NETWORK CONTROLLED ROBOT

ANUSHA K.

CHITRA VIJAYAN

DEEPA B. NAIR

FATHIMA JIHAN

IHSANA MUHAMMED

CHAPTER 1

INTRODUCTION

1.1 CURRENT SCENARIO

The world as it exists today os becoming more secure. The technology is being used to make sure that human lifes are as secure as possible. The security systems include a lot of areas of the engineering desciplines. Computer science has also played a key role in security related matters. Data security is the main among them. It also stands as a key support in implementing various security measures by making things much more easier to do.

The national security is actively handled mainly by the military organisations. Theses organisations have extensively used technological advancements a lot. Organisations like DRDO is working on high end technologies that shall further enhnace the security of the nation and its civilians.

One of the important inventions is the remote survelence using robot. Robots are also used to do lot of risk assosiated jobs remotely. Currently what they do is connect a robot to a computer using a wired or a wireless connection. This computer has a terminal to control the functions of the robot. When a camera is attached to it at various positions, the controlling person can have a more presice control over the robot.

1.2 NEED FOR THE SYSTEM

Current application is indeed already a lot complicated. But what if a war scenario comes where in the person controlling the robot is not qulified enough or matured enough to make decisions? The control has to be transfered to a senior officer not physically present in the area. This shall involve establishing a network connection that transfers the commands from the client present far away from the scene to the server that is physically closer to the roobot.

But why do we need another server in between the robot and the controlling computer? The reason is to abstract the details regarding the specific robot from the controlling person's software. If a robot fails and a more sophisticated robot has to take in charge, the server program alone needs to be upgraded, that shall provide the same commands to the client software. This is indeed an advantage in the war scenario where in the robots robustness is often challenged.

1.3 FEATURES OF THE SYSTEM

Network Robot Controller consists of a platform-independent application in Java which allows remote control of a robot along with video streaming from a camera as video source mounted on the robot. The robot is connected to computer via parallel or serial port. In case of using serial port the robot has a controller in it that controls the motion of the robot based on the input received from the user. The user is provided with a GUI that shows the video that is streamed from the camera mounted on the robot. Also the GUI has controlling buttons that may be used to control the motion of the robot.

CHAPTER 2

SYSTEM STUDY

2.1 EXISTING SYSTEM AND ITS DRAWBACKS

The current system already has complicated controlling mechanism that controlls the robot connected to the system. The system that connects to the robot is well aware of the capabilities of the robot. The drawback is that in a case where the functionalities are too many for a single person to handle, there is no way for more than one person to send data to the robot.

Limitation is also present in the field of distance between the robot and the controller. This distance is limitted by the capability of the communication device used. This often results in high cost of implementation either for hardware or for cabling, since re-using the existing cable is not feasible.

2.2 PROPOSED SYSTEM

The proposed system is a client – server architecture in which the server is responsible for the communication with the robot. The client is connected to the network in which the server is present. This way the client communicates with the server computer to control the robot. The detailed description of the client and server softeware is described next.

2.2.1 SERVER

The server is provided with a GUI, that shows the client system address and port from which the request is coming. It also has a text area showing the progress of events at the client side. This is inferred from the commands send by client system. The GUI in server side is just for aiding in the processing.

Socket in the server side receives data send from the client side. In the server program, receive function, called by the server socket instance, will accept data. According to client data, specific cases are declared. Then the command is read. Then for each case, the specific console message is displayed and an eight bit code corresponding to that is outputed through the parallel port. For example, if the command read is control_pressed_up, then an 8 bit code, an integer value, corresponding to that is outputed to the system's parallel port. This will be sent to the robot.

For video streaming, JMF is used. The video from robot is streamed to the network. RTP protocol is used here. In RTP protocol, the encoded video data is made to chunks of data of specific size along with a header containing the encoding and sequence number information and this is encapsulated in a UDP packet and is given to the network. The Manager implements an instance of Media Locater. The Media Locater, like a URL or a file, locates a data source and will act as a data sink accepting the video data from the camera mounted on robot acting as video source. Then the Media Locater finds proper destination for the video packet.

In JMF, the main process includes an input system, a processing system and an

output system. The format of the data coming as input and output must be known. The input system takes data in its source format. Here the media frame is taken by the input system. In the processing state the format of data is converted to the RTP packet format. These RTP packets are given by the output. This processed data is sent to the server socket by the Media Locater taking it as the destination. Then in the next stage, server socket acts as the data source of the processed RTP packets and the Media Locater will take the client socket as data sink. This is how streaming is done.

