Performance of AMI-CORBA for Field Robot Application

Nanang Syahroni, Jae Weon Choi
Guidance, Navigation, and Control Laboratory
School of Mechanical Engineering and RIMT
Pusan National University, Busan 609-735, Korea
nsyahroni@yahoo.com, choijw@pusan.ac.kr

ABSTRACT

The objective on this project is to develop a cooperative Field Robot (FR), by using a customize Open Control Platform (OCP) as design and development process. An OCP is a CORBA-based solution for networked control system, which facilitates the transitioning of control designs to embedded targets. In order to achieve the cooperation surveillance system, two FRs are distributed by navigation messages (GPS and sensor data) using CORBA event-channel communication, while graphical information from IR night vision camera is distributed using CORBA Asynchronous Method Invocation (AMI). The QoS features of AMI in the network are to provide the additional delivery method for distributing an IR camera images will be evaluate in this experiment. In this paper also presents an empirical performance evaluation from the variable chunk sizes were compared with the number of clients and message latency, some of the measurement data's are summarized in the following paragraph.
In the AMI buffers size measurement, when the chuck sizes were change, the message latency is significantly change according to it frame size. The smaller frame size between 256 bytes to 512 bytes is more efficient for the message size below 2Mbytes, but it average performance in the large of message size a bigger frame size is more efficient. For the several destination, the same experiment using 512 bytes to 2 Mbytes frame with 2 to 5 destinations are presented. For the message size bigger than 2 Mbytes, the AMI are still able to meet requirement for more than 5 clients simultaneously

Keywords: Field Robot, OCP, CORBA, AMI

1. INTRODUCTION

The hierarchical systems for autonomous Field Robot (FR) control systems with real-time communication capability, which are mission control station (MCS) to draws the mission planning, decision control algorithm, and maintain coordination of several type of FR have emerged as a topic of significant interest to controls community. The small FR type that possible implementation fitting in rucksack envelope, provides by navigation system sensors for all weather, daylight and nigh vision, to performs for scout mission to search, detect, recognize the mines or targets, and communication relay[1]. Another type of FR for squad supporting mission with limited speed, mobility and heavy, but carrying large of supplies and equipments for weapon control system.[2]

The basic idea behind most of the current FR navigation system is uses the satellite navigation system to acquire their locations, combining with the environment information's which are collected by their sensors to provide a representation of the surround environment to emerges the situation awareness [3]. In the autonomous ground vehicles that use integrated GPS/INS systems, combining with sensor data to follow prescribed paths and to sense the environment around them[4]. One operates in smooth terrains, and the other is intended for cross-country operation high-accuracy positioning for path guidance in the presence of large structures such as buildings, rocks and trees.[5]

In order for a multiple field robots to travel from one location to another's, they have to aware the position and contiguity among them by exchanges the information using wireless local area network. The distributed dynamic host configuration protocol for nodes in a Mobile Ad-hoc Network (MANET) enables nodes to configure the network parameters of new nodes entering the network. Specifically to addressing the problem of assigning unique IP addresses to MANET nodes in the absence of a DHCP server.[6]

The middleware communication among several FR by exchanges the data or their system performance needs the rapid response times needed in a distributed network to support time critical calculation algorithm for joint missions. A mission base station will use the human level perceptual processing algorithm to acquire accurate and timely information to make time critical decision. Implementing this system, middleware communication requires state of the art real-time networked communication to shape and response to emerging situation rapidly. Remote object invocations to shared communication channels and publisher/subscriber communication to real time channels also were considered in many application.[7]

2. HARDWARE SYSTEM

The significant advances in surveillance or reconnaissance over wide area could be substitute by the improvements of sensor technologies [8]. In the figure 1 is illustrated our FR use an All Terrain Vehicle (ATV), it consist a IR camera manipulator, ultrasonic sensors for avoid a collision, sensor for heat and rotation of engine, steering and break system sensor, and GPS receiver are connected into PC interface terminal.

