PROPOSED DRAFT Proceedings of IEEE/ACM:
33rd Annual Simulation Symposium
April 16-20, 2000, Washington, D.C.

Controller Driven VRML Animation of the NIST Robotic Welding Cell

Keith Stouffer
National Institute of Standards and Technology
100 Bureau Drive, Stop 823
Gaithersburg, MD 20899-8230
(301) 975-3877
keith.stouffer@nist.gov

ABSTRACT

Virtual objects in a web-based environment can be interfaced to and controlled by external real world controllers. A Virtual Reality Modeling Language (VRML) welding cell was created that models a robotic arc welding cell (the Automated Welding Manufacturing System project,) located at the National Institute of Standards and Technology (NIST). The VRML welding cell contains a model of a 7 degree-of-freedom robot, a welding table, torch and various fixtures and parts. The VRML robot is interfaced to, and can be controlled by, the real world robot controller. This is accomplished by a socket connection between the collaborator's web browser and the real world controller. The current joint angles of the robot, which are stored in a world model buffer in the controller, are collected by a Java applet running on the web page. The applet updates the VRML model of the robot via the External Authoring Interface (EAI) of the VRML plug-in. Virtual welds, a series of VRML cylinders, are also dynamically created every 100 ms on the part based on the current robot position and colored according to the calculated weld quality of that section of weld obtained from the real world controller. This allows a collaborator to visually determine where a bad section of weld has occurred without being present in the physical welding lab.

NOMENCLATURE

AWMS - Automated Welding Manufacturing System project
EAI – External Authoring Interface
ISO – International Organization for Standardization
JPEG – Joint Photographic Experts Group
NAMT – National Advanced Manufacturing Testbed
NIST – National Institute of Standards and Technology
NML – Neutral Messaging Language
RCS – Real-Time Control System
RRC – Robotics Research Corporation
VRB – VRML Review Board
VRML – Virtual Reality Modeling Language
XDR – eXternal Data Representation

DISCLAIMER: Any mention of commercial products within this paper is for information only; it does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.

INTRODUCTION

The Automated Welding Manufacturing System (AWMS) [1], shown in Figure 1, is a testbed in automated gas metal arc welding (GMAW). The goals of the AWMS project are to develop interface standards and intelligent, science-based control technology to increase productivity, improve quality, and reduce the cost of welding system integration. The controller consists of a hierarchy of modules based on the NIST Real-Time Control System (RCS) Reference Model Architecture [2]. Potential customers include shipbuilders, heavy equipment manufacturers, and automotive manufacturers.

The work described in this paper was performed under the Virtual NAMT (VNAMT) project which is part of the larger Systems Integration for Manufacturing Applications (SIMA) [3] program at NIST. The goal of the VNAMT project is to develop an internet-accessible, multimedia environment that brings the capabilities of the National Advanced Manufacturing Testbed (NAMT) [4] and its projects to the web. The collaborator has access to live and stored video, client-controlled pan/tilt cameras in labs, VRML models of equipment and labs, live and stored data and multimedia PowerPoint presentations. The VNAMT project, in collaboration with the AWMS project, developed a remote access web page that allows a collaborator to monitor a remote welding operation with a PC and an internet connection.

Figure 1: AWMS welding cell

VRML MODEL OF THE WELDING CELL

The Virtual Reality Modeling Language (VRML) is a recently ratified International Organization for Standardization (ISO) standard (ISO 14772) [5] file format for the description of geometry and behavior of 3D computer graphics. VRML is the most widely used open file format for integrating 3D computer graphics with web-based content. VRML provides a robust foundation upon which 3D applications can be built that are portable and distributable via the World Wide Web. It defines most of the commonly used semantics found in today's 3D applications such as hierarchical transformations, light sources, viewpoints, geometry, animation, fog, material properties, and texture mapping.

The Robotics Research Corporation (RRC) robot was originally modelled in 3D using a robotic simulation package from Deneb Robotics called TeleGRIP. This simulation package allows for design, evaluation, and off-line programming of robotic workcells, incorporating real world robotic and peripheral equipment, motion attributes, kinematics, dynamics and I/O logic.

