AIAA 2003-0459

Client-Server Java Programming For Wireless Mobile Robots

Lyle N. Long, Anupam Sharma, and Frederic Souliez
The Pennsylvania State University
University Park, PA

41st Aerospace Sciences Meeting
6 - 9 January 2003, Reno, Nevada

Client-Server Java Programming for Wireless Mobile Robots

Lyle N. Long[1], **Anupam Sharma**[2], **and Frederic Souliez**[3]
Department of Aerospace Engineering
The Pennsylvania State University
http://www.personal.psu.edu/lnl
lnl@psu.edu

Abstract

A client-server application for the remote control of a wireless mobile robot (a small four-wheel vehicle) is described. The application requires an onboard computer (e.g. a PC-104 or a laptop) running a Java Virtual Machine (JVM) on the vehicle. The mobile robot is the client, which is controlled by the server. The server can be any desktop machine or another mobile robot. We show how mobile robots can be controlled using any computer on the internet through a wireless Ethernet network. This project uses commercial off-the-shelf (COTS) hardware and software. The ultimate goal of this effort is to build autonomous or semi-autonomous air-borne vehicles which can run with little or no human intervention, but for the time being we are working with ground-base vehicles to develop the hardware and software. A small traditional radio-controlled aircraft (e.g. 2 meter wing span) could easily carry the computer required by the client. The approach we are using essentially gives the power of large desktop or Beowulf clusters to very small mobile robots, since the client is a complete internet-based PC. The client is basically an intelligent agent. This approach can be extended to allow multiple vehicles to communicate with each other and with other computers on the internet. This is not an approach that can be used for real-time remote control of air-borne vehicles, but it is expected to be useful for autonomous and semi-autonomous vehicles, where it would be useful to occasionally send commands to the vehicle. This approach will also allow very powerful computers such as Beowulf clusters or parallel supercomputers to be used to control numerous mobile robots, since they too can act as servers.

Introduction

The concept of remotely controlling a vehicle was first used for military purposes by German motor-boats to ram enemy ships in World War I (WWI). They used radio waves to communicate with the remote motor-boats. Remote control technology was enhanced further in WWII and is now used in a wide variety of commercial and military products.

Remote control may be done using real, physical connections between the remote object and the user, or it may be done with wireless methods. Wireless remote control is often more desirable, especially when the distance between the user and the remote object is large. Over the years, researchers have tried different means of making wireless connections using flashes, ultrasonic waves, radio waves and infrared waves. All these have a restricted range of application. The idea of using computers to enhance the remote control is relatively new. Since the internet connects almost every part of the globe now, remote objects anywhere in the world can be controlled using networked computers if the interface controlling the remote object is a computer. If we couple wireless networking technology with networked computers, we can effectively achieve wireless control from global distances. This can be useful for a variety of purposes; including weapons systems, search and rescue operations, mine clearing, exploration, and reconnaissance.

Copyright 2002, Lyle N. Long, published with permission by AIAA
[1] Professor, Associate Fellow AIAA
[2] Graduate Research Assistant, Member AIAA
[3] Graduate Research Assistant, now at BMW, Munich, Member AIAA

In this paper we discuss the remote control of a vehicle using Java [Ref. 1 and 2]. We chose Java because it is platform independent, and we wanted our remote object to be controlled from any machine. Also, Java provides an easy implementation of Graphical User Interfaces (GUI) which is highly desirable. This is essentially a model problem to explore the use of networked computers for the remote control of mobile robots. In this paper we present some ideas which can help develop the concepts of semi-autonomous systems and groups of semi-autonomous systems for devices with embedded computers.

Wireless mobile robots which can be controlled via the internet will basically be very capable platforms for intelligent agent devices. The robots described herein are relatively small, but have fairly significant onboard processing power. These systems could have onboard cameras with neural network-based image processing. They could also have onboard GPS and many different types of sensors.

Our goal was to remotely maneuver a vehicle which has an onboard networked computer from another computer on the internet. The embedded computer has a wireless ethernet card which has a range of about 300 meters, but these will work over several miles (line of sight) with larger antennae and more powerful base units (access points). We implemented Client-Server programming to achieve remote control. The computer on the vehicle is called the client, and the machine which controls the vehicle is called the server. The client machine is always on and running the client program which listens for messages from the server. The idea is to have autonomous or semi-autonomous vehicles which can maneuver with little human intervention. Of course, at this moment our server is controlled by a human being, but this job can be taken over by a computer in the future.