2.2.2 CLIENT

The client is provided with a Graphical User Interface. The GUI includes a frame having 4 control buttons, UP, DOWN, LEFT & RIGHT .Another frame is provided to show the streamed video, from the camera mounted on robot acting as video source. By viewing the video from robot, user can give proper commands to control robot locomotion accordingly. When the UP button is pressed, the robot is commanded to move forward, DOWN to move backward, LEFT & RIGHT to move to left& right respectively.

The GUI can easily be implemented using SWING. The GUI is event driven. ie; the pressing& releasing of each of the control button is considered as an event. On the occurrence of an event, for eg. if UP button is pressed, that event is binded to the event function code, where corresponding action to be taken is specified.

In the client program, for every event say a UP button pressed & then released. The UP button release event will be bind to the code present in a function Release. There the code can specify to print that message. Then a variable or an integer can be

passed as an argument, say UPR, while calling a function to out the command through a socket at the client side. The client is provided with a socket to communicate with the server socket. An instance of the Socket class is created & that object does the send function, accepting the argument UPR will return a variable or an integer to be sent to the server as data. For each event, a distinct integer or variable is assigned while programming and this is the data that is passed to the server.

If a UDP socket is used, then in order to send the data, sendto() function is used. It has arguments the client socket descriptor, data buffer, size of data buffer, flag, destination address & its size. The destination address includes server system address & the port address of the server socket. Thus from the client socket, the data in a buffer can be send to the server socket. This data is interpreted by server program & correspondingly an 8 bit code is assigned for each of them &passed to its parallel port for transmission.

CHAPTER 3

THE ROBOT

3.1 INTRODUTION

The Robot has 2 DC motors connected to the wheels that power the motion of the robot. A motor driver for driving the motor and a power source for driving the motor are also present. The locomotion of robot is as per the instructions given by the user. The commands from client are processed by server and are sent to the robot connected to its port. Robot has a camera mounted on it acting as a video source. The video of its surroundings is streamed to the server & is then sent to the client. The client sends control information to move the robot according to the video provided by this camera mounted on the robot.

3.2 INTERFACING ROBOT WITH PC

A robot can be connected to a PC in two ways. We can either connect it through the PC's serial port or through its parallel port. Here we are connecting the robot to PC's parallel port, for easier implementation. Let's have a look at both of them.

In serial port communication we can transfer only one bit at a time. In this type of communication, we have to use a voltage converter & a micro controller along with motor driver and DC motors in the robot. The data from the serial port of the system is interpreted by the microcontroller. Atmega 8L microcontroller can be used for this purpose. The information like baud rate, number of bits/frame and the parity

information are needed to be specified in both Server as well as in microcontroller programming.

For proper data transmission, the voltage range of (-12)V - 12V in serial port must be converted into the microcontroller input range of 0V - 5V. For this purpose MAX-232 voltage converter is used. The motor driver L239D is used to convert the voltage signals from microcontroller output and passes it to 12V DC motors of 100 rpm connected to it. This drives the motors.

Unlike serial port communication, through parallel ports the whole 8 bits can be sent in parallel. This is one of its greatest advantages as it provides transmission 8 times faster! Also parallel port communication makes the robot circuit much simpler. It is because, while using parallel port communication there is no need of using a microcontroller for interpreting data from the server. The data can be directly fed into the motor driver and thus can drive the motor easily. The parallel port data output voltages are in the range of 0V – 5V. Thus it also eliminates the requirement of a voltage converter, making the robot circuit much simpler. This is the reason why we are interfacing the robot to server's parallel port.

3.3 ROBOT LOCOMOTION

The robot is a three wheeled object that has two DC Motors at the rear and a simple wheel in the front. The DC Motor is a simple motor that works at 12V. The turning of the robot to the left and right may be attained by turning only one of the rear wheels at a time. When the left wheel is alone rotated the robot turns to the right and when the wheel in the right is turned the robot turns to left. To attain better

turning we can turn the second DC motor in the opposite direction. For example, if we want to turn right, we will rotate the left wheel forward and the right wheel backward. The two DC Motors are controlled by a motor driver and that is connected to the parallel port of server. The forward and the backward motion of the robot is simply done by rotating the two rear wheels together in the required direction.

3.4 ROLE OF MOTOR DRIVER

The data coming from the parallel port is fed to the motor driver. It drives the motors according to the input given. Here we are using motor driver L293D. The L293 is an integrated circuit motor driver that can be used for simultaneous, bi-directional control of two small motors. It comes in a standard 16-pin, dual-in line integrated circuit package. The motors can be moved forward, backward, left or right according to the output of the motor driver.

