DESIGN OF A MULTI-THREADED DISTRIBUTED TELEROBOTIC FRAMEWORK

Mayez Al-Mouhamed, Onur Toker, and Asif Iqbal

College of Computer Science and Engineering
King Fahd University of petroleum & Minerals
Dhahran 31261, Kingdom of Saudi Arabia.
mayez/onur/aiqbal@ccse.kfupm.edu.sa

ABSTRACT

A telerobotic system consists of master (client) and slave (server) stations which are usually connected by a computer network. A reliable real-time connection between master and slave systems is proposed using *Distributed Components (.NET Remoting)*. This has a number of benefits such as software reusability, ease of extensibility, debugging, and data encapsulation. It is based on most advanced software tools like *.NET Framework* that promise definite advantages over *DCOM (Distributed Component Object Model)* and *RPC (Remote Procedure Call)*, previously used for distributed applications. The components communicate with each other using *.NET Remoting* and *SOAP (Simple Object Access Protocol)* that automatically handle the network resources and data transfer while isolating the components from network protocol issues. This enhances the data security as well as facilitates easy deployment. Implementing telerobotics using the proposed approach gives the advantage of a multi-threaded execution needed to effectively realize multi-streaming of force, command and stereo data over a LAN.

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

1. INTRODUCTION

Telerobotic technology [1] is enhancing minimally invasive surgery (MIS) through improved precision, stability, and dexterity. The surgeon moves a dexterous master arm (client) that is scaled down to a slave arm (server) inside the patient's body. Reliable and secure real-time streaming of force feedback and stereo vision are critical in many telerobotic applications

In [2] a telerobot is implemented using TCP/ATM in which two LANs are connected to an ATM backbone. Specification of Quality-of-Service (QoS) includes application timing, criticality, clock synchronization, and reliability. This is accomplished by using a constant bit rate (CBR) ATM connection allowing a tightly constrained transmission delay which is suitable for real-time applications. Random delays in the robot closed loop control affect stability and performance.

Real-time network and protocol transmission delays, jitter [3], and processing times need to be optimally reduced to ensure guaranteed quality of service for robot commands, stereo vision, and force feedback. In a computer network, the communication delays and traffic capacity vary with flow direction and irregularly change with traffic conditions.

Real-time operating system VxWorks is used to compute a scattering transformation to guarantee state stability. A distributed component based design for telerobotics using *DCOM/ActiveX* approach [4] is proposed to integrate web technologies and telerobotics together with environmental constraints. Operator views 3-D model, control paths, and issue commands through supervisory control.

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. These distributed components use *SOAP* as a way to communicate with each other and implement data remoting for real-time updates of sensor data and process statuses.

Although Java has been used in many telerobotic systems but we choose *.NET Framework* because of the following reasons:

1. We target to use the proposed framework on a commodity LAN where Microsoft technologies give optimized performance [5]. Java is recommended for Internet-based cross-platform environments.

2. *.NET* components can be easily deployed to work across firewalls.

3. *CLR (Common Language Runtime)* used by *.NET Framework* is similar to JVM (Java Virtual Machine) because it also compiles the source code into platform-independent bytecode [6].

Also it should be noted that in a typical scenario when both client and server use *.NET* based components with TCP channels, highly optimized data transfer is obtained [7].

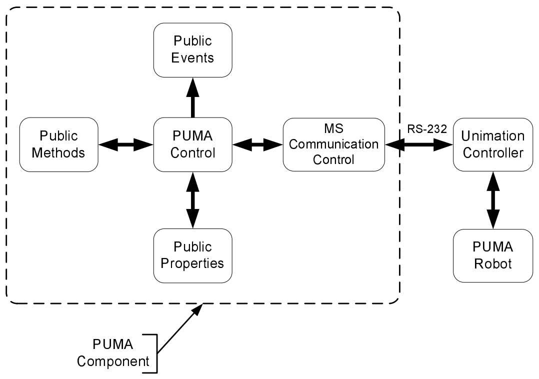

Figure 1. Block diagram of PUMA Component

The organization of the rest of the paper is as follows. Section 2 presents our distributed framework. In Section 3 we compare our approach to others. We conclude in Section 4.