Fig.1 Field Robot System

The design methodology in this research is use the customize Open Control Platform (OCP) to integrating control technologies and resource, which is using Common Object Request Broker Architecture (CORBA) real-time distributed computer technology to coordinated distribute data, organize the interaction among hierarchically components, and to support dynamic reconfiguration of the components. In this paper we review the AMI CORBA as tools for supporting the graphical data exchange among several FR systems.

During this research, the two simulators of FR systems and one Mission Control System (MCS) are prepared. In the each FR hardware system, the four CIC DC-Geared Motor (Model JC-35L/H-12) installed in each FR with a 12-V nominal voltage was used as actuators in lieu of expensive. The IR Camera mechanical system needs a one motor driver for 360 degrees horizontal rotations, while each steering, engine and braking system needs one motor driver respectively. Each motor is driven by an H-Bridge power amplifier using 6 amperes complementary silicon transistor TIP41 and TIP42, were made by Mospec. It can deliver continuous power of 60W from each transistor within peak voltage of 100V. Motor drivers are drives using PC parallel port line with PWM output which generated by computer program for precise torque control. During initial testing, this motor drivers ware observed to produce high temperature in 12V, which can damage the power transistor as the main electronic components. Therefore, a resistor with higher resistance was used to clip the bias current caused by the optocoupler AN25, and a NAND gate 74LS00 was used to perform as inverting any PWM output from PC parallel port with the standard 5V digital output.

A four motor drivers are connected to the IEEE 1284 standard parallel port use 25 pin female (DB25) connector (to which printer is connected). On almost all the PCs only one parallel port, but we can inserting additional ISA/PCI parallel port cards. The Status, Data, and Control lines are connected to there corresponding registers inside the computer. As a typical PC, the base address of LPT1 is 0x378 (0x278 for LPT2). The Data register resides at this base address (0x378), Status register at base address + 1 = 0x279 and the Control register is at base address + 2 = 0x27a.

The CPU runs the software that handles all hardware controls, TAO CORBA-based communication, and the filters for the IR camera signals. A PC-type Intel Pentium III-733MHz from Samsung (model M2761) was used as the FR central processing unit. It has 128MB of main memory, a 20GB hard disk, two serial ports, and one parallel port.

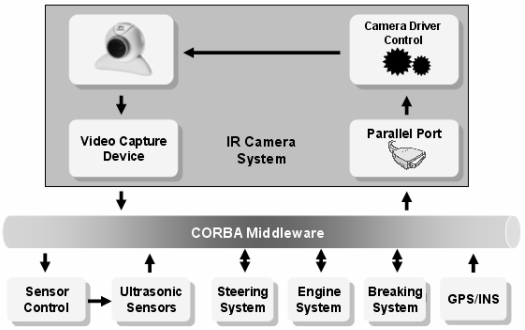

Fig.2 FR System Middleware Interconnection

The graphical information come from video device using the components that were used in the design and testing of the system were low cost commercial products that are readily available. An IR camera Asung ACM-7212DNC IR Led Dome CCD Camera along with a Video Capture Card was used to capture data for analysis via an USB interfacing to PC. This allowed the capture of PAL standard data, formatted for 640x480 captures at a frame rate of 25 frames per second.

The FR communication modeling and simulation are developing and ongoing using wire LAN, complement to MANET. Simulation models of this FR are used, partly to ensure the performance throughout the development phase, in order to obtain the results that fully represent reality the models are constantly validated against the performed tests.

3. SOFTWARE SYSTEM

Two software applications were developed separately for both the field robot PC_{FR} computers and the mission control system PC_{MCS} of the overall system experiment, the PC_{FR} runs the application for the decision control system and trajectories algorithm, while the PC_{MCS} run the mission planning algorithm and performance analyze.

The software applications are designing by customizes Open Control Platform (OCP) middleware structure as

shown in figure bellow. The OCP has its heritage in the CORBA-based and designed to support all levels of control development for Unmanned Aerial Vehicle (UAV) in Boeing. The major components of the OCP software include run-time framework and middleware, simulation environment, and tool integration. One goal of OCP is to bring middleware enabled software development to universal vehicle management-type processing.