In order to view the 3D model of the RRC within a web browser, the model is translated to VRML using a translator program [6] that translates devices and workcells generated by TeleGRIP into VRML. The translation includes the geometry information as well as the kinematic information of the device. Figure 2 shows the translated VRML model viewed within a web browser using a VRML plugin such as Cosmoplayer or Blaxxun Contact.

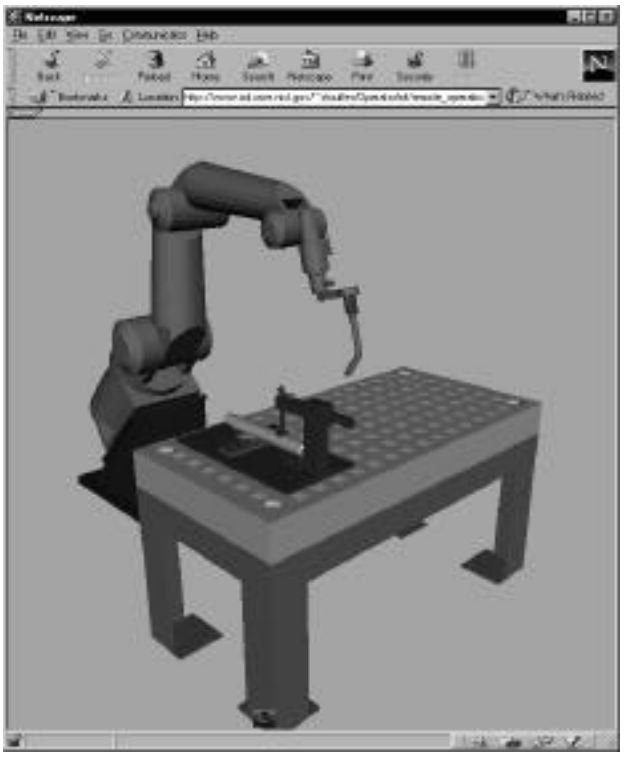

Figure 2. VRML model of the welding cell including the RRC robot

REMOTE ACCESS PAGE

Figure 3 shows the remote access web site [7] that was developed. The page contains a collaborator-controlled pan/tilt/zoom camera that sends video to the collaborator, a VRML model of the RRC that is controlled by the real world welding controller located at NIST, a real-time Java display for welding process data, and an archive of previous welds. This remote access web site allows a collaborator to monitor a remote welding operation with a PC and an internet connection.

