A Multi-Threaded Distributed Telerobotic Framework

Mayez A. Al-Mouhamed, Onur Toker, and Asif Iqbal,

***Abstract*— A reliable real-time client-server telerobotic system is proposed using a *Distributed Component Framework* to promote software reusability, ease of extensibility, debugging, and data encapsulation. .NET remoting is used for automatic handling of the network resources and data transfer while isolating the components from network protocol issues. A client-server transfer of live stereo video provides the operator 3D views of the slave scene with augmented reality (AR) framework and services. Overall distributed framework and design independence improves the portability and modularity of the proposed telerobotic system. A multi-threaded execution is proposed for streaming of force, command, and for the transfer of live stereo video data. The proposed framework provides a useful integrated software and hardware environment to enhance man-machine interactions using stereo visualization and AR in real-time telerobotic systems.**

***Index Terms*— Distributed application framework, reflected force feedback, man-machine interface, 3D visualization, telerobotics.**

I. Introduction

ROBOTIC technology [1], [2] is enhancing surgery through improved precision, stability, and dexterity. Depth perception and haptic sensing [3], [4], [5] are needed at the surgeon's console. Needed media data have increasing sampling rate which requires tradeoffs between quality and sampling frequency. A central problem is how to design a man-machine interface for the implementation of effective telerobotic systems that extends human manipulative skills over a distance. For this a real-time framework that efficiently and pervasively integrates physical robot, sensors, software, and hardware is highly desirable.

Internet telerobots [6] (ITRs) can be driven by anyone and include robots that navigate undersea, drive on Mars, visit museums, float in blimps, handle protein crystals, etc. The study of ITRs enabled the assessment of communication delays and the analysis of supervisory control.

A WWW remote supervisory control architecture combining computer network in an autonomous mobile robot with collision avoidance and path planning is proposed in [7]. Sheridan[8] defines a model of supervisory control. The operator provides system commands to a human interactive computer that controls a task interactive computer which translates higher level goals into a set of commands.

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

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

IPP Multisensorics, Center for Sensor Systems, ZESS, University of Siegen, D-57068, Germany. iqbal@ipp.zess.uni-siegen.de

Hu et. al.[9] proposed JAVA for network interfacing and video as well as the use of C++ for the robot controller for Internet-based telerobotic system. TCP/IP sockets [10] is one highly reliable method for sending information over the Internet using VxWorks real-time multitasking system. By prioritizing the tasks the user can control the order of task execution and the amount of CPU time allocated to each task.

A component-based distributed control for Internet teleoperations using DCOM and JAVA is proposed in [11] to explore the foot of a volcano using a mobile robot. JAVA is used for database connectivity and path planner GUI and DCOM for network connectivity. JAVA virtual machine (MS VM) is proposed to bridge the gap between JAVA and DCOM. The robot position feedback is provided by two paths, one from the GPS (Global Positioning System) data and the second one from the visual feedback.

A DCOM-based distributed component system [12] is proposed to integrate web technologies and telerobotics together with environmental constraints. The main feature is a DCOM/ActiveX based supervisory control server operating over the Internet. Operator views 3-D model, control paths, and issue commands through supervisory control. Ho [13] developed an Internet based telerobotic system using JAVA and VRML. JAVA-based frame is used as grabbing software to move an image from camera to main-memory.

A CORBA-based distributed robot object model [14] is proposed for mobile robotics. Object communication uses timely leased communication patterns associated with the availability of some resources. A distributed perception strategy [15] is proposed for internet robotics using the perception-motion process (behavior) and task planning (mission). The objective is a robot that can find neighboring Internet sensors for improved reliability. For this, behavioral objects are transported via Internet to favor local interaction which improves stability and synchronization when Internet becomes unreliable. A distributed robot architecture [16] is proposed for modularity for integrating learning aspects in a mobile robot. The robot functions are designed as hardware agents whose resources and notifications are managed by an agent manager. Behavioral functions are flexible programs that are created by task knowledge learning and managed by an agent manger. An environment model is used for tasks, skills, and objects. Distributed objects communicate using CORBA publish-subscribe mechanism.

The classical remote procedure call is redesigned for resilience, transparency, and event-driven notification [17]. CORBA is used to connect objects across heterogenous processing nodes. Using CORBA event service two interaction

models are proposed: (1) an event channel be operated using proxies for client and server for communication with multiple clients where active servers push events on registered clients, or (2) a notification scheme where a client dynamically subscribes to a set of events which define event priority and lifetime.

A subsumptive, distributed, vision-based robotic architecture is proposed in [18]. To improve system fault tolerance different behavioral strategies are coded into multiple loops (fine-to-coarse) where each loop adds to the competence level of the loops below. The loops are independently processed and their results ranked by an arbitrator using application-based criteria to decide which loop should instantly control the robot. The decoupling of video processing and communication is similar to proposed multithreaded execution for concurrent frame acquisition and transmission.

We propose a reliable, real-time, telerobotic framework using object-oriented distributed components technology. .NET remoting technology is used to allow a software component to be developed in one computer and used in many different computers. All updates on an instance of a component (object) can be seen in all connected computers as if it is a local component. Stereo vision greatly enhances the operator's efficiency but imposes severe requirements in terms of bandwidth to transfer real-time video data in a client-server environment. A client-server framework for grabbing and live video transfer is proposed.

The organization of paper is as follows. In Section II the robotic aspects of the multi-threaded distributed framework are presented. In Section III the software system design is presented. In Section IV the live stereo video transfer is presented. In Section V the results are briefly presented. In Section VI a comparison to others is presented. We conclude in Section VII.

II. A MULTI-THREADED DISTRIBUTED FRAMEWORK

The aim is to extend natural eye-hand motion coordination through a computer network while preserving human manipulative dexterity in scaled working environments. The objective is to develop a multi-disciplinary telerobotic research environment integrating motion, vision, and haptic senses to experience telerobotic system interactions, man-machine interfacing, and computer aided teleoperation (CAT).

