A Client/Server Message Oriented Middleware for Mobile Robots

Danilo H. F. Menezes
Federal University of Sergipe/Computer Department, São Cristovão, Brazil
Email: danilohfm@dcomp.ufs.br

Marco T. Chella and Hendrik T. Macedo
Federal University of Sergipe/Computer Department, São Cristovão, Brazil
Email: {chella,hendrik}@ufs.br

***Abstract*— Mobile robots offer a wide range of applications in educational and research fields. Sensing, planning, control, reasoning, and learning are human-like capabilities that can be artificially replicated in a computer-based robotic system as software applications. The development of software for mobile robot application is a complex task due to the fact that the wide range of robots are composed of heterogeneous hardware components which need different device software drivers and different low-level communication protocols. This paper presents a Client/Server Message Oriented Middle for Mobile Robots which separates the user application from the hardware and low-level implementation. The middleware has been already ported to two different robotic platforms. As a case study, a path planning and self-localization algorithms have been implemented and applied successfully.**

***Index Terms*—mobile robots, robotic middleware, message oriented middleware**

I. INTRODUCTION

Mobile robots offer a wide range of applications in educational and research fields. The first has been validated in AI [12], [18], Programming Languages [8] and Image Processing [16]. The second can be exemplified by some complex issues such as real-time face detection [9], robot navigation [7], [16] and self-localization [21].

As [2] states, sensing, planning, controlling, reasoning and learning are human-like capabilities that can be artificially replicated in a computer-based robotic system as software applications, that implement data structures and algorithms devised on a large spectrum of theories, from probability theory, mechanics, and control theory to ethology, economy, and cognitive sciences.

The development of software applications for mobile robots is a complex task due to the fact that the wide range of robots are composed of heterogeneous hardware components that need different device software drivers and different low-level communication protocols. As a result, the development of reusable and portable software is definitely not a trivial task, this means that when a change of platform is needed for the same application, it will have to be almost completely rewritten and great effort is spent to the installation and configuration of such new device software drivers.

In order to face these problems, we propose a Client/Server Message Oriented Middleware(MOM) which separates the user application from the hardware and low-level protocols, creating a less hardware-dependent application. Using the MOM, the application programmer just need to be aware of the high level protocol which is a human-like language and can be almost the same for most platforms. The translation to the low-level protocol is made by the server side where also relies all the device software drivers and its configurations. The middleware has already been ported to two different robotic platforms: the Lego Mindstorm NXT [9] and a low-cost robotic platform developed at Federal University of Sergipe (RPFUS) [3]. As a case study, self- localization and path planning algorithms have been implemented for both platforms and applied successfully.

The rest of the paper is organized as follows. Some related works to middleware for robotics are presented in section 2. In section 3, the Client/Server MOM is detailed. Experiments performed with both platforms are described in section 4. Finally, some concluding remarks are presented in section 5.

II. RELATED WORKS

Several researchers and research groups are working on middleware solutions for robotics. In this section we point out some highly related initiatives.

Miro [22] is an object-oriented middleware for robots developed by University of Ulm, Germany. The main motivation of using object-oriented middleware is to improve the software development process for mobile robots and enable the interaction between robots and enterprise information. Miro allows inter-process and cross-platform interoperability for distributed robot control.

Orca [1] is a middleware framework for developing component-based robotics. The main goal of Orca is to enable software reuse in robotics. Orca enables the implementation of a distributed component-based robotic

doi:10.4304/jsw.7.5.1155-1163

system by allowing the user himself to define interfaces and communication mechanisms.

The goal of RT-Middleware [13] is to build robots and their functional parts in a modular structure at the software level and to simplify the process of building robots by simply combining selected modules. These goals are to allow system designers or integrators to build customized robots for a variety of applications in cost and efficient manners. Another important goal is to make robots more intelligent by distributing their necessary resources over a network. RT-Middleware provides the necessary services to enable implementing robotic applications that need these types of distributed applications.

ASEBA [15] is an event-based middleware, messages are only transmitted when a relevant event occurs, that supports distributed control and efficient resource utilization of multiprocessor robots. This middleware is designed for robots with several processors that communicate through a shard bus.

Player/Stage system [11] is a middleware platform that provides infrastructure, software drivers and some algorithms for mobile robotic applications. Player serves as an interface to many different robotic devices and provides drivers for many hardware modules. Some of the main features of this middleware are transport protocol-independence and modularity.

A good overview of some other middleware proposals for robotic applications can be seen at [13].

III. The Client/Server Message Oriented Middleware

A Message Oriented Middleware (MOM) is a category of middleware which provides a layer between the high level applications and the (robotic) platforms where they actually run. The MOM replaces the direct communication between the involved parts by message exchanging system.

The main goal of Client/Server MOM for mobile robots is to provide easy software reusability and avoid the configurability of device software drivers and low-level protocols. In order to achieve this task, the Client/Server MOM is structured in a distributed view (Fig. 1).

Client/Server communication takes place on TCP Socket I/O channels. On the server side, abstract operations are defined, which allows specific implementations for different robotic platforms (Fig. 2).