A GUI to maneuver the vehicle was designed to run on the server machine. Through the GUI the user can set the throttle and the direction of the vehicle. The GUI also shows an image of the area in front of the vehicle from the onboard camera. There are essentially two parts to solving this problem: (1) We need an interface between the software and the hardware i.e. the servos on the vehicle, and (2) we need the Client-Server program to do the remote control. We discuss these in the following sections.

The Hardware Interface

We chose a radio-controlled electric truck (Traxxas Stampede) as the mobile robot. This is a popular 1/10th scale radio control vehicle which comes with an onboard receiver. The truck has a good shock absorber-spring suspension system, a slipper clutch, and a differential. The receiver is normally connected to the speed controller and a single servo that controls the steering. It typically uses a large 1500 mAh 7.2 volt battery. Figure 1 shows the vehicle with a small Toshiba Libretto laptop, and Figure 2 shows the vehicle with a PC-104 computer. We basically replaced the standard radio-control receiver with the computer and some other electronics. The onboard computer drives the servo motors through the serial port. The laptop and the PC-104 both have wireless ethernet cards which communicate with a wireless access point that is on the internet. It would be fairly straight-forward to add GPS hardware to the systems as well. We have now begun a larger version of this system using a larger mobile robot, an example image of one of these is shown in Figure 3. These will allow us to use larger computers and incorporate more sensors.

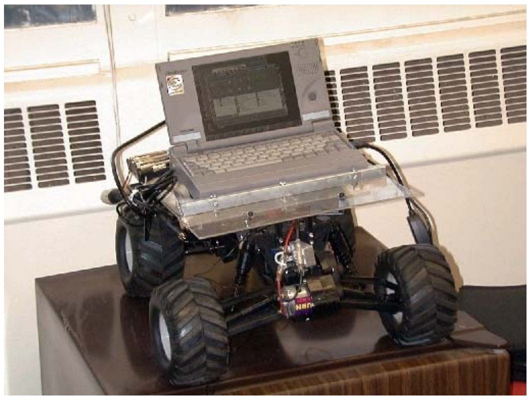
Figure 1. Vehicle with Toshiba laptop.

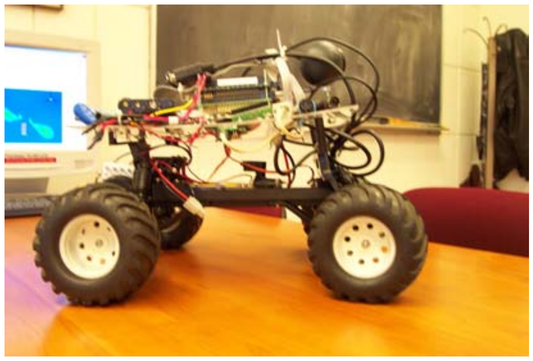
Figure 2. Vehicle with PC-104 computer.

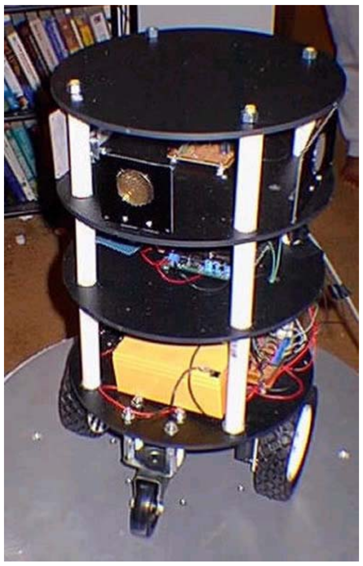

Figure 3. Zagros Max99 mobile robot with 12 inch diameter platforms [from Ref. 3]

The Toshiba laptop was a Libretto model 70CT, which has a 120 MHz Pentium processor and 32 MB memory. It has a 6 inch diagonal TFT matrix screen. It also has a serial port, parallel port, and a type III PC card port. The maximum battery life is 4.5 hours. With the larger battery this system weighs 2.2 pounds and is 8.3 x 4.5 x 1.4 inches. It can run MS Windows or Linux. This laptop was purchased for $500, but they are no longer in production,. A good, more modern replacement for this would be the new Fujitsu Lifebook P-series.