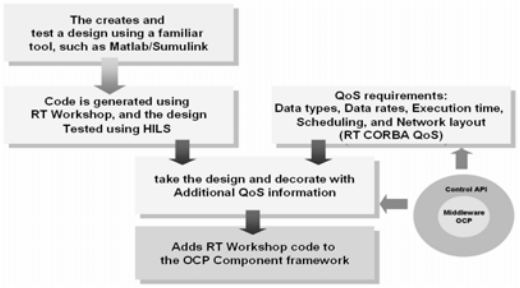

Fig. 3 OCP Development Process

The OCP software infrastructure development as shows in the figure 3, for the PC_{FR} is focused on trajectories, control algorithms, and reconfiguration system, then PC_{MCS} latter is used to send start/stop commands for health monitoring and mission strategies, for post experiment is for data analysis and plotting. A remote PC_{FR1}, PC_{FR2}, and base station PC_{MCS} computer communicate via LAN with standard Ethernet 10Mb of bandwidth.

4. MIDDLEWARE STRUCTURE

Middleware is software that provides a substrate through which software components can communicate with each other. It sits between the operating system and the application software and transparently handles low-level details generally required for data transfer between applications and over the network. Middleware can be used when the application software is located on the local processor or on a distributed computing system linking many processors across a network. A CORBA is a middleware software standard developed by a consortium called the Object Management Group (OMG). A basic feature of CORBA is the object request broker (ORB), which handles remote method calls.[9)]

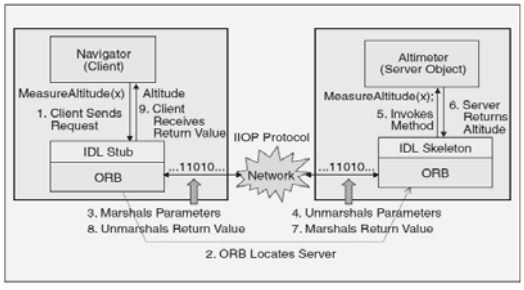

Fig.4 CORBA Distributed Communication.

As depicted in the figure 4, when an object calls a method of another object distributed elsewhere on the network, the ORB intercepts the call and directs it. The client object does not need to know the location of the remote server object, a principle known as location transparency that greatly simplifies the programming of distributed applications. The way this works is that a programmer specifies an interface for each object using a standard interface definition language (IDL). These interfaces are compiled into what are referred to as client IDL stubs, which act as proxies for the actual objects, and server object IDL skeletons. All components are registered with and interact through the ORB. When a client object invokes a method it does so as if it is making a method call to a local object, but it is actually invoking it on a proxy that is the IDL stub. The method call goes through the ORB, which locates the server object. If the server object is remote, the ORB needs to send the request to the remote object's ORB over the network. This involves marshalling the parameters, which means translating data values from their local representations to a common network protocol format, such as the Internet Inter-ORB Protocol (IIOP) standard defined by OMG.[10)]

On the server object side, the parameters are de-marshaled and the method invocation is passed to the IDL skeleton, which invokes the method on the actual server object. The server returns the requested value in a similar fashion through the ORB. Note that the client does not have to be aware of where the server object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface.

The OCP provides an open, middleware-enabled software framework and development platform for controls technology. The customize OCP middleware applications are written in C++. It includes a real-time CORBA component which leverages the ACE and TA0 products developed by the Distributed Object Computing (DOC) research team at Washington University.[11)] TA0 provides some real-time performance extensions to CORBA. The Asynchronous Method Invocation (AMI) is one of the invocation methods in the Common Object Request Broker Architecture (CORBA) using Internet Inter-ORB Protocol (IIOP) to perform data objects exchange in the client/server application hierarchy [12)]. The AMI client application uses this service by sending the command to download the preferred graphical information for further processing. The AMI server response by sending several chunks of a message by iteration transfer simultaneously. The number of chunk is generated automatically according the amount in conjunction with AMI without requiring multiple threads. The basic design of the AMI programs is to allow both the client and the server is to other tasks without having to wait for a given task to complete.