The interaction occurs as Fig. 3 illustrates. First, the application requires the connection get started. Next, the MOM client side creates the connection with the server side. As soon as the connection is established, the application starts sending high level messages in human-like language through the MOM client side interface. “ROTATE 90” and “MOVE 20”, for instance, could concern instructions for making the robot turn *90º* to the right and moving itself straight ahead 20 centimeters, respectively. The client side receives the message and

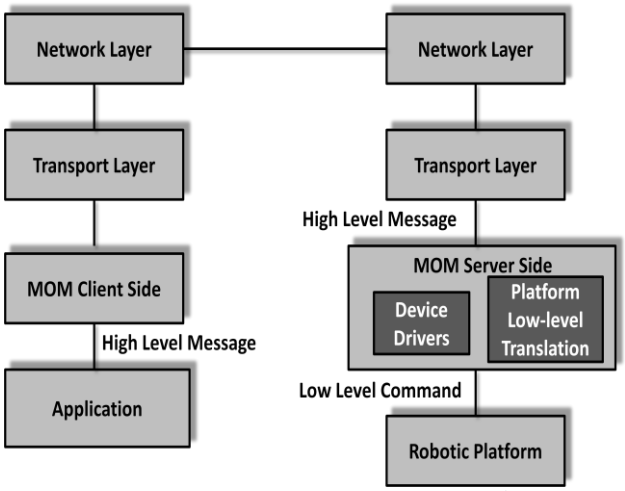

Figure 1 – Client/Server MOM multi-tier architecture fosters reuse.

pushes it over the network toward the server side. All the client messages are blocking messages in order to synchronize both sides of MOM. The server side receives the high level command and uses its translation function to translate the high level command into a low-level platform command or into another intermediate command that can be, in turn, sent to another server, in order to perform last low level translation.

All these command translations are transparent to the application programmer, who just needs to be aware of the high level protocol which, in turn, is shared to the whole set of robotic platforms. Device software drivers installation and configuration are performed on the server side, hiding it from the application programmer. This transparency aims to reduce application development costs and take the programmer focused only in his main task.

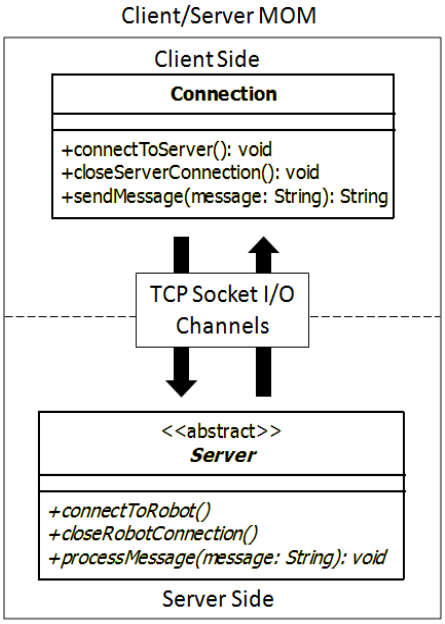

Figure 2 - Client/Server MOM communication via TCP socket. Different robotic platforms share common abstract definition of the server.

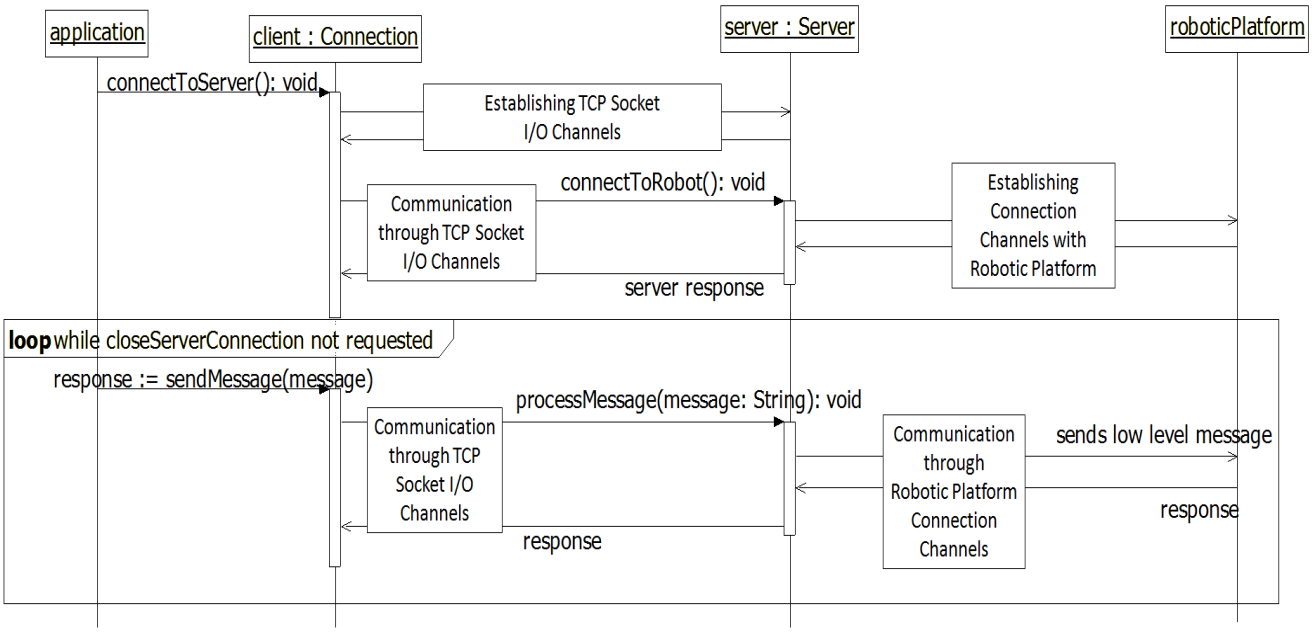

Figure 3 - The diagram of Fig. 4 illustrates how the framework relates to Client/Server MOM.

