DESIGN AND IMPLEMENTATION OF AN INTUITIVE DOMAIN SPECIFIC LANGUAGE FOR A BIONIC-ORIENTED EMBEDDED SIX LEGGED ROBOT

Franz Wiesinger, Daniel Hölzl and Michael Bogner
Embedded Systems Design
University of Applied Sciences Upper Austria
Softwarepark 11
4232 Hagenberg, AUSTRIA
email: franz.wiesinger@fh-hagenberg.at, daniel.hoelzl@fh-hagenberg.at, michael.bogner@fh-hagenberg.at

ABSTRACT
This paper demonstrates the development of a domain specific language for a bionic-oriented six legged walking robot. It shows how the complex application domain of robotics can be simplified to enable unskilled users to model motion sequences for a given moveable robot, by keeping the option for sophisticated movements. For the development of the language the Coco/R parser generator was used. To further simplify the workflow, additionally a development environment was implemented based on the Eclipse Rich Client Platform and the Xtext framework.

KEY WORDS
Domain specific language, software language engineering, bionics, robotics, Eclipse Rich Client Platform, Xtext

1 Introduction

With the progress of the digital age the demand for skilled workers with profound knowledge in software development is increasing steadily. For companies which focus their business field outside the information technology this circumstance causes difficulties. Either they must fund expensive further training of their employees or they have to purchase knowledge from external providers. A lot of the given challenges, like machine control are limited to specific application areas. Instead of complex high level languages like C++ or Java, the challenges could be solved with easy-to-learn domain specific languages [2].

The objective of this paper is to show a case study of a simple language and a supportive development environment. The application domain of the new language is robotics, where the focus lies on a six legged walking robot, developed inhouse at the University of Applied Sciences Hagenberg. The created language should enable unskilled users to model complex motion sequences for the moveable robot.

2 Related Work

In [5] Frigerio et al. present their realization of a domain specific language to describe the kinematics model of a robot. Their work points out, that the usage of DSLs allows the developer to deal with the high level description of the robot and further efforts can be focused on open research questions.

Fischertechnik is a German company producing construction sets for children. These are used in education for teaching about machines, motorization and mechanisms. One of their products is the Robot TX, which can be programmed with the proprietary tool ROBO Pro. A further similar educational robot project is the Lego Mindstorms NXT kit.

3 Criteria of the Created Language

What requirements did we impose on this new language? First of all, the main criteria is the simplicity of the language. All commands should be intuitive and should represent the real situation like in nature. An unskilled user only can interpret the meaning of commands, that trigger immediately reactions on the robot. The language should be usable without knowledge in software engineering.

A further criteria of the language is the scope of functionality. All features of the target platform have to be covered. No functionalities should get lost through simplification.

The target platform is a six legged walking robot, manufactured inhouse at the institute. Only a limited amount of memory and computational power are available on the embedded device. This has to be considered during the design of the language.

Another aspect that has to be considered is the abstraction level. The firmware of the walking robot provides mainly low level commands. As an example there are commands to change the angle of a servo. Theoretically, all motions of the robot could be realized with these simple commands. Here is a higher level of abstraction necessary and commands like MoveLeg or Rotate should be available.

A simple language requires also an easy way to make use of it. A corresponding development environment, which allows easy handling of the language and good functionalities to transfer the program to the target platform is needed.
Further the language should fulfill industrial standards. The development of the language should be done with approved tools, which lead to an adequate degree of stability and safety.

4 Target Platform

The target platform is an inhouse developed six legged walking robot. The robot is powered by 18 model aircraft servo motors. So there are three servos on each leg, which means three degrees of freedom for each leg. The skeleton of the robot is realized with ply wood. Excluding the costs of the development, the robot is a low budget product, with expenses of about 250 Euros.

The self developed hardware platform called Sandbox is responsible for the management of the servos. It consists of a field-programmable gate array (FPGA) in combination with a micro controller. On the FPGA is the so called Robotics Chip implemented. It generates the pulse-width modulation (PWM) signals for the regulation of the servo motors and contains an implementation of the cordic algorithm, which is an efficient iterative algorithm to calculate trigonometric, exponential or logarithmic functions. Especially for real time applications and embedded devices, like the walking robot, this ability is important. The additive 8-bit micro controller has 128 kilobytes flash memory and can be programmed via USB in the language C. Its primary purpose is the communication with the Robotics Chip. Figure 1 shows a design drawing of the walking robot.

5 Design of the Language to Meet the Criteria

The implemented language tries to copy the movement abilities of insects [1]. All insects have in common that they use six legs to move. This characteristic is also applicable to our robot. For easier identification the legs on the robot are numbered.