2. A MULTI-THREADED DISTRIBUTED FRAMEWORK

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

2.1. Server Side Components

The server components are: (1) PUMA Component, (2) Force Sensor Component, 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 The details of the functionalities of these components and interfaces will be discussed in the following sub-sections.

2.1.1. 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. A block diagram of the PUMA component is shown in Figure 1.

Some important public methods exposed by PUMA component include *ConnectRobot* that connects server to slave robot, *InitializeRobot* sends a program to robot that repeatedly moves the robot in either joint or cartesian space in an incremental fashion which reduces the communication data payload.

A command for an incremental joint motion $\Delta\theta$(a 6×1 vector) is sent directly to robot. A command for an incremental cartesian motion is specified in hand frame translation vector ΔX (3×1) and orientation matrix ΔM (3×3). PUMA computes the new robot hand position $X_{new} = G(\theta_P) + \Delta X$ and orientation matrix $M_{new} = M_P.\Delta M$, where $G(\theta)$ is the direct kinematic model of slave arm and M_P is the current robot hand orientation matrix. PUMA computes the inverse kinematic for X_{new} and M_{new} and finds the corresponding joint vector $\Delta\theta$ which is sent to robot.

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. 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.

2.1.2. 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. Similar to the PUMA component shown in Figure 1. 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.

2.1.3. Decision Server Component

DecisionServer is a component that provides an autonomous loop on the server to support supervisory telerobotic control. The is a higher abstraction layer which is used as an agent to implement local robot impedance control and workspace scalability functions. This layer can also accommodate the repeatability of a set of movement commands. A block diagram describing the hierarchy of the server system including DecisionServer is shown in Figure 2. The human operator is at the highest level of hierarchy and interacts with the system using a UI(user interface).

2.1.4. 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 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

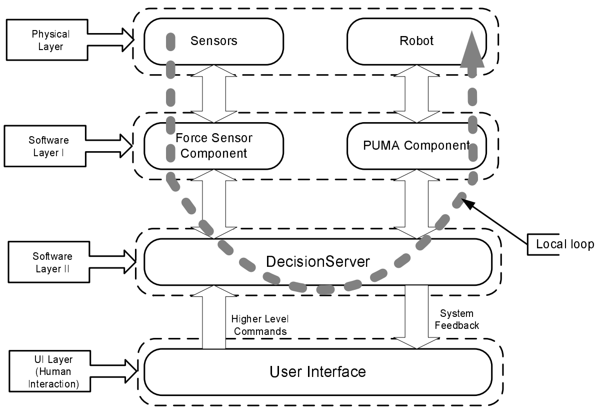

Figure 2. Component Hierarchy on the Server Side

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.

2.2. Client Side 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).

2.2.1. Decision Server 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*.

2.3. 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 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

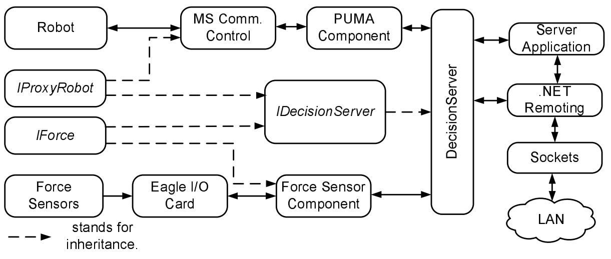

Figure 3. Integrated Scheme - Server Side

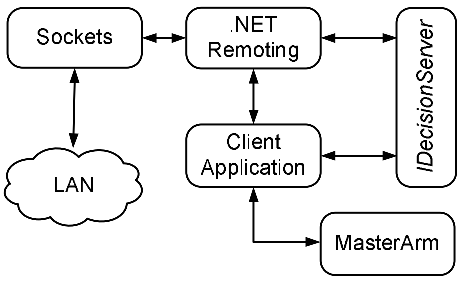

Figure 4. Integrated Scheme - Client Side

DecisionServer events over to the client or *IDecisionServer* interface. Shim classes are thin assemblies visible to both the server and the client. A diagram showing the events being forwarded with the help of shim classes is shown in Figure 5.