In this experiment, the CORBA Naming Service will use to bind and resolve and object reference dynamically, rather than using an Interoperable Object Reference (IOR) static file. Operations in the Interface Definition Language (IDL) will use exceptions to propagate problems back to

the clients. In this chapter illustrates how to implement a simple client and server using CORBA, IIOP, and AMI. Client applications can use this service to download and display files from a CORBA server on the network[13].

```
module Graphic_Server
{
   typedef sequence<octet> Chunk_Type;
   interface Picture_Iterator
   {
      boolean next_chunk (in unsigned long offset,
                          out Chunk_Type chunk);
      void destroy ();
   };
   exception Error_Result {
             short status;
  };
   struct Data_Type
   {
      string modification_date;
      string content_type;
   };
   interface Iterator_Generator
   {
      void get_iterator (in string pathname,
                         out Response_Iterator contents,
                         out Data_Type metadata)
         raises (Error_Result);
   };
};
```

Fig.5 IDL for AMI experiment

The IDL structure for the experiment is shown in the figure 5. The client first activates its callback object, and then asynchronously registers a reference to its callback object with the server's iterate generator. The iterate generator then creates an AMI reply callback handler for the requested file that asynchronously sends chunks of data to the client's callback object. After creating and running the callback handler, the iterate generator returns the metadata containing the content type and modification date of the file to client.

Since the callback was registered the iterate generator using AMI, an AMI reply handler called iterate handler on the client side will receive and handle the metadata returned from the iterate generator. The iterate handler then passes the received metadata to the callback object. The callback will spawn an external viewer once both the metadata and the entire file content have been received. The callback object is designed to correctly handle the case where the content is received before the metadata, and vice versa. The core functionality as depicted in the figure 6 below:

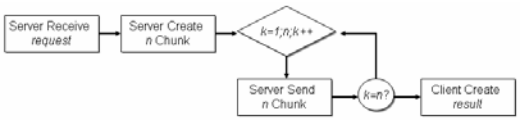

Fig.6 AMI Chunk Iterator

Client applications will use an iterator in conjunction with AMI to download and display files from multiple CORBA server one chunk at a time simultaneously, without requiring multiple threads. This design will help improve the memory management overhead on the client and server. In the server side it reads the name of the pathname the data want to download. It then initializes the client-side ORB and uses resolve initial reference to obtain a reference to a naming service. This object reference is then downcast via narrow function to an object reference for a naming context interface, which is then used to resolve the object reference that the server bound earlier. After narrowing this to the Server interface, the get iterate operation is called via the object reference to obtain the chunk iterator, which is used to download the file. The client invokes the send next chunk method on the iterator, passing in the offset and the object reference to the client's reply handler. To relax this constraint would require some type of offset parameter to the next chunk callback to perform reassembly if chunks for the same file arrived out of order.

When next chunk returns a chunk of the file, the contents are written into a temporary file created in on the local host. Then, an external viewer is spawned to display the file. The type of viewer to spawn is determined by examining the content type metadata returned by the server. The call back functionality of this program is as depicted in figure 7 below:

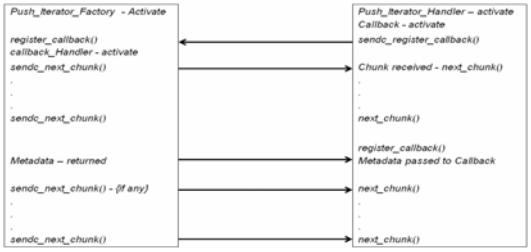

Fig.7 AMI Call Back Functionality

The figure 8 below is shows the monitor console of the AMI server in this experiment. The 4 different clients download the information from AMI server in standard chunk size of 512 bytes and every chunks have its a sequence number.