CONNECTION DIAGRAMS

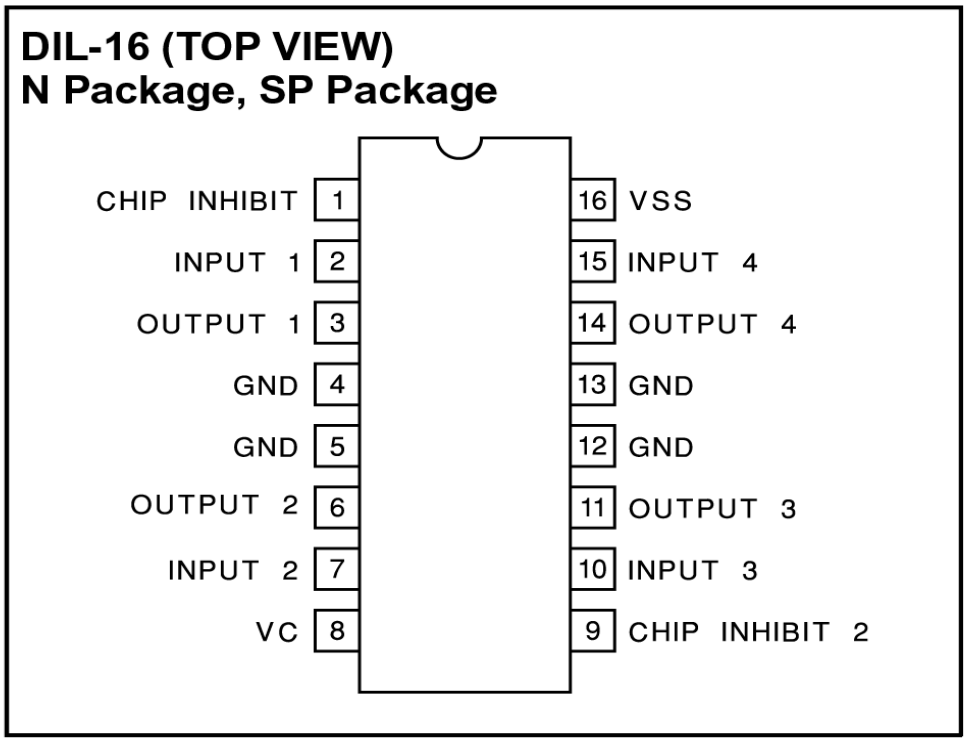


Fig .1 Connection diagram of L239D

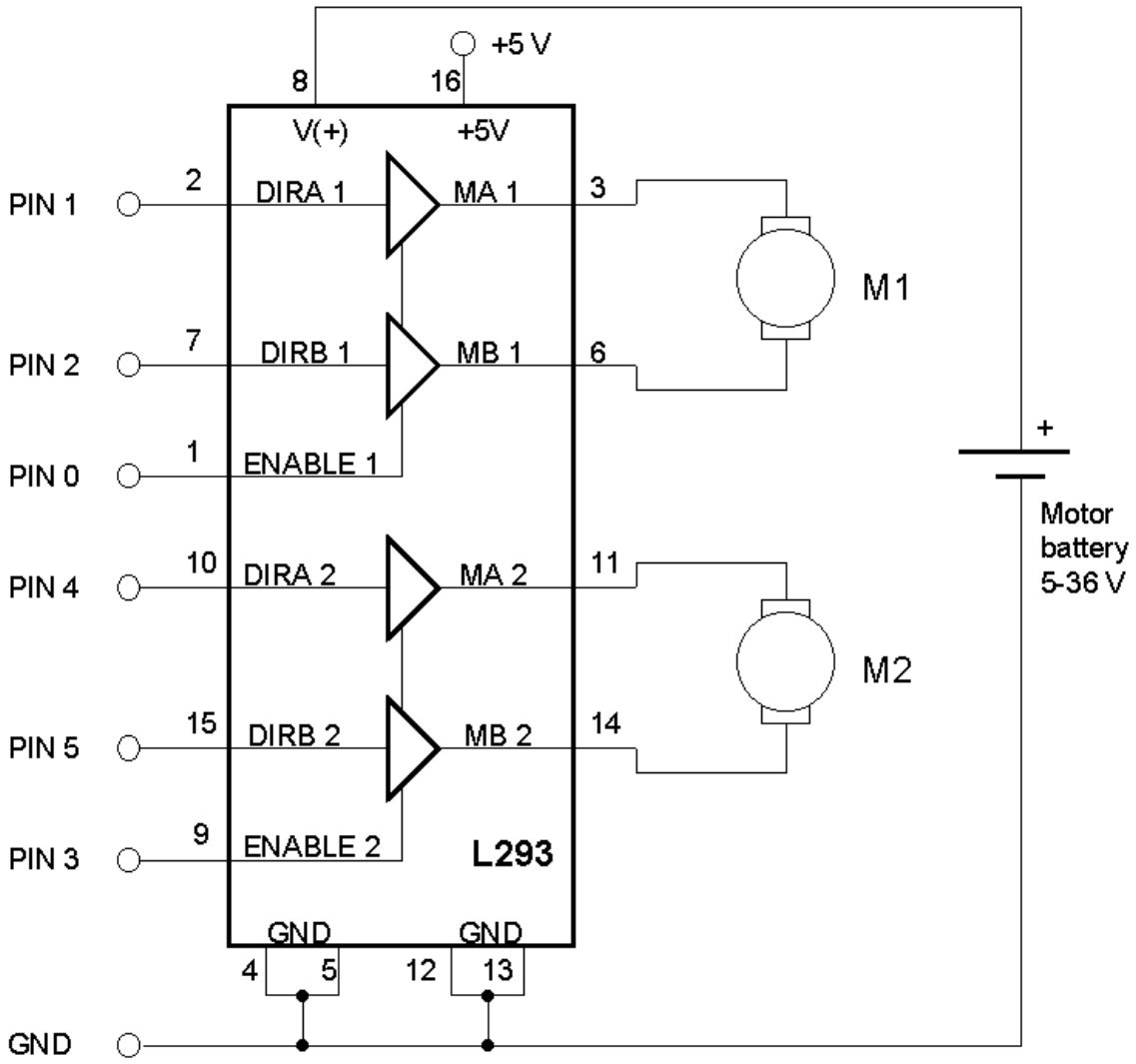


Fig. .2 Internal circuit of L239D

This is a figure showing how the pins are connected to the two dc motors. Consider the case of controlling motor M1. Making ENABLE pin High, if the DIRA1 is made high & DIRB1 is made low, then M1 turns right. Keeping ENABLE High, if DIRA1 is made Low & DIRB1 is made High, then M1 turns left. This is how the motor driver controls the robot's movment direction.

CHAPTER 4

DC MOTORS

4.1 INTRODUCTION

Two 12V, 100 RPM, DC geared motors are connected to the wheels of the robot to provide locomotion. The DC motor, operation is based on simple electromagnetism. A current carrying conductor generates a magnetic field; when this is then placed in an external magnetic field, it will experience a force proportional to the current in the conductor, and to the strength of the external magnetic field. NR-DC-ECO is high quality low cost DC geared motor.

It contains Brass gears and steel pinions to ensure longer life and better wear and tear properties. The gears are fixed on hardened steel spindles polished to a mirror finish. These spindles rotate between bronze plates which ensures silent running. The output shaft rotates in a sintered bushing. The whole assembly is covered with a plastic ring. All the bearings are permanently lubricated and therefore require no maintenance. The motor is screwed to the gear box from inside.

4.2 MOTOR SPECIFICATIONS

- Total length: 46mm
- Motor diameter: 36mm
- Motor length: 25mm
- DC supply: 4 to 12V
- RPM: 100

- Brush type: Precious metal
- Gear head diameter: 37mm
- Gear head length: 21mm
- Output shaft: Centered
- Shaft diameter: 4mm and 6mm
- Shaft length: 22mm
- Gear assembly: Spur
- Torque: 0.25 to 7Kg/cm

