Distributed Agents For Remote Service Reconstruction In Field-bus and IP networks

Amit Walinjkar, Nick Whittaker

Department of Computing and Mathematics, Manchester Metropolitan University
Chester Street, Manchester UK
amit.w.vjti@gmail.com
N.Whittaker@mmu.ac.uk

Abstract- Computing IP networks use the TCP/IP protocol extensively for connection-oriented service access, and UDP/IP is used for best-effort connectionless data transfer. Commonly used RPC mechanisms like Java RMI, IIOIP and CORBA use TCP/IP which has connection set-up and management overheads. For systems with high speed and high frequency RPC (Remote Procedure Calls), connection management overheads can affect the system performance and overload the server. Such is the case with a number of measurement and controls engineering applications where the field devices communicate continuously with the field bus server, at a high frequency. The control software for such systems have to consider the real time performance issues, load balancing and fault tolerance. Moreover, the plug-n-play field-bus Human Machine Interfaces (HMI) require reconfiguration at every node they are installed and therefore automated re-configuration and re-construction could save time and effort for measuring and monitoring processes. To reduce the communication overheads caused due to connection management, RPC over UDP/IP can be used as an alternative to RPC over TCP/IP, for high frequency RPC calls. In addition, the server load can be sufficiently reduced by transferring the remote service parameters to the client end (for re-construction), instead of executing the entire service at the server end for each client. Reconstructing the remote service at the client end by using these service parameters, aided with RPC implementation over datagrams, can improve the system performance and help perform load balancing and fault tolerance tasks.

Keywords-- Distributed Agents, Field-bus, RPC, HMI, RPC over UDP

I. INTRODUCTION

In this paper we present a distributed agents model for dynamically reconstructible HMI for field-bus and IP networks. The aim was to develop a distributed agent model using RPC over datagrams for remote service transfer, reconstruction and load balancing. A suitable controls engineering application was required for this purpose, one which involved a remote service that is accessed at high frequency, has relatively longer duration of execution and is relatively compute intensive. Having considered several applications, from web services running in commercial banking environment to real time compute intensive application in aerospace systems, more general Foundation Field-bus model in controls engineering was chosen. The service exposed by the server is a diagnostic service that monitors the performance of a HVAC system and simulates its operation on a graphical user interface (GUI). The diagnostic service would take readings from a set of nine pressure sensors and the control unit would actuate a set of nine pneumatic valves all commissioned over a field bus [1] (section 3.12.3 pg.96) network. Such a system normally consists of diagnostic software called the ‘Human Machine Interface’ (HMI), installed on the machine that controls and monitors the field bus segment it is connected to. This software is also used to program the Programmable Logic Controller (PLC) that controls the field devices (sensors/actuators). A Profibus or Modbus segment can contain a maximum of 32 devices including the diagnostic monitoring machine and PLC (as shown in Fig 1).

Fig. 1 Example of a Field-bus segment with HMI, PLC and field devices

It is the diagnostic and monitoring service which has to be reconstructed and re-configured on a remote client machine using agent models. Multiple remote client machines may request the service from remote locations as, field-bus networks are deployed in large industrial or commercial complexes. The service component would accept the readings from the pressure sensors after a time interval of 1 second (an assumption), and would calculate the average fluid pressure flowing through the connecting pipes in an HVAC system. The readings for fluid pressure are taken from flow meters [6] (Flow meters, pg. 207) and the actuators control the valves [6], (pg 233,234) depending on the pressure in the pipes.

Once this controls engineering application was taken as the system under consideration the effectiveness of RPC over datagrams approach was developed and tested. The communication model was finalized and the interacting client and remote entities were modelled as agent software components which facilitated the information exchange seamlessly and transparently.
II. LITERATURE REVIEW