The PC-104 was obtained from EMJembedded.com, and is a MOPS LCD6. This has a 166 MHz Pentium processor. It has two serial ports, a parallel port, IDE interface, onboard VGA, floppy drive interface, USB port, and keyboard port. We added a PCMCIA port and a wireless Ethernet card, a 64 MByte solid state disk, and a hard-drive. This system is lighter than the Toshiba, since it does not have to carry the display and keyboard. The PC-104 can be run from batteries through a power converter (the board requires 5 volts). You could fairly easily spend $1000 for a fully-ready PC-104 system.

Java

We chose Java for this application because it is object oriented, has GUI capabilities, has remote method invocation (RMI), has integrated graphics, and it can easily drive devices through serial and parallel ports. All of these could also be accomplished using C++, however. C++ is also object oriented. GUI's can be incorporated into C++ using FLTK [Ref. 9]. Graphics can be incorporated using OpenGL [Ref. 10]. Remote method invocation can be done with CORBA or even Unix sockets, which we have implemented in a library called POSSE [Ref. 11 - 14]. In addition, there are readily available C++ libraries for accessing serial and parallel ports. Java is said to be portable, but it is difficult to really claim that it is more portable than C++. Thread programming is a part of the Java language and not part of C++, however POSIX thread libraries are readily available for C++. So you could do what we have done here with C++, but it would require several of the above libraries.

When Java was created there was a huge amount of fanfare. This is typical of new technology, as shown in the Gartner Hype Cycle [Ref. 15] in Figure 4. In our opinion, Java is roughly in the "slope of enlightenment" period. Java is a very good programming language, but it has been a bit oversold. Microsoft has not helped the situation by continually arguing with Sun over its use and implementation. Java has also suffered from performance issues, but some recent compilers have demonstrated that it can achieve very good performance [Ref. 16 - 17]. The Java Grande Forum [Ref. 18] has some good suggestions for how Java can be improved, these are mainly related to how Java performs floating point operations. One of the good features of Java is its memory management and automatic garbage collection, but this is a drawback for real-time applications. This has led to a Real-Time Java implementation [Ref. 19], but there are still concerns about whether this implementation can be used in safety-critical applications. There is also the issue of the need for Java to work well in legacy systems, and to interact reliably with C++ and Ada. C++ is not perfect either, since it includes features left over from C. In safety-critical systems it is still very difficult to develop highly reliable code without memory leaks or unpredictable behavior.

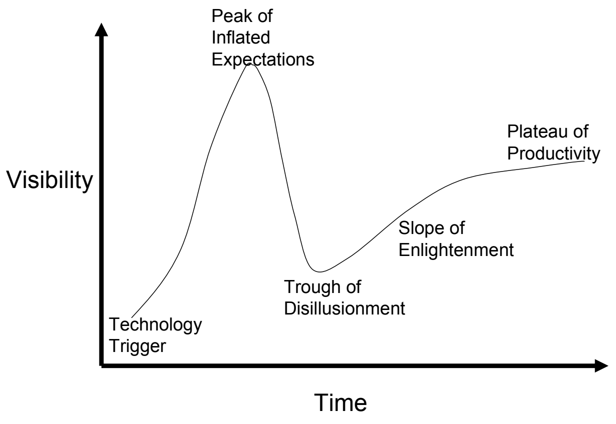

Figure 4. Gartner Hype Cycle [Ref. 15]

One of the amazing aspects of this project is that the entire code is only 730 lines long. This includes the servo control, the remote method invocation, the GUI, and the camera source. This is possible because so much of the functionality that we needed was already part of Java. In addition, the clients and servers can be completely different types of machines and operating systems.

The Client-Server Program

The second part of the project was to be able to communicate with the onboard computer on the vehicle in real time. This was achieved by Java Client-Server programming. The vehicle is the client and the server machine can be any computer on the network. The server program is always listening to calls from the client and responds appropriately. Java provides Remote Method Invocation (RMI) for invoking remote method calls on different Java Virtual Machines (JVMs). This is illustrated in Figure 5. RMI is similar to CORBA, and allows one to use remote objects as though they were local. The client-server programming essentially requires two programs and an interface. The two programs are for the client and the server and they may run on different JVMs. The interface declares the remote methods which can be called on a remote object.

