Off-Line Robot Programming Framework

Vitor Santos Bottazzi
Industrial Electronic Engineering Department
University of Minho
Braga, Portugal 4710-057
Email: bottazzi@dei.uminho.pt
..
Jaime Francisco Cruz Fonseca
Industrial Electronic Engineering Department
University of Minho
Braga, Portugal 4710-057
Email: jaime.fonseca@dei.uminho.pt
Telephone: (351) 253 510193
Fax: (351) 253 510189

Abstract— **The industrial robot programming is a work for specialist in robotics. Today, this work is very hard because there are many robot manufacturers with different languages and different programming environments. Although, off-line programming is an way that can reduce drastically the machines stop time to maintenance.**
With the use of object oriented design patterns, it is possible minimize the time spent in robot programming. In this work is proposed a off-line programming environment. This tool is based in one abstract model to program robots, encapsulate in java classes. This way has the main advantage of best source code utilization. Grouping the business classes in modules by functionalities, we can reduce complexity between low matching. Recognized patterns like Facade and Template Method will construct the base to develop this programming framework. The programming robot languages tested in this work was Rapid, Karel and Melfa Basic IV, respectively languages used by ABB, Fanuc and Mitsubishi constructors.

Index Terms— **Framework, off-line programming, robot code generator.**

I. INTRODUCTION

Quality and efficiency are characteristics that influence world-wide economy. After the Total Management Quality(TMQ)(Patterson, 1995) and other quality patterns appearance, product cost reduction is the main topic discussed by the employs. The employers technologic level adopted defines the competitive difference between businesses, and increase concurrence. So, tools development that makes easy robot programming and may reduce the manufacture cell integration process costs can be an interesting solution. Actually many employs are working in the robotic segment and the robot programmer have to flexibly programming over many different robots interfaces in many different robot languages. The teach pendant(TP), a joystick plugged in the robot controller, is one of these used interfaces for robot programming. This joysticks has layout, weight, buttons and command sequences completely different between manufacturers. This interaction become too difficult for professionals that do not have an affinity with robotics. Knowledge about how to welding, paint or manipulate industrial tools are also very important to automate the task. Beyond the robot common knowledge, the programmer need to know the automatized process bringing real profits on product quality, equipment durability, and decrease manufacture time. Using a text editor or some specialized robot programming tool, an welder may program a robotized welding task optimizing significantly the process. But commonly, it programs are made manually by TP, spending a lot of time and requiring available equipment to do it. During this process, the programmer using teaching techniques, have to be able to program a motion sequences those results in a specific task. The main steps to program an articulated arm are described by (Fuller, 1999): First is defined the problem. After that, outlines the steps of an solution. Next express the steps in the computer robot language, then enter the steps as instructions into the robot controller, and finally check out the resulting solution. Specifically enter the steps as instructions into the robot controller is described in the picture 1.

The off-line programming method was created to diminish the integration cell time. But actually, the off-line programming has not brought significative gains to the manufacture cell integration, also to reduce the robot programmer worked hours. The assembly and program manufacture cell tools were projected without the necessary abstraction, to generalize the robot programming problem. The available tools presents in the robot kits, can program and interact only with it platform, files and libraries. The demand grows by an unified tool that interact between different manufacturers solutions, turning easy robot programming to the companies.

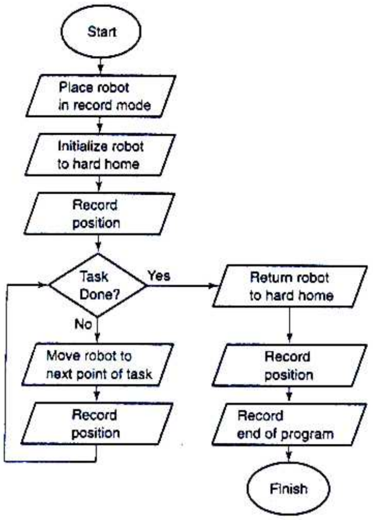

Start
Place robot in record mode
Initialize robot to hard home
Record position
Task Done?
Yes
No
Return robot to hard home
Move robot to next point of task
Record position
Record position
Record end of program
Finish

Fig. 1. (Fuller, 1999) Point-to-Point Robot Programming Process.