Indeed, both server side and translation are platform dependent because the software drivers are usually different from one robotic platform to another. The client side is platform independent except for some minor modifications that may be required in the high level protocol. As a consequence, the application can be thoroughly ported from one robotic platform to another.

It is noteworthy that the provided interfaces and the high level protocol are both programming language-independent.

IV. Case Study: Path Planning and Localization Algorithms

As a case study of the Client/Server MOM we have made a series experiments. In the experiments the robotic platform is put in one place at a previous known environment and has the task to go to another determined place within it without run into any obstacle. In order to achieve this main task two high level tasks in robotics, path planning, to plan the path to the initial place to the destiny place, and self-localization, to keep track of its position while executing the path, were performed by two different platforms, the Lego Mindstorm NXT and a robotic platform developed at Federal University of Sergipe (RPFUS). The programming language used in the implementation was Java.

A. Use of Client/Server MOM

The robotic platforms high level protocols and its respective functionalities can be seen in Table I. , the Xs indicate that the respective message belongs to the respective robotic platform protocol.

In the "RANGEPOSITION" message it's important to notice that if A is equal to 90 the sensor will be pointing to the place in front of the robot Similarly, when A is equal to 180 and 0, the sensor will be pointing to places exactly at right and left of the robot, respectively.

All the messages receive a response from the server in order to know that everything has gone well on the server side. The "READULTRASONIC" and "READINFRARED" responses are their readings of the sensor.

Table I
HIGH LEVEL PROTOCOLS.

Message	Functionality	RPFUS	Lego
"FORWARD"	Makes the robot move straight forward until it receives a "STOP" message.	X	X
"BACKWARD"	Same as above but moves backwards.	X	X
"ROTATE A"	Turn left Aº, if A is negative turn right Aº.	X	X
"STOP"	Makes the robot stop.	X	X
"MOVE A"	Makes the robot move A *cm* straight forward if A is positive, otherwise moves backward A *cm*.	X	X
"READULTRASONIC"	Request a read to the ultrasonic sensor attached to the robot.		X
"READINFRARED"	Request a read to the infrared sensor attached to the robot.	X	
"RANGEPOSITION A"	Uses a servomotor to make a sensor point at Aº.	X	

While the high-level protocol was known, information about device software drivers installation and configuration as well as the low-level protocol of each robotic platform were completely hidden from the application programmer.

B. Motion and Sensor Framework

Using the Client/Server MOM structure, a motion and sensor framework was implemented in order to encapsulate the high level protocol. The diagram of Fig. 4 illustrates how the framework relates to Client/Server MOM.

A Pilot instance encapsulates sendMessage calls of a established connection and implements high level protocols (Fig. 5). Each type of sensor implements its own read method, encapsulating the high level protocol message, with string readings.

Sensors that measure distances, like ultrasonic and infrared ones, must also implement the getDistance method of DistanceSensor interface and convert string readings in the appropriated data type. In order to allow the use of other kind of sensors, the framework can be easily extended with the provision of other similar interfaces.

The MobileRobot class encapsulates the Pilot's class methods and is in charge of openning and closing the connection with the. A mobile robot may have zero or more sensors, indeed.

It is important to notice that although the framework is dispensable to the programming of robots, it greatly facilitates the job.

The framework also shows how powerful the middleware is due to the fact that the Pilot class just uses the high level protocol messages related to the robot motion and this class remains exactly the same for all robotic platforms.

C. Robotic platforms

The Lego Mindstorms NXT robotics kit is composed of light, sound, touch and ultrasonic sensors. The programmable brick control all these sensors and three servomotors. A brief description of the brick's most important properties is: 32-bit ARM7 microcontroller, Communications: Bluetooth class II V2.0, USB Port 256 Kbytes FLASH/64 Kbytes RAM, Wireless (12Mbit/s) and 6 AA batteries or Rechargeable Lithium battery.

In the case study NXT has been mounted as a mobile robot, moving around through two servomotors attached to rubber wheels. The ultrasonic sensor had been also attached to the robot. The ultrasonic sensor, which belongs to NXT set, consists of a transmitter that sends 40KHz sound signals, and a microphone that receives the sound back. The sensor has a range of 0cm to 255cm with an accuracy of +/-3cm. Robot behavior configuration is done through firmware LeJOS NXJ, a tiny Java operating system which implements advanced control algorithms programming interface(*API*). The version used in this work is LeJOS NXJ 0.85, which runs Java applications uploaded to the robot.

The RPFUS is a low cost platform that uses off-the-shelf components and it's easy to use. The communications between PC and RPFUS are over a wireless data. The system is composed by a base communication board connected to PC in USB port and a robot controller board. Radio modules with ZigBee [23] technology are used to implement the wireless link. Base and controller board have a microcontroller with a firmware that provides all functionalities of the system.

The RPFUS eletronic circuit is based in a microcontroller PIC18F2550 a RISC 12 MIPS chip with transceiver for full speed USB 2.0, peripherals like analog to digital converter, pulse with modulation (PWM) module, pins for digital input/output and USART (serial communication).