We present a *Multi-Threaded Distributed Framework* (MTDF) for Telerobotics. The proposed framework consists of components and patterns that establish how components interact with each other. The server station components (Section III-A) are (1) a slave arm components that consist of a robot PUMA (S_{PUMA}) and a Force (S_F) components, and (2) a video (S_V) component (Section IV). The client station components (Section III-B) are (1) a master arm component that consists of a Motion (C_M) and a Force (C_F) components, and (2) a video component (C_V). All client (server) components are concurrently run as independent threads on the client (server) computer. For example, one pattern consists of real-time thread S_V (also S_F and C_M) that is logically interconnected to C_V (also C_F and S_{PUMA}) to which it sends

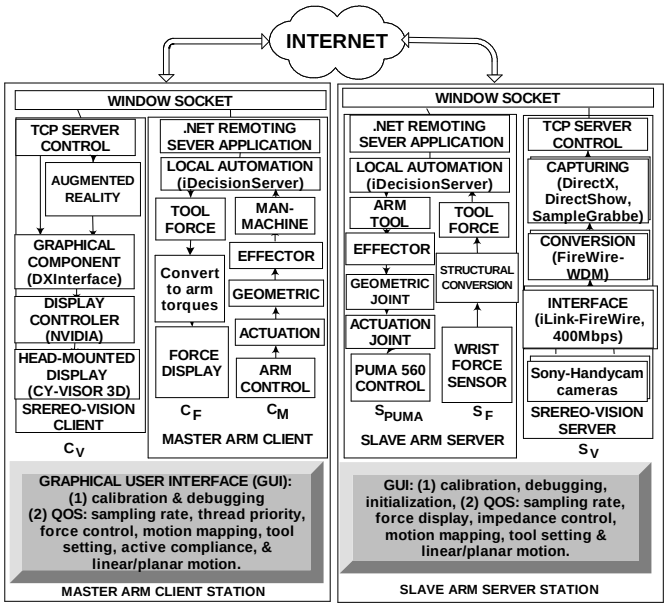

Fig. 1. A layered representation of client-server telerobotic system

data through the network. A layered representation of proposed client-server telerobotic system is shown in Figure 1. In the following we present the robotic aspects of the proposed *Multi-Threaded Distributed Framework* (MTDF).

1) Server motion coordination: The kinematics of the 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), 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 defined by its position vector T_t and its orientation matrix M_e^t of frame R_t with respect to frame R_e. The position of the tool point is defined by $T_w = E_w + M_w^e(\theta)M_e^t T_t$. 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 is:

$$\Delta E_w = M_w^t(I - \Delta M_t)T_t + \begin{cases} \Delta T_w & \text{Operator-tool} \\ M_w^t \Delta T_t & \text{Operator-world} \end{cases} \quad (1)$$

where $M_w^t = M_w^e M_e^t$. The new effector orientation matrix (controllable) becomes $\Delta M_e = M_e^t \Delta M_t M_t^e$. To avoid cumulative errors in the above equation Gram-Schmidt orthogonalization [19] is used to guarantee that the columns of ΔM_e represent unit vectors that are orthogonal. The PUMA reads current joint vector θ and computes 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 $\theta^+ = G^{-1}(E_w^+, M_w^{e+})$ provides the joint vector θ^+ that moves the tool by the commanded translation ΔT and rotation ΔM. θ^+ is sent to the slave arm motion controller. Incremental change in operator hand frame R_{op} is superimposed on tool frame R_t. For example, when R_{op} is tilted the remote tool frame R_t is tilted by the same

angle.

2) Server active compliance: The force sensor consists of two parallel plates p_1 (frame R_e) and p_2 (frame R_s) interconnected by three elastic links. The force sensor used is developed by the authors [20]. The motion of p_2 with respect to p_1 is measured by a (1) translation vector ΔS_e, and (2) orientation matrix ΔM_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 R_e and 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 $M_e^t = \Delta M M_s^t$ the tool deflection vector is $\Delta T_t = M_t^s \Delta M^{-1} \Delta T_e$.

Active Compliance (AC) control consists of converting the measured force into an incremental motion for the slave tool. The force (F_t) and moment (C_t) vectors are computed using 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. F_t and C_t are used to: (1) display the reflected force feedback at the client station, and (2) implement active compliance mechanism as a force control strategy as shown in Section III-A.3.

3) Assistance functions: Server teleoperation assistance functions can be activated by buttons at the client station. The assistance functions are:

1) Relative or world mapping: operator motion is mapped to: (1) world frame, or (2) tool frame in tool manipulation tasks.
2) Floating tool mapping: operator motion is dynamically mapped to the slave tool frame by defining the tool frame position and orientation at some point of interest for the task. This may greatly reduce the number of iterations done by the operator to set up the manipulated object in a given position and orientation.
3) Planar or linear motion: constraining some tool motion axes to linear or planar motion and leaving the other axes under direct operator manual control.
4) Force Control: implements active compliance by continuously sensing the force exerted on the tool, evaluates a proportional force error based on a desired force, and converts the error into a position increment to reduce the force error. Here the user may select setting up active compliance over a sub-set of tool axes while other axes are kept under position control. In this case the selected components of computed force F_t and moment C_t vectors are feedback as elementary tool translation $\Delta T = A F_t$ and rotation $\Delta M = B C_t$, where A and B are two 3×3 matrices that determine the selected axes.

4) Client CAT support: The client station have a set of button-controlled teleoperation functions which are: (1) real-time rendering of the operator motion, (2) indexing, (3) space scalability, and (4) impedance control. In the following we present each of these functions.

Real-time rendering of the operator motion and display of force feedback are implemented as follows. There are two major inputs: (1) the joint vector read from position sensors and (2) force data coming from the remote side. Arm kinematic model $G_M(\theta)$ allows computing the current operator hand position vector X^+ and orientation matrix M^+, where θ is the arm joint vector. Using last references X and M, it computes the variations $\Delta X = X^+ - X$ and $\Delta M = M^t M^+$ with respect to reference. The client sends the above computed variations to the slave arm as an incremental motion command for the slave tool frame. The client also outputs the received force feedback as master arm motor torques to display the force feedback on operator hand.