Actually, the personal computer(PC) is able to manipulate strings and solve matrix operations. We propose to use this intrinsic PCs

Proceedings of the Joint International Conference on Autonomic and Autonomous Systems
and International Conference on Networking and Services (ICAS/ICNS 2005)
0-7695-2450-8/05 $20.00 © 2005 IEEE

features, through an software implementation that knows the robot programming languages. Respecting the working area(Murray, 1994) and using an communication protocol to interact with the robot controller is possible develop off-line robot programming.

Sensors technology associated with the power processing of PCs is bringing new possibilities to industrial applications. Obstacle detection and computer vision are applications examples that are being tested and improved every year (Fuller, 1999). The interface proposed process the input data in the system and generate an specific command sequence to the robot, that will reproduce the idealized motion by the user. The target activity: weld, assembly, inspection or another one, can be programmed by the welder, for example. He will program a motion sequence described by the manipulator through software, using a common PC. The emphasis of the present work, is develop a code generator to different robot platforms, starting an research line on off-line programming area. This research intends to demonstrate that the manufacturing cell integration can be accelerated, the communication between different platforms of robots can be optimized and costs with specialized people can be reduced. Actually, the robot programming is still a hard work (Wrn, 1998). Some causes are: difficulty of available equipment reserve to improve it, complex handling, and technologic approach demanded to learn it.

The main target of this research is develop an tool that makes easy robot programming and learning it. A little approach of object oriented design pattern(OODP) (Helm, 1995) techniques was used in this code generator and will be discussed at section II. In section III frameworks characteristics will be presented. Section IV shows the generator code business classes dynamic. In section V are showed the experimental results. Finally, in section VI the conclusion and future works.

II. USED PROJECT TECHNIQUES

The abstraction is a very important feature when thinking about industrial machine programming. Code abstraction makes possible work with industrial equipment that is in the coordinates motion command universe. A good modelling of basic mechanisms structures like: joints, axes, programs and points can facilitate the routines implementation and strings manipulation composing the motion command that will be transmitted to the robot controller. At this section we will explain how programming techniques helps to represents these structures and organize sequentially robots control commands.

A. DESIGN PATTERNS - DP

Experience is an evaluated characteristic in every kind of business. Also in oriented object programming(OOP) is useful adopt renamed programmers successful experiments and relates, called Design Patterns.

Code reusability is one of the benefits brought by OOP. But, projecting an reusable object oriented software is a hard work. In the project phase, it is necessary partition the problem in general modules capable to be used in the future in convenient situations. During the analysis process is necessary identify class and communication models(or templates) between objects that can be reused in the object oriented system. Specific development problems are targeted by OODP bringing flexibility, elegance and reusability code. Resuming, DP is an high level mechanism that relates the best programmers practices(successful experiences) to solve software projects common problems. The DPs (Helm, 1995) used at present work was: Factory, Singleton, Facade, Memento, Command e Template Method. Next we will discuss this OOP techniques applied in the generator code development, his labels, common problems solved by this practices, it micro-architecture, and implantation consequences.

B. FACTORY

This pattern centralizes the objects creation with high changing project probability in a single class. It cam be understood like an objects factory. The demand of a class that centralize the objects creation was the motivation to idealize it. Prohibiting any other object of business class to instance directly objects. The source problem is: if an signature method changes in a specific class, it will be necessary change all of direct instances to it class, becoming hard and complex work, considering that this object is being instanced by many classes. The proposed solution was develop an centralized solution class, responsible to create and return instance references from all called objects. If some object charger method changes, in the factory class, it will be centralized in the getInstanceX() method, where ”X” is the name of a class that contain the object attributes(identity) changed. The strategy was analyze the list of classes that has an big probability to change and implements the getInstance() methods to them in the class factory.

```
  Example:
;This method returns an instance to the
"AbbPoint" object charging by an general "Point"
object, the Rapid commands to ABB robots.
method getNewAbbPoint(Point p)
{
return new AbbPoint(p);
}
```

C. SINGLETON

Is a pattern that prohibit the duplication of objects in RAM memory. Your meaning is ”unique instance”. It was idealized because during the program execution many objects are recreated without real requirement, reflecting in memory and processing overhead. The main problem attacked by it DP is the memory wastefulness caused by the sub-utilization of created objects in memory increasing the overhead caused by the garbage collector(java virtual machine service that monitories and deallocate the unreferenced objects in memory).

