A Modular Distributed Telerobotic Client-Server System

Mayez A. Al-Mouhamed[1], Mohammad Nazeeruddin[2], and Asif Iqbal[3]

Abstract— **Pervasive telerobotics aims at providing portable computer aided assistance using small, light, and highly interactive client-server telerobots. A reliable real-time multithreaded interface between a master arm and a slave arm is proposed using *Distributed Components .NET Remoting*. The advantages are (1) automatic handling of the network resources and transfer while isolating the components from network protocol issues, and (2) compiling into platform-independent bytecode. The functionalities of master and slave arms are independently designed. The operator maps his hand motion to the remote tool making abstraction of the structure of master and slave arms. Force feedback measured at the tool tip is transmitted and displayed on operator hand. Real-time streaming of stereo vision provides stereoscopic views of remote tool. Design and performance of proposed multi-threaded execution is presented to effectively realize multi-streaming of force, command, and stereo data in a distributed and modular framework.**

Keywords: DCOM, Distributed Application Framework, Force Feedback, Stereo Vision, Telerobotics.

I. Introduction

Telerobotics [1] is needed for minimally invasive surgery for enhancing dexterity, precision, and stability. The surgeon moves a dexterous master arm (client station) that is scaled down to a slave arm (server station) located inside the patient's body. Pervasive telerobotics aims at providing portable computer aided assistance using small, light, and highly interactive telerobots. The integration of real-time streaming of stereo vision, haptic feedback, and computer assisted telerobotics are targeted. An effective communication interface is needed to enable operations using portable platforms making telerobotics an effective tool for remote surgery. Reliable real-time streaming of force feedback and stereo vision are critical in many telerobotic applications.

A distributed component for telerobotic DCOM [2] allow integrating web technologies and telerobotics. DCOM-ActiveX based supervisory control is a server operating over the Internet as backbone. Operator views 3-D model, control paths, and issue commands through supervisory control which provides reduced human attendance, robustness control, and flexibility to ease the task of upgrading the system. MS VM (Microsoft Virtual Machine) is used to bridge the gap between JAVA and *DCOM*.

(1) Department of Computer Engineering, College of Computer Science and Engineering (CCSE) King Fahd University of Petroleum and Minerals (KFUPM), Dhahran 31261, Saudi Arabia. mayez@ccse.kfupm.edu.sa

(2) Department of Systems Engineering, CCSE, KFUPM, Dhahran 31261, Saudi Arabia. onur@ccse.kfupm.edu.sa

(3) Department of Systems Engineering, CCSE, KFUPM, Dhahran 31261, Saudi Arabia. iqbal@ccse.kfupm.edu.sa

In[3] VB 6.0 and TCP ActiveX based client-server framework is proposed. TCP read/write operations are slow because of the many software layers involved such as application, custom protocol, TCP ActiveX control, and Windows Sockets. Ho[4]'s JAVA based fame-grabbing software takes about 1 s for camera-DRAM transfer with a video rate 0.33 Fps.

Internet produces random transmission delays which degrades the quality of streaming force feedback and stereo data. TCP/IP sockets [5], [6] and VxWorks real-time multitasking OS are used for reliable streaming over Internet using task prioritization to control CPU time. However, packet arrival delay varies from 50 ms to 100 ms, for small packets, over the US. UDP packets do not preserve their chronological order. TCP/IP can be reasonably used for packet with 256 bytes with a 10Hz sampling rate.

Telerobotic systems using TCP/ATM [6] provide QoS specification of timing, criticality, clock synchronization, and reliability. The sampling intervals reported are 0.4, 0.3, and 0.2 s for TCP/IP, raw ATM, and TCP/IP over ATM, respectively. Augmented reality [7], [8], consists of overlaying virtual graphics over real images of remote scene to allow the operator to see how his action fits into the scene before executing his commands.

We propose a reliable real-time connection between master and slave stations using *.NET based Distributed Components*. For this we designed various telerobotic components, interaction methods, and secure communication support while isolating the components from network protocols. We designed the functionalities of the master and slave arms based on hiding the details of each and direct mapping of operator hand motion to the remote tool, streaming of tool stereoscopic views, and displaying force feedback measured at the tool tip on operator hand.

The organization of the rest of the paper is as follows. Section 2 presents our client server telerobotic framework and its functionalities. In Section 3 we compares our approach to others. We conclude in Section 4.

II. A Telerobotic distributed framework

In this section we describe the server and client telerobotic components and their interactions with each other in a distributed application.