The indexing function is defined as follows. In direct teleoperation the variation in operator hand position and orientation $(X^+ - X, M^{-1} M^+)$ is evaluated and transmitted to server. During indexing the system continuously sets up the reference to current (X, M) and disable transmission to slave arm.

The scalability function is defined as follows. The increment in master position vector (ΔX) and orientation matrix (ΔM) are scaled-down before being transmitted to the server. For this the variation in the operator hand orientation matrix (ΔM) can be seen as made of three euler angles, i.e. $\Delta M = R_x(\alpha_x) R_y(\alpha_y) R_z(\alpha_z) = R_{xyz}(M)$, where R_u is a rotation matrix about axis u and R_{xyz} is the product of three rotation matrices sets for ΔM. Since ΔM is known, we inverse the above equation and find the three angles as $(\alpha_x, \alpha_y, \alpha_z) = R_{xyz}^{-1}(\Delta M)$. Using an operator-controlled scale factor s, the scale function becomes:

$$(\Delta X, \Delta M) = ((X^+ - X) * s, R_{xyz}((R_{xyz}^{-1}(\Delta M)) * s))) \quad (2)$$

To avoid singularities, the three Euler angles are computed for the variation in the operator orientation matrix ΔM. Due to speed of operator motion and real-time mapping, the computed angles are small which avoids the singular area.

The impedance control is implemented as follows. The master arm dynamics [21] determine the motor torque τ as $\tau = D(q)\ddot{q} + C(q, \dot{q}) + G(q)$, where q joint vector, $\dot{q}$ joint velocity, $\ddot{q}$ joint acceleration, $D(q)$ is the inertia matrix, $C(q, \dot{q})$ is the coriolis and centrifugal coefficients, and $G(q)$ is the gravity term. We compute terms $C(q, \dot{q})$, $G(q)$, and $D(q)$ which enables finding the motor torque:

$$\tau = \alpha \ddot{q} + \beta \dot{q} + C(q, \dot{q}) + G(q) + \tau_{ff} \quad (3)$$

where term $\alpha \ddot{q} + \beta \dot{q}$ is generated based on the operator motion, terms $C(q, \dot{q})$ and $G(q)$ are used to compensate for dynamic effects and gravity, and τ_{ff} is the force feedback. The overall dynamic motion equation becomes:

$$\tau_{ff} = (D(q) - \alpha)\ddot{q} + \beta \dot{q} \quad (4)$$

where term $D(q) - \alpha$ represents the reduced master arm inertia (impedance) and $\beta \dot{q}$ is a motion damping factor. The values of the parameters α and β are experimentally determined based on a performance/stability criterion.

III. Software system design

A. Server side components

The server components are: (1) PUMA Component S_{PUMA}, (2) Force Sensor Component S_F, and (3) Decision

Server Component. In addition to these components, we also have three interfaces known as (1) Proxy Robot Interface (2) Force Sensor Interface, and (3) Decision Server Interface which will be presented subsequent sections.

1) PUMA component: S_{PUMA} acts as a software proxy of the robot for which commands are issued to the component as they are issued to the robot. Some important *public methods* exposed by S_{PUMA} include *ConnectRobot* that connects server to slave robot, *InitializeRobot* sends a program to robot that repeatedly moves the robot in an incremental fashion. Some properties are: (1) reading robot angles, (2) computing the position vector and orientation Matrix of robot hand frame, (3) setting up specification of robot tool frame, (4) setting up the communication between server and robot, etc. The events invoked by S_{PUMA} include: (1) Data received from PUMA, (2) some errors occurred with PUMA, (3) Robot moved to a new location, and (4) PUMA status changed.

2) Force sensor component: The force sensor component S_F reads the robot wrist force sensor and creates a stream of reflected force feedback directed to the master station. S_F is implemented as a separate thread, the priority of which can be adjusted during runtime to allow for the management of CPU usage. In .NET remoting technology, events can be issued from one station and received in another, i.e. automatic transfer of some data. Events do contain certain parameters, for example MouseClick(3,5) event indicates a mouse click event at position at 3 and 5. Here a new instance of S_F creates a new thread and waits until the sensing is triggered. After the reading has started, it informs the parent application (PA) of the availability of a new force packet. The PA uses an event handler at the higher level of application hierarchy. The event directly transfers the force information to the client. Similarly the component also provides *StopReading()* function to abort the force sensing thread. Sensing can be triggered again using *StartReading()*.

There are three public properties exposed by S_F. The *SensorThreadPriority* sets the priority as one out of five OS levels. The *TimerValue* sets a time interval between two successive readings. The *ThresholdValue* activates the force event only when there is noticeable change in one of the force values.

3) Decision server component: DecisionServer is a component that provides a supervisory control such as active compliance, impedance control, and assistance functions. The DecisionServer is a server abstraction layer to allow: (1) active compliance control, (2) supervisory commands like space scalability and indexing. A four-layer hierarchy including DecisionServer is shown in Figure 2, where physical layer refers to robot and force and lowest layer refers to the user interaction level.

4) Server side interfaces and .NET remoting: Proposed telerobotics is based on object oriented distributed application. The Decision server interface allows the client to receive the events fired by the DecisionServer instance on server side. An interface is a set carrying definitions of public methods and properties. It serves as a contract [22] for any component that implements this interface. Any client that accesses or executes the methods of a component on the server needs an access to the server assembly or component. By giving a reference to an interface that the server component implements, we can hide the actual component or assembly from the client which improves overall system security.