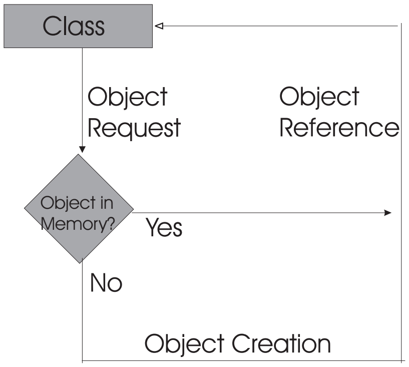


Fig. 2. Singleton pattern.

The present solution was, to verify during object creation, if it still exists in memory. The direct consequences to it practice will: decrease process overhead caused by garbage collector and increase the objects useful life, reflecting in a better memory resources allocation . The follow strategy was represented in picture 2. If the object still exists in memory is returned an reference to him, else the object is created and also is returned an pointer to it new object.

```
  Example:
;This method verify the existence of an
object that knows Karel Language used by Fanuc
Robotics. If it is, returns an reference to this
object, also create a new object and returns it
reference.
method FanucPoint getNewFanucPoint()
{
exists in memory a FanucPoint?
If yes, return a reference to it;
Else create new FanucPoint and return a pointer
to it in memory;
}
```

D. FACADE

The facade pattern decouple user interface and business classes, through a knew default data input point. The meaning of facade can be understood like ”consensus”. It was implemented to make possible connect different interfaces with the business classes, through an centralized data input, decoupling interface and business layers. That problem is: if the interface changes, also will be necessary changes the businesses classes to compatible it. This pratique suggests the creation of a class that hide interface layer system complexity and alow interaction with any kind of input, like command prompts, database querys, applets or simple applications. The direct consequences will be decrease interface interaction complexity and decouple interface from business layer. Thus all dependencies between involved entities in the use cases, will be transparent to the user. Another visible gain is, the easy software maintenance. Reflect in fast interface modifying and new interface input creation.
The strategy was implements a Charger class, that knows how to talk with the business layer and known by interfaces that want to use the service. This way will make possible change the presentation layer(interface) without big software set adjustments. The picture 3 show the functionality of this pattern. Is possible see in the example above how Charger class talks with business layer and the data inputs centralized model.

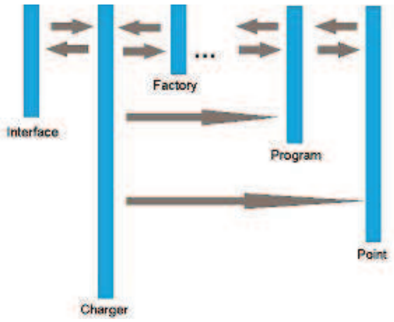


Fig. 3. Facade pattern.

```
  Example:
Charger Class method ProgramTypeCharger()
{
;request a new robot program to factory receiving
the protocol and the program name from interface.
}
method PointCharger()
{
;request new coordinate point to factory Charger
and insert the command in the program using
supplied parameters from interface.
}
method FileCharger()
{
;request the file creation from interface to
storage class setting program and protocol used.
}
```

E. MEMENTO

It is a pattern that stores the last recent memory interactions with the software, making possible recoup effected operations. Can be understood like ”Undo”. This demand was perceived when the user, seeking an program creation though line needs to undo(turn back) some interactions in the code. The problem detected is: how to store the working structures? The proposed solution was store the objects state in a list, to restore it if necessary. Thus the user can undo executed steps if it is necessary. The main objective of this pattern is creating an user-friendly interface. The strategy is create a collection that memorize the user steps, defining a structure that stores and restore the objects attributes, making it persists in memory.

```
  Example:
;Two lists are instanced in the Program class the
first is used to generate the robot program and
the second to implements the memento pattern.
method Program()
{
String idprog;
List PointsList;
List PointsListVarCoord;
}
```