2.4. A Multi-threaded Distributed Telerobotic System

The multi-threaded distributed telerobotic system (Fig. 6 and 7) 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.

3. COMPARISON

Ho[8]'s JAVA based fame-grabbing software took 1 second for camera-DRAM transfer with a video rate 0.33 Fps against 25 ms and 12 Fps using DirectX image acquisition.

Yeuk et. al [4] use MS VM (Microsoft Virual Machince) to bridge the gap between JAVA and *DCOM*. However, the proposed *.NET Framework* is used for development of all GUIs and core system components thus freeing us from using any intermediatory services like MS VM within the framework.

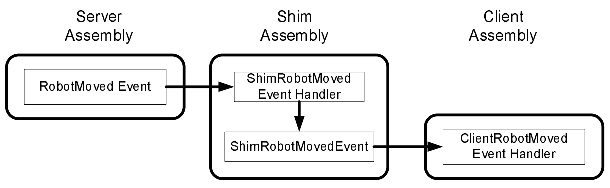

Figure 5. Events forwarding Using Shim Classes

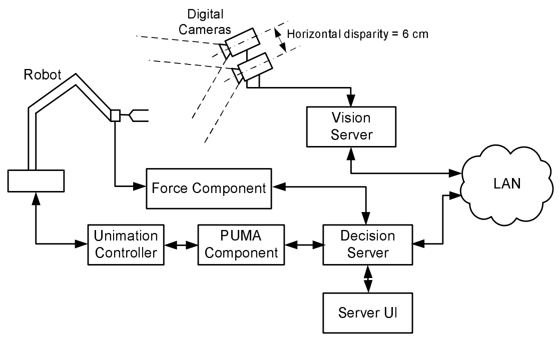

Figure 6. Server side of the distributed framework

In the case of custom protocols like Al-Harthy's[9] VB 6.0 and TCP ActiveX based client-server framework, the TCP read/write operations are very slow because of the many software layers involved such as Application, Custom protocol, TCP ActiveX control, and Windows Sockets etc. While in the proposed setup, 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. See [10] for details on performance.

4. ACKNOWLEDGEMENT

This work is supported by King Abdulaziz City for Science and Technology (KACST) under research project grant AR-20-80. We also acknowledge computing support from KFUPM.

5. CONCLUSION

A reliable real-time connection between master and slave systems is proposed using *.NET Remoting based Distributed Components*. Our approach uses tools that automatically handle the network resources and data transfer while isolating the components from network protocol issues. This liberates us from defining custom protocols for client-server interaction. Also this scheme provides flexible deployment environment in a sense that no pre-registration of components is required on the host machines which is a clear advantage over *DCOM*. In addition to providing a truly multi-threaded environment, the use of *.NET* components on both client and server sides guarantees fastest telerobotic interaction in a closed environment like a LAN.

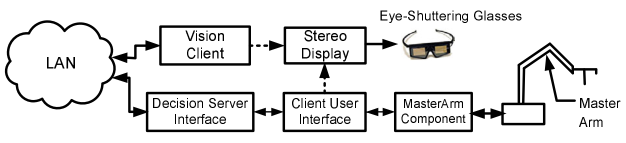

Figure 7. Client side of the distributed framework

6. REFERENCES

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

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

[3] X. Ning; T. J.Tarn. Action synchronization and control of internet based telerobotic systems. *IEEE Inter. Conf. on Robotics and Automation*, 1:219–224, 1999.

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

[5] C. Sorensen. A comparison of distributed object technologies. *Technical Report# DIF8910, The Norwegian University of Science and Technology*.

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

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

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

[9] 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.

[10] M. Al-Mouhamed; O. Toker; A. Iqbal. Performance evaluation of a distributed telerobotic framework. *ICECS'03, Sharjah, United Arab Emirates*, 2003. To be held in December, 2003.