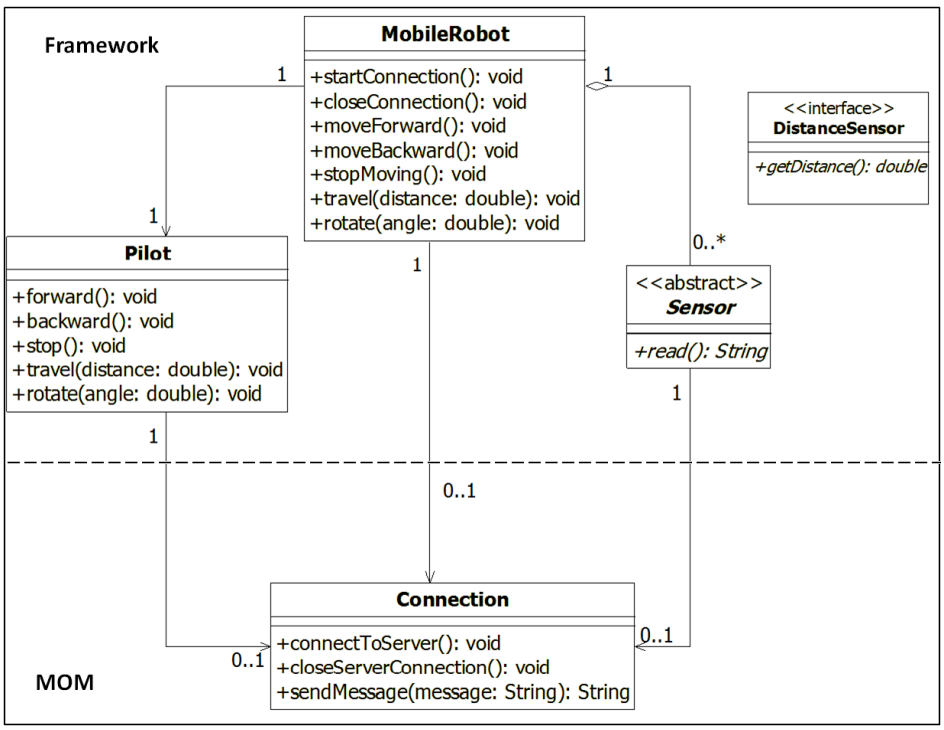

Figure 4 - The diagram of Fig. 4 illustrates how the framework relates to Client/Server MOM.

```
1:  public class Pilot{
2:     private Connection conn;
3:     public Pilot(Connection conn){
4:          this.conn = conn; }
5:     public void forward(){
6:          conn.sendMessage("FORWARD"); }
7:     public void backward(){
8:          conn.sendMessage("BACKWARD"); }
9:     public void stop(){
10:         conn.sendMessage("STOP"); }
11:    public void rotate(double angle){
12:         conn.sendMessage("ROTATE "+angle); }
13:    public void travel(double distance){
14:         conn.sendMessage("MOVE "+distance); }
15: }
```

Figure 5 – Pilot class definition

The firmware implements the USB stack that enumerated the device with a CDC (communications device class). The PC computer host sees base communication board as a virtual serial port. This way any programming tool with resources to access hardware serial port can be used to connect to base communication board, send and receive commands to ERB. Messages received from host are converted to serial and sent to ZigBee module, messages received from ERB via ZigBee are converted to USB. The controller board has functionalities to control actuators(servomotors), sensors(using analog inputs), decode commands and manage communication with base board.

RPFUS has been mounted as a mobile robot, that uses a differential assembly steering system with two servomotors attached to rubber wheels. A distance sensor based on infrared light(infrared sensor) had been also attached to a third servomotor. The third servomotor function is to make the infrared sensor point in a specific direction in order to decrease the noise that would be created if the entire platform had to turn to take the sensor reading in a specific direction and then try to get back to position, which is what the Lego actually does.

Using the framework, the read and getDistance implementations of the Lego's ultrasonic sensor and RPFUS infrared sensor are shown in Fig. 6 and Fig. 7 respectively.

The method which is implemented in order to encapsulate the message to move the servomotor which directs the infrared sensor is illustrated in Fig. 8.

As we can see, the only difference between them is limited to the protocol messages.

D. Environment description

The environment map is illustrated in figure 9. We have chosen a grid as its computational representation [5]. The graph vertex and edge sets are build as follows. First, the environment is approximated by a grid of equal sized cells. There are two categories of cells: occupied cells and free cells. Each free cell is represented by a vertex in the graph. Next, a neighborhood style for the cells is chosen and thus each cell's vertex have an edge to each of his free cell neighbors.

A cell size of 20 cm and the Von Neumann neighborhood style have been chosen (Fig. 10). The environment approximated by a grid and the graph structure are illustrated in Fig. 12.

E. Path planning algorithm

Path planning is achieved by breadth-first search algorithm[4].

Given a graph $G = (V, E)$, where V is its vertex set and E its edge set, and a distinguished source vertex s, breadth-first search systematically explores the edges of G to "discover" every vertex that is reachable from s until it finds the end vertex f [4]. The algorithm is illustrated in Fig. 13.

```
1:  public double getDistance(){
2:     String reading = read();
3:     return Double.parseDouble(reading); }
4:  @Override
5:  public String read(){
6:     return conn.sendMessage("READULTRASONIC");
7:  }
```

Figure 6 – Lego's ultrasonic getDistance and read methods.

```
1:  public double getDistance(){
2:     String reading = read();
3:     return Double.parseDouble(reading); }
4:  @Override
5:  public String read(){
6:     return conn.sendMessage("READINFRARED");
7:  }
```

Figure 7 – RPFUS's infrared getDistance and read methods.

```
1:  public void setRangePosition(int angle){
2:      conn.sendMessage("RANGEPOSITION "+angle);
3:  }
```

Figure 8 – Method that move the third servomotor.