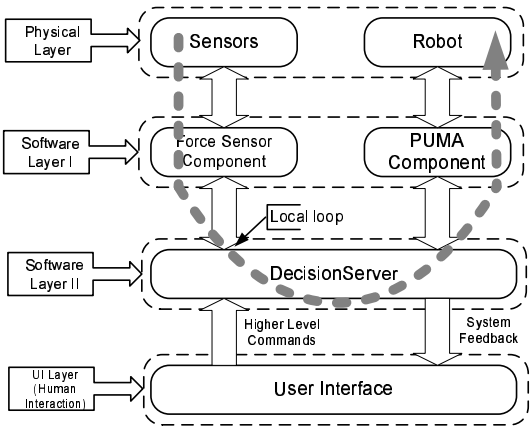

Fig. 2. Component hierarchy on the server side

To access references to both the PUMA and Force Sensor components, two interfaces are defined: *IProxyRobot* and *IForceSensor*. Further we define another interface *IDecisionServer* which inherits both the *IProxyRobot* and *IForceSensor* interfaces. Using this approach we are able to define a unified set of public members (methods, properties and events) that are required to be implemented in the form of DecisionServer component on the server side. The integrated scheme incorporating all the components on server side is shown in Figure 3.

.NET remoting is used to publish an instance of DecisionServer component on the network which is uniquely identified to potential clients. A client can get a reference to this instance through an *IDecisionServer* interface. .NET remoting enables accessing objects using SOAP (Simple Object Access Protocol). Any object in the distributed application can be referenced using the above scheme as if it was available on the same machine.

B. Client side components

The client contains the *IDecisionServer* interface that allows referencing the server side component through .NET remoting as shown in Figure 4. In addition to *IDecisionServer*, there are instances of .NET remoting and client Graphic User Interface (GUI).

1) Decision server interface: The DecisionServer is inherited from IDecisionServer and in turn from IProxyRobot and IForceSensor interfaces. .NET remoting is responsible for making socket calls to the client and we may choose either network protocol for these requests.

DecisionServer provides a mechanism for remote execution of any event handler program. The client assembly must be known to the DecisionServer which violates object oriented philosophy and reduces security. For this *Shim Classes* are used as agents to forward DecisionServer events over to the *IDecisionServer* interface. Shim classes are thin assemblies visible to both the server and the client. DecisionServer

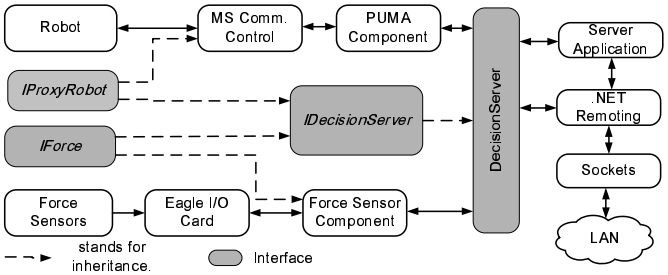

Fig. 3. Integrated scheme - server side

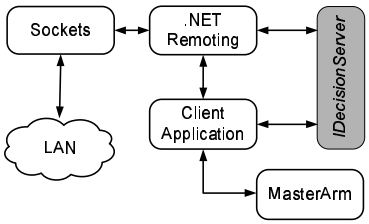

Fig. 4. Integrated scheme - client side

invokes the event which is received by an event handler hooked by shim classes. This event handler then calls the event handler of the client (*IDecisionServer*).

Care must be taken while receiving events from the server and writing event handlers for them because these are synchronous events which means that the thread that is invoking the event on the server side will be blocked until all the event handlers for this event are executed. To send a single TCP packet, eventually a system call is needed, which is an atomic operation, e.g. it cannot be pre-empted due to a higher priority packet. So manipulating threads during the invocation of the events may cause deadlocks in the distributed client-server environment.

2) MasterArm component: Public methods used by MasterArm are used to: (1) start and stop reading the master arm position (Inherited from *Force* component), and (2) write the given force data to the master arm in a separate thread. Now we describe some of the public properties. One property computes the change in position vector and orientation matrix after the position data ready event is fired. A property is used to find/set whether a master arm is engaged or not to control computation of arm kinematics. A get/set property is used to indicate whether to provide force feedback to the master arm or not. Other properties are also used to compute the local impedance control function. A block diagram of the *MasterArm* is given in Figure 5.

C. Integrated scheme of client-server components

The integrated scheme incorporating all the components on client and server side is shown in Figures 3 and 4. The DecisionServer is inherited from *IDecisionServer* and in turn from IProxyRobot and IForceSensor interfaces. In order to cause an event handler subprogram to be called (invoked) on client side for any event invoked by DecisionServer, we must provide DecisionServer, access to the client assembly.

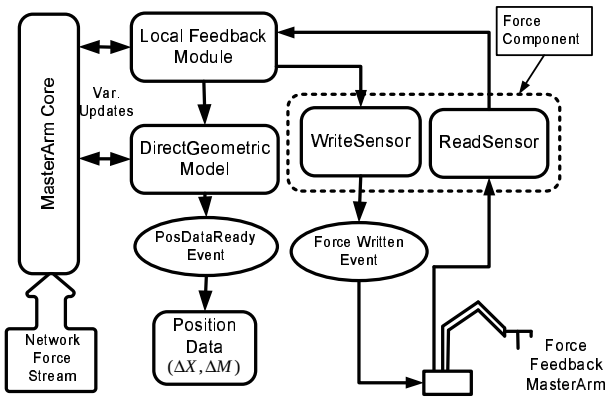

Fig. 5. MasterArm component

This introduces severe deployment limitations as described in Section III-B.1. Here DecisionServer invokes the event which is received by an event handler hooked by shim classes. This event handler then calls the event handler of the client (*IDecisionServer*). This allows hiding the server and client assemblies from each other.

D. A multi-threaded distributed telerobotic system

The distributed approach leads the logic of the system be distributed in different software components. The multi-threaded aspects stem from the simultaneous threads running on the server carrying out: grabbing of stereo video data, reading force sensors, sending control signals to the robot, reading the feedback from the robot servo controller, and sending and receiving all of this information to one or more clients. Two cameras generate pictures which are sent to the client using the vision server. The operator may issue commands to the DecisionServer which in turn makes use of PUMA and Force Sensor components. The client side uses the GUI as well as master arm to issue commands to the slave arm on remote side. The vision client receives the synchronized video data using windows sockets and provides a stereo display.