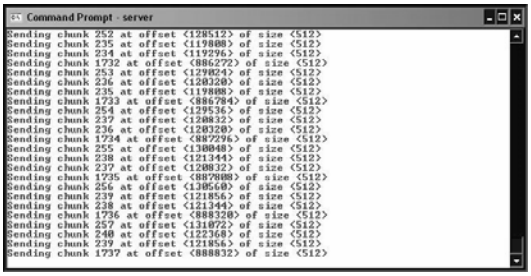

Fig.8 CORBA AMI Console

5. EXPERIMENTAL RESULT

In conducting the experiments began with a single FR machine measurement of each of its system performance, sensors properties, and than moved to multiple FR machines connected with a CORBA network. With these experiments we have a various results, methods, and performances which are obtained on different classification. However only the most interesting subsets which are related to AMI are reported as shown in following figures, most of them are documented for the

message latency and throughput of the network that effected by data sizes and number of clients.

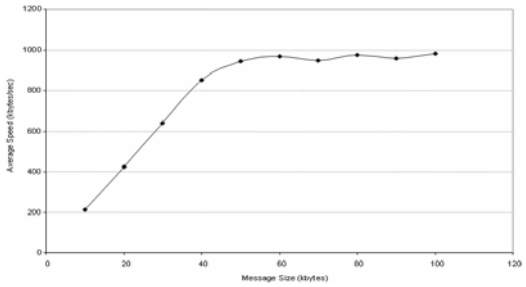

Fig.9 AMI Average Speed vs Message Size

The first result shown in figure 9 is a comparison of the performance or speed of standard 10Mb Ethernet network. On average, the time needed to send a message of the specified size through the CORBA using Ethernet transaction took approximately 1000 kb/sec for the message above 50kbytes. The 512 bytes buffer size was chosen for system compatibility reason; however the message latency in small size of message is not efficient.

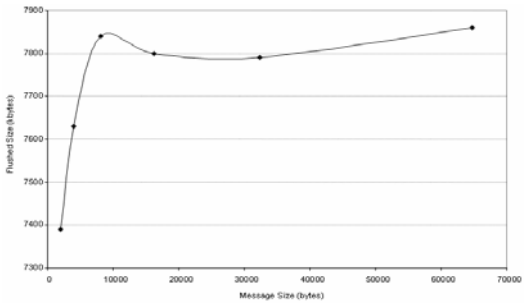

Fig.10 AMI Maximum Throughput vs Message Size

In the figure 10, the TCP throughput of the CORBA for the standard Ethernet network is nearly achieved their maximum theoretical throughput at 8 Mb/sec, or 80% of theoretical maximum speed of Ethernet. Because the CPU of the FR machine has enough speed to process any data transfer, and buffer size is also small size.

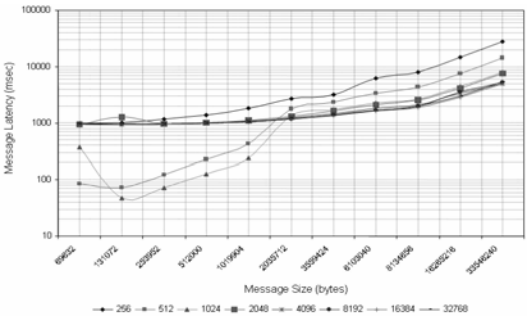

Fig.11 AMI Message Latency vs Variable Chunk Sizes

In the figure 11, since the AMI buffers size were change, the message latency is significantly change according to it frame size. The smaller frame size between 256 bytes to 512 bytes is more efficient for the message size below 2Mbytes, but it average performance in the large of message size a bigger frame size is more efficient.

For the several destination, the same experiment using 512 bytes frame with 2 to 5 destinations are shown in figure 12. For the message size bigger than 2 Mbytes, the AMI are still able to meet requirement for more than 5 clients simultaneously, but the major revelation had not been discovered in this experiment, especially to uncover the effective programming parameters and number of clients that will be affected to the overall AMI performance significantly. This research program is an ongoing project in GN&C Laboratory while AMI is consider to send the large graphical information and CORBA event channel is consider distributing the navigation information over the network.