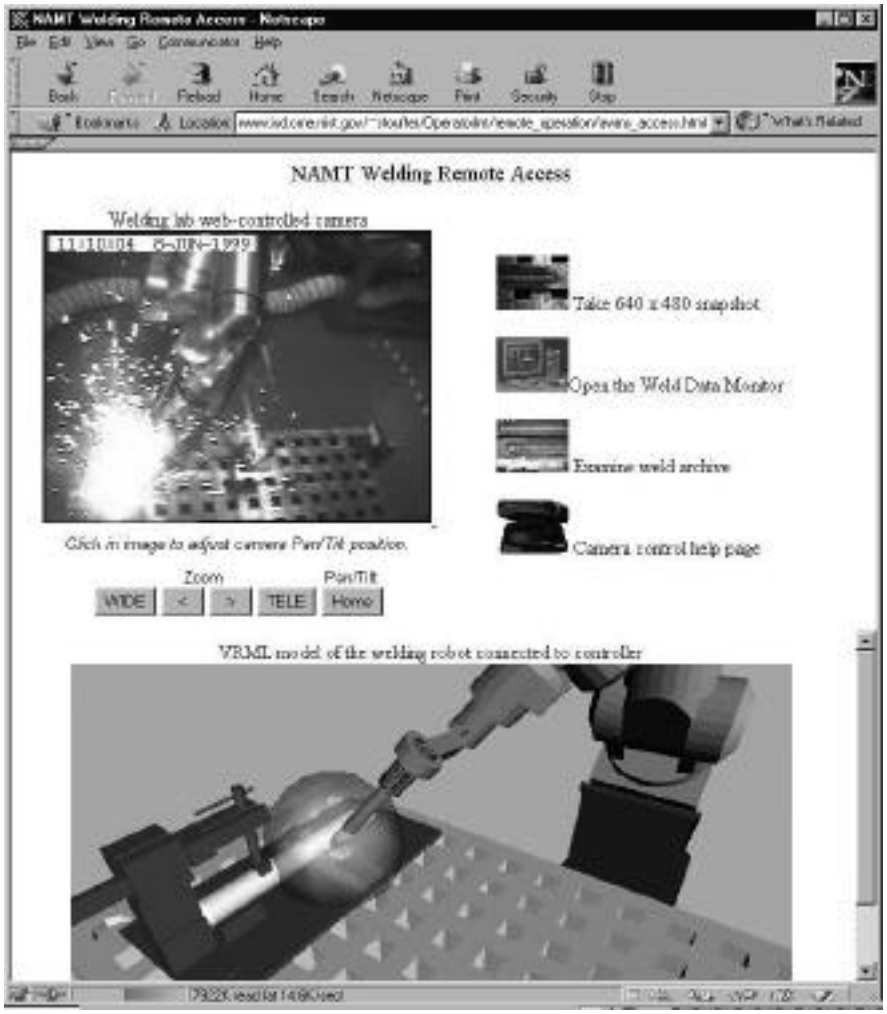

Figure 3. Welding Remote Access Web Site

CONTROLLABLE PAN/TILT/ZOOM CAMERA

The welding remote access page contains a collaborator controllable pan/tilt/zoom camera. This allows a collaborator to view live video of the welding operation. The video within the remote access page is a server-pushed JPEG image with an imagemap overlaid on it. The resolution of each image is 320 x 240 pixels. Each image is fairly large (20 Kbytes), therefore, frame rates are slow (1 frame per second). A higher frame rate could be obtained by using a lower resolution image, but in the case of welding, a high resolution image is paramount for examining the physical weld quality remotely. Each image of the video has an overlaid imagemap, allowing the collaborator to click on the image where the new center of interest should be. The required pan/tilt angles are calculated and the camera moves to recenter the image. The zoom of the camera is controlled by WIDE (zoom out) and TELE (zoom in) buttons on the page. Figure 4 shows the video of the welding cell with the camera zoomed out and Figure 5 shows the camera zoomed in for inspection of the weld.