IV. Stereo vision for telerobotics

Stereo vision enhances operator's efficiency during telemanipulation [23]. A client-server framework for live transfer of video data is implemented using Microsoft Visual C# and Microsoft DirectX which provides COM interfaces for graphics related functionalities such as DirectShow. The later provides an interface for capturing and playback of video data. The server captures two stereo images and upon a request from the client it sends video data to the client using windows socket. SampleGrabber (DirectShow) is used to capture video frames coming through a live video transfer as shown in Figure 6.

The client establishes a graphic display, establishes a connection with server, and displays the incoming pictures using a head-mounted display (HMD). A graphics device interface

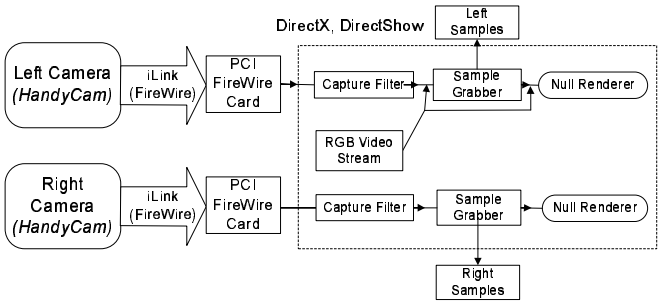

Fig. 6. Sample grabber and camera interfacing at the server

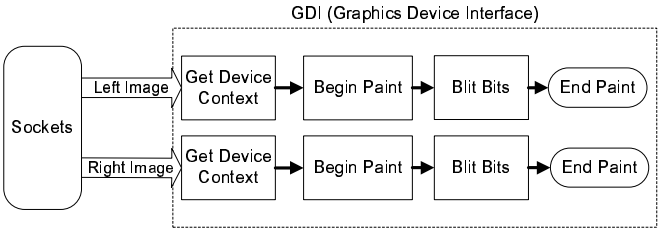

Fig. 7. Stereo vision display on client side

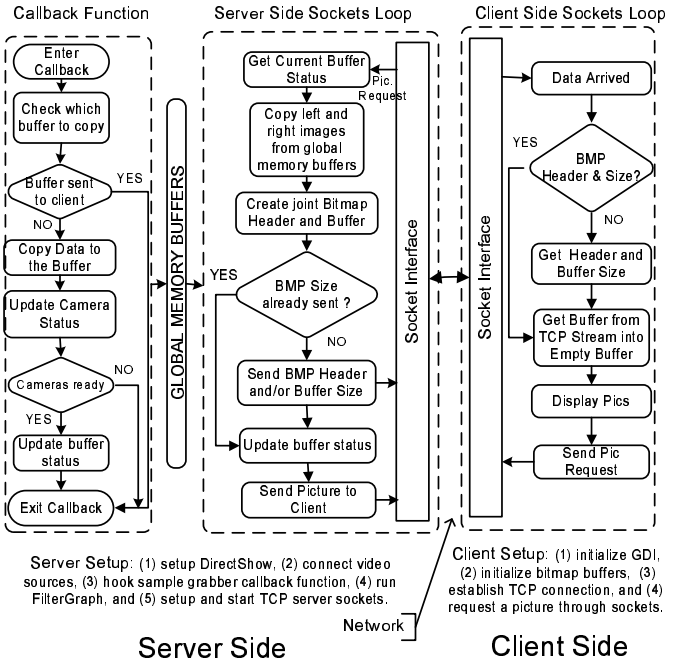

Fig. 8. Live stereo video transfer using the optimized scheme

(GDI) of Windows graphical environment is used to (1) communicate between the application and the device driver, (2) perform the hardware-specific functions that generate output, and (3) display the received picture (see Figure 7). Microsoft bit block transfer API called *BitBlt() or Blit* is used to copy an image from one device context to another.

A. A distributed framework for relaying stereo vision

Synchronous windows sockets are used for video transfer at server station (see Figure 6). To maximize video transfer rate two thread-based schemes are used: (1) a single buffer with serialized transfer, and (2) double buffer, concurrent transfer.

In the single buffer with serialized transfer, the SampleGrabber component of DirectShow [22] uses a callback function to inform the completion of one video frame at server. Two thread instances of SampleGrabber running at the same time to capture the video coming from two cameras. The data is copied by SampleGrabber to some global memory buffer to be sent to the client through sockets. After hooking of callback function onto SampleGrabber, FilterGraph (DirectShow) starts the video capturing. The last step of server socket is to transfer the video data. The server waits for a request of a picture from client to start video data transfer.

On the client, the GDI is set to draw the received pictures. The server flushes the previous bitmap buffers, grab left and right images using callback functions, create a bitmap information header for these images and transfer to the client. The client gets the buffer size (TCP stream), prepares the bitmap buffer, receives the bitmap information header, copies the bitmap data from the sockets into the buffer, requests for new picture, and draws the 3D stereo picture.

In the double buffer scheme, concurrent transfer allows pipelining the execution of video capturing and live video transfer as shown in Figure 8. For this, two buffers are used, one for each stereo frame on the server. When a picture is received, the camera callback function is invoked which accesses a variable shared by multiple threads indicating whether the buffer was copied in the previous callback of this very camera. The camera status is used to synchronize the stereo frames for the left and right pictures. It is updated after copying the data to the buffer. If both cameras are ready, updating the buffer status enables the sending thread to transfer the content of this buffer over to the client. In case the second camera has not finished copying the picture to the buffer, buffer status is not updated.

The sending thread is responsible for receiving requests from the client. It checks the buffer status to determine the buffer, creates bitmap headers, and retrieves the buffer size. If the information has not already been sent to the client, it is sent. Otherwise, the server continues with the sending of buffer data only. The client proceeds in the same manner as with single buffer approach except that it does not receives the bitmap information header and buffer size with each stereo frame. It retains the bitmap information header and buffer size to properly display and read the required number of bytes from windows sockets.