A system of distributed agents is considered here to illustrate agent operation concepts and agent management techniques. Agents can be visualized as software components distributed over a private network or the Internet that are presented with a search template to search information they need. This information is hosted at the service agent end for subsequent queries and information retrieval using agent management schemes and RPC communication models. When the updates are available at the service node, the remote agents are notified of the changes, so that appropriate actions can be taken. The information updates can be periodic or aperiodic in nature, so various techniques to sample this information will need to be considered. Agents have sensing elements that receive inputs from the environment that hosts them and they act or respond to these events. Yet another study of agents says agent can have beliefs, intention, goal and actions and they react to the events and send messages [1]. Goals are initiatives or the tasks the agents need to perform autonomously with the logic built within them. Beliefs are the knowledge parameters and variables in the environment that hosts the agent. Intention is a plan that prescribes a set of actions. Actions are responses that agent generate on a set of events. Multiple agents hosted in a distributed environment need to communicate amongst themselves by passing messages. These messages may have a format and a set of rules (protocol) for inter-agent communication, discussed later in the sub-topic inter-agent protocol. [1]

Agents can be classified according to its mobility as static or mobile. The mobile agents are able to commute and relocate to some destination on the network. E.g. Aglets, IBM Aglet workbench is an environment that allows aglets to relocate on the network in a secured framework. The workstations where the mobile agents would visit need to have a mobility server (Tahiti) installed and security permissions enabled for that particular workstation. Yet another classification could be: collaborative agents, collaborative learning agents, interface agents, smart agents. [5].

Collaborative agents are agents that communicate with each other in a distributed system and hence not much different from distributed agents that communicate using an inter-agent protocol. Learning agents and smart agents derive their functionality from artificial intelligent systems where agents are managed as a framework of artificial neural network and the agents in the network learn from various training and learning methods, viz. supervised and unsupervised. In Supervised learning, a network is given an input set along with its desired output set. For each set of input, the network compares its own actual output and the desired output. The network then tries to reduce the difference by adjusting parameters in the neurons (agents in this case). This process is repeated until the difference in desired and actual output is sufficiently reduced. On the other hand, a network in unsupervised learning is given only an input set, and depending on the performance criteria and parameters, the network is allowed to train and self organize itself. The network is monitored to find out, whether it is performing sufficiently well according to the performance parameters. [7]. Agents in a neural network can have a feed-forward (connection between neurons don’t form a loop) or a multi-layer perceptron model (multiple layers or neurons connected in feed-forward fashion); and forward pass or back propagation algorithms can be used to train the agents. Having trained with the input set, the agents can then respond to events depending on the logic they are trained with. However, neural networks aim at minimizing the difference between the actual output and the desired (approximately known) output of the system; and agent software is aimed at exploring (unknown until accessed) the network, so it is difficult to proclaim the effectiveness of some neural network algorithms in implementing agent based systems.

An alternative approach to manage a multi-agent system could be grid computing. The term Grid was coined in the late 90s to describe a set of resources distributed over wide-area networks that can support large-scale distributed application. These resources can be loosely coupled computing elements which can be managed as agents and perform collaboratively. [8].

In order to manage agents in a grid, a workflow plan needs to be created which consists of workflow design, workflow scheduling, fault tolerance and data movement. Workflow can be depicted using directed acyclic graphs showing flow of control between tasks among agents. Workflow of tasks can be sequential, parallel or by choice (conditions on a task need to be satisfied before a task executes). An agent management system (AMS) can be used for workflow scheduling of tasks on agents remotely. Scheduling can be centralized where the AMS controls the task scheduling on the agent, or it can be decentralized where agents are independent to schedule tasks on them. Scheduling can be managed hierarchically where some agents can control the scheduling of tasks on other agents according to some scheme. Initiation of the scheduling process can be static or dynamic. Static initiation of schedules can be user-directed where users initiate tasks depending on their own knowledge and information, or it can be simulation based, where, tasks are run in a simulated environment and the best performing task is initiated. Dynamic initiation is prediction based, where the AMS or the agents decide which task would be the best to schedule depending on the dynamic information and results available with the agent. A just-in-time scheme can also be used to initiate scheduling where; a decision to schedule tasks is made depending on the situation and information (conditions, parameters, thresholds) at that time. Tasks executing on the agent in a grid environment need to be fault tolerant so they can handle faulting situations in the system network.

III. PROPOSED SYSTEM:

The proposed system consists of distributed agent components communicating in a federated publish-subscribe mode using an intelligent inter-agent protocol.
Distributed agents have been modelled as client, server and discovery agents.