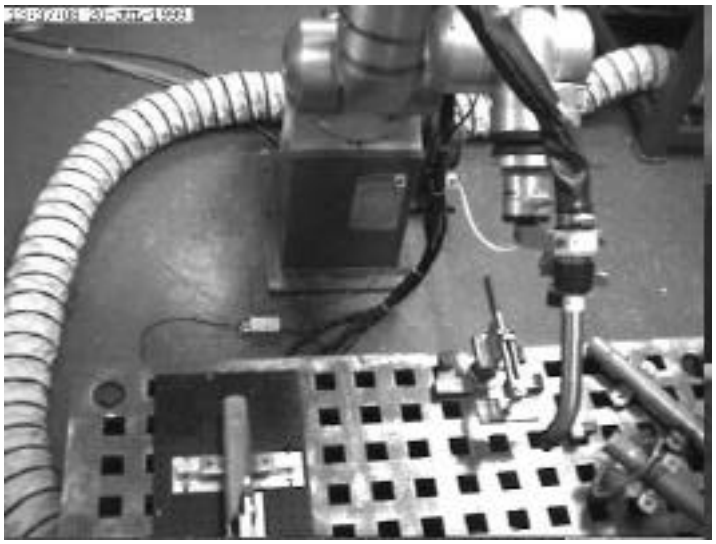

Figure 4. View from the controllable camera

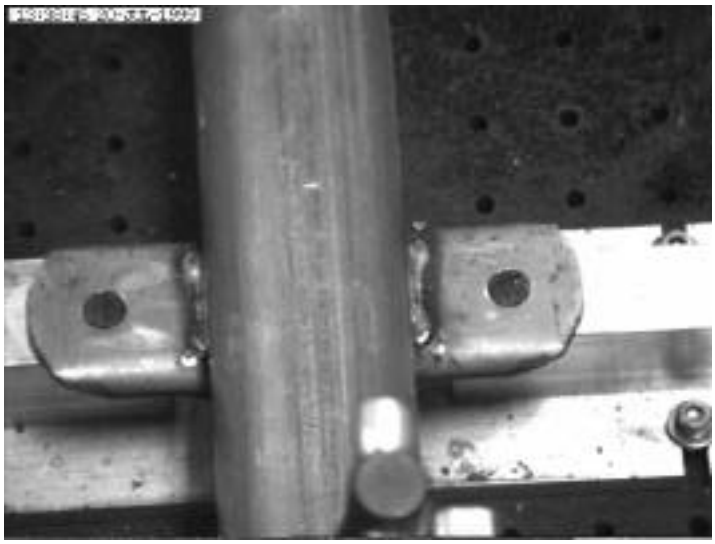

Figure 5. View from camera with maximum zoom

CONTROLLING THE VRML MODEL WITH THE CONTROLLER

The controller driven VRML model of the weld cell augments the capabilities of the camera system by giving collaborators a 3D perspective of cell motions and events with faster update rates. The VRML robot is interfaced to and controlled by the real world robot controller. This is accomplished by a Neutral Messaging Language (NML) [8] socket connection between the collaborator's web browser and the real world controller. This socket is created by a Java applet running on the remote access page. The current joint angles of the robot, which are stored in a world model buffer in the controller, are collected by the Java applet. In general, the

communications between an NML client and server is fairly simple. The server starts, opens a configuration file [9], reads whether to use Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) and a port number, and then binds to the port and listens for requests. Then a client starts and reads the same configuration file (which tells the port whether to use TCP or UDP, and which host the server is running on), connects to the appropriate port, and sends requests to the server. The client can send requests to write a message, read a message only one-time, or read a message and automatically be sent updates whenever the data changes or at a particular rate.

TCP PROXY

There are several obstacles to running controller driven VRML animations within a web page. Web browsers usually prevent Java applets running inside them from connecting to any hosts other than the web server they were loaded from. Many institutions also have firewalls in place that prevent any host outside the wall from directly contacting hosts inside the wall. One way to circumvent these problems is to run a TCP proxy. A TCP proxy is a generic program that forwards TCP data from one host to another in order to provide a very limited bypass of these security restrictions.

The TCP proxy is started on the web server with command line arguments indicating the host with the NML server and the TCP port number. It binds that port on the web server it and listens for connections. Each time a client connects to the TCP proxy, the TCP proxy connects to the NML server on the controller. It forwards any requests made by that client to the server and any data sent by the server back to the client in such a way that neither the client nor the server is aware that a TCP proxy is involved at all.

LOW BANDWIDTH CONNECTION

When a collaborator visits the web page, a VRML model of the welding cell is downloaded to their machine. Once the VRML model has been downloaded to the client's machine, only the joint positions need to be sent across the socket from the controller (server) to the web browser (client) to move the joints of the VRML robot to the current controller position. All of the graphics of redrawing the position of the robot are handled on the client machine. The data that is sent across the socket consists of 10 floats (7 joint angles to control the robot and the x, y, and z position of the weld tip to create VRML welds,) converted to the eXternal Data Representation (XDR), so the format is not processor specific. This data (10 floats = 40 bytes) is sent across the socket every 100 ms, therefore giving a data rate of 400 bytes/sec. This low bandwidth is usually handled easily, even on a slow network connection.

EXTERNAL AUTHORING INTERFACE (EAI)

Once the data has been gathered from the controller by the Java applet, the applet must commnicate with the VRML robot to animate it. For communication between a VRML world and an external environment (the applet), an interface between the two is

needed. A proposed interface, called the External Authoring Interface (EAI) [10], defines the set of functions on the VRML plugin that the external environment can perform to affect the VRML world. In essence, the EAI provides a method for developing custom applications that interact with, and dynamically update a VRML scene. These outside applications "talk" to the VRML scene. The EAI allows an external program (the applet) to access nodes in a VRML scene using the existing VRML event model. The EAI Spec has been submitted to the VRML Review Board (VRB) [11] for ISO formalization. Cosmoplayer 2.1 and Blaxxun Contact are two VRML plugins that incorporate the EAI.