A. Telerobotic server components

The server components are: (1) PUMA component and its functionalities, (2) Force Sensor Component and its functionalities , and (3) Decision Server Component. In addition we have three interfaces known as (1) Proxy Robot Interface (2) Force Sensor Interface, and (3) Decision Server Interface which will be presented in the following sub-sections.

1) PUMA Component: PUMA component acts as a software proxy of the robot for which commands are issued to the component as they are issued to the robot. Whenever robot changes its states, the component updates itself automatically to reflet these changes.

Some important public methods exposed by PUMA component allow operating the robot as an independent agent capable of moving the robot in a variety of motion modalities and exception handling. The PUMA component accepts a user defined tool frame of reference as (1) robot base frame (world), (2) robot wrist frame, or (3) robot tool frame. Some of the robot statuses are (1) connection to robot is not detected or robot not initialized, (2) robot is connected but not initialized, (3) initialization is pending, (4) robot is ready to receive a motion parameter, (5) robot is moving, etc. The events invoked by PUMA component include: (1) Data received from PUMA, (2) some error occurred with PUMA, (3) Robot moved to a new location, and (4) PUMA status changed.

2) Functionality of PUMA component: Operating and positioning tools in small areas poses access constraints as well as unexpected contact forces. 3D anatomical models (MRI or CT-scan) are guide the tool to the organ location. There is pressing need to map the surgeon hand to the operating tool both in position and force control which is presented here as augmented functionalities to the server and client stations.

The kinematics of slave arm is represented by means of three frames: (1) a fixed world frame (R_w) at arm origin, (2) an effector frame (R_e) located at arm terminus, and (3) a user defined tool frame (R_t). The controllable frame R_e is represented by its 3×1 position vector ($E_w(\theta)$) and its (3×3) orientation matrix ($M_w^e(\theta)$), where θ is the slave arm joint vector and w refers to R_w. The tool frame R_t is user or system defined by its position vector T_t and orientation matrix M_e^t of tool frame R_t with respect to frame R_e. The position of the tool point is defined by:

$$T_w = E_w + M_w^e(\theta) M_e^t T_t \tag{1}$$

The slave station receives a command to translate the tool frame R_t by ΔT_w and to rotate it by ΔM_t. The operator motion can be efficiently mapped onto the tool frame when the translation is specified in tool frame, i.e. ΔT_t. The new arm controllable position vector (controllable) is:

$$\Delta E_w = M_w^t (I - \Delta M_t) T_t + \begin{cases} \Delta T_w & \text{Operator-tool} \\ M_w^t \Delta T_t & \text{Operator-world} \end{cases} \tag{2}$$

where $M_w^t = M_w^e M_e^t$. The new effector orientation matrix (controllable)is:

$$\Delta M_e = M_e^t \Delta M_t M_t^e \tag{3}$$

The PUMA component reads current robot joint θ as a 6×1 vector which allows computing current effector position $E_w(\theta)$ and orientation $M_w^e(\theta)$. The target effector position and orientation are $E_w^+ = E_w(\theta) + \Delta E_w$ and $M_w^{e+} = M_w^e(\theta) \Delta M_e$. The inverse kinematic model $\theta^+ = G^{-1}(E_w^+, M_w^{e+})$ of the slave arm allows finding the joint vector θ^+ that moves the tool by the commanded translation ΔT and rotation ΔM. The new joint vector θ^+ is sent to slave arm motion controller.

3) Force Sensor Component: The force sensing component (FSC) reads the robot wrist force sensors and creates a stream of reflected force feedback directed to the master station. FSC is implemented as a separate thread, the priority of which can be adjusted during runtime to allow for the management of CPU usage.

A new instance of FSC creates a new thread with a default *normal* priority and waits until the sensing is triggered. After the reading has started, it continues sensing the force information at a pre-specified, alterable, default frequency. The public properties exposed by FSC are: (1) SensorThreadPriority used to set the thread priority that is one out of five OS levels. (2) TimerValue used to set a time interval between two successive readings.

