prototype includes all three layers as shown in Fig. 1. Currently in the highest layer the movement functions are implemented. The mid layer provides full control over robot’s sensors and motors. It also exposes timer and event services. In the low-level communication layer, the iCommand, DirectSerial, and NXT++ communication modules are implemented. Current implementation for the SWI-Prolog environment (www.swi-prolog.org) has been provided by Masters students: Piotr Hołownia [8], with help of Paweł Gutowski [9] and Marcin Ziolkowski [10]. For the full information see https://ai.ia.agh.edu.pl/wiki/mindstorms:nxt_prolog_api.

Below some excerpts of Prolog code for the highest layer are presented, including selected setup and movement predicates.

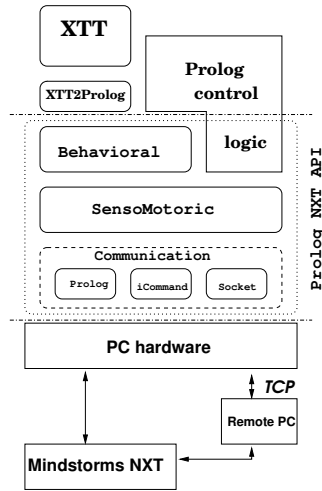


Fig. 1. Prolog NXT API Design

```

nxt_set_robot(WheelCircumference, AxleLenght, LeftMotor, RightMotor,
Reverse, TouchPort, SoundPort, LightPort, UltrasonicPort) :-
nonvar(WheelCircumference), nonvar(AxleLenght),
nonvar(LeftMotor), nonvar(RightMotor), nonvar(Reverse),
retractall(nxt_robot(_,_,_,_,_,_,_,_,_)),
assert(nxt_robot(WheelCircumference, AxleLenght, LeftMotor, RightMotor,
Reverse, TouchPort, SoundPort, LightPort, UltrasonicPort)).
nxt_stop :-
nxt_robot(_,_,_LM, RM, _,_,_,_), nxt_motor(LM, 0), nxt_motor(RM, 0).
nxt_go(Speed) :-
nxt_robot(_,_,_,_,_ TouchP, _,_,_),
nxt_go(Speed, force),
trigger_create(_ ,nxt_sensomoto:nxt_touch_sensor(TouchP, 1),nxt_stop).
nxt_go(Speed, force) :-
Speed \= 0, nxt_robot(_,_,_LM, RM, _,_,_,_),
nxt_motor(LM, Speed), nxt_motor(RM, Speed).
nxt_go(Speed, Angle) :-
nxt_robot(_,_,_,_,_ TouchP, _,_,_), nxt_go(Speed, Angle, force),
trigger_create(_ ,nxt_sensomoto:nxt_touch_sensor(TouchP, 1),nxt_stop).
nxt_go(Speed, Angle, force) :-
nxt_robot(_,_,_LM, RM, _,_,_,_),
nxt_motor(LM, Speed, Angle), nxt_motor(RM, Speed, Angle).
nxt_go_cm(Speed, Distance) :-
nxt_robot(WC, _,_,_,_,_,_,_), Angle is round(Distance/WC*360),
nxt_go(Speed, Angle).
nxt_go_cm(Speed, Distance, force) :-
nxt_robot(WC, _,_,_,_,_,_,_), Angle is round(Distance/WC*360),
nxt_go(Speed, Angle, force).
nxt_turn(Speed, Angle) :-

```

```

    Angle > 0, nxt_robot(_,_,LM,RM,_,_,_,_,_),
    nxt_motor(LM,-Speed,Angle), nxt_motor(RM,Speed,Angle).
nxt_turn(Speed,Angle) :-
    Angle < 0, nxt_robot(_,_,LM,RM,_,_,_,_,_),
    MinusAngle is -Angle,
    nxt_motor(LM,Speed,MinusAngle), nxt_motor(RM,-Speed,MinusAngle).

```

The declarative Prolog syntax simplifies the design of the control logic.