Fig. 12 shows the grid approximation of the environment map. Occupied are represented by blank cells and gray cells represent the free cells. The graph vertices and edges are highlighted by gray dots and gray lines, respectively.

The algorithm always finds the shortest path, in number of edges, from s to f if it exists and runs in $O(|E|)$. Using the parent structure is easy to rebuild the path until the source s by executing the algorithm illustrated in Fig. 13 using f as parameter. The path is the inverted order of vertex in queue L.

F. Self-localization Algorithm

The localization problem can be probabilistic modeled as a Bayes Filter [19]. Mathematically, what we are trying to predict is the position $x = [x, y, \theta]^T$ at time k.

$$p(x_k | Z^k) \quad (1)$$

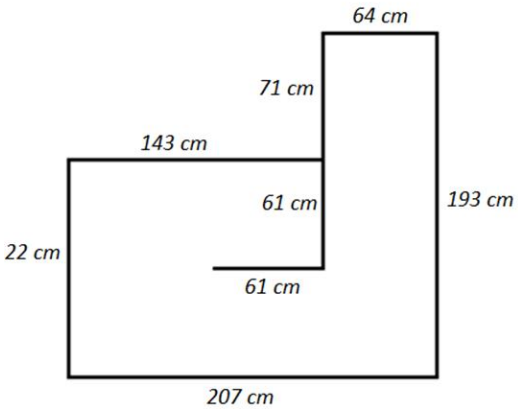

Figure 9 – Environment map.

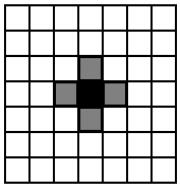

Figure 10 – The von Neumann neighborhood. Gray cells are the neighborhood of the black cell.

where $Z^k = \{ z_k, i = 1..k\}$ and z_k and x_k means, respectively, the measurements and the position at time *k.*

This probability density function can be calculated recursively in two phases by the measurements, z_k, and actions, u_k, over the time. The first phase is the *prediction phase* where the action made by the robot is taken into account, because of that its phase is also known as *motion model*, and derives (1) into

$$p(x_k|\, Z^{k-1}) = \int p(x_k|\, x_{k-1}, u_{k-1}) p(x_{k-1}|\, Z^{k-1}) dx_{k-1} \quad (2)$$

The second phase is called *update phase* where the measurements are taken into account deriving (2) into (3). This phase is also known as *measurement model*.

$$p(x_k|\, Z^k) = \frac{p(z_k|\, x_k)\, p(x_k|\, Z^{k-1})}{p(z_k|\, Z^{k-1})} \quad (3)$$

Using $\alpha = p(z_k|\, Z^{k-1})$ as a normalization factor we obtain

$$p(x_k|\, Z^k) = \alpha\, p(z_k|\, x_k)\, p(x_k|\, Z^{k-1}) \quad (4)$$

```
1:   Algorithm_BFS( G , s , f ):
2:       Q = ∅;
3:       enqueue(Q,s);
4:       while Q ≠ ∅ do
5:           u = dequeue(Q);
6:           if u == f then
7:               quit while;
8:           endif
9:           if u wasn't marked yet then
10:              mark u;
11:              for each edge e = (u,w) do
12:                  if w wasn't marked yet then
13:                      enqueue(w);
14:                      parent[w] = u;
15:                  endif
16:              endfor
17:          endif
18:      endwhile
```

Figure 11 – The breadth-first search algorithm

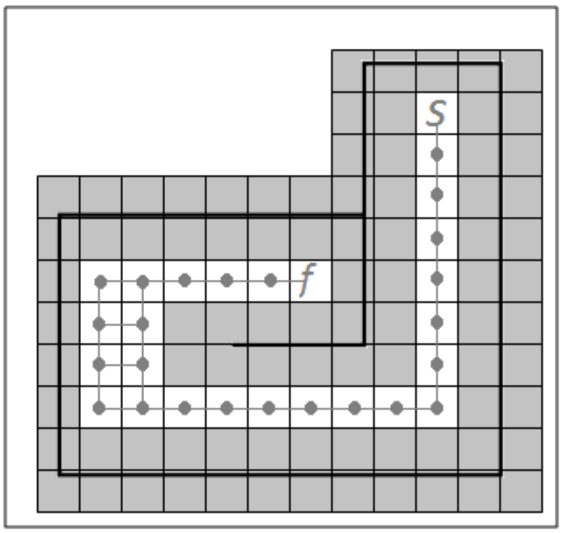

Figure 12 - Grid approximation of the environment map. Breadth-first searches a path from s to f.

```
1:   Build_Path( v ):
2:       if v == s then
3:           enqueue(L,v);
4:           return;
5:       endif
6:       enqueue(L,v);
7:       Build_Path( parent[v] );
```

Figure 13 – Build path algorithm

From now on we just need to define a $P(x_0)$. For the Bayes Filter implementation we have implemented the Monte Carlo Localization (MCL) algorithm [20]. This algorithm discretizes the probability function in a set containing M particles, $X_t = \{x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[m]}\}$, and each these has a *importance weight*. Let the importance weight of the *i-th* particle at time *t* be $w_t^{[i]}$ we have

$$\sum_{i=1}^{M} (\alpha\, w_t^{[i]}) = 1 \quad (5)$$

At the beginning of the algorithm all particles are at the correct start position. At the same time their importance weights are $\alpha w_t^{[i]} = M^{-1}$, that defines $p(x_0)$.