F. COMMAND

This pattern generates interface command decoupling actions of it causing events. The name of this pattern can give na idea of his functionality. Command pattern is necessary because the user is forced to execute system functionalities through data input interface objects(Menu, Button, CheckBox,...). The user interface generation tools has menus and buttons objects responsible for ”commands” entered by the user. Action are responsibility of business classes. The user interface objects should not implement the explicit action, because only the business layer should know how to do it.

The solution adopted is showed in picture 4. All current interface instanced objects know only a reference to the method responsible to execute the user desired action. It decouples interface and business layer functions. The chosen strategy is, the interface objects only will know the system default input point(the Charger class implemented with Facade pattern). Thus interface objects invoked by the user, have to reference the method located in the business layer to execute

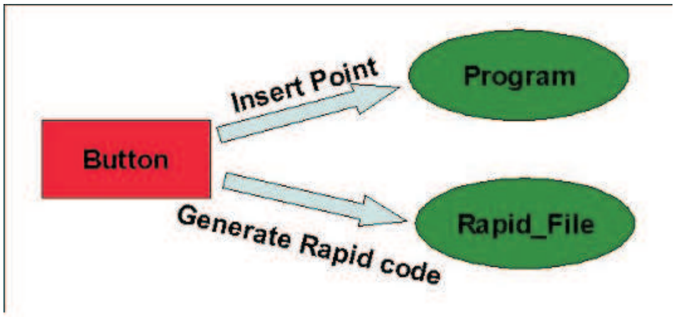


Fig. 4. Command pattern.

actions, using the common input point, the Charger class.

```
 Example:
;Interface Applet
class Applet
{
Charger charge = new Charger();
;The method ProgramGeneratorButton() delegates
through an object instance called "charge" the
responsibility, robot program file generation.
method ProgramGeneratorButton()
{
this.charge.FileCharger();
;call FileCharger() through instanced object
charge.
}
}

class Charger
{
;The Charger class is remodelled like Command
pattern receiving requisitions and distributing
competencies.
int prot;
List prog;
Factory f = new Factory();
;Method FileCharger() calls the responsible robot
program file creator object setting the generic
program and the desired protocol.
method FileCharger()
{
RapidFile createfile = f.getFileInstanceRapid();
createfile.FileCreator(this.prog,this.prot);
}
...
}
```

G. *TEMPLATE METHOD-TM*

This pattern helps to define an skeleton algorithm. This skeleton will be used to specialize subclasses that inherit the object abstract common model.

It allows an father class refinement from child classes realizing it reality. This technique consists in create a template to be specialized by child classes. It classes will materialize its behavior over charging father class methods through polymorphism, showed in picture 5. The TM direct consequence is increase code reuse. Although reflects in couple caused by the implemented inheritance between the abstract and child classes. This practice allied to dynamic binding are the bases to a framework building. A little introduction of frameworks will presented in the next section. An TM example is showed below.

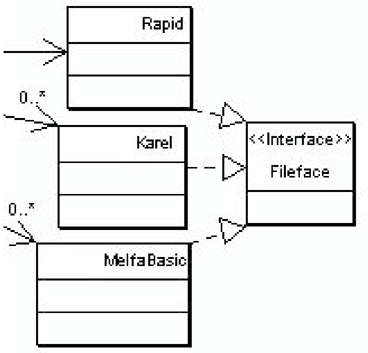


Fig. 5. Template Method pattern.