Nodes in VRML can be named using the DEF construct. Any node named with the DEF construct can be accessed by the applet and is referred to as an accessible node. Once a pointer is obtained, the events (eventIns and eventOuts) of that node can be accessed. A Java applet communicates with a VRML world by first obtaining an instance of the `Browser` class. This class is the Java encapsulation of the VRML world. It contains the entire `Browser` Script Interface as well as the `getNode()` method, which returns a `Node` when given a DEF name string. Only DEF names in the base VRML file are accessible. Names in Inline files and those created with `createVRMLFromString()` or `createVRMLFromURL()` are not accessible.

Once a `Node` instance is obtained from the `getNode()` method of the `Browser` class, its eventIns and eventOuts can be accessed. The `getEventIn()` method of the `Node` class returns an eventIn when passed a string with the desired eventIn name. The `getEventOut()` method of the `Node` class returns an eventOut when passed a string with the desired eventOut name. ExposedFields can also be accessed, either by giving a string for the exposedField itself (such as rotation) or by giving the name of the corresponding eventIn or eventOut (`set_rotation()` for the eventIn and `rotation_changed()` for the eventOut). In order to access and manipulate the VRML joints (each a separate DEFed Node), a VRML joint class was developed. The Java code for the `VrmlJoint` class follows:

```
import java.applet.Applet;
import vrml.external.field.*;
import vrml.external.exception.*;
import vrml.external.Node;
import vrml.external.Browser;

public class VrmlJoint
{
   Node myNode;
   EventInSFRotation myRotation;
   String nodeName;
   int axis;
   float[] axisValue = {0.0f, 0.0f, 0.0f, 0.0f};

   public VrmlJoint(String name, Browser browser, int a)
   {
```

```
        // Get the handle of the browser
        myNode = browser.getNode(name);
        myRotation = (EventInSFRotation) myNode.getEventIn("set_rotation");
        axis = a;
        nodeName = name;
        axisValue[axis] = 1.0f;
     }
    public void updateValue(float newValue)
    {
        // Send new joint values to VRML robot via the External Authoring Interface
        axisValue[3] = newValue;
        myRotation.setValue(axisValue);
    }
}
```

The current joint angles of the robot along with the position of the welding tip, which are stored in the world model buffer in the controller, are collected by a Java applet on the web page. The applet then updates the VRML model of the robot via the EAI of the VRML browser. The source code for the java applet that communicates between the VRML envirnment and the real world welding controller is included in the Appendix of this paper.

DYNAMIC CREATION OF VRML WELDS

Virtual welds, a series of VRML cylinders, are also dynamically created on the part based on the current robot position and the calculated weld quality of a section of weld obtained from the real world controller. A VRML weld section is created every 100 ms via the EAI using a VRML cylinder prototype [12]. The color of the VRML weld can be based on the calculated quality of the weld in that 100 ms interval. The weld quality is calculated from the data obtained from a collection of sensors including voltage and current [13]. This would allow a collaborator to visually determine where a bad section of weld has occurred without being present in the welding lab. Figures 6 and 7 show the VRML model of the welding cell during a weld and immediately following a weld.