4) Force sensor component functionality: A 6 dof force sensor is implemented at the wrist of the slave arm to provide (1) measurement of external forces, and (2) passive compliance of the tool. The sensor consists of two parallel plates p_1 (frame R_e) and p_2 (frame R_s) interconnected by three elastic links. The elementary motion of p_2 with respect to p_1 is measured by a differential (1) translation vector ΔS_e of the origin of R_s, and (2) orientation matrix ΔM_e of R_s measured in R_e. The sensor structure allows finding ΔS_e and ΔM_e as functions of the six sensing signals. The sensor frame R_s is located between the effector frame R_e and the tool frame R_t. An external force applied to the tool causes a deflection vector $\Delta T_e = \Delta S_e + (\Delta M_e - I) M_s^t T_t$ to the tool frame origin as well as a change ΔM_t in R_t orientation as $\Delta M_t = M_t^s \Delta M M_s^t$. Since $M_e^t = \Delta M M_s^t$ we can compute the tool deflection vector in its frame R_t:

$$\Delta T_t = M_t^s \Delta M^{-1} \Delta T_e \tag{4}$$

The force (F_t) and moment (C_t) vectors applied to the tool are computed using the tool linear and rotational compliance vectors ΔT_t and $M_t^s \Delta M M_s^t$. Using the passive compliance matrices for linear (K_l) and rotational (K_r) motion of the tool we compute the force $F_t = (f_x, f_y, f_z)^t = K_l \Delta T_t$ and moment $C_t = (c_x, c_y, c_z)^t = K_r \Delta M_t$ vectors. The tool force and moments vectors F_t and C_t are used to: (1) display the reflected force feedback at the client station, and (2) implement active compliance mechanism at as supervisory functions at the slave arm level.

5) Decision Server Component: DecisionServer is a component that provides an autonomous loop on the server to support supervisory telerobotic control. It belongs to a higher abstraction layer which is used as an agent to implement local robot automation functions. A block diagram describing the hierarchy of the server system including DecisionServer. The human operator is at the highest level of hierarchy and interacts with the system using a UI(user interface).

6) Server Side Interfaces and .NET Remoting*:* An interface is a set of definitions of public methods and properties. It servers as a contract for any component that implements this

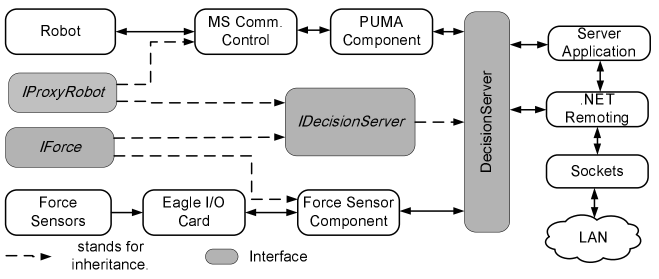

Fig. 1. Integrated Scheme - Server Side

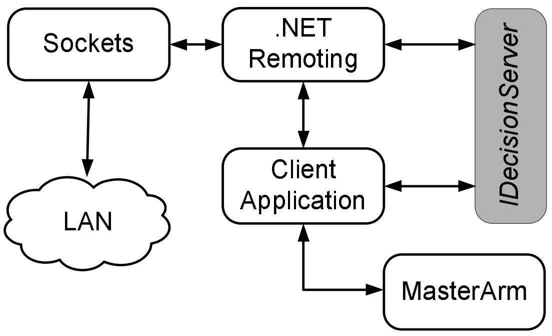

Fig. 2. Integrated Scheme - Client Side

interface. This scheme allows hiding the actual component or assembly from the client which increases security from potentially unsafe clients as well as gives the developers, freedom to easily amend the logic of the server methods while the interface remains unchanged.

In order to communicate with both the PUMA and Force Sensor components, we define two interface named *IProxyRobot* and *IForceSensor*. *IDecisionServer* inherits both of these interfaces. This allows defining a unified set of public members (methods, properties and events) that are required to be implemented in the form of DecisionServer component. Now *.NET Remoting* is used to publish an instance of DecisionServer component on the LAN that is identified to the client by a unique object identifier. *.NET Remoting* enables us to access objects using SOAP(Simple Object Access Protocol) which isolates the network protocol issues from the development of a distributed application.

B. Telerobotic client components

The client contains the *IDecisionServer* interface to reference the server side component through *.NET Remoting*. In addition to *IDecisionServer*, there are instances of *.NET Remoting* and client GUI(Graphic User Interface).

Decision Server interface named as *IDecisionServer* contains all the definitions to execute methods on PUMA and Force Sensor components. Following the initialization of the client, the system carries an empty un-referenced copy of *IDecisionServer*. Once a network connection with the server is established, the client gets the reference to the server side instance of DecisionServer. Now *IDecisionServer* refers to the published instance of DecisionServer and the client side can access the server side instance of DecisionServer as a local component through *IDecisionServer*.

The integrated scheme incorporating all the components on client and server side is shown in Figures 1 and 2.