A. Problem statement.

In a typical RPC mechanism the time taken to obtain the results from the remote server would depend on two factors:

1) The time taken for the request/response packets to travel to and from the client and the server.
2) The time the remote procedure takes to execute on the server.

The first parameter is non-deterministic and would depend on the network traffic and the second parameter would depend on the system configuration (processor speed, system memory, background processes running etc.) of the host machine on which the remote procedure executes. So, if the remote procedure takes ‘n’ units of time on the server and the time taken for the message packets to travel to and from client and server machines be ‘t1’ and ‘t2’ respectively, then the total time to obtain the results from a RPC would be at least \( t1 + t2 + n \) units and this value is non-deterministic in nature. In addition there is an added overhead on the client-server hosts for the connection management which involves the SYN/ACK packet flow and socket options (socket linger/ keep-alive options), assuming the communication takes place on TCP/IP sockets.

For applications that take longer duration to execute on the server to process the request with a relatively high number of client connections, the server may face load balancing problems. If the RPC call is a complex task that involves high number of floating point operations or compute intensive input output operations, then with a high number of client connections the server performance may deteriorate. With a very high number of client connections, each requesting such a complex service, the server may even crash or refuse serving clients any longer. In cases where it is not cost effective to have such systems deployed, the complexity will have to be handled at the software design and component implementation level. As an alternate solution to replicate the compute intensive remote service on each client, the service object is transferred to the client machine, and the server responds to the client requests with only the 'relevant information', that the service object might require for its execution, thus saving the server of some processing time. The relevant information transferred can be maps, schematics or meta-data that can be used to construct the service at each client end. The solution can be implemented using a scheme similar to MANTA system using Panda [3] (section 2.2 and section 3) for RPC over datagrams. However, instead of performing serialization of objects at server end and then de-serialization at the client end, only byte streams or relevant meta-data information is passed to the client, which then reconstructs the service using this meta-data information.

According to Amdahl’s Law of scalability for fixed-load (as the there are fixed set of instructions to read data from sensors).

Referring Fig. 2
\[
S_{n1} = \frac{T}{T_h + h_n} \\
S_{n2} = \frac{S_{n1}}{n} = \frac{1}{n(T_h + h_n)}
\]

B. Models and Prototypes:

The operation of continuously accepting the readings from the PLC (time consuming I/O operation) every second for all the devices connected to the PLC and then calculating the average over a period of time can be considered as a compute intensive task, especially if several clients request the service at the same time. The service is run on the remote clients and only the 'schematics' of the field-bus network and the 'readings' are sent to the client machine. The server machine is relieved of the some load of maintaining several client connections each requesting a compute intensive task over
A real time solution was thought of in the beginning, which was to take readings from the sensor devices in individual real time threads and make them available at the remote client end. However, even if the readings were obtained on real time threads, the communication between the agents was to be implemented over a non-deterministic Ethernet (LAN) medium, which may not suit deterministic real time communication. So, the readings were taken at a predefined interval of 1 second and the operation can be simulated at the server end with no hard real time constraints. A simulation of readings on the server component is a practical consideration as the readings are taken at an interval of 1 sec and not in the order of micro or milliseconds. Initially, an experiment was conducted to generate data packets for all the sensors on the field-bus network adhering to Modbus/TCP [4] protocol, a field-bus similar to Profinet. It was found that it takes 400 microseconds to 1000 microseconds or more to form individual packet corresponding to each sensor device. The contents of the packet were as in TABLE 1:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceType</td>
<td>Sensor/Valve</td>
</tr>
<tr>
<td>Protocol</td>
<td>MODBUS</td>
</tr>
<tr>
<td>DeviceAddress</td>
<td>1...18</td>
</tr>
<tr>
<td>FunctionCode</td>
<td>0x 04 // instruction code to take readings from the sensor device</td>
</tr>
<tr>
<td>DataField</td>
<td>55 // reading from the sensor device</td>
</tr>
</tbody>
</table>

A data packet would be accepted by the server component from the diagnostic and monitoring tool on the field-bus network. It is assumed that such a provision is made to the diagnostic HMI software, whereby it would interface with a software component and exchange data packets corresponding to the readings it obtains from the sensor devices. It would convert the Modbus/Profinet frames to a suitable data structure and format mentioned above. The HMI component was simulated and java.util.Random.nextInt() function was used to generate sensor reading. A variation from 400 microseconds to 1000 microseconds was found to create data packets with same structure and format. This is due to the background processes running in the background on a Windows XP, Pentium III machine. The application under consideration is not under hard real time constraints, hence such variations (microseconds) would not interfere with the performance of the server component as the frequency to take sensor readings itself is 1 second, which is much larger than the variations found.