In the main sensomotoric layer Prolog predicates for reading the sensors, as well as controlling the motors are provided, as shown below.

```

nxt_motor(_,Speed) :-
    nonvar(Speed), Speed > 900, writeln('Rotational speed is to high!').
nxt_motor(_,Speed) :-
    nonvar(Speed), Speed < -900, writeln('Rotational speed is to high!').
nxt_motor(Motor,Speed) :-
    nonvar(Motor),nonvar(Speed), Speed > 0, nxt_actions_motor_forward(Motor,Speed).
nxt_motor(Motor,Speed) :-
    nonvar(Motor),nonvar(Speed), Speed < 0,
    BackwardSpeed is -Speed, nxt_actions_motor_backward(Motor,BackwardSpeed).
nxt_motor(Motor,Speed) :-
    nonvar(Motor),var(Speed), nxt_actions_motor_get_speed(Motor,Speed).
nxt_motor(Motor,0) :-
    nonvar(Motor), nxt_actions_motor_stop(Motor).
nxt_light_sensor(Port,Value) :-
    nonvar(Port), nxt_actions_light_sensor(Port,Value).
nxt_light_sensor_LED(Port,Setting) :-
    nonvar(Port), nxt_actions_light_sensor_LED(Port,Setting).

```

Using a single predicate `nxt_motor/2` it possible to *both* set and get the speed of the motor. Another predicate `nxt_motor/3` supports rotating the motor at the given speed until a specified angle is reached or for the given time.

The event and time-based triggers are provided with the main predicates: `trigger_create/3` which allows for creating an event trigger, and `timer_create/3` that creates a time-driven trigger, both can run any Prolog predicate. They use SWI multithreading capabilities allowing for soft-real time control.

The lowest level communication protocol provides an iCommand-based control module, that implements get/set commands from the middle layer. It is implemented using SWI Prolog Java interface called *JPL*.

```

nxt_actions_motor_forward(Motor,Speed) :-
    jpl_get('icommand.nxt.Motor',Motor,MotorHandle),
    jpl_call(MotorHandle,'setSpeed',[Speed],_),
    jpl_call(MotorHandle,'forward',[],_).
nxt_actions_motor_backward(Motor,Speed) :-
    jpl_get('icommand.nxt.Motor',Motor,MotorHandle),
    jpl_call(MotorHandle,'setSpeed',[Speed],_),
    jpl_call(MotorHandle,'backward',[],_).
nxt_actions_motor_get_speed(Motor,Speed) :-
    jpl_get('icommand.nxt.Motor',Motor,MotorHandle),

```

```

    jpl_call(MotorHandle, 'getSpeed', [], Speed).
nxt_actions_motor_stop(Motor) :-
    jpl_get('icommand.nxt.Motor', Motor, MotorHandle),
    jpl_call(MotorHandle, 'stop', [], _),
    jpl_call(MotorHandle, 'setSpeed', [0], _).
nxt_actions_light_sensor(Port, Value) :-
    not(nxt_actions_light_sensor_connected(Port, _)),
    retractall(nxt_actions_light_sensor_connected(_, _)),
    jpl_get('icommand.nxt.SensorPort', Port, PortHandle),
    jpl_new('icommand.nxt.LightSensor', [PortHandle], SensorHandle),
    assert(nxt_actions_light_sensor_connected(Port, SensorHandle)),
    nxt_actions_light_sensor(_, Value).
nxt_actions_light_sensor(_, Value) :-
    nxt_actions_light_sensor_connected(_, SensorHandle),
    jpl_call(SensorHandle, 'getLightPercent', [], Percent),
    Value is Percent.

```

This solution works flawlessly in both GNU/Linux and Windows environments. A purely Prolog-based module for serial connection under GNU/Linux is also being finalized. This module will be ported to the Windows environment.