CHAPTER 5

MOTOR DRIVER - L293D

5.1 DESCRIPTION

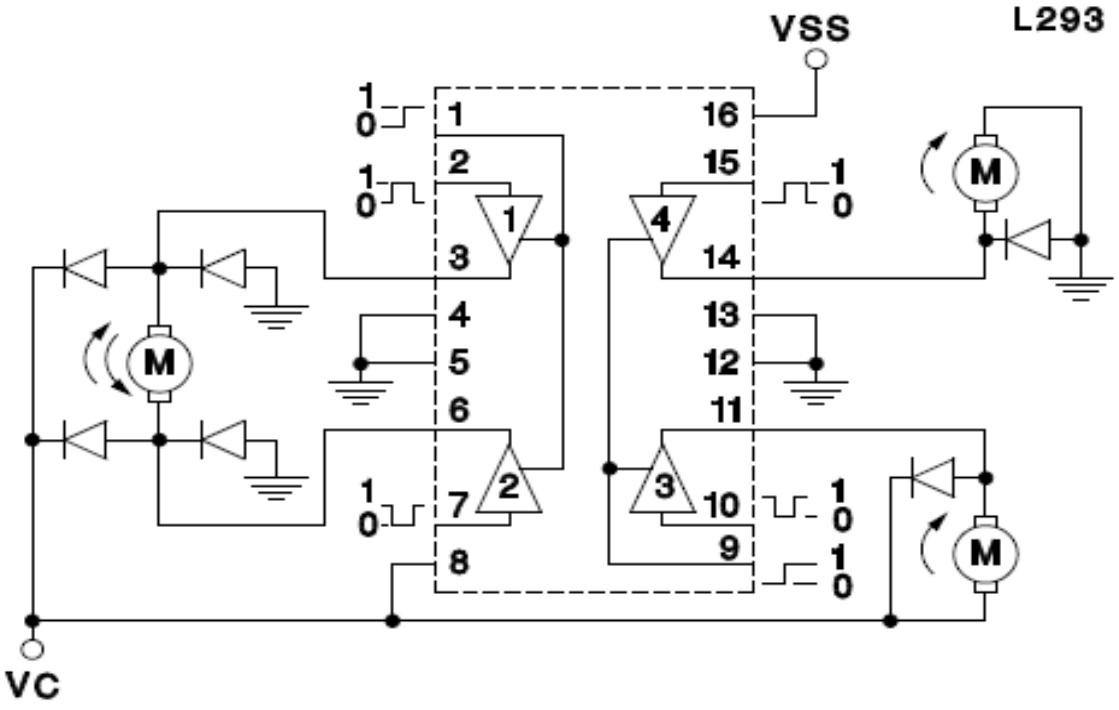
The L293 and L293D are quad push-pull drivers capable of delivering output currents to 1A or 600mA per channel respectively. Each channel is controlled by a TTL-compatible logic input and each pair of drivers (a full bridge) is equipped with an inhibit input which turns off all four transistors. A separate supply input is provided for the logic so that it may be run off a lower voltage to reduce dissipation. Additionally the L293D includes the output clamping diodes within the IC for complete interfacing with inductive loads. Both devices are available in 16-pin Batwing DIP packages. They are also available in Power S0IC and Hermetic DIL packages.

The Device is a monolithic integrated high voltage, high current four channel driver designed to accept standard DTL or TTL logic levels and drive inductive loads (such as relays solenoides, DC and stepping motors) and switching power transistors. To simplify use as two bridges each pair of channels is equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included. This device is suitable for use in switching applications at frequencies up to 5 kHz. The L293D is assembled in a 16 lead plastic packaage which has 4 center pins connected together and used for heatsinking.

5.2 FEATURES

- Output Current 1A Per Channel (600Ma for L293D).
- Peak Output Current 2A Per Channel (1.2A for L293D).
- Inhibit Facility.
- High Noise Immunity.
- Separate Logic Supply.
- Over-Temperature Protection.

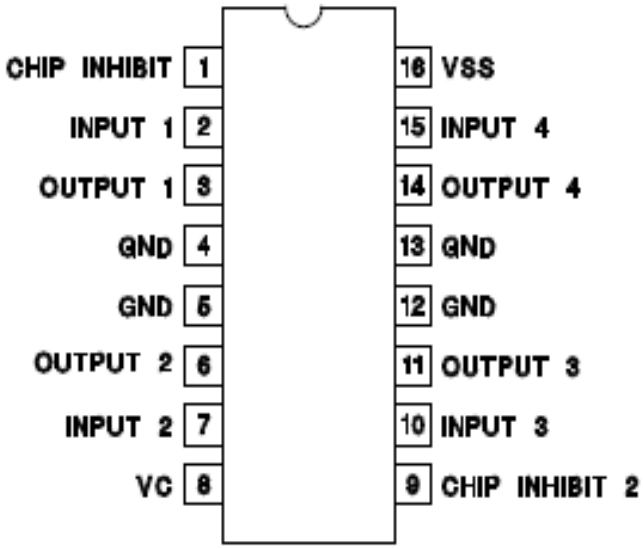
5.3 BLOCK AND CONNECTION DIAGRAM



Fig

CONNECTION DIAGRAMS

DIL-16 (TOP VIEW)
N Package, SP Package



CHAPTER 6

JAVA MEDIA FRAMEWORK

6.1 ARCHITECTURE

The JMF component architecture is very flexible, and its components can generally be classified in three groups:

• **Input** describes some sort of media that is used as an input to the rest of the process.

• **A processor** performs some sort of action on the input. A process has a distinct input and output. A large number of processes are available, and can be applied to an input or a group of inputs. These processes can be chained together so that the output from one process is used as an input to another process. In this manner multiple processes may be applied to an input. (This stage is optional -- our first two examples contained no real data processing, only an input from a file and an output through the Player.)

• **Output** describes some sort of destination for media.From this description, you might be thinking that the JMF component architecture sounds quite like that behind a typical stereo or VCR. It is easy to imagine using JMF in a manner similar to that of turning on a television or adjusting the sound on a stereo.