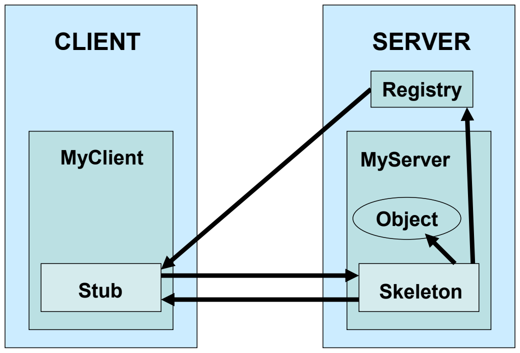

Figure 5. Java Remote Method Invocation (RMI) [Ref.6]

The client-server interface has two methods (or functions): (1) ``getThrottle'' : to fetch the throttle value from the remote object (server), and (2) ``getYaw'' : to fetch the yaw value (for steering). The interface only declares the remote methods, it does not define the action they perform. This interface has to extend the java.rmi.Remote interface to inherit the basic features of RMI. Also, each remote method declared in this interface has to throw a java.rmi.RemoteException. This is required because while working on networked machines there is always a possibility of a connection failure which has to be caught or thrown. One should also note that only the methods defined in the remote interface can be used with the remote objects.

The Server Program

The GUI is coupled to the server so the client (vehicle) does not have to worry about how the maneuvering is done - it can be manual or programmed. The GUI only interacts with the server which is on the same machine and hence the network is not clogged by just transferring data for Graphical interfacing. The only data which runs through the ``external'' network is the value of throttle and yaw (stored as integers) (and the camera image), which are read at a specific rate. The integer values of throttle and yaw are the ``private'' members of the object which is accessed by the client through the remote methods declared in the interface.

As shown in Figure 6, the Graphical user interface provides two controls to the user : (1) throttle value which can range from -100 units (reverse) to 100 units, and (2) yaw value which ranges from -45 to 45 degrees. The user specifies the value by sliding the ``JSliders'' provided by the {javax.Graphics} class. The event handlers (mouse motion listeners) are embedded into the sliders which listen to any ``drag'' events from the mouse. Thus when the user drags the throttle slider, the event handler associated with throttle changes the value of throttle in the server object, and similarly with the yaw (steering) control. An onboard camera image appears inside the GUI also when the vehicle is running.

Another important thing the server has to do is to register itself in the ``rmiregistry''. The rmiregistry is like a directory of remote objects. It contains the addresses where the remote objects are located in the server. When a client has to fetch the server object (the remote object), it contacts the rmiregistry, looks up the address and fetches the object from that address. The server is registered with the name ``Server'' on the localhost (IP = 127.0.0.1). Once registered, a server can be accessed using the protocol:

``rmi://machineIP/ServerName''.

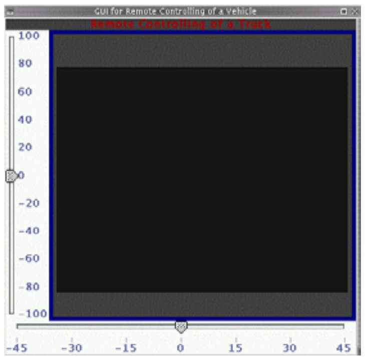
GUI for Remote Controlling of a Vehicle
Remote Controlling of a Truck
100
80
60
40
20
0
-20
-40
-60
-80
-100
-45
-30
-15
0
15
30
45

Figure 6. Simple Graphical User Interface (GUI) for Vehicle Control

The Client Program

The client program requires two different communications: (1) with the server to get throttle and yaw values, and (2) with the hardware (servos). The server program needs to do two things: (1) establish contact with the client, and (2) get a remote reference to the remote object so that it can be accessed as a local object. The contact is established by contacting the rmiregistry running on the server and locating the remote object. This is done using the following.

```
 Server server =
  (Server)Naming.lookup("rmi://host/Server");
```

The above actually does both tasks at once. It contacts the server and also stores a reference to the remote object in ``server" which is of the type ``Server" (interface). Once we have a remote reference to the server object we can access its members which are ``throttle" and ``yaw" using the methods declared in the remote interface. "getThrottle()" and "getYaw()" are used to achieve this. Now that we have the values of throttle and yaw available remotely, we can write them to the servos and thus have our vehicle running. This, however, is not a trivial task.