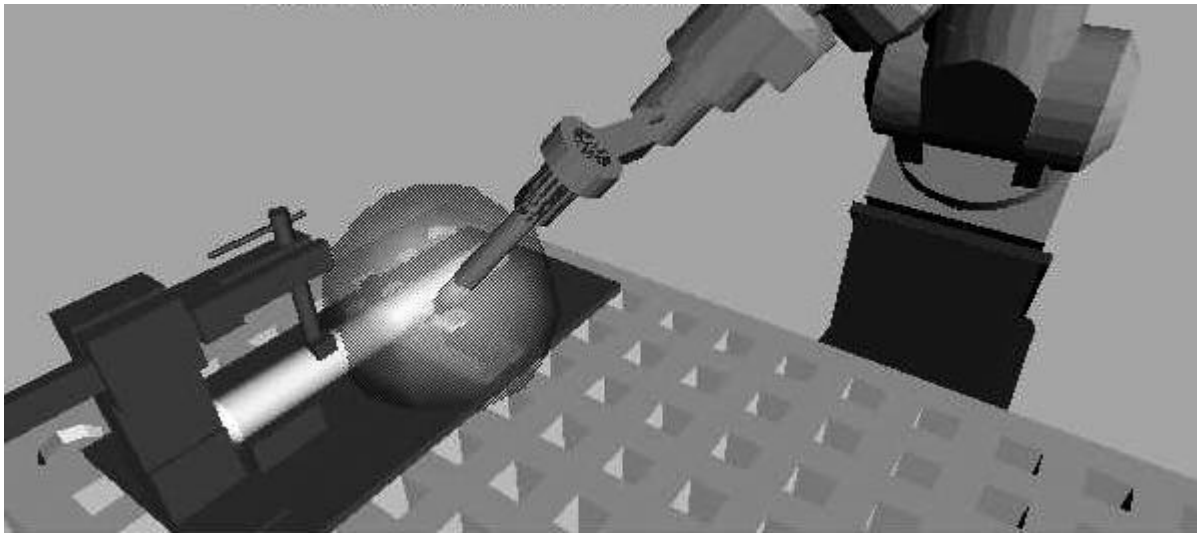

Figure 6. VRML model of the welding cell during a weld. Note the spark field (sphere around weld).

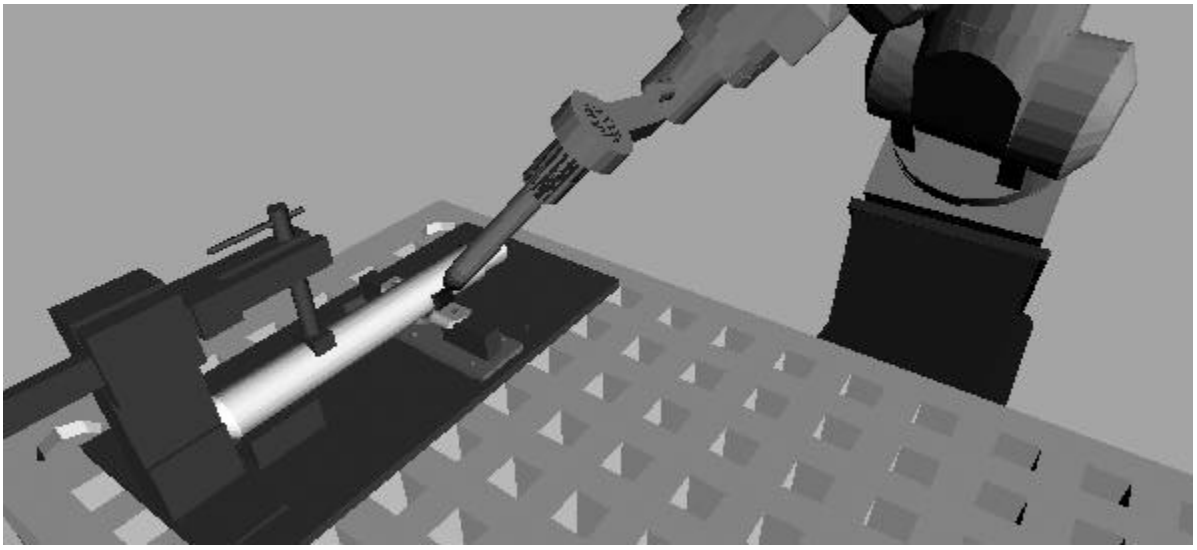

Figure 7. VRML model of the welding cell immediately following a weld. Note the dynamically created weld.