It is to be noted here that the specifics of the field-bus network or the HVAC systems is not of much relevance to the application, as the theory focuses on the transfer of readings obtained from the HMI to the remote clients, and not on how the readings are obtained from the field-bus network.

C. Implementing RPC over Datagrams:

To test the performance of RPC over datagrams as against RPC over TCP, another experiment was conducted by implementing the RPC mechanism over datagram sockets instead of the more common TCP sockets. A remote interface was implemented with just one function getReadings() which executed an empty 'for loop' 1000 times before returning an integer. In the second test the same loop was run 1,000,000 times. A high loop variable was chosen to simulate a relatively longer function call. The RMI (TCP) version of remote implementation took about 110 to 150 ‘microseconds’ to return results back to the RMI client and about 115 to 160 ‘milliseconds’ to execute the loop 1,000,000 times before returning back the results to the client. The RMI over TCP version was compared with the RMI over Datagrams (UDP) version, which took 210 to 250 microseconds to execute the remote loop 1000 times and return back the results to the UDP client. It took about 500 to 550 microseconds to execute the loop 1,000,000 times remotely and return back. The table (Fig 3) summarizes the results from the tests run.
The RMIDatagramClient issues a Lookup() on the RMIDatagramLookup (Fig 4) class which implements the static method Lookup(). This class creates a datagram socket and sends the name of the remote interface and remote procedure in the form of a string. (“IGetReadings.getReadings”) to the RMIDatagramServer, which resolves the request through the RequestResolver class that has registered the GetReadingsImpl object. The remote method is executed and the results are returned to the client over a datagram socket. The RequestResolver class is analogous to the rmi registry in Java RMI implementation. Based on the results obtained from the tests, RPC over UDP approach was chosen to be implemented in situations where a continuous high frequency remote procedure calls were expected. The agent components were modelled using java sockets API as the communication infrastructure.

The proposed system was then conceived to consist of at least the following:

1) A human machine interface service that could be made available remotely.
2) A client agent that manages the communication between the HMI service and the remote entity.
3) One or more remote agents that subscribe to this service via the client agent.
4) The remote GUI that encapsulates and represents the service transferred remotely.

Two events are modelled as service for the proposed application:

1) The event when the sensor reading exceeds threshold limit which actuates the valve.
2) Change in the schematics of the field-bus network. E.g. replacement of the sensor/valve types (analogue sensors to digital) that may take place if a fault develops in a sensor.

These two events are simulated on the HMI and the events are sent to the remote clients subscribing to the HMI service, with minimum delays. The system focuses on publish subscribe events, TCP/UDP communication mechanisms to choose from.

From Fig 5 the agent that accepts the information from the HMI service and transmits it to the subscribers is the Client Service Agent (CSA). Modelled as ClientAgent, this agent accepts readings and the resources (RDF) information from the HMI and sends it to the Remote Agent (RA). The server component makes its services available to the remote clients through a Client Service Agent (CSA) and the remote clients would subscribe to these services through a Remote Agent (RA). To manage such a scheme a discovery service was implemented where the CSA would publish its service through an Agent Discovery Service (ADS) and the RA would subscribe to it.
D. Inter-Agent Information Exchange:

The information exchanged between agents include:

1) **Data Frame** consisting of sensor readings and control parameters sent from the CSA to the RA.
2) **RDF file** containing the field-bus network information. A set of 9 sensors and 9 valves are considered as a case.