We create an object of type "Servo" which provides two methods to write to the two servos associated with throttle and steering control. Then we pass the values of throttle and yaw to these methods and the methods write them to the respective servos. The process of accessing the control parameter values and writing them on the servos continues in an indefinite loop until the user shuts the connection or there is some error in connecting to the server. This constitutes a fully functional client-server program and a remote controlled mobile robot.

To compile and run the programs you need to compile the client, server, and implementation programs using javac. Then the stub and skeleton class files are created by running the implementation class file through rmic. The stub is actually a client side proxy for the remote object. The skeleton is a server-side function that actually interacts with the remote object. You also need to start the rmi registry on the server-side (using "rmiregistry &" in Linux and "start rmiregistry" in MS Windows). The final step is to run the server code on the server, and the client code on the client. You may also have to start the webcam software.

Serial Port Interface

To control the servo motors we use the javax.comm library [Ref. 4] by passing messages to them through a serial port. This library can also be used with a parallel port, and could be used for a wide variety of applications including printers, scanners, bar-code readers, etc. The javax.comm library provides an event-style interface based on the Java event model. Figure 7 illustrates the various interface layers required. This portion of the project was tested by writing a sample program where the user inputs a servo position (in degrees), and the servo responds by rotating through the prescribed angle. A couple of methods were then added to this Java class which take the angle as a parameter and it was used to rotate the servos. These servos were then connected to the throttle and the steering control

In between the servo motors and the serial port we use a Mini Serial Servo Controller (SSC II) [Ref. 5]. Figure 8 shows a Mini SSC II, which can be connected to up to eight servo motors. It connects to the serial port through a standard phone jack and is powered by a 9 volt battery.

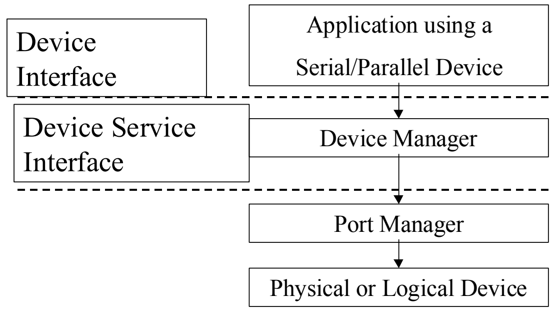
Device Interface
Application using a Serial/Parallel Device
Device Service Interface
Device Manager
Port Manager
Physical or Logical Device

Figure 7. Javax.comm library interface layers

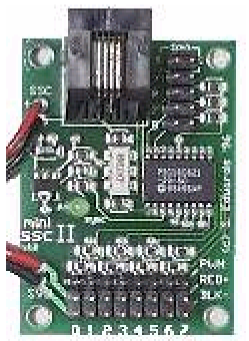
Figure 8. Mini SSC II (Ref. 5)

Here we describe some basic programming aspects regarding the javax.com library for serial/parallel port communication.

```
        CommPortIdentifier portId;
        Enumeration portList;
        SerialPort serialPort;
```

The above three variables are essential to using the serial ports via the javax.com library. portId and portList provide the program with the list of existing ports to be used for serial or parallel communications. serialPort is an extension of the abstract class CommPort that contains all necessary attributes to communicate with serial or parallel devices. The process required to open a data stream with one serial port in the servoControl code is described below:

```
portList=CommPortIdentifier.getPortIdentifiers();

while (portList.hasMoreElements()) {
  portId=(CommPortIdentifier)portList.nextElement();
     if(portId.getPortType()==
          CommPortIdentifier.PORT_SERIAL) {
        if (portId.getName().equals("COM1")) {
          try { serialPort = (SerialPort)
                 portId.open("ServoControl", 2000);
          } catch (PortInUseException e)
```

In a typical java try/catch procedure, a list of available ports is returned to the program via the enumeration portList of possible ports. The next step is to associate an application ServoControl to a serial port. The targeted serial port has a type (PORT_SERIAL) and a name (COM1 on Windows, /dev/tty on Solaris). This application name assigned during the opening phase of the port combined with the 2000 millisecond time-out allow several applications to use that same port. The synchronization process takes place by associating a synchronization byte to the port.