For example, the simple act of recording a favorite TV show can be thought of in these component-based terms:

• Input is the TV broadcast stream, carrying both audio and video information on the same channel.

• Process is a recording device (that is, a VCR or any number of digital devices) converting the analog or digital audio and video broadcast stream into a format suitable for copying to a tape or some other media.

• Output is the device writing the formatted track (audio and video) to some type of media.

6.2 PLAYING VIDEO

One of the great features of JMF is that you don't need to know anything about the media file types in order to configure a media player; everything is handled internally.The main difference in dealing with video media rather than audio is that we must create a visual representation of a screen to be able to display the video. Luckily, JMF handles many of these details for us. We will create a Player instance, and obtain many of the visual components to create our visual media viewer directly from JMF objects.

Video player example:

The video player example is designed to run from the command line, but this example is GUI based. We start by invoking the application and passing in the file name of the media. Next, the application displays a window with components that let us manipulate the media.

In the opening lines of the MediaPlayerFrame we define the class and extend the javax.swing.JFrame class. This is how we define our media player as a separate window on the desktop. Any client that creates an instance of our media player class may then display it by using the show() method defined on the JFrame class.

Below is a sample screenshot showing the MediaPlayerFrame playing an MPEG movie:

Fig

6.3 WORKING WITH VISUAL COMPONENTS

All of the above interface methods return an instance of the java awt component class.Each instance is a visual component that may be added to our frame. These components aretied directly to the Player, so any manipulation of visual elements on these components will cause a corresponding change to the media displayed by the Player.

It is important that we ensure each of these components is not null before we add it to our frame. Because not every type of media player contains every type of visual component, we should only add components that are relevant for the type of player we have. For instance, an audio player generally does not have a visual component, so getVisualComponent() returns null. You would not want to add a visual component to the audio player frame.

6.4 MEDIA LOCATOR

The last big difference between our new GUI-based media player and our first simple player is that we'll use a MediaLocator object rather than a URL to create the Player instance, as shown below:

```
public void setMediaLocator(MediaLocator locator) throws IOException,
  NoPlayerException,CannotRealizeException{
                                    setPlayer(Manager.createRealizedPlayer(locator));
    }
```

We'll discuss the reason for this change in a later section. For now, think of a diaLocator object as being very similar to a URL, in that both describe a resource cation on a network. In fact, you may create a MediaLocator from a URL, and you may t a URL from a MediaLocator. Our new media player creates a MediaLocator stance from a URL and uses that to create a Player over the file.

6.5 JMF PROCESSORS

When we are working with JMF, the processor component of the application is

represented by an instance of the Processor interface. You should already be somewhat familiar with the Processor, as it is an extension of the Player interface. Because the Processor inherits from the Player interface, it also inherits all of the valid states from the Player. In addition, the Processor adds two more states: Configuring and Configured. These extra states (and associated events) are used to communicate when the Processor is gathering information from the input stream.

For our final example application, we will create a Processor to convert audio encoded in the MP3 format into a format suitable for broadcast over a network. We will discuss the steps to create simple Processor in a later panel.

6.6 DATA SOURCE AND SINK

There are a few ways to represent the output phase of the JMF process model. The (simplest and the one we will use in the final example)is the javax.media.DataSink interface. A DataSink reads media content and renders it to some destination. In the audio-format conversion scenario at the beginning of this section, the MP3 (output) file would be represented by the DataSink. In our final example, we will use a DataSink to actually do the work of broadcasting audio media over a network. A DataSink is created through the Manager class by specifying a DataSource (the input to the DataSink) and a MediaLocator (the output of the DataSink).

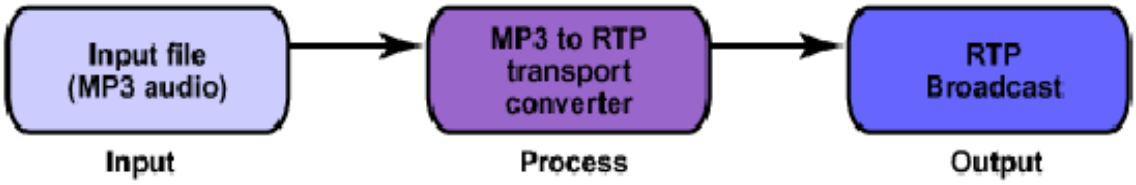
A DataSource instance represents input data, which is used in Players, Processors, and DataSinks. The output of a Processor is also represented as a DataSource object.This is how Processors can be chained together to perform

multiple operations on the same media data. This is also how the output from a Processor can be used as input to a Player or to a DataSink (which would render the media to an output destination).