1) **Data Frame format:** (Fig 6 and TABLE 2)

<table>
<thead>
<tr>
<th>Fields</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, ….9</td>
<td>Numbers indicating the Device Addresses.</td>
</tr>
<tr>
<td>Val1…Val9</td>
<td>Values indicating sensor readings corresponding to each of the 9 sensor devices.</td>
</tr>
<tr>
<td>THRESHLD</td>
<td>The threshold value for the fluid pressure, which when exceeded raises an alarm and actuates the valve.</td>
</tr>
<tr>
<td>TIMESTMP</td>
<td>The time when the data frame is generated on the service node. This time stamp is sent to the remote client to calculate the transmission time for each data frame sent.</td>
</tr>
<tr>
<td>SCHEMTCS</td>
<td>Scheme adopted for field devices representing the type of sensors and their locations on the field-bus network. Schemes could be one of the two: SCH1 for analogue sensors and SCH2 representing some analogue sensors (those at address location 1, 2, 4, and 6) replaced with digital sensors.</td>
</tr>
</tbody>
</table>

2) **RDF file** is an XML file consisting of <DEVICE>, <Protocol> and <DevAddress> elements. The document also provides the IP address of the machine that hosts the HMI service (Fig 7)

![Fig 5 Distributed agent architecture for implementing AMS](image)

![Fig 6 Inter-agent Data Frame format.](image)

![Fig 7 RDF File format for Field-bus HMI and agents interaction.](image)

E. **Inter-Agent Protocol:**

The data frame (Fig. 6 and TABLE 2) contains the control information which remote agents can use to take appropriate actions. On receiving the data frame, the remote agent instantiates the remote GUI and updates it according to the readings from the data frame. If the sensor reading for a sensor exceeds the THRESHLD value in the control field then the GUI shows the actuated valve at a corresponding location in the schematics.

The inter-agent communication takes place through Java sockets. TCP was compared with the UDP mode of transmission and since UDP is a faster mode of transmission, data transfer from client agent to remote agent takes place over Multicast Datagram Sockets. The readings are generated at a high frequency on the service node and they are to be displayed on the remote GUI with a minimum delay. Using datagram sockets for all the remote clients involves obtaining the subscription registry from the discovery agent and sending datagram packets to all the socket addresses on which the remote agents are ready to receive. However, a more effective mode of transmission is implemented by using ‘multicast sockets’; where the client and the remote agents create Multicast Sockets, at a predefined multicast address, and the remote agents join and leave the multicast group.

IV. **Performance Analysis and results:**

The performance of the agent and service components developed is tested with Netbeans Profiler 5.5 and the network performance is tested using network protocol analyzer Wireshark.
The graphs in Fig 9 indicate that a larger heap memory space is created when the HMI begins execution, which is due to a large number of GUI components being displayed on the GUI with their own AWT threads. Similarly, the threads graph shows an increase in the number of threads running in parallel, again due to the 'Java swing' components in execution with their own AWT event handling queue. In Fig 8 it can be observed that, initially the packets from the field-bus HMI get dropped due to the increase in high frequency calls and due to the AMS the packet traffic is routed to remote clients and remains constant due to load balancing.

V. CONCLUSIONS:

RPC mechanisms using datagrams was found to be an effective mode of data transfer for the services that take longer execution times on the server, are compute intensive, have I/O overheads and are accessed remotely at a high frequency. By passing minimal information required to reconstruct and replicate the service remotely, an effective method of remote service invocation can be implemented for such services. The Datagram version of RPC scales only for remote methods with lengthier execution times as was observed from the comparative graphs obtained from the experiment conducted to compare RMI TCP and RMI UDP.

Agent based modelling was found to be an efficient approach to design intermediate components that facilitate exchange of services based on a set of goals, beliefs and intentions. The service exchange remains transparent to the publishing and the subscribing entities.

The major hurdle to communicating systems with hard real time constraints is the software execution time of communicating entities. The composition and decomposition of data frames as they pass through various OSI layers, increases data transmission overheads. The transition overheads of data packets as they pass through multiple layers can be greatly reduced by combining layer 3 and 4 and layer 1 and 2, so that there are less overheads of adding header and trailer bytes to the transmitting frames at every layer it transitions through. Furthermore, a high performance gain can be achieved if the layers 1 to 4 were implemented at a hardware level on the network interface card. This might give rise to issues of synchronization between the software components that may have to read data frames from the interface card and almost instantaneously convert them to application layer information. Solutions to these issues remain a matter of further research.

VI. REFERENCES:


