A Bluetooth-based Architecture for Android Communication with an Articulated Robot

Sebastian van Delden and Andrew Whigham
Division of Mathematics and Computer Science
University of South Carolina Upstate
800 University Way, Spartanburg, SC 29303
svandelden@uscupstate.edu, whighama@email.uscupstate.edu

***Abstract*—In industrial robotic environments, there are many different robots performing a variety of tasks. Each robot is controlled by its own teach pendant or via a networked socket application. However, to monitor the status or make minor changes to the programming of the robot, the user must obtain access to the teach pendant or terminal. In an effort to eliminate this need, this paper introduces an android platform that communicates with robots over a Bluetooth connection.**

Keywords—Articulated Robotics; Android; Bluetooth; Manufacturing.

I. INTRODUCTION

Many factories and manufacturing plants use various articulated/industrial robotic systems to help humans with repetitive, precise, and/or dangerous tasks. These tasks, for example, range from: the repetition of pick-and-place part handling, to the dangerousness of mass welding, to the precise placement and assembly of sensitive electronic components. The majority of these robotic systems are manually controlled and programmed via a "teach pendant." To a lay-person, the teach pendant can be an intimidating and complex device which diminishes a 'typical' manufacturing plant employee's abilities to modify the robot's functionality. Furthermore, each robot manufacturer provides its own unique and proprietary programming interface. For example, programming in Stäubli's (http://www.staubli.com) current VAL3 environment is different from their previous V+ environment which is far different from Fanuc's (http://www.fanuc.co.jp/eindex.htm) TPP environment. In an effort to rectify these issues, this paper introduces a client/server software architecture which allows Android clients to view, edit, and monitor industrial robotic programming wirelessly via Bluetooth, and which is decoupled from the robotic-specific programming language. The primary benefit of this system is that it would allow a non-expert to adjust the programming of, or interact with, a range of robotic systems using a ubiquitous Android-enabled smart phone or tablet device.

The basic physical architecture for this project is the Bluetooth client/server architecture. An Android enabled device, acting as a client will communicate with a Bluetooth server running on a computer that has a wired network connection with the robot. Each robot in the environment will have its own Bluetooth server running on its own computer. The computer will not only run a Bluetooth server to communicate with the Android device, but also a socket program that is able to send and receive data from the robots.

Currently, much of the research dealing with the wireless communication of robots focuses on the successful navigation and interaction [1-3] of mobile robots across a busy environment or an implementation of a network infrastructure that allows for communication across a large network [4]. Because our research deals with stationary robotic systems, this project allows for greater flexibility of two-way communication between the mobile control-device and the robotic arm. Unlike much of the related research on wirelessly controlling a robot [5-7], our approach presents a more generic model where any robot can be used instead of tailoring the implementation to the advantages of one robot.

The remainder of this paper is organized as follows. Sections II and III will discuss the system architecture of the Android application and Bluetooth server, respectively. Section IV will demonstrate two practical and useful applications of this research, and section V will discuss conclusions and present future research goals.

II. ANDROID APPLICATION ARCHITECTURE

Android development [8-9] is unique in that it completely revolves around "activities." Activities are components of the overall application that provide windows with which a user can interact. An application usually contains several or even many activities that are related to each other. Activities are composed of "views" or graphical user interface components such as buttons and text boxes. These views are called by the activity, usually when it is created. An activity can be called from another activity or it can be designated as the main activity, in which case it is created when the application is initially launched. Each activity has a life cycle. The activity life cycle contains several states including "start," "resume," "pause," "stop," and "destroy." The start state refers to when an activity is about to become visible. The resume state is called when the activity has become visible. The pause state is called when another activity is about to take focus from the activity. The stop state refers to when an activity is no longer visible, and the destroy state refers to when an activity is about to be removed from memory. Each activity must have each of these states implemented. When an activity calls another activity, it executes the pause state which saves the current state of the activity. If the new activity requires any data from

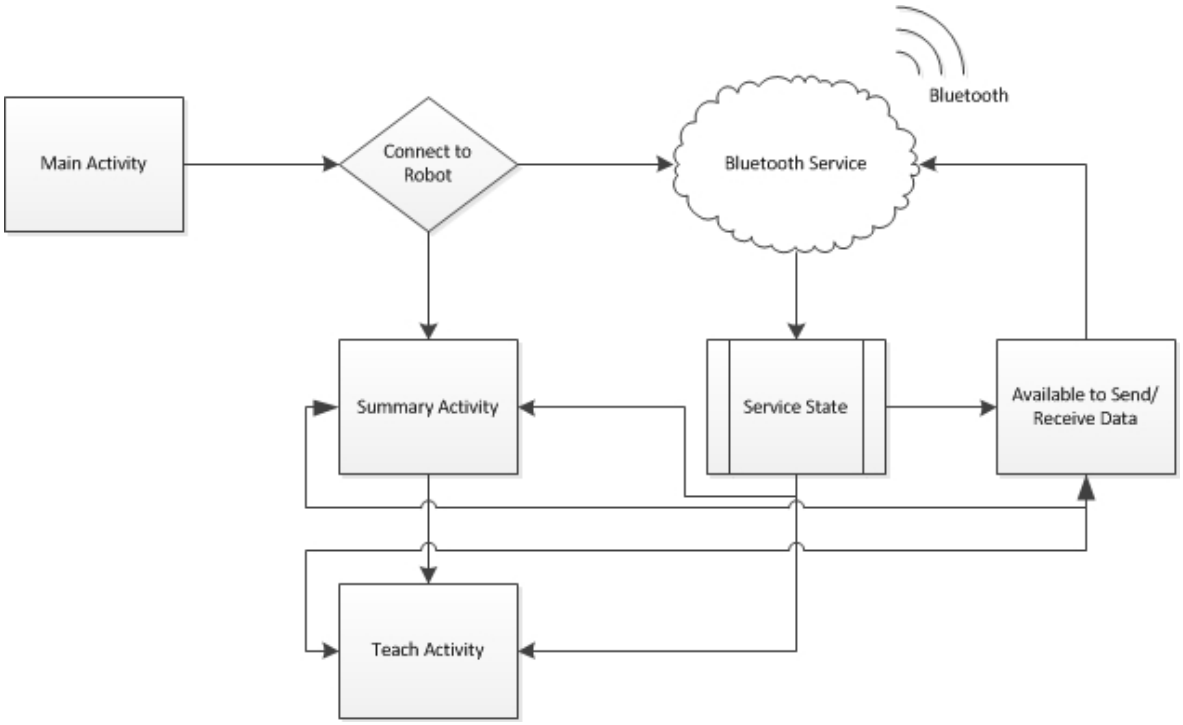

Figure 1. Android Application Architecture.

the pausing activity, it must be passed to the new activity when it is called or else the data will be inaccessible. Another way of passing data between activities is by creating a static service that all of the activities can access throughout the lifetime of the application. In the case of a static service, each activity must inherit the necessary functions to fully control the service.

A. System Architecture

As seen in Figure 1, when the application connects to a robot, it starts the Bluetooth service. In order to preserve the Bluetooth services throughout the lifetime of the application, it is necessary to encapsulate them into a container and distribute the container among the activities as needed or else the services would be lost upon pausing or destroying an activity. For each activity to successfully interact with the Bluetooth services, a Bluetooth service handler is required in every activity. The handler monitors whether there has been a state change and what actions need to be taken on such a state change for a particular activity. For example, the different states are "connected," "connecting," "listen," and "none." Based on those states, the handler must determine whether or not an activity can write data, listen for data, or "pop Toast." Popping Toast in Android refers to displaying a temporary dialog on the screen. For example, if the Bluetooth service handler is programmed to pop Toast on connecting to a robot, then a dialog exclaiming "Connected to Robot!" could be displayed on the screen. After an activity determines the current state of the Bluetooth services, it can then determine whether or not it can send data or listen for data.