5.1 Physical Construction of the Robot

To be able to explain the language, first of all the physical construction of the robot should be elaborated.

Each leg of the robot consists of three elements. Following the structure of an insect, these elements are named Coxa, Femur and Tibia. The body of the robot is called Corpus. In figure 2 a leg of the robot is shown. Coxa, known as hip, is the element, that is directly mounted to the body. With the first servo this element can be moved in the horizontal plane in an angle of about 100 degrees. The second element of the leg is called Femur, also known as shank. It is the first of the two vertically aligned leg elements. The last element has the name Tibia (shin) and is like Femur moveable in the vertical plane. By combining all three leg elements and the usage of inverse kinematics a free selectable position in space can be reached. The point of origin is located at the beginning of the first leg element and the tool center point is located at the end of the last element. Due to mechanical restrictions not all positions in space can be reached.

By combining the movement of all six legs both translational and rotatory movements are possible. The rotatory movements in detail are Roll, Pitch and Yaw. Roll means tilting the robot by the longitudinal axis. Pitch is the forward and backward movement around the transversal axis. A rotation around the vertical axis is called Yaw.

5.2 Abstraction Level and Resulting Commands

The user should never get to know about the complexity of the kinematic calculations.

In actual fact, all movements can be reduced to a handful of commands. At first all leg elements have to be controllable individually. The commands to accomplish this task are called MoveCoxa, MoveFemur and MoveTibia. Important when designing these commands is not choosing abstract constructs. Instead all commands should represent the reality and should trigger immediately reactions.

The commands for single leg elements are based on angle values. Further it should be possible to move to a free selectable position. Therefore the command MoveLeg exists. It is based on position values in the Cartesian coordinate system. The required kinematic calculations to transform coordinates to angle values become abstracted.

In the next abstraction level, also the Cartesian coordinates of single legs get abstracted. The command Walk
allows the robot to move straight ahead and the command Rotate causes a rotation around the vertical axis.

Additionally the Corpus of the robot can be positioned individually. The command CorpusLift causes a horizontal change of position. With the command CorpusPitch a tilting around the transversal axis can be achieved. CorpusRoll finally tilts the robot around the longitudinal axis.

To be able to put fix temporal breaks during a motion sequence the command WaitOnTime can be used. Manual controlling of the robot can be carried out with keys mounted to the controller board. Therefore the command WaitOnKey exists.

5.3 Grammatical Structure and Further Language Features

All these theoretically considered commands have to follow an accurately defined grammar. The language fulfills the simplified EBNF grammar listed in table 1.

A motion sequence is defined in a program. It consists of an optional constants declaration block and the main block, which contains the sequentially processed commands. Constants can be used to define repetitive values or to set meaningful names for individual parameters, for example a certain angle position.

A command in the language can be attributed with properties. If the property is not used, its default value will be used. The possible properties depend on the command. Pure servo commands only have properties about the desired angle position. The command MoveLeg on the other hand has the properties X, Y and Z for the position in space.

Properties of a command can be assigned with integer values, with constant values declared in the constants declaration block, or with predefined values like Min or Max.

5.4 Example of the New Language

Table 1 shows a short example movement programmed in the new developed language and the resulting C code. The movement starts with a key press on the robot. First, the robot moves the Tibia element of the second leg to a certain angle. After a fixed time span the height position of the body is changed. Finally the first leg is moved to a certain position. The property Legnumber has the default value one and is therefore not specified in the code.

At the first view the generated C code does not seem to be more complicated compared to our language. But considering persons without software development skills there is a difference. Without software development knowledge they don’t know anything about interrupts, structures or even identifiers. Our main goal was a simple language. Therefore, we avoided complex concepts like object-orientation or polymorphism. Furthermore, the user does not need to be worried about the cordic algorithm and the kinematic calculations.

6 Overall Workflow - From the Language Model to the Binary Code

Figure 3 shows the complete overview from the domain specific language to the finally executable binary code for the target platform.

![Figure 3. Overall workflow from the model to the binary code](image)

Starting position at first is an attributed EBNF (Extended Backus Naur Form) file. It defines the grammar of the language and is required as input for the Coco/R parser generator. Coco/R [3] has been developed at the Johannes Kepler University Linz and is available under the GNU General Public License. Coco/R uses the EBNF to generate a scanner and a parser, which will be used as the heart of the compiler. By the definition of additional attributes for terminal tokens in the EBNF, the transformation into the semantic model is made easier. Together with the code generator, which translates the semantic model into the C code, the parser, the scanner and the semantic model form the compiler.