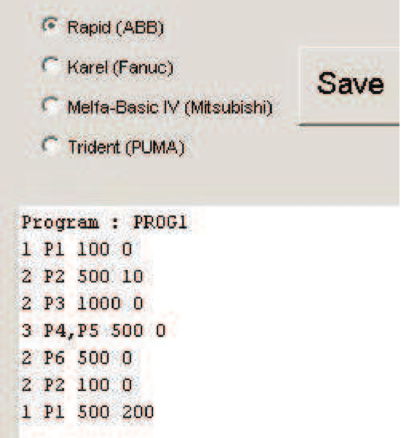
```
 Example:

 ;Abstract Class
abstract class TemplateFile
{
abstract method ProgramHeader ();
abstract method ProgBodyVarsDeclaration();
abstract method CloseProgram();
method FileCreator()
{
;create a robot program and return a report
message.
}
}

 ;Concrete Class
class KarelFiles extends TemplateFile
{
method PointsCoordVarsReturn()
{
;while exists variable names in list, write
(varName+":POS(0,0,0,0,0,0’’);");
}
method ProgramHeader()
{
write("PROGRAM "+ this.nome);
write("-- ! LANGUAGE KAREL 2");
write("-- ! MEMORY 8192");
write("-- !ROBOT ARC MATE 120iB");
write("-- TEACHPOINT DECLARATIONS");
}
method ProgBodyVarsDeclaration();
{
;Var declaration
write("VAR");
;insert coordinates and quadrants seetings using
PointsCoordVarsReturn();
;Main program
write("BEGIN");
;insert motion command lines;
}
method CloseProgram()
{
;insert the end instruction into robot program;
}
}
```

III. FRAMEWORK

Is an work layer where all knowledge about the predefined activity is encapsulated. Some characteristics of OO like dynamic binding, polymorphism, and inheritance make easy structures modelling it. These techniques were widely used to reusable, abstract, and specialized object creation. Make objects interact and communicate dynamically is the most important thing in the framework building. The framework concept is so important to develop the robot code generator presented in this research, because it has to generate many different specific robot languages. Sometimes to implant the framework model is necessary redesign all the application to support its powerful interaction. Wherefore, the OODP and framework documentation is so important, prevents completely remodel the projected system if appear some located change demand.

IV. CODE GENERATOR FOR THE ROBOTS

Quick Teach(ABB), WinOLPC(Fanuc), and Cosirop(Mitsubishi) are some of off-line programming simulators available in industry. This simulators are able to command its respective robots, using his proprietary robot programming languages, without worry in interaction between this platforms. In this work, the programs structures composed in the code generator were builded using the basic robot motion commands found at ABB, Mitsubishi, and Fanuc technical manuals. Using robot programming common parameters like: Motion type, Point Name, Manipulator position coordinates, displacement velocity, and precision target. This knowledge is represented into special classes and validating it with the convenient DP was possible implements an intermediate generic language responsible to represent robot commands. It intermediate language is showed in picture 6 and can be used to write the same program to 3 different robot platforms. Insert a new robot language becomes easy through new protocol class creation shall the OO project nature. We can see in the picture 6, the 4 columns that compose the robot intermediate code. The first column represents the motion type, the second one the name of position variables, the third is the motion velocity, and finally fourth column shows the point reach precision.



● Rapid (ABB)
○ Karel (Fanuc)
○ Melfa-Basic IV (Mitsubishi)
○ Trident (PUMA)

```
Program : PROG1
1 P1 100 0
2 P2 500 10
2 P3 1000 0
3 P4,P5 500 0
2 P6 500 0
2 P2 100 0
1 P1 500 200
```

Fig. 6. Intermediate generic language.

Components reuse is an OO language intrinsic feature and can be reached through many ways. Good practices to increase reuse are applying the OODP techniques described in section 2. The chosen language was Java 2 Enterprize Edition(J2EE) to bring this powerful features and easy project maintenance. It exceptional capability attends fully to OO concepts making viable the code generator development. The Project is composed basically in 5 packages that subdivide the off-line programming basic responsibilities. This packages was: interf, optimize, service, protocol, and persist. How its packages works will be explained in this section. The development model adopted was 3 layer project. This model is divided in presentation, business, and persistence layer.