B. An augmented reality system for telerobotics

Augmented reality consists of overlaying the graphics on real stereo images. A camera model is used to project 3D points on 2D image plane. The full perspective transformation between world and image coordinates is conventionally analyzed using the pinhole camera model and camera calibration proposed in [24]. When scene depth is small as compared to distance from camera to scene, the weak perspective projection is a linear approximation model [25]. In this case image plane coordinates and their pixel addresses can be related by an affine transformation. A GUI was designed to support the user

carrying out camera identification by pointing to four non-coplanar reference points forming a frame of reference. This allows finding left and right camera projection matrices. The fiducial points should be 20 cm distance from frame origin and camera set at no less than 1.5 m from the scene.

1) Superimposing graphics using DirectX API: The image data retrieved from the *StereoSocketClient* comes in bitmap format. It can be displayed using the *DXInterface* component, and HMD. *DXInterface* is designed using *DirectX* Application Programming Interface (API) [26]. Off-screen surfaces are used to superimpose real or virtual images as well as drawing and blitting. The *BackBuffer* service of *DirectX Device* allows indexing and fast switching of primary and off-screen surfaces. The Hardware Abstraction Layer (HAL) and graphics processor control flipping the addresses of front and back buffers. Without image shearing or tearing, the next image drawn on the HMD comes from the previous back buffer. While the previous front buffer is now back buffer and is ready to be used for the coming video frame. New image is acquired from the network while the drawing of the current image is in progress. In short, the stereo video is updated on local display in a page-by-page format and not pixel-by-pixel which reduces time delays.

2) Component framework: In this sub-section the AR stereo vision client and server components are presented.

Client side components: Listed below are the components providing stereo vision and AR functionality on the client side:

1) The StereoSocketClient generates compatible bitmaps from the binary video and receives the live video transfer sent by the vision server developed using MFC (Microsoft Foundation Classes) client-server setup.
2) IdentifyCamera computes camera projection matrices using a set of four non-coplanar points as vector edges of a frame [24]. The GUI uses the images provided by StereoSocketClient in computing the left (M_l) and right (M_r) projection matrices.
3) RobotModel locates the future position of the robot gripper based on the current command. *RobotModel* acts as a local proxy of the PUMA robot.
4) DXInterface (Figure 9) handles (1) augmentation of real video, (2) synchronization of real and virtual data, (3) projection on video surface, and (4) page flipping.

Server side: Server side acquires and sends the stereo image data through windows network sockets. However, only the client side is responsible for the major AR functions. The server side for the stereo video client-server framework is used in the AR framework.

3) The complete augmented reality system: All of these components have been combined together to form a complete AR system on the client side. The system provides the AR functionality through the following steps:

1) Read operator motion using the *MasterArm* component.
2) *MasterArm* provides motion parameters to *IDecisionServer* and *RobotModel* components.
3) *IDecisionServer* executes the incremental move command on remote *DecisionServer*.
4) *RobotModel* provides the new 3D position of gripper to *DXInterface* component.

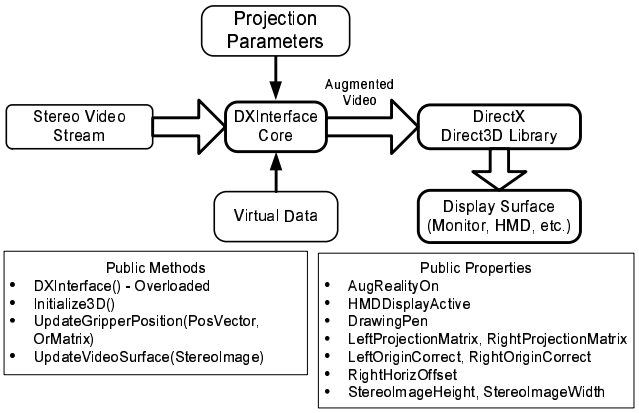

Fig. 9. An overview of DXInterface Component

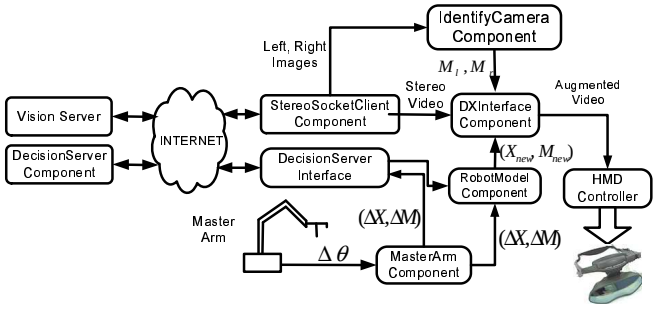

Fig. 10. Block diagram of complete AR system on client side

5) *DXInterface* acquires a frame of remote scene from *StereoSocketClient* as well as two projection matrices from *IdentifyCamera* at the system initialization.
6) *DXInterface* projects a ball at the gripper position in 2D image and sends both images to HMD display controller.
7) When the *IDecisionServer* receives the *OnMove* event from the remote side, the angular position of the robot is sent to the *RobotModel* to update the local model.

An architectural overview of the AR system present on the client side, is given in Figure 10.

It is important to update the local robot angles upon the invocation of *OnMove* event from the server side because there may be some differences between the move command arguments and the current position of robot due to mechanical and mathematical round off errors. Also it is to be noted that the *IDecisionServer* uses *.NET remoting* for network streaming of component data and force data while Vision Server and *StereoSocketClient* use raw windows sockets to live transfer of binary stereo video data.

The system has the ability to remember the identification of cameras and other projection related data across different runs by preserving these values to the permanent memory in a special format. So, the identification is required only when the cameras or the objects have been moved from their previous locations.

V. BRIEF RESULTS

Extensive performance evaluation and experimentation were carried on for the propose telerobotic system. Brief results are

reported in this section.

Analysis of telerobotic delays through three campus routes was carried out while streaming of video, force, and commands. A sampling rate of 120 Hz is achieved for force feedback and 50 Hz for operator commands. Stereo video transfer operates at a rate of 17 fps. Total reference delays for force and stereo are 8 ms and 83 ms, respectively. Overall round-trip delay is 183 ms (5.5 Hz) when slave arm is operated at 10 Hz. More details can be found in [27].