The particles move according to the motion model and their weights are updated according to the measurement model. At the end of the update phase, a key process called *importance resample* is performed in order to select the most probable particles to represent the robots pose. The Monte Carlo Localization algorithm is illustrated in Fig. 14 [19].

The performance and computational burdenare directly affected by the number of particles used to represent the probability function. In the case study we have used *1500* particles.

G. Correction module

A correction module has been implemented in order to correct robot's position each time the localization algorithm realizes it differs from previously determined by path planning with a threshold tolerance.

The correction relies on vector properties. Let (x_i, y_i) and (x_c, y_c) be the actual and correct position of the robot respectively. Furthermore let θ_i be the current robot's orientation. First two vectors $A = (\cos\theta_i, \sin\theta_i)$ and $B = (x_c - x_i, y_c - y_i)$ are created as illustrated in Fig. 15. In order to do the correction we need to know the angle between the vectors A *and* B. The dot product between A and B asserts (6) and (9). Using (6), (7) and (8) we can evaluate (9) obtaining (10).

$$A \cdot B = x_c \cos\theta_i - x_i \cos\theta_i + y_c \sin\theta_i - y_i \sin\theta_i \quad (6)$$

$$||A|| = 1 \quad (7)$$

$$||B|| = \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2} \quad (8)$$

$$\theta = \cos^{-1}\left(\frac{A \cdot B}{||A||||B||}\right) \quad (9)$$

$$\theta = \cos^{-1}\left(\frac{x_c \cos\theta_i - x_i \cos\theta_i + y_c \sin\theta_i - y_i \sin\theta_i}{\sqrt{(x_c - x_i)^2 + (y_c - y_i)^2}}\right) \quad (10)$$

As soon as we know θ the correction move made by the robot is to turn θ and move ||B|| cm straight forward to get into the correct position.

H. Experiment results

The experiment was run 10 times for each robotic platform and each robotic platform failed in one out of ten times. Both robots presented similar execution for path planning over 10 runs. Some of these executions are illustrated in Fig. 16. Variance is slightly higher for the RPFUS. Fig. 17 illustrates two screenshots of MCL execution. Blue dots represent the particles spread out whereas red lines concern infrared or sonar directions.

```
1:   Algorithm_MCL( X_t , u_t , z_t, M):
2:     N_t = X_t = ∅;
3:     enqueue(Q,s);
4:     for m = 1 to M do
5:       x_t^[m] = motion_model(u_t, x_{t-1});
6:       w_t^[m] = measurement_model(z_t, x_t^[m], m);
7:       N_t = N_t + ⟨x_t^[m], w_t^[m]⟩;
8:     endfor
9:     for m = 1 to M do
10:      draw m with probability α w_t^[m] ;
11:      add x_t^[m] to X_t;
12:    endfor
13:    return X_t
```

Figure 14 – Monte Carlo Localization(MCL) algorithm.

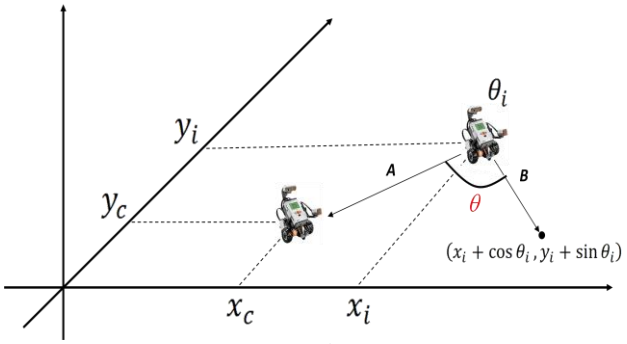

Figure 15 – Construction of the vectors *A* and *B*.

Client/Server MOM is in order to provide software reusability since almost no rewriting of code was done neither to the path planning algorithm nor to the self-localization one.

Minor modifications were necessary in Sensor class and takeReadings() method to support self-localization algorithm. The reason for this is that there is no servo motor able to direct the ultrasonic sensor in the Lego platform. As a consequence, the entire robot should move itself to accomplish such task. Fig. 18 highlights an excerpt of the implementation differences concerning the takeReading method for both robotic platform.

It is important to notice the another feature of the use of the proposed Client/Server MOM. The application programmer didn't have to be aware about the Bluetooth stack configuration, the ZigBee configuration and software drivers installation, how was made the communication between Lego and Lejos and its API

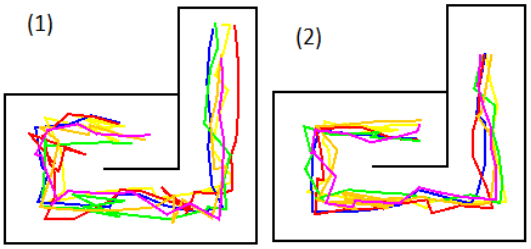

Figure 16 – Execution of path planning. Figure shows the set of paths performed by RPFUS and Lego over some runs.