The routine handling the yaw control of the truck is listed below (a similar function exists for the throttle control). The servo to which the port is connected has to be identified (here the servo motor responsible for the yaw angle of the truck) and the amount of yaw (messageToPort) is also output to the port as a byte value (from 0 to 256, corresponding to a ± 45º yaw angle).

```
public static void writeToYawServo(byte
          messageToPort, byte servoYaw_idvalue) {
     try {outputStream.write(message_sync);
          } catch (IOException e) {}
     try {outputStream.write(servoYaw_idvalue);
          } catch (IOException e) {}
     try {outputStream.write(messageToPort);
    } catch (IOException e) {}
}
```

Video Images

The mobile robot also has a camera onboard so the remote user can see the view in front of the vehicle. For this we use a standard PC web camera. It is possible to capture the images in Java on the client and then send the images to the server using Java, however currently we use a simpler approach. We used a simple webcam package (e.g. Webcam2000) which continually grabs the image from the camera and saves it to disk. You could also install the Apache webserver [Ref. 7] on the onboard computer (on the robot). The camera can store images at a frequency of about one per second. The image is stored in a public directory which can be accessed from remote machines using the HTTP protocol. On the server side, Java was used to continuously poll the client images. This requires creating a URL object (in a try-catch block), and then using

```
 image =
   Toolkit.getDefaultToolkit().createImage( url );
```

in MediaTracker. The image can be drawn in the server GUI using the drawImage method. This code was run in a separate thread in the server implementation code. Thus the images can appear on a server thousands of miles from the client (robot).

The update speed of these images is fairly slow, so the above approach is only suitable for low-speed vehicles. We are currently working on incorporating more robust vision systems, including stereographics. Stereo camera systems can be obtained from SRI [Ref. 8] and other vendors. High resolution graphics requires fairly high bandwidth networking or robust compression algorithms. A video that runs at 30 frames/second at 1024x768 resolution and 24-bit color would require roughly 566 Mbits/second without compression. This is higher than the peak speed of Fast Ethernet, and might be more than is possible using gigabit ethernet. The peak speed of wireless Ethernet is only 11 or 55 Mbits/second, for IEEE 802.11b and 802.11a, respectively. So an 802.11b network can only sustain, at most, about 8 frames/second at 320x240 resolution and 8-bit color.

Future Hardware Platforms

While Java is not perfect and has been somewhat oversold, it has a very natural place in the embedded systems market. It may soon be in the "plateau of productivity" regime, and serve a very important role. We believe the pace of Java acceptance will accelerate due to the recent release of several new Java processors. Many of these run Java code in native mode without the need for an operating system. Table 1 shows several different embedded processors that are now available. To use many of these you would also need a development kit or board, which typically costs a few hundred dollars. Website addresses are given, in case the reader would like more information. The processors cover a wide range of performance, price, and power consumption ranges. All can run Java, but some (such as the OOpic) seem to stray fairly far from the standard. The table shows nominal power consumption for each one also, but this would vary dramatically depending on what other devices are connected to the processors (especially an RF antennae). The power consumption is a very important item for some applications such as micro air vehicles, since battery weight often dominates those designs.

The Javelin Stamp is very interesting, but it runs a subset of Java. So it would be difficult to port some existing Java code to it. The main limitations of the Javelin Stamp environment are [20]:

- Single Thread
- No Garbage Collection
- Subset of Primative Data Types
- Subset of Java Libraries
- Strings are ASCII
- No Interfaces
- One Dimensional Arrays

Even though the user cannot implement threads, the Javelin does have several Virtual Peripherals that do run as separate threads, and these are very useful. Some of these are: pulse width modulation (PWM), analog to digital (ADC), digital to analog (DAC), and serial port interfaces (UART). You can use up to six of these. The absence of garbage collection is probably a good thing for embedded systems. The primitive data type limitations are important to mention. Variables declared as int are 16 bits, so they can vary from -32,768 to 32,767. Also, there are no floating point variables, which will make it difficult to implement some algorithms onboard.

While some of the Java processor boards include Ethernet (e.g. TINI), the others do not. And none of them currently include wireless Ethernet. There are, however, RF devices available. For example, there are transceivers from RF Digital that cost less than $100.