The system is currently configured so that the Android application instigates all communication. Once it sends a message, it changes its state to listening and puts a lock on the service until it receives data back. Putting a lock on the Bluetooth services does not cause unnecessary delay throughout the application, due to "empty receipts" that both the Android application and server can send. An empty receipt is a flag that confirms that the receiver did indeed receive the last message. In the case of a dropped or lost message, the lower network layers take care of the retransmission of the output buffer.

B. Android Application Visual Walkthrough

Upon launching the Android application entitled "Robot Manager," the main activity is launched as seen in the left picture of Figure 2. The user is presented with two buttons, a back button which exits the program and an add robot button which allows the user to select a robot from previously paired robots or scan for new robots. Once connected to a robot, the summary activity is started and the robot information is populated. From this screen the user can send commands to the robot for execution by filling out the text box and clicking the send button. The back button on this screen destroys the summary activity and returns the user to the main activity. If the "Teach Points" button is clicked the teach activity is launched (right most image in Figure 2) and the user can

Figure 2. Android Activities

retrieve all trained locations from the robots memory and manually edit and update them. The back button on this activity returns the user to the summary activity.

III. Bluetooth Server System Architecture

The goal of the Bluetooth server is to provide an operating-system-independent solution that can act as a transparent layer in between the robot and the Android device. This layer must receive data through the Bluetooth connection, parse it, and forward it to the robot with as little latency as possible. The latency during the reverse direction is even more critical. If data transfer is slow from the robot controller to the robot controller component, then there is a possibility of receiving garbled data.

A. OPP versus SPP

There are two Bluetooth profiles that can be used for transmitting data: Object Push Profile (OPP) and Serial Port Profile (SPP). OPP is a protocol for sending (pushing) and receiving (pulling) any type binary object over Bluetooth radio communication. The unique and advantageous aspect to this profile is that the client, not the server, instigates all transfers. This means that the client can call methods on the server to achieve the desired action. In many Bluetooth libraries where OPP is implemented, it is given the name Object Exchange (OBEX). SPP is a serial port emulation profile for Bluetooth, which adheres to all serial port specifications. In Bluetooth communication, SPP is effective for sending streams of data in between the client and server and vice versa. The disadvantage of SPP is that for every connection and transfer, the server must be programmed to receive predetermined amounts of data. Whereas with OBEX, the length of the data is determined by the size of the object to be transmitted which is included in the header of the packets sent.

B. System Architecture

As shown in Figure 3, the Bluetooth server receives XML data from the Android phone. This data is then sent to the XML Service for parsing and then sent back to the server. The server then evaluates the message type. If the message is a command for the robot, the command is forwarded to the robot controller component and in turn forwarded to the robot controller for execution. If the message is a status update request, then a request is forwarded to the robot controller which sends the update data back to the server which then forwards the data back to the Android phone. The robot controller component maintains a queue of commands received from the phone to forward to the robot. The controller component also keeps a set of time stamped location and program data that is updated regularly, to be forwarded back to the phone upon request.

C. System Implementation

There are three parts to the implementation of the server: the Bluetooth services that it provides, the socket connection to the robot, and the interface that bridges the two. The entirety of the server part of this project is written in the Java programming language. To take advantage of Bluetooth in Java, an external library is required. There were several options for Bluetooth libraries but after evaluation, Bluecove (http://www.bluecove.com) was chosen. Bluecove is a Java library that implements the Windows, Mac, and Bluez Linux Bluetooth stacks. At the time of this project's development, Bluecove 2-1-1 was the current version. The Bluetooth service is an extension of a Bluetooth adapter. Upon execution, a service is broadcasted on the adapter using a unique UUID. A Bluetooth socket that contains an input stream is then created. The input stream is monitored for incoming data from the Android phone. When the phone connects and starts transmitting data, a 512-byte buffer is created to house the incoming data. When all of the data has been received, the