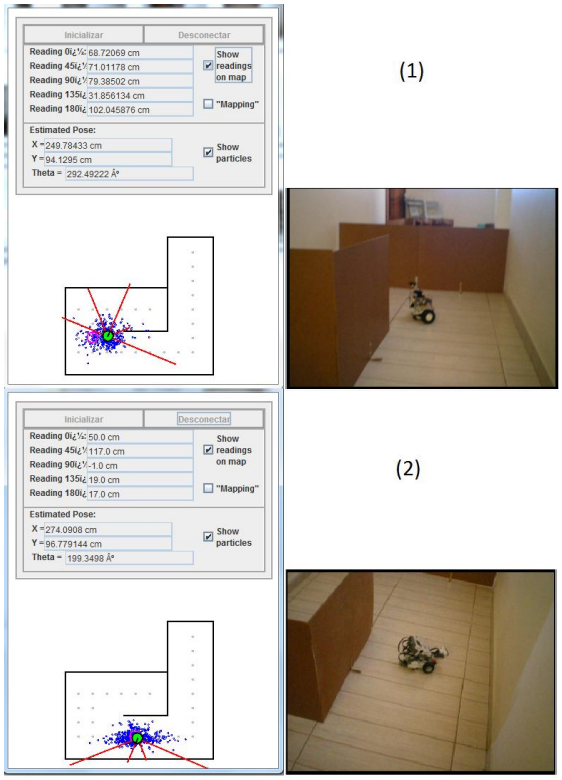

Figure 17 – Application running on RPFUS(1) and on Lego(2).

installations, what programming language was used on the server side and installation and configuration of other software device drivers that are specific to each robotic platform. This transparency to the application programmer decreases the time required in order to develop the software application as far as software reusability does.

Figures 19 and 20 show the instances of Fig. 1 for the RPFUS and Lego, respectively.

V. Conclusion

In this paper, we presented a Client/Server Message Oriented Middleware (MOM) for mobile robots in order to improve software reusability in the field of mobile The Client/Server MOM abstracts the low-level protocols

LEGO

```
1:  public void takeReadings(){
2:     double range;
3:     range = ultrasonic.getDistance();
4:     readings.setRange(1,range,false);
5:     rotate(45);
```

RPFUS

```
1:  public void takeReadings(){
2:     double range;
3:     setRangePosition(90);
4:     range = infrared.getDistance();
5:     readings.setRange(1,range,false);
6:     setRangePosition(45);
```

Figure 18 – Differences between implementations of takeReadings method for Lego(1) and RPFUS(2) platforms, respectively.

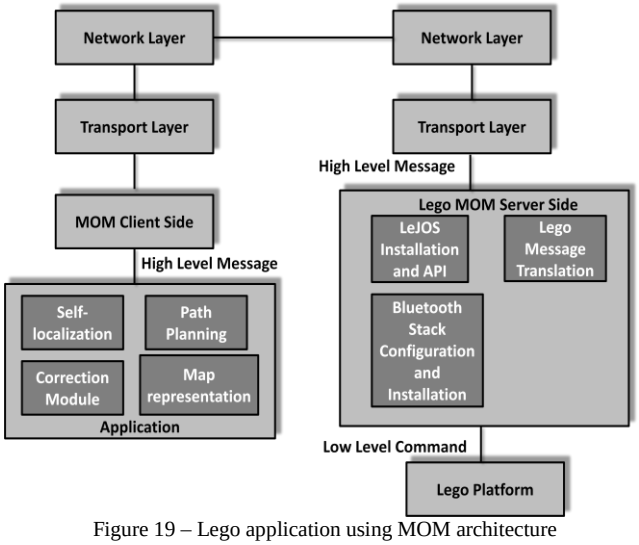

Figure 19 – Lego application using MOM architecture

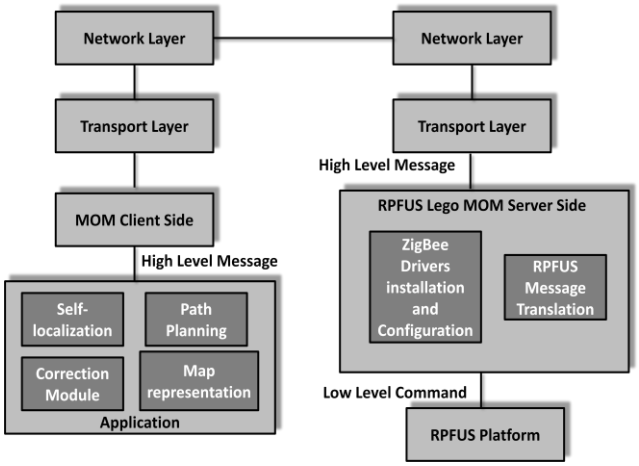

Figure 20 – RPFUS application using MOM architecture

robots and to reduce the cost and time spent in developing these kind of applications.

The Client/Server MOM abstracts the low-level protocols used used to control the robotic platforms into high level API that is more understandable by application programmers. Therefore it also hides the necessity of installing and configuring the device software drivers which is all made by his server side that became transparent to the application programmer. The approach showed to be very powerful in these aspects.

A case study with a application which implemented two high level tasks in robotics, localization and path planning, was made using two different robotic platforms. The application was ported from one platform to another without great modifications and with showing that the use of the Client/Server MOM really improves mobile robots software reusability and development cost.

Acknowledgment

This work was supported in part by a grant from Programa Institucional de Bolsas de Iniciação Científica(PIBIC)/Conselho Nacional de Desenvolvimento Científico e Tecnológico(CNPq) and

Fundação de Apoio à Pesquisa e à Inovação Tecnológica do Estado de Sergipe(FAPITEC/SE).

REFERENCES

[1] Makarenko A., Brooks A. and Kaupp T., "Orca: Components for Robotics," In International Conference on Intelligent Robots and Systems (IROS), pp. 163-168, Oct. 2006.