The final destination of a DataSink is specified by a MediaLocator object. Just as before, the MediaLocator represents a network resource; that is, it's where the media stream will be rendered.

6.7 MEDIA TRANSMITTER

On the transmitting machine, the process model would look something like this:



In fact, the source code for the MediaTransmitter object contains the following three lines:

```
private MediaLocator mediaLocator = null;
private DataSink dataSink = null;
private Processor mediaProcessor = null;
```

These three instance variables may be mapped directly into the above process model diagram, as follows:

• The mediaProcessor variable is our processor; it will be responsible for converting our audio media from the MP3 file format to a format suitable for transmission over the RTP protocol.

• The dataSink variable is our output block.

• When we create the DataSink we will specify a MediaLocator, which is the actual destination of the DataSink.

When we run our processed media through the DataSink, it will be transmitted to whatever location(s) we have specified in the MediaLocator.

6.8 JMF AND RTP

Many network-friendly features are built directly into JMF, which makes broadcasting and receiving media over a network very easy for client programmers. When a user on a network wants to receive any type of streaming media, he shouldn't have to wait for the entire broadcast to download to the machine before viewing the media; the user should be able to view the broadcast in real time. This concept is referred to as streaming media. Through streaming media, a network client can receive audio being broadcast by another machine or even intercept a live video broadcast as it is happening.

The Real-Time Transport Protocol (RTP) is defined in IETF RFC 1889. Developed to carry extremely time-sensitive data over a network in a quick and reliable manner, RTP is used in JMF to provide users with a way to transport media

streams to other network nodes.

In this section, we'll walk through our final example application. Here, you'll learn how to broadcast an MP3 file stored on one machine to other machines in the same network. The actual MP3 source file never leaves the host machine, nor is it copied to any other machine; rather, it will be translated into a format that can be broadcast using RTP and sent over the network. Upon being received by a client, the source file (now in the form of RTP packets) may again be translated, this time to a format suitable for play on the receiving machine.

An RTP MediaLocator conforms to the following form, which looks like a typical URL:

rtp://address:port/content-type

Let's look at each piece of the above URL specification:

• address is the address to which the media will be transmitted. To transmit in unicast mode(to one specific IP address), the address should be the IP address of the intended receiving machine. To transmit in broadcast mode (to all machines within a subnet), the address should be the subnet address with 255 as the last section. For example, if I were on the subnet denoted as 192.168.1 and I wanted to broadcast to all nodes, I could specify 192.168.1.255 as the address; this would enable each node in the subnet to listen to the broadcast media.

• port must be a port that has been agreed upon by both transmitters and receivers.

• content-type is the type of streamed media. In our case this will always be audio.

CHAPTER 7

JAVA COMMUNICATION API

7.1 INTRODUCTION

The Java(tm) communications API can be used to write platform-independent communications applications for technologies such as voice mail, fax, and smartcards, embedded systems, and point-of-sale equipment. This download features API documentation in HTML format, reference implementations for the Solaris and Win32 platforms, and sample software. This version of the Java communications API contains support for RS232 serial ports and IEEE 1284 parallel ports.

7.2 FEATURES

1. Javax.comm and gnu.io are released in Linux platform.

2. Extended portmapping to provide control over which ports are exposed in the CommPortIdentifier list, as well as over the names assigned to them.

There are three levels of classes in the Java communications API:

- **High-level** classes like CommPortIdentifier and CommPort manage access and ownership of communication ports.
- **Low-level** classes like SerialPort and ParallelPort provide an interface to physical communications ports. The current release of the Java communications API enables access to serial (RS-232) and parallel (IEEE 1284) ports.

- **Driver-level** classes provide an interface between the low-level classes and the underlying operating system. Driver-level classes are part of the implementation but not the Java communications API. They should not be used by application programmers.

7.3 BASIC SERVICES

The package provides the following basic services:

- Enumerate the available ports on the system. The static method CommPortIdentifier.getPortIdentifiers returns an enumeration object that contains a CommPortIdentifier object for each available port. This CommPortIdentifier object is the central mechanism for controlling access to a communications port.

- Open and claim ownership of communications ports by using the high level methods in their CommPortIdentifier objects.

- Resolve port ownership contention between multiple Java applications. Events are propagated to notify interested applications of ownership contention and allow the port's owner to relinquish ownership. PortInUseException is thrown when an application fails to open the port.

- Perform asynchronous and synchronous I/O on communications ports. Low-level classes like SerialPort and ParallelPort have methods for managing I/O on communications ports.