The DecisionServer is inherited from *IDecisionServer* and in turn from IProxyRobot and IForceSensor interfaces. In order to use an event handler on client side for any event invoked by DecisionServer, we must provide DecisionServer, access to the client assembly. This introduces severe deployment limitations. To overcome this problem, we use *shim* classes as intermediatory agents to forward DecisionServer events over to the client or *IDecisionServer* interface. Shim classes are thin assemblies visible to both the server and the client.

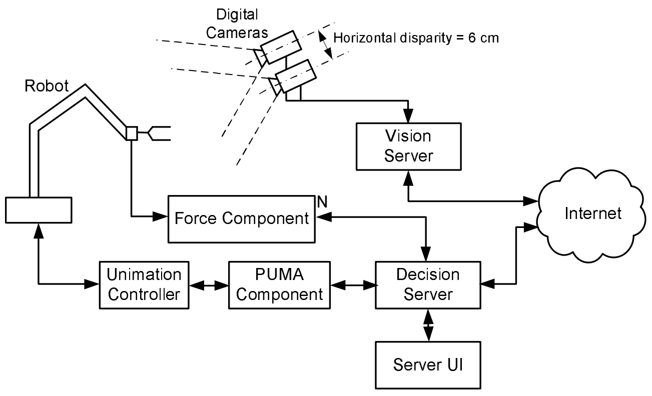

Fig. 3. Server side of the distributed framework

1) Functionalities of the client station: The operator handing the master arm may easels exit a geometric area which he can efficiently teleoperate the slave arm, i.e. dexterity area. In this case the teleoperation becomes very difficult. One needs to shift the master arm to operator dexterity area without affecting the current position of slave arm. This defines the master-shift function which must be activated by the operator hand handling the master arm in an optimized man-machine interface.

Denotes by (E, M) the previous operator position vector and orientation matrix at the master arm and p a boolean that is 0 during the active periods of the master-shift function, i.e. when connection between master and slave arms is disabled. The client must send motion increments to map the slave arm $(\Delta E, \Delta M) = (E^+ - E, M^{-1}M^+)$ when the connection is active.

The operator may need to scale-down his motion in the neighborhood of a critical task location to increase the motion accuracy. In this case the increment in master position vector (ΔE) and orientation matrix (ΔM) need to be scaled-down before being mapped to those of the slave arm. We evaluate three orientation angles for the operator hand frame. The operator orientation matrix M can be seen as made of three euler angles, i.e. $M = R_x(\alpha_x)\ R_y(\alpha_y)R_z(\alpha_z) = R_{xyz}(M)$, where R_u is a rotation matrix about axis u and R_{xyz} is the product of three rotation matrices sets for M. Since ΔM is known we inverse the above equation and find the three angles as $(\alpha_x, \alpha_y, \alpha_z) = R^{-1}_{xyz}(\Delta M)$. Using a user defined scale

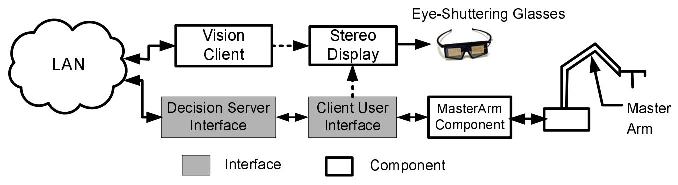

Fig. 4. Client side of the distributed framework

factor s the scale function becomes:

$$(\Delta E, \Delta M) = ((E^{+} - E) * s, R_{xyz}((R_{xyz}^{-1}(\Delta M)) * s))) \quad (5)$$

C. A Distributed Telerobotic System

The multi-threaded distributed telerobotic system (Fig. 3 and 4) allows simultaneous activation of many threads like grabbing and transfer of stereo video data, reading force sensors, sending and receiving robot control signals over the LAN to one or more clients.

Two digital cameras generate stereo pictures which are sent to the client. Both the stereo data and the distributed component calls share the same LAN using different ports for data transfer. The client uses the GUI as well as a 6 dof master arm to issue commands to the slave arm on remote site. The vision client receives the synchronized stereo data from the LAN through windows sockets and provides a stereo display of the remote scene to the operator using eye-shuttering glasses.