[2] Brugali, D. and Prassler, E., "Software Engineering for Robotics [From the Guest Editors]," *Robotics & Automation Magazine, IEEE* , vol.16, no.1, pp.9-15, March 2009. doi: 10.1109/MRA.2009.932127

[3] Chella, M. T., Robotic tool with Scratch Language. In: Robocontrol 2010 4th Workshop in Applied Robotics and Automation, 2010, Bauru. Robocontrol 2010 4th Workshop in Applied Robotics and Automation. Bauru : UNESP. 2010.

[4] Cormen, T.H., C.E. Leiserson, R.L. Rivest and C. Stein, Introduction to Algorithms, Section 22.2: Breadth-First Search. 2nd Edn., MIT Press, McGraw-Hill, New York. 2001.

[5] Crous, C. B., Autonomous robot path Planning (Master's thesis). 2009. URL: http://www.cs.sun.ac.za/~abvdm/mhp/CarlCrous.pdf , July 2011.

[6] Dinh, H. and Inanc, T., Low cost mobile Robotics Experiment with camera and sonar Sensors. In: ACC'09: Proc. of the 2009 Conference on American Control. pp. 3793–3798. IEEE Press. 2009.

[7] Eggert, D., Using the Lego Mindstorms NXT robot kit in an Introduction to C Programming class. Journal of Computing Sciences in Colleges 24(6), 8–10. 2009.

[8] Gasperi, M.; Hurbain, P. and Hurbain, I., Extreme NXT : Extending the LEGO Mindstorms NXT to the next level.[S.l.]: Apress. 2007.

[9] Lee, T., Real-Time Face Detection and Recognition on LEGO Mindstorms NXT Robot. Advances in Biometrics 4642, 1006–1015. 2007.

[10] LEGO. Lego Mindstorms NXT. URL: http://mindstorms.lego.com/en-us/default.aspx, July; 2011.

[11] Kranz, M., Rusu, R., Maldonado, A., Beetz, M. and Schmidth, A., "A Player/Stage system for Context-Aware Intelligent Environments," in Proc. of the System Support for Ubiquitous Computing Workshop (UbiSys), Sep. 2006.

[12] McNally, M. and Klassner, F., Demonstrating the Capabilities of MindStorms NXT for the AI Curriculum. In: American Association for Artificial Intelligence. 2007.

[13] Mohamed, N., Al-Jaroodi, J. and Jawhar, I., "Middleware for Robotics: A Survey," *Robotics, Automation and Mechatronics, 2008 IEEE Conference on* , vol., no., pp.736-742, 21-24 Sept. 2008 doi: 10.1109/RAMECH.2008.4681485

[14] Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T. and Yoon, W., "RTMiddleware: Distributed Component Middleware for RT (Robot Technology), IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3555-3560, Aug. 2006.

[15] Magnenat, S., Longchamp, V. and Mondada, F., "ASEBA, an event-based middleware for distributed robot control" Workshops DVD of International Conference on Intelligent Robots and Systems (IROS), Oct.-Nov. 2007.

[16] Solano-Aragón, C. and Alanis, A., A Multi-agent Architecture for Controlling Autonomous mobile robots using fuzzy logic and Obstacle Avoidance with Computer vision. Bio-inspired Hybrid Intell. Systems for Image Analysis and Pattern Recognition pp. 215–246 . 2009.

[17] Stevenson, D.E. and Schwarzmeier, J.D., Building an Autonomous Vehicle by Integrating lego Mindstorms and a web cam. In: SIGCSE '07: Proc. of the 38th SIGCSE technical symposium on Computer science education. pp. 165–169. ACM . 2007.

[18] Talaga, P. and Oh, J., Combining AIMA and LEGO Mindstorms in an Artificial Intelligence course to build real world robots. Journal of Computing Sciences in Colleges 24(3), 56–64. 2009.

[19] Thrun, S., Burgard, W. and Fox, D, Probabilistic Robotics, Section 2.4: Bayes Filters and Section 8.3: Monte Carlo. 2000.

[20] Thrun, S., Fox, D., Burgard, W. and Dellaert, F. Robust monte carlo Localization for mobile robots. Artificial Intelligence, 101:99-141 18. 2000,

[21] Tribelhorn, B. and Dodds, Z.: Evaluating the Roomba: A low-cost, Ubiquitous Platform for Robotics Research and Education. In: 2007 IEEE Int. Conf. on Robotics and Automation. pp. 1393–1399, 2007.

[22] Utz, H., Sablatnog, S., Enderle, S. and Kraetzschmar, G., "Miro - Middleware for mobile robot applications,"*Robotics and Automation, IEEE Transactions on* , vol.18, no.4, pp. 493- 497. 2002. doi: 10.1109/TRA.2002.802930.

[23] ZigBee, Digi International [Online]. URL: http://www.digi.com/, July 2011.

Danilo H. F. Menezes was born in Aracaju, Brazil. He is an undergraduate student majoring computer science at the Federal University of Sergipe since 2008.

Marco T. Chella is an adjunt professor of computer science at the Federal University of Sergipe. He received his Ph.D in Eletrical Engineering from Universidade Estadual de Campinas, Brazil, in 2006.

Hendrik T. Macedo is an associate professor of computer science at the Federal University of Sergipe. He received his Ph.D. from the Federal University of Pernambuco, Brazil, in 2006. Currently, he occupies the position of Vice Coordinator of postgraduate program in Computer Science at Federal University of Sergipe. His research interests include both theoretical and experimental artificial intelligence and, in particular, simulated and phisical multiagent systems.