CONCLUSIONS

Virtual objects in a web-based environment can be interfaced to and controlled by external real world controllers. This allows animation of actual processes for collaborative verification. A VRML environment was created that models a robotic arc welding cell. The VRML welding cell contains a model of a 7 degree-of-freedom robot, a welding table, torch and various fixtures and parts. The VRML robot is interfaced to, and is controlled by, the real world robot controller. This is accomplished by a socket connection between the client's web browser and the real world controller. The current joint angles of the robot, which are stored in a world model buffer in the controller, are collected by a Java applet running on the web page. The applet then updates the VRML model of the robot via the EAI of the VRML plug-in. Virtual welds, a series of VRML cylinders, are also dynamically created every 100 ms on the part based on the current robot position and colored according to the calculated weld quality of that section of weld obtained from the real world controller. This allows a collaborator to visually determine where a bad section of weld has occurred without being present in the physical welding lab.

ACKNOWLEDGEMENTS

We would like to thank the SIMA program for the continuing support of this work and the members of the AWMS project (Bob Russell, Jim Gilsinn, Joe Falco, and Bill Rippey) for the technical assistance required to complete this work.

REFERENCES

[1] AWMS project brochure
http://www.isd.cme.nist.gov/projects/brochure/Welding.html

[2] Albus, J.S., Meystel, A.M.,"A Reference Model Architecture for Design and Implementation of Intelligent Control in Large and Complex Systems," International Journal of Intelligent Control and Systems, Vol. 1, No. 1 pp. 15-30

[3] Systems Integration for Manufacturing Applications (SIMA) Program
http://www.nist.gov/sima

[4] National Advanced Manufacturing Testbed
http://www.mel.nist.gov/namt/

[5] VRML VRML2.0 Specification ISO/IEC CD 14772, 1996

[6] Translator program from Deneb Robotics format to VRML
http://ovrt.nist.gov/projects/mfg/SIMA/deneb2vrml/deneb2vrml.html

[7] NAMT Welding Remote Access Page
http://www.isd.cme.nist.gov/~stouffer/OperatorInt/remote_operation/awms_access.html

[8] Shackleford, W.P., "The NML Java Programmer's Guide"
http://www.isd.cme.nist.gov/proj/rcs_lib/NMLjava.html.

[9] Shackleford, W.P., "Writing NML Configuration Files"
http://www.isd.cme.nist.gov/proj/rcs_lib/NMLcfg.html.

[10] External Authoring Interface Working Group
http://www.vrml.org/WorkingGroups/vrml-eai/

[11] VRML Review Board
http://www.web3d.org/vrb/

[12] VRML cylinder prototype
http://www.isd.cme.nist.gov/~stouffer/OperatorInt/remote_operation/Cyl_p.wrl

[13] Quinn, T.P., Madigan, R.B., Sensing of Arc Welding Process Characteristics for Welding Process Control, U.S. Patent Number 5,756,967, 1998.

APPENDIX

Source code for the java applet that communicates between the VRML environment and the real world welding controller:

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;
import vrml.external.field.*;
import vrml.external.exception.*;
import vrml.external.Node;
import vrml.external.Browser;

public class RRCapplet extends Applet implements Runnable
{
   final static String getcmd = "get\n";
   final static String quitcmd = "quit\n";
   RRCconnect myConnection;
   RobotStatus myRobotStatus;
   WeldStatus myWeldStatus;
   boolean stayAlive;
   Thread myThread;
   float[] prev_position;
   float[] cur_position;
   float[] position;

   boolean weldingStarted;

 // Added for External Authoring Interface
   Browser browser;
   Node root;
   VrmlJoint[] myJoints;
   Node[] weld;
   Node sparkNode;

   EventInSFVec3f spark_location;
   EventInMFNode addChildren;
   EventInMFNode removeChildren;

   public void init()
   {
      myJoints = new VrmlJoint[7];
      prev_position = new float[3];
      cur_position = new float[3];
      position = new float[3];

      // Make the NML connection to the RRC controller

      myRobotStatus = new RobotStatus("applet");
      myWeldStatus = new WeldStatus("applet");
      myConnection = new RRCconnect(getCodeBase().getHost(), 3010);
   }
```