III. Evaluation

The client station is interfaced to: (1) a locally made master arm, and (2) an HMD, both are used to interface the operator to the client station. The slave station consists of a PUMA 560 arm with 6 dof arm and a wrist force sensor. The operator hand motion is mapped to the tool held by the slave arm (PUMA). Using our distributed framework three streams are actives: (1) a stereo video (not described here) stream, (2) a reflected force feedback transmitted from server to client and displayed on operator hand through the master arm, (3) a stream of operator motion commands flowing from client to server.

The server is 2.0 GHZ P-IV which is connected to 100 Mbps LAN. Each force packet is 48 bytes. Each video picture is 288×360 pixels and each pixel is 3 bytes. Each stereo frame (two pictures) is 0.6 MB and requires a bandwidth of 5 Mbps/Frame. The server throughput on network of streaming only force packets is about 1 KHz. Multistreaming of Force and video leads to a force packet rate of 250 Hz which drops to 76 Hz during active video intervals. Optimized video transfer provides a throughput of 58.94 ms per stereo frame rate (17 fps).

Typical telerobotic experiments are carried out: (1) task-1 consists of setting up the slave tool in a given position by direct command from the operator hand motion, (2) task-2 consists of object manipulation and stacking, and (3) task-3 consists of manipulation of bottled liquid. The distributed framework has been shown to be reliable during the above tasks. The GUI and .NET remoting proved to be useful in mirroring the state of slave arm at the client station.

IV. Comparison

Compared to[4] our camera-DRAM transfer and video rate are 25 ms and 12 Fps using DirectX image acquisition, respectively. Compared we proposed *.NET Framework* for development of all GUIs and core system components thus freeing us from using any intermediatory services like MS VM within the framework. Compared to [3] the components directly communicate with each other through windows sockets using *.NET Remoting* providing shorter round-trip time. For example a command takes 55 ms in the case of the cited framework as compared to around 1 ms in our case. .NET has embedded type signatures which allows component debugging across different languages, a missing feature in Java and Corba. .NET is highly recommended for mission-critical applications running under Windows.

V. Conclusion

A portable real-time telerobotic interface between master and slave arms is proposed using *.NET Remoting based Distributed Components*. The advantages of .NET are (1) ease of deployment to work across firewalls,(2) compiles the source code into platform-independent bytecode [9], and provides highly optimized data transfer [10] for symmetric configuration for the client and server. Design independency in the master and slave modules lead to mapping the operator hand motion and force feedback to the the remote tool. Overall distributed framework and design independence improves the portability and modularity of the proposed telerobotic system. Thread engineering proved to be effective in achieving a sampling rate of 17 Hz for stereo video, 76 Hz for force feedback, and 50 Hz for operator commands.

VI. Acknowledgement

The authors acknowledge computing support and conference attendance support from King Fahd University of Petroleum and Minerals (KFUPM). This work is supported by King Abdulaziz City for Science and Technology under research project grant KACST AT-20-80.

References

[1] R. D. Howe; Y. Y. Matsuoka. Robotics for surgery. *Annual Review of Biomedical Engineering*, pages 211–240, 1999.

[2] Y. E. Ho; H. Masuda; H. Oda; L. W. Stark. Distributed control for teleoperations. *IEEE/ASME Transactions On Mechatronics*, 5(2):100–109, June 2000.

[3] A. Al-Harthy. Design of a telerobotic system over a local area network. *M.Sc. Thesis, King Fahd University of Petroeum and Minerals*, January 2002.

[4] T. Ho. System architecture for internet-based teleoperation systems using java. Master's thesis, Department of Computing Science, University of Alberta, Canada, 1999.

[5] W. J. Book; H. Lane; L. J. Love; D. P. Magee; K. Obergfell. A novel teleoperated long-reach manipulator testbed and its remote capabilities via the internet. *Proc. of the IEEE Inter. Conf. on Robotics and Automation*, 3, No. 6:1036–1041, 1997.

[6] F. Goktas; J. M. Smith; R. Bajcsy. Telerobotics over communication networks. *IEEE Conference on Decision and Control*, 3:2399–2404, 1997.

[7] R. Marin; P.J. Sanz; J.S. Sanchez. A very high level interface to teleoperate a robot vis web including augmented reality. *Inter. Conf. on Robotics and Automation*, pages 2725–2730, 2002.

[8] P. Cohen J. Gu; P. Augirre. Augmented reality interface for telerobotic application. *6th IEEE Workshop on Apps. of Computer Vision*, 2002.

[9] J. Singer. JVM versus CLR: A comparative study. *2nd International Conference on the Principles and Practice of Programming in Java*, 2003.

[10] Microsoft. MSDN library. *http://msdn.microsoft.com/default.asp*.