The effectiveness of the framework and concurrent execution of its various computing and communicating threads has been assessed in the experimentation of the following tasks: (1) peg-in-hole insertion, (2) assembly of a water pump, (3) operating drawers, (4) pouring of water, and (5) wire-wrapping. The above experiments [28] involve the completeness, modularity and flexibility of proposed telerobotic framework when rich and heterogeneous sensory data (video, force, and command) was exchanged between client and server. A summary of results is as follows: (1) teleoperation tools are very effective and need to be developed, (2) advanced motion coordination reduces teleoperation time and operator mental effort, (3) active compliance at server station is more effective than operator reaction using force feedback. More details can be found in [29].

VI. Comparison to others

Even et. al [30] describes a computer aided telerobotic system as an integrated planning scheme including interactive 3D modelling, programming, and execution. The proposed telerobotic framework may provide the above system a pervasive network connectivity and mobility as well as a structured software framework for implementing real-time interactions.

In [31] a client-server framework is designed using VB 6.0 and custom protocol with TCP ActiveX controls. This scheme exposes TCP read/write operations to application, custom protocol, TCP ActiveX control, and Windows Sockets etc. While in a distributed setup, the components directly communicate with each other through windows sockets using .NET remoting. This difference is achieved by using the distributed component based approach in place of TCP based custom protocols.

An object oriented distributed application (OODA) could also use JAVA that provides RMI as remote calling function, graphics services, and 2D and 3D support. Hu et. al.[9] proposed JAVA for network interfacing and video as well as the use of C++ for the robot controller for Internet-based Telerobotic Systems. Ho[13] used Java-based framework for frame grabbing as compared to the use of DirectShow in our scheme. Compared to [12], .NET framework is directly used for all GUI development as well as the core system components making it a unified solution. This avoids the use of middleware services like MS VM within the framework. JAVA and CORBA are intended to be cross-platform environments thus requiring lot of JIT (just-in-time) compilation and virtual machines to interpret code on different operating systems. In addition they provide no support for hardware-accelerated graphics APIs that are critical for live video visualization on PCs.

Compared to [15], the behavior in proposed telerobotic framework is represented by the local position, force, and active compliance loops and the planer is being the human operator and AR system. Compared to [18], in proposed framework the coarse correction made by the remote teleoperator adds to the competence of fine active compliance loop during the period of contact with the environment. Compared to [17], in proposed framework there are three servers and three clients at any given time. Object interconnection is implicitly done using .NET remoting.

We proposed an object-oriented distributed component framework and .NET remoting for (1) remote procedure call, (2) hiding details of network interface, (3) an automatic notification and data messaging mechanism between client and server. .NET does not require components registration thus breaking the interdependency in the development phase. To develop off-the-shelf components and programs we used Windows 2000 because of support provided for hardware accelerated graphics, .Net remoting, and thread scheduling and the priority needed for multi-threaded execution. .NET may be used for off-the-shelf real-time applications as it casts off every possible overhead. .NET has embedded type signatures which allow component debugging across different languages, a missing feature in JAVA and CORBA. Process variables like real-time sensor data and robot-states are relayed to the client-side using implicit inter-component communication.

VII. Conclusion

In this paper a real-time *Distributed Component Telerobotic Framework* has been described using object-oriented distributed programming paradigm with Visual C #, .NET remoting, DirectX, and TCP sockets. .NET remoting is used to automatically handle the network resources and data transfer while isolating the components from network protocol issues. This approach frees the programmer from defining custom protocols for client-server interaction. It also provides flexible deployment environment without pre-registration of components on host machine which is an advantage over DCOM. The framework is implemented as a modular multi-threaded system for live transfer of stereo video, transfer of master-slave commands, and streaming of force feedback. The functionalities of master and slave arms are independently designed. A multi-threaded execution with DirectX graphical tools has been used to promote concurrency in grabbing, transmitting, receiving, processing, and displaying image data using HMD technology. The client station provides components that support AR tools like camera model identification, and graphical slave arm components. The proposed framework is useful to enhance man-machine interactions with 3D perception and AR, and to serve as an integrated environment to develop telerobotics.

References

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

[2] A. Rovetta, R. Sala, Xia Wen, and A. Togno. Remote control in telerobotic surgery. *IEEE Trans. on Sys., Man, and Cybernetics, Part A*, 26(4):438–444, 1996.