One of the authors (Long) has programmed a small three-wheeled robot using the Javelin Stamp, which is shown in Figure 9. It was fairly straight-forward to program robot motion and object avoidance with infrared emitters and detectors. In the past we have programmed these using BASIC, and the ability to program in Java makes an enormous difference. The code is much more capable and maintainable. These small embedded processors are excellent candidates for Java.

Processor	**Manufacturer (website)**	**Speed**	**Memory**	**Size (cm x cm)**	**Ether-net**	**Power (mA)**	**Java**	**Cost ($)**
OOPic	Savage Innovations (www.oopic.com)	2 KIPS	0.2 KB	9 x 5	No	10	Yes	49
Javelin Stamp	Parallax (www.parallaxinc.com/)	8 KIPS	64 KB	3 x 1.5	No	50	Yes	89
JStamp	Systronix (www.jstamp.com) (www.ajile.com)	3 M Byte codes/sec.	2.5 MB	5 x 2.5	No	80	Yes	149
TINI Board	Dallas Semiconductor www.ibutton.com/TINI/)	40 MHz	1 MB	10 x 3	Yes	250	Yes	67
PC-104	JumpTec (www.adastra.com)	266 MHz	128 MB	10 x 10	Yes	1400	Yes	800

Table 1. Summary of some Java-capable processors.

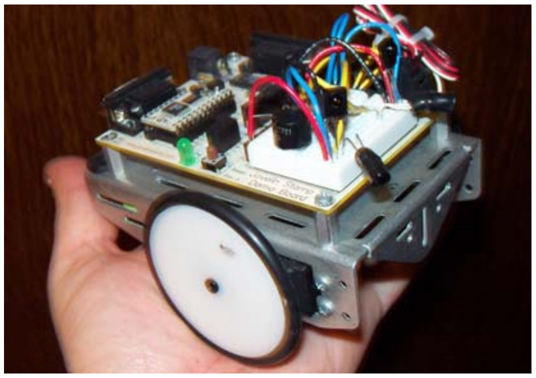

Figure 9. Three-wheeled mobile robot with Javelin Stamp processor and infrared sensors.

Conclusions

We have created a fairly functional remote controlled vehicle. The remote control can be done from anywhere in the world using a computer on the internet. The remote control is wireless due to the wireless ethernet card in the vehicle. Some very basic controls are currently provided to the user, as this is just an example to show how client-server programming can be used with embedded devices to achieve control of vehicles. Java is shown to be a very useful language for this application due to its ability to do client-server programming, GUI's, graphics, object oriented programming, and multiple threads.

With the rapid advances being made in embedded processors and compilers, we will see many more devices using Java.

References

1. http://www.java.sun.com/
2. Horstmann, C. and Cornell, G., "Core Java," Sun, 1999.
3. https://www.zagrosrobotics.com/
4. http://java.sun.com/products/javacomm/javadocs/Package-javax.comm.html
5. http://www.seetron.com/ssc.htm/
6. http://java.sun.com/products/jdk/rmi/
7. http://www.apache.org/
8. http://www.ai.sri.com/~konolige/svs/
9. http://www.fltk.org/
10. http://www.opengl.org/
11. Modi, L. Long, N. Sezer-Uzol, and P. Plassmann, "Scalable Computational Steering System for Vizualization of Large-Scale CFD Simulations," AIAA Fluids Conference, AIAA paper 2002-2750, St. Louis, 2002
12. Modi, L. Long, and P. Plassmann, "Real-Time Visualization of Wake-Vortex Simulations using Computational Steering and Beowulf Clusters," Parallel Computing Conference, Portugal, 2002
13. Modi, A. “Real-Time Visualization of Aerospace Simulations using Computational Steering and Beowulf Clusters,” Ph.D. Dissertation, Penn State Univ., Aug., 2002.
14. http://posse.sourceforge.net/
15. http://www4.gartner.com/
16. Genovesi, D. and Long, Lyle N., “An Object Oriented Approach to the Direct Simulation Monte Carlo Method,” Java Grande, Nov., 2002.
17. Genovesi, D., “An Object Oriented Approach to the Direct Simulation Monte Carlo Method,” M.S. Thesis, Penn State Univ., May, 2002.
18. http://www.javagrande.org/
19. http://www.rtj.org/
20. http://www.javelinstamp.com/docs/manual/Javelin_Stamp_Manual_v1.0a.pdf