The compiler now can be called with source code in the new language. If the code is valid, C code gets generated. This code is precisely tuned for the target platform. Finally, the code can be translated to binary code for our target platform with the AVR compiler. Additionally, the Robotics Library for the usage of the Robotics Chip will be linked. It contains the low level commands of the robots firmware.

By the usage of free available, industrial approved tools, like the Coco/R parser generator and the WinAVR compiler, a general stability of the results can be guaranteed. Because these tools exist since years and in this way are subject to continuous improvement, a useful result with a minimal error rate can be expected. From an economic viewpoint, the reusage of existing tools leads to considerable time saving.
7 Integrated Development Environment

The IDE is based on the Eclipse Rich Client Platform [6], which allows the development of plugin based, modular desktop applications. Especially, the reusability and extensibility of existing program parts is a remarkable feature of the Eclipse Rich Client Platform.

An advantage of the plugin based architecture is, that the contents actually get loaded if needed. As a result the launching time of RCP applications can be shortened and the whole application is more scaleable.

A significant help of the development process was the framework Xtext [4]. It is an open source framework for developing domain specific languages maintained by the company Itemis and part of the Eclipse Modelling Framework. Internally the framework uses the ANTLR parser generator. Xtext allows to generate plugins for RCP applications. One of these plugins provides a text editor part, which includes among others the features code completion, syntax highlighting, dynamic validation and automatic code formatting for our domain specific language. Xtext generates these functionalities partially ready to use or in code stubs which have to be adapted to our needs.

7.1 Assistance for the User during the Developing Cycle

The IDE supports the developer during the whole development cycle, starting from the creation of a new project, till the download of the completed program onto the target platform.

7.1.1 Individual Validation

For unexperienced users error recognition is especially important. With immediate feedback still during typing, the user gets informed about incorrect input as soon as possible. The error recognition does not only detect syntax errors. Additionally semantic constraints are validated. This is possible, since the constraints of the application domain are known. This includes checking of angle values or coordinates against the mechanical restrictions. As the error recognition takes place immediately, the user could learn
from his mistakes and avoid them in future. For the implementation of the individual validation code injection with the Google Guice Framework was used, to be able to adapt the generated UI plugin.

7.1.2 Code Completion

Although the language has only a few commands, not all commands and corresponding properties can be memorized entirely. With code completion this is not required any more. The parser in the Xtext framework knows what syntactical rules currently have to be fulfilled. Thus, it is possible to suggest only those inputs, which are currently allowed. For easier handling of parameters, for example the desired X position in the coordinate system, the language provides the predefined values \textit{Min}, \textit{Max} and \textit{Home}. So the user does not need to know the limits or default values.

7.1.3 Syntax Highlighting

Basis of the syntax highlighting is the semantic model of the language, which was generated with the help of the Xtext framework. It is updated on every text input. Every semantic object in the model has a corresponding position in the source file. The coloring was realized with a presentation reconciler provided by the Xtext framework.

8 Usage of the Development Cycle

Figure 5 shows the usage of the development cycle. After writing the source code, supported by the IDE, it can be compiled to binary code for the target platform. Via USB the final program can be downloaded to the controller board of the walking robot.

All steps can be executed within the integrated development environment. For the download process to the robot, the free tool Atmel Flip was integrated in the development environment.

9 Conclusion and Outlook

This work points out the advantages of the usage of domain specific language for the controlling of mobile walking robots. With adequate abstraction it was possible to reduce the complexity of the application domain. As a result our developed language enables users without knowledge in software development to model complex motion sequences for a six legged walking robot. The demonstrating language is based on the movement of insects. Therefore the language is intuitive and easy to learn. Further the language covers all relevant functionalities of the target platform. By the usage of free available, industrial approved tools, like the Coco/R parser generator, it was possible to reach the objective of an easy language in a shorter timeframe than expected. The development of a supportive IDE completes the result. It was implemented with the help of the Xtext framework. For unexperienced users the features of the IDE i.a. syntax highlighting, code completion or automatic validation are a necessary support. Especially the implementation of the automatic source code validation points out the advantage of the usage of domain specific languages. As the constraints of the target platform are known, it is possible to check semantic restrictions additionally to the syntactic rules. This semantic restrictions are for example limited angle values or boundaries in the coordinate system.

During the development of the language it became apparent that the communication with the robot is currently quite onesided. Except the keys, mounted to the controller board, the robot has no feedback possibilities for the developer. It is not possible to query the current angle position of a servo motor. Neither there are sensors to capture the environment of the robot. However, the developed language was designed to be expandable with little effort.

References