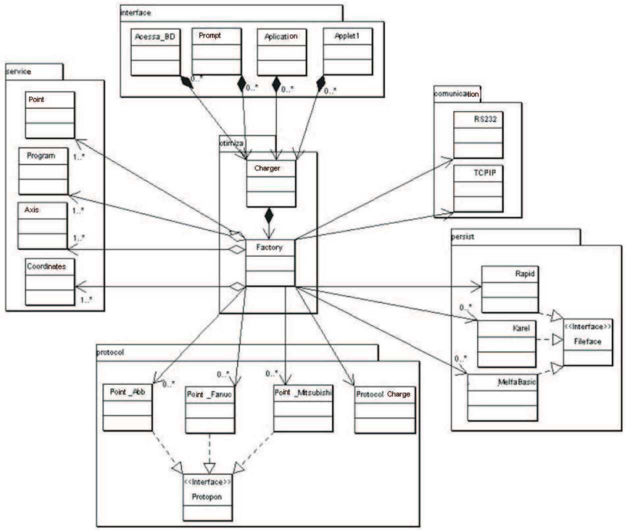


Fig. 7. Classes Diagram.

A. *PRESENTATION LAYER*

The interf package represents the interface layer and is responsible to system data input. It can be an database query, an application running in the host PC or an Applet browser showed in picture 7. At present moment, the package interface is the unique framework module that is completely decoupled of the business layer. It can was explained widely at section II-D.

B. *BUSINESS LAYER*

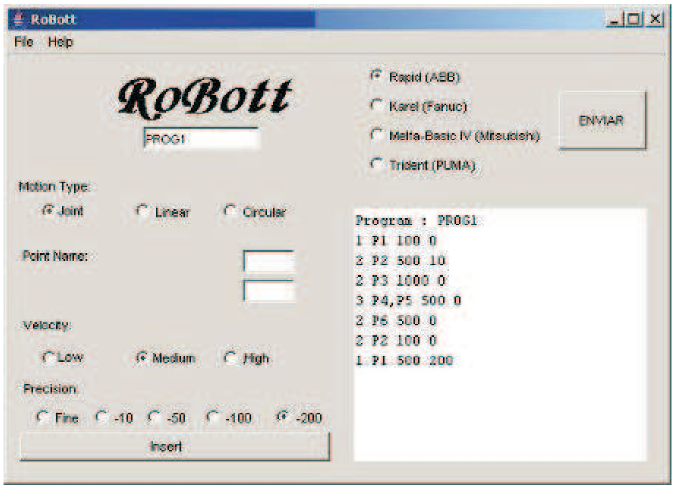
It was drew up by service, protocol, and optimize packages. At service package is generated the intermediate code using input interface instructions. It package hold the most important system classes because there is generalized points, axis, coordinates, and robot program. In protocol package was the classes responsible to convert the intermediate code to specific robot code chosen at interface layer. Optimize package is a set of classes that regulates interaction between different business classes. In other word, this package brings benefits to application centralizing data input and object creation through Charger and Factory classes respectively.

C. *PERSISTENCE LAYER*

It main function is taken care of robot code generated store. The persist package automatize the natives manufacture languages structure and extension knowledge through Java classes. Each class responsible to an specific robot language template. This Project adopts OO techniques using J2EE applied to robot programming. The proposed software intends to enable professionals that were being substituted by robots work, to program it. Using an class formalized knowledge, this software have to facilitate program an robot motion type, with it own precision and velocity between two different points in a 3D reality. Everyone PC can be used to programming and communicate with robots. Because the portability brought by Java virtual machine and the PC communication devices are compatible with all contemporaries robots. The abstraction adopted using the OODP described in section II allow increase interfaces, communication protocols, and robot languages.

V. EXPERIMENTAL RESULTS

The experience proposed to demonstrate the software functionalities was, write the acronym UCSAL in a white board using two different robots. The robots were programmed using the code generator described in this research and showed in the picture 8. To explain the experience and the software utility, was idealized the motion sequence to write the acronym UCSAL in the white board using a pen griped by the manipulator. To make the character ”U” is required 1 circular motion, to ”C” another circular motion, to ”S” two circular motions, to ”A” tree linear motions and finally to ”L” two linear motions. Beyond the motion type, for each displacement, is essential set up point names, velocities, and precision. Is also necessary set up points out of the white board between the characters writing, to define the attack angle of the pencil. Like the coordinates capture was not also implemented, it is been made by teaching, using the joystick linked to the robot controller to set the xyz coordinates.