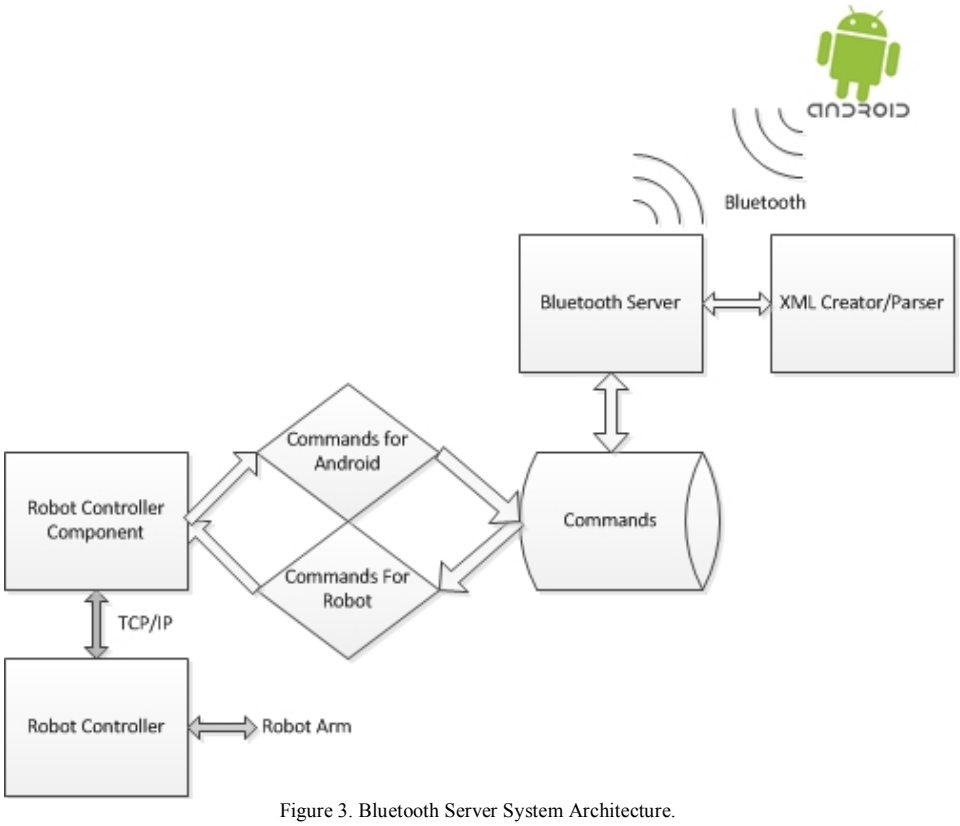

Figure 3. Bluetooth Server System Architecture.

XML file is reconstructed from the received bytes. The XML is then forwarded to the socket service. The socket service that the server provides receives XML from the Bluetooth service and parses command information from the XML. The service then sends this information and commands to the robot and the robot processes the data accordingly. The robot then sends information and status updates back to the server. The socket connection then wraps the data into XML and sends it to the Bluetooth service. The Bluetooth service then opens up a Bluetooth connection to the phone and transfers the data to the phone.

IV. Practical Applications

A. *Welding and Pick-and-Place*

To demonstrate one of the practical uses of this application, a typical manufacturing floor environment was simulated. Two Stäubli RX-60s robotic systems were set up with looping programs. The first simulated a spot welding line by visiting a starting point just above an apparatus containing a model vehicle and then quickly visiting six points around the model. The second simulated a palletizing line where the robot was programmed to pick a cylinder up from a pallet and place it into another pallet and vice versa. Each systems was equipped with the typical "stop emergency" and "stop normal" commands that are commonplace in factories. The stop emergency command brakes the robot immediately while the stop normal command allowed the robot to finish its current cycle in the program. The robots were then connected to the Bluetooth server application and the Android application was started. This demonstration showed the ease in which a user could switch between robots running different programs quickly and send those robots commands. Both commands executed very well on both simulations and there was very little latency upon issuing the stop emergency command to the robots via the phone. This demonstration was videotaped and can be viewed here:

http://www.youtube.com/watch?v=iPuTz2j_rjo

B. *Altering Location Data*

An additional feature that the application offers is the ability to alter trained locations as well as manually create new locations for the robot. All of the trained locations are populated in a drop down box in the Teach activity of the Android application. Upon selecting one of the locations, text boxes populate with either the joint-angle or Cartesian point data depending on which type of point is selected. For the purpose of these experiments, a joint-angle point is represented by

using a pound sign in front of the variable name and a precision point is represented by not using a pound sign. So the variable 'a' is a Cartesian point while '#a' is a joint-angle point. The values in the text boxes can be changed. The far right picture in Figure 2 depicts this component of the user interface. Upon clicking the update button, the location of the selected variable is updated. This experiment was also successfully conducted on a Stäubli RX-60.

C. System Limitations

Despite the advantages of this implementation, there are several possible issues that may arise to challenge the viability of this research as an effective solution. First is dealing with the limited range of Bluetooth radio. Bluetooth has a range of around 32 feet, and, in a manufacturing environment, this might not be enough range to reach all appropriate robotic systems that are working together on a specific task. Another shortcoming of Bluetooth radio is that it is only designed to support seven simultaneous connections with no broadcast mode. Again this may be a shortcoming on an actual factory line where the number of robotics arms in use may exceed seven. The outcome may lead the application to only support the robotic arms with the seven strongest signals. Latency is also a possible issue that must be further studied as this will affect the quality of the user experience while working with the robots.

V. Conclusions and Future Research

Although this research is still in an early stage of development, it has already proven to succeed in several of its goals. It has successfully demonstrated to provide a robot independent and operating-system-independent application interface. It has also proven to allow for meaningful two-way communication between the Android controller and the robot which would allow a non-expert to interact with and adjust the functionality of a manufacturing process which uses articulated robotic systems.

We are currently extending this system to allow for real-time control of a robotic arm with the built-in accelerometer and gyroscope of an Android-enabled smart phone. Furthermore, besides stopping applications normally or immediately, and manually adjusting location/point information, we are working with industry leaders to develop a concise set of interface capabilities that will provide broad coverage of typical usages in manufacturing environments. The end product of this research is an application which runs on any Android-enabled device and allows a non-expert to intuitively communicate with a variety of industrial robotic equipment.

Acknowledgements

We would like to thank SEW Eurodrive for funding this work and the Stäubli Corporation for providing the robotic equipment in our laboratory. We would also like to thank Kenneth Pestka and Sean Brakefield for their technical expertise and guidance.

References

[1] E. Cardozo, E. Guimaraes, L. Rocha, R. Souza, F. Paolieri, and F. Pinho, "A Platform for Networked Robotics," IEEE/RSJ International Conference on Intelligent Robotics and Systems, Taipei, Taiwan, 2010.

[2] Z. Wang, L. Liu, and M. Zhou. "Protocols and Applications of Ad-hoc Robot Wireless Communication Networks: An Overview," The International Journal of Intelligent Control Systems, vol. 10(4), pp. 296-303, 2005.

[3] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A Survey of Socially Interactive Robots," Robotics and Autonomous Systems, vol. 43, pp. 143-166, 2003.

[4] S. Hashish, and A. Karmouth, "Mobility-Based Generic Infrastructure for Large Scale Sensor Network Architecture," The IEEE International Conference on Communications, pp. 1-6, Ottawa, Ontario, 2009.

[5] C. Connolly, "KUKA robotics open architecture allows wireless control", Industrial Robot: An International Journal, vol. 35(1), pp.12 – 15, 2008.

[6] T. Song, J. Park, S. Jung, and J. Jeon, "The development of interface device for human robot interaction", Control, Automation and Systems, 2007

[7] P. Neto, J. Pires, and A. Moreira, "Accelerometer-based control of an industrial robotic arm", Robot and Human Interactive Communication, 2009.

[8] Z. Mednieks, "Programming Android," O'Reilly Media, ISBN: 1449389697, 2011.

[9] R. Meier, "Professional Android 2 Application Development," Wrox Publishers, Second Edition, ISBN: 0470565527, 2010.