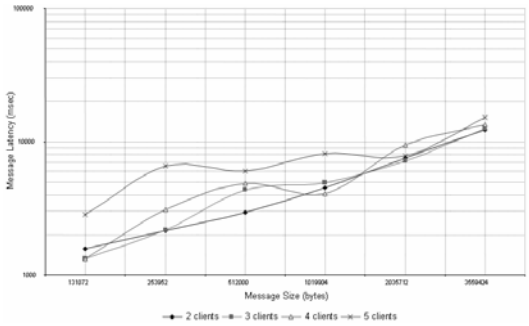

Fig.12 AMI Message Latency vs Number of Clients

6. CONCLUSIONS AND FUTURE WORKS

A number of additional experiments and measurements could be conducted to further research in this area. First, the QoS features of the wireless network could be used to help providing additional assurances for timely delivery for distributing the navigation messages, scanned images, and real-time video information with variable chunk sizes. Second, by increasing the number of the destination than the network is heavily loaded could be performed to determine how the data routing distribution performs in the face of varying network congestion. And also AMI can help improving the scalability of CORBA applications by tuning the chunk size and minimizing the number of client threads required to perform two-way invocations.

In previous section, we present empirical results that show how AMI implementation helps to increase application scalability to distribute the large graphical information, and we demonstrate the efficiency of the AMI implementation distinctly, by comparing message latency and throughput of AMI. However the major revelation had not been discovered in these experiment steps, especially to uncover the effective CORBA programming parameters

according to the network parameters that will be affected to the overall AMI performance significantly.

ACKNOWLEDGEMENT

This work was supported by Grant No.(R012003000104300 2004) from the Basic Research Program of the Korean Science & Engineering Foundation.

REFERENCES

[1]. Cang Ye, and Borenstein, J., A method for mobile robot navigation on rough terrain, Proceeding ICRA '04. Vol. 4 , April 26-May 1, 2004, p:3863- 3869.

[2]. Fregene, K., Madhavan, R., and Kennedy, D., Coordinated control of multiple terrain mapping UGVs, Preceeding ICRA '04, Vol. 4, p:4210 – 4215.

[3]. Anthony Stentz and Martial Hebert, A Complete Navigation System for Goal Acquisition in Unknown Environments, Autonomous Robots, Volume 2, Number 2, August 1995.

[4]. S. A. Roth and S. Singh, Application of robust, high-accuracy positioning for autonomous ground vehicles, AUVSI Unmanned Systems North America 2004, August, 2004.

[5]. Golda, D.,a Iagnenima, K., and Dubowsky, S., Probabilistic modeling and analysis of high-speed rough-terrain mobile robots, Proceeding ICRA '04. Volume: 1 , April 26-May 1, 2004, Pages:914 – 919.

[6]. Sanket Nesargi, Ravi Prakash, MANET: Configuration of Hosts in a Mobile Ad Hoc Network, Proceedings of INFOCOM 2002.

[7]. M. Mock and E. Nett , On the coordination of Autonomous Systems, 5th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, Monterey, 1999.

[8]. Insop S., Karray F., Guedea, F., A Distributed real-time system framework design for multi-robot cooperative systems using real-time Corba**,** IEEE International Symposium on Intelligent Control. 2003, Pages:793 – 798.

[9]. T. Samad and G. Balas, Software-Enabled Control: Information Technology for Dynamical Systems. John Wiley & Sons/IEEE Press, 2003.

[10]. Object Management Group, CORBA Messaging Specification, OMG Document 98-05-05, May 1998.

[11]. D. C. Schmidt, and S. Vinovski, Programming Asynchronous Method Invocations with CORBA Messaging, C++ Report, SIGS, Vol. 11, Feb. 1999.

[12]. Mayur Deshpande, Douglas C. Schmidt, Carlos O'Ryan, and Darrell Brunsch, The Design and Performance of Asynchronous Method Handling for CORBA, Proceedings of the Distributed Objects and Applications (DOA) conference, Irvine, Nov. 2002.

[13]. Alexander B. Arulanthu, Carlos O'Ryan, Douglas C. Schmidt, Michael Kircher, and Jeff Parsons, The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging, Proceedings of the IFIP/ACM Middleware 2000 Conference, New York, April 3-7, 2000.