```
public void start()
{
   // Get the handle of the browser
   browser = (Browser) vrml.external.Browser.getBrowser(this);

   // Get the handles for each joint of the robot
   myJoints[0] = new VrmlJoint("rrc_j1", browser, 1);
   myJoints[1] = new VrmlJoint("rrc_j2", browser, 2);
   myJoints[2] = new VrmlJoint("rrc_j3", browser, 2);
   myJoints[3] = new VrmlJoint("rrc_j4", browser, 2);
   myJoints[4] = new VrmlJoint("rrc_j5", browser, 2);
   myJoints[5] = new VrmlJoint("rrc_j6", browser, 2);
   myJoints[6] = new VrmlJoint("rrc_j7", browser, 2);
   root = browser.getNode("d_rrc");
   sparkNode = browser.getNode("spark");
   addChildren = (EventInMFNode) root.getEventIn("addChildren");
   removeChildren = (EventInMFNode) root.getEventIn("removeChildren");

   spark_location = (EventInSFVec3f)  sparkNode.getEventIn("set_translation");
   if(!myRobotStatus.connect())
   {
      System.out.println("Could not connect to Robot Status Buffer");
   }
   else
   {
      weldingStarted = false;
   }

   if(!myWeldStatus.connect())
   {
      System.out.println("Could not connect to Weld Status Buffer");
   }


    if(myConnection.makeConnection())
   {
      stayAlive = true;
      myThread = new Thread(this);
      myThread.start();
   }
}

public void stop()
{
   stayAlive = false;
   System.out.println("Stopping");
   myConnection.disconnectSocket();
   System.out.println("Stopped");
}

public void run()
{
   String data;
   while(stayAlive)
   {
      myConnection.sendData(getcmd);
```

```
            data = myConnection.getData();
            parseData(data);
        }
    }

private void parseData(String s)
    {
        int ix;
        String token;
        float[] value;

        StringTokenizer st = new StringTokenizer(s, " ");
        ix = st.countTokens();
        value = new float[7];

        // Parse the data from the socket
        if(ix == 8)
        {
            token = st.nextToken();
            for(ix = 0;ix < 7; ix++)
            {
                token = st.nextToken();
                value[ix] = Float.valueOf(token).floatValue();
            }

            // Send new joint values to VRML robot via the External Authoring Interface
            for(ix = 0;ix < 7; ix++)
            {
                if(stayAlive)
                {
                   // Updating the new joint values here
                    myJoints[ix].updateValue(value[ix]);
                }
            }
            // Get weld status and if welding, perform the following
            if(myWeldStatus.isWelding())
            {
                // Get the current robot position
                cur_position = myRobotStatus.getStatus();

                position[0] = cur_position[0]/1000f;
                position[1] = cur_position[2]/1000f+0.9144f;;
                position[2] = -cur_position[1]/1000f;

               // Move the VRML spark to the weld tip via the External Authoring Interface
                spark_location.setValue(position);

                if(!weldingStarted)
                {
                    for (ix = 0; ix < 3; ix++)
                    {
                        prev_position[ix] = cur_position[ix];
                    }
                    weldingStarted = true;
                }
```

```
            // Create the VRML weld section using  the cylinder proto and createVRMLfromstring function
            weld = browser.createVrmlFromString("Cyl {p1  " + prev_position[0] + " " + prev_position[1] + " " +
                    (prev_position[2]+914.4f) + " p2 " + cur_position[0] + " " + cur_position[1] + " " + (cur_position[2]+914.4f) +
                     "color 1 0 0 radius 10}");

            // Add the weld section to the world via the External Authoring Interface
            addChildren.setValue(weld);

            // Set previous position to the current position
            for (ix = 0; ix < 3; ix++)
            {
               prev_position[ix] = cur_position[ix];
            }
           }

         else
         {

            weldingStarted = false;

           // Remove the spark (move offscreen)
            position[0] = 0f;
            position[1] = 100f;
            position[2] = .2f;
            spark_location.setValue(position);
          }
      }

      else
      {
         System.out.println("Error:  Only have " + st.countTokens() + " tokens");
      }
   }
}
```