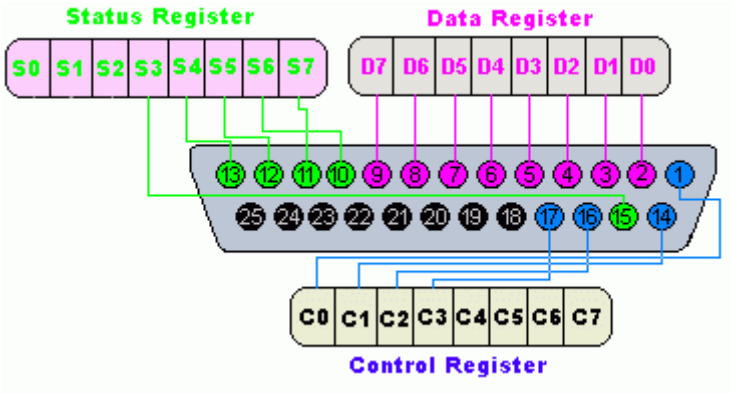
- Receive events describing communication port state changes. For example, when a serial port has a state change for Carrier Detect, Ring Indicator, DTR, etc. the SerialPort object propagates a SerialPortEvent that describes the state change.

7.4 PARALLEL PORT

It is the parallel communications port. Parallel Port describes the low-level interface to a parallel communications port made available by the underlying system. Parallel Port defines the minimum required functionality for parallel communications ports. Parallel port is a simple and inexpensive tool for building computer controlled devices and projects.

7.5 HARDWARE SPECIFICATION

The pin outs of DB25 connector is shown in the picture below:



The lines in DB25 connector are divided in to three groups, they are:

1) Data lines (databus)

2) Controllines

3) Status lines

As the name refers, data is transferred over data lines, control lines are used to control the peripheral and of course, the peripheral returns status signals back computer through Status lines. These lines are connected to Data, Control And Status registers internally . The details of parallel port signal lines are given below

PinNo	Signal name	Direction	Register - bit	Inverted
1	nStrobe	Out	Control-0	Yes
2	Data0	In/Out	Data-0	No
3	Data1	In/Out	Data-1	No
4	Data2	In/Out	Data-2	No
5	Data3	In/Out	Data-3	No
6	Data4	In/Out	Data-4	No
7	Data5	In/Out	Data-5	No
8	Data6	In/Out	Data-6	No
9	Data7	In/Out	Data-7	No
10	nAck	In	Status-6	No
11	Busy	In	Status-7	Yes
12	Paper-Out	In	Status-5	No
13	Select	In	Status-4	No
14	Linefeed	Out	Control-1	Yes
15	nError	In	Status-3	No
16	nInitialize	Out	Control-2	No
17	nSelect-Printer	Out	Control-3	Yes
18-25	Ground	-	-	-

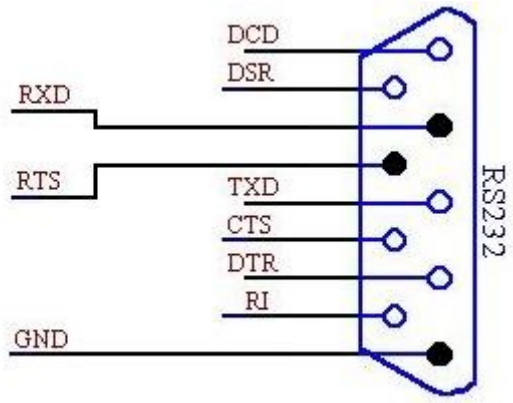
7.6 SERIAL PORT

An RS-232 serial communications port. SerialPort describes the low-level

interface to a serial communications port made available by the underlying system. SerialPort defines the minimum required functionality for serial communications ports. In serial port communication we can transfer only one bit at a time. In this type of communication, we have to use a voltage converter and a micro controller along with motor driver and dc motors in the robot.

The data from the serial port of the system is interpreted by the microcontroller. Atmega 8L microcontroller can be used for this purpose. The information like baud rate, no. of bits/frame and the parity information are needed to be specified in both Server as well as in microcontroller programming.

Serial Port Pin Diagram



From these 9 pins we only use three of the following

RXD: This pin detects the signals generated by the device.

RTS: This pin supplies power to the device

GND: This pin is ground pin.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

A robot controller as mentioned above was successfully built using java as the software programing language, on a GNU/Linux system using parallel port and other hardwares involved. It could successfully be controlled from a network client.

The robot controller has lot of features that may be implemented in it in the future. The main one is the remote control. In order to increase the mobility of the robot we can also implement an RF module in it. Also a lot of applications like in military or in local area surveillance.

REFERENCES

- Java documentations, http://java.sun.com/javadoc/
- Java API specifications, http://java.sun.com/references/api/