Another module wraps the NXT++ (<http://nxtpp.sourceforge.net>) exposing the robot control through a TCP socket. The low level communication module connects to the robot using a network connection, and a simple text-based protocol, representing the original LEGO Protocol [5]. This approach allows for a *remote* control over a robot, where the controlling host is not directly connected to the robot. This opens up some flexible control possibilities in a distributed environment. It is unfortunately out of scope of this paper.

The Prolog API has been successfully tested on number of simple control algorithms. A selected algorithm is discussed in the following section.

6 Example Algorithm

Below a simple obstacle avoidance algorithm using the API is presented.

```

state(go_foward). % state(go_step_foward).
start(Thresold) :- nxt_ultrasonic_sensor(Value), Value > Thresold, start_go.
start(Thresold) :- nxt_ultrasonic_sensor(Value), Value =< Thresold, start_turn.
start(_) :- go(0,0,0).
start_go :- state(go_foward), nxt_go(0,speed,force).
start_go :- state(go_step_foward), nxt_go_cm(distance,speed).
start_turn :- state(go_foward), nxt_turn_degrees(90,speed).
start_turn :- state(go_step_foward), nxt_turn_degrees(-90,speed).

```

7 Future Work

The original contribution of this paper is the pure Prolog API for controlling the LEGO Mindstorms NXT robot platform. It can be used as a generic solution

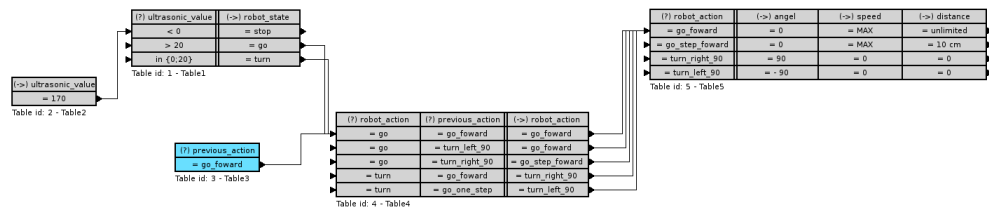


Fig. 2. Rule-based XTT representation of the control algorithm

for programming the NXT in Prolog. It also serves as a foundation for a higher-level visual rule-based programming within the XTT knowledge representation. While this is a ready solution, there are some directions for future work.

Extending the behavioral layer is an important direction. The current implementation is mainly focused on simple mobile robots. A robot configuration abstraction layer is in the works. It would allow to define different types of robots.

An ultimate goal is to provide an integration layer with the visual algorithm design with the XTT knowledge representation. Such a design of the example discussed in Sect. 6 is shown in Fig. 2. Since the XTT interpreter is built with Prolog, this is just a matter of time and experiments. This would provide a complete open robot design environment based on the rule-based programming.

References

1. Holland, J.M.: Designing Mobile Autonomous Robots. Elsevier (2004)
2. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd edn. Prentice-Hall (2003)
3. Craig, J.J.: Introduction to Robotics: Mechanics and Control. 3rd edn. Prentice Hall (2004)
4. Mario Ferrari, Giulio Ferrari, D.A.: Building Robots with LEGO Mindstorms NXT. Syngress (2007)
5. The LEGO Group: LEGO MINDSTORMS NXT Communication Protocol. (2006)
6. The LEGO Group: LEGO MINDSTORMS NXT Executable File Specification. (2006)
7. Nalepa, G.J., Ligeza, A.: A graphical tabular model for rule-based logic programming and verification. *Systems Science* **31**(2) (2005) 89–95
8. Hołownia, P.: Mindstorms nxt prolog api prototype. Knowledge Engineering Project (MIW), G. J. Nalepa supervisor (2008)
9. Gutowski, P.: Mindstorms nxt prolog api prototype, icommand module. Knowledge Engineering Project (MIW), G. J. Nalepa supervisor (2008)
10. Ziołkowski, M.: Mindstorms nxt prolog api prototype, socket module. Knowledge Engineering Project (MIW), G. J. Nalepa supervisor (2008)