PROG1

● Rapid (ABB)
○ Karel (Fanuc)
○ Melfa-Basic IV (Mitsubishi)
○ Trident (PUMA)

Motion Type: ● Joint ○ Linear ○ Circular

Velocity: ○ Low ● Medium ○ High

Precision: ○ Fine ○ -10 ○ -50 ○ -100 ● -200

```
Program : PROG1
1 P1 100 0
2 P2 500 10
2 P3 1000 0
3 P4,P5 500 0
2 P6 500 0
2 P2 100 0
1 P1 500 200
```

Fig. 8. Application interface.

The experiment was successful. Creating an intermediate program using RoBott and seeting the coordinates by teaching. It was able write the acronym UCSAL using ABB and Mitsubishi golems. The file transfer of robot programs, at present moment is done by floppy disk.

VI. CONCLUSION AND FUTURE WORKS

The facilities brought by OO Project nature makes possible in a posterior implementations add sensors and artificial intelligence techniques updating the software application area.

The system, after some changes, can be centralized in a server showed in picture 9 and interact with a motion coordinates data base of a manufacture cell, or facilitate program(task) migration between different robots. Also is possible develop new protocol classes allowing work with other machines, that have the same action principle of robots, for example: a CNC or a five axis machine.

The implementation of motion coordinate inputs, through an 3D simulation and PC to controller communication was not made.

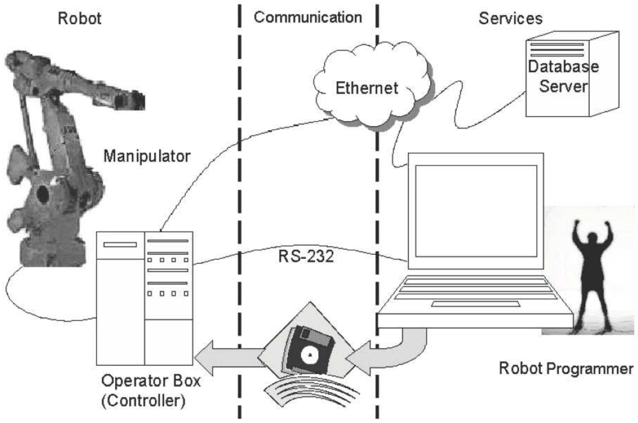


Fig. 9. Communication proposed to future works.

Because the development focus was made an portable software, capable to write robot programs to different manufacturers languages. Hence actually, RoBott is able to generate particular files to different platforms through different interfaces running in different operating systems. The next step is use piece neutral project files like: step-nc, iges, dxf, and sat files. Capturing the intrinsic piece coordinates through his neutral projects and calculating the extrinsic coordinates will allows the piece manipulation. The position coordinates input, simulation, and the PC to robot communication will be discussed in a better chance.

Acknowledgements. This research was carried on in the context of the Master Industrial Electronic Engineering. Project Supported by the Programme ALBAN, the European Union Programme of High level Scholarships for Latin America, scholarships no. E04M033540BR. Thanks to CIMATEC - Technology and Manufactory Integrated Center of SENAI-BA, Brazil.

REFERENCES

Fuller, J. L. (1999). *Robotics: introduction, programming, and projects*. Prentice-Hall Inc., London, 2nd edition.

Helm, R. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, ISBN 0-201-63361-2, 1st edition.

Murray, R. M. (1994). *A Mathemetival Introduction to Robotic Manipulation*. CRC Press, Florida, 1st edition.

Patterson, J. G. (1995). *ISO 9000: Worldwide Quality Standard*. Crisp Publications, Inc., California, 2nd edition.

Wrn, H. (1998). Automatic off-line programming and motion planning for industrial robots. In *ISR98, 29th International Symposium on Robotics 1998*. ISR Press.