[3] L. Stocco and S.E. Salcudean. A coarse-fine approach to force-reflecting hand controller design. *Proc. of the Inter. Conf. on Robotics and Automation*, 1:404–410, 1996.
[4] R.D. Howe. A force-reflecting teleoperated hand system for the study of tactile sensing in precision maniplulation. *IEEE Inter. Conferance on Robotics and Automation*, pages 1321–1326, May 1992.
[5] H. Iwata. Artificial reality with force-feedback: Development of desktop virtual space with compact master manipulator. *ACM SIGGRAPH Computer Graphics*, 24 (4), 1990.
[6] K. Goldberg and R. Siegwart, editors. *Beyond Webcams: an Introduction to Online Robots*. The MIT Press, Cambridge, MA, USA, 2001.
[7] R. C. Luo and T. M. Chen. Remote supervisory control of an autonomous mobile robot via world wide web. *Proc. of IROS '97*, 2:1163 – 1168, 1997.
[8] T. Sheridan. Supervisory control. *G. Salvendy (Ed.) Handbook of human factors and ergonomics*, pages 1295–1327, 2002.
[9] Huosheng Hu, Lixiang Yu, Pui Wo Tsui, and Quan Zhou. Internet-based robotic systems for teleoperation. *International Journal of Assembly Automation*, 21(2):143–151, May 2001.
[10] W.J. Book, H. Lane, L.J. Love, D.P. Magee, and 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(6):1036–1041, 1997.
[11] Y.E. Ho, H. Masuda, H. Oda, and L.W. Stark. Distributed control for tele-operations. *Proc. of the 1999 IEEE/ASME International Conf. on Adv. Intelligent Mechatronics*, pages 323–325, September 1999.
[12] Y.E. Ho, H. Masuda, H. Oda, and L.W. Stark. Distributed control for tele-operations. *IEEE/ASME Transactions on Mechatronics*, 5(2):100–109, June 2000.
[13] Teresa Ho. System architecture for Internet-based teleoperation systems using java. Master's thesis, Department of Computing Science, University of Alberta, Canada, 1999.
[14] D. Wang, X. Ma, and X. Dai. Web-based robotic control system with flexible framework. *IEEE Inter. Conf. on Robotics and automation*, pages 3351–3356, 2004.
[15] X. Hou and J. Su. New approaches to internet based intelligent robotic system. *IEEE International Conf. on Robotics and Automation*, pages 3363–3368, 2004.
[16] S. Knoop, S. Vacek, R. Zollner, C. Au, and R. Dillmann. A CORBA-based distributed software architecture for control of service robots. *IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems*, pages 3656–3661, 2004.
[17] M. M. Arnoretti, S. Bottazzi, M. Reggiani, and S. Caselli. Evaluation of data distribution techniques in a CORBA-based telerobotic system. *Intl. Conf. on Intelligent Robots and Systems*, pages 1100–1105, 2003.
[18] G. N. DeSouza and A.C. Kak. A subsumptive, hierarchical, and distributed vision-based architecture for smart robotics. *IEEE Trans. on Sys., Man, and Cyber.-Part B: Cybernetics*, 34:5:1988–2002, 2004.
[19] The Gram-Schmidt algorithm. *http://www.math.hmc.edu/calculus/ tutorials/gramschmidt/, 2002*.
[20] M. Al-Mouhamed, M. Nazeeruddin, and N. Merah. Design and analysis of force feedback in telerobotics. *Submitted for publication*, December 2005.
[21] J.P. Desai and R.D. Howe. Towards the development of a humanoid arm by minimizing interaction forces through minimum impedance control. *Proc. IEEE International conference on Robotics and Automation, ICRA*, 4(2):4214 – 4219, 2001.
[22] Microsoft's DirectX and .NET websites. *http://www.microsoft.com/directx/ and http://www.microsoft.com/net/, 2002*.
[23] Xi Ning and T.J. Tarn. Action synchronization and control of Internet based telerobotic systems. *Proc. of IEEE Inter. Conf. on Robotics and Automation*, 1:219–224, 1999.
[24] O.D. Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? *In Proc. 2nd European Conference on Computer Vision, LNCS 588, Springer-Verlag*, pages 563–578, 1992.
[25] N. Hollinghurst and R. Cipolla. Human-robot interface by pointing with uncalibrated stereo vision. *Image and Vision Computing*, 14(3):171–178, 1996.
[26] Microsoft. MSDN library. *http://msdn.microsoft.com/default.asp, 2002*.
[27] M. Al-Mouhamed, O. Toker, and A. Iqbal. Performance evaluation of a multi-threaded distributed telerobotic framework. *Submitted for publication*, December 2005.
[28] Telerobotics video clips. *http://www.ccse.kfupm.edu.sa/researchgroups/ robotics1/, April, 2003*.
[29] M. Al-Mouhamed, M. Nazeeruddin, and Sayed Islam. Experimentation of a multi-threaded distributed telerobotic framework. *Submitted for publication*, December 2005.
[30] E. Even, E. Fournier, and R. Gelin. Using structural knowledge for interactive 3D modeling of piping environments. *Proc. of IEEE Inter. Conf. on Robotics and Automation*, pages 2013–2018, Apr. 2000.
[31] M. Al-Mouhamed, O. Toker, and A. Al-harthy. A 3D vision-based man-machine interface for telerobotics. *IFAC Inter. Conference on Intelligent Control and Signal Processing*, April 8-11 Portugal, 2003.

PLACE PHOTO HERE

Mayez Al-Mouhamed (Al Dandashi) was born in Tal-Khalakh (Syria) in 1950. He received his B.S., M.Sc., and Ph.D. (Doctorat D'Etat) degrees in Electrical Engineering from the University of Paris XI, Paris-France, in 1975, 1977, and 1982, respectively. From 1976 to 1983 he served as a Ph.D. student in the Department of Electronics and Nuclear Instrumentation (DEIN and STEPPA-EMH), Center for Nuclear Studies, Saclay, France. In 1983 he joined the Computer Engineering Dept., King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia. Currently he is a Professor of Computer Engineering. In 1992-93 he served as visiting Associate Professor at the Information and Computer Science Dept., University of California Irvine. He has two registered European patents in the area of Robotics. His research interests is in Computer Architecture, Parallel Architectures and Algorithms, Parallel compiler, Computer Vision, and Robotics.

PLACE PHOTO HERE

Onur Toker got his M.S. degree in Electrical Engineering, and Mathematics, and Ph.D. degree in Electrical Engineering in 1992, 1994, and 1995 respectively, all from the Ohio State University, Columbus OH. He worked as a postdoc researcher in Eindhoven University of Technology, Holland, and in University of California, Riverside CA. He joined KFUPM in the period 1997-2005 as an Associate Professor. He is currently at Fatih University, Electronics Engineering Department, 34500 Buyukcekmece, Istanbul, Turkey. His research interests include real-time Java, mechatronics, embedded systems, robust control, and telerobotic systems.

PLACE PHOTO HERE

Asif Iqbal was born in Khushab, Pakistan in 1975. He received his B.Sc. degree in Mechanical Engineering from University of Engineering and Technology, Lahore, Pakistan in 1999. After working in industry for one year, he joined King Fahd University of Petroleum and Minerals (KFUPM) as research assistant in 2001. He received his M.S. degree from KFUPM in Systems Engineering in 2003. Currently he is a PhD candidate at IPP Multisensorics, Center for Sensor Systems (ZESS), University of Siegen, D-57068, Germany. His research interests lie in real-time control, teleoperation, system identification, and embedded systems.