Library of Distributed Computing using Object Oriented Language

Phua Ging Sheng , Tay Choo Chuan , Zulkifilie Bin Ibrahim

Abstract - **Currently, there are Java based solutions for real time distributed computing system especially in industrial automation domain. Imagine how beneficial to extend yet adapt Java platform into automation and user can monitor yet control anything at anywhere by only using a Java based hand-held devices. This research has been done on creating a new library of distributed computing using object oriented language.**

Keywords - Object oriented, industrial, distributed, computing.

I. INTRODUCTION

For the industrial automation industry, using Java has several benefits. First, as hardware is updated and newer hardware controllers replace older ones, no change in the binary software is necessarily required [1]. Users will no longer be forced into purchasing one brand of hardware or software. Second, hardware independence allows developers to simulate the run-time environment on a desktop computer, greatly reducing development time. With the simple addition of low-level hardware simulation, the same application class files that run on the target PLC can also execute on a desktop computer for development and debugging [2]. Once the code is completed, the class files can be executed on the target hardware for final testing and quality assurance. This saves valuable time during the development process, since standard Java development environments may be used. Another benefit is that debugging is required only on a single version of source code, once for all platforms. This reduces the project overhead and maintenance by greatly simplifying project build configurations, reducing coding errors and simplifying bug tracking and change requests. Java classes file idea could be generic to cover a wide range of potential problems, or custom-designed to diagnose a specific problem. Combining the ability to patch files on the fly with remote debug and fix, the Java environment provides a powerful and compelling tool for software development, for industrial automation and other industries with similar requirements. Java programming is also a partial solution to the problem of fielding an interactive, platform independent, lightweight, compatible system. The application must let users manipulate information interactively, as is commonly done with desktops. Java applications, once written, can run on any Java-enabled platform, thereby minimizing the time, effort, and energy needed to support the fielded system. Java's scalability and portability make it a prime candidate for industrial automation controlling and monitoring, with these a few advantages which Java extends HMI capabilities to machine (or areas of a plant) where it is cost-effective to install a PC or dedicated display. In advanced situations, user can carry monitoring and diagnosis applications on hand-held, wireless, Java-enabled personal digital assistants. Java simplifies remote monitoring in several ways.

II. GOAL

Library of classes that implement internet protocol were designed to provide interaction over client server paradigm. All libraries were then integrated and interacted over a system which allows access to automation distributed control via hand held device over wireless internet network protocol. Imagine how beneficial to extend yet adapt Java and hand held device into automation where user can monitor yet control anything at anywhere. Through this approach, Java extendibility and adaption in different environment were proven successfully.

III. DESIGN AND IMPLEMENTATION

There are totally three libraries that were designed to meet the requirement for distributed controlling using handheld device which is Front Component, Client Component, and Server Component. The entire three components were actually library software frameworks that have written using object-oriented Java language which can suite in any type of platform. The three component concepts were all designed based on the distributed and internetworking techniques.

A. Front Component

Front Component is an object oriented high level language library program or software that provides job functions and user interface screen which users can interact in order to do something or any request. Front Component can be integrated into handheld device or any mobile devices to interact with Client Component. In a word, a Front Component represents a single screen with a user interface which users can interact using handheld device in order to do something or any request. It is a functional unit of an application, which may be invoked by another Front Component or application. An application may incorporate numbers of Front Component. One Front

Component may be set as the default component which will be directly executed by the user.

B. *Client Component*

Client Component is a library designed to provide robust support of base HTTP protocol for client side. It is compiled and integrated into handheld device or any mobile devices to interact with Front Component. Client Component is a client side HTTP transport library which used to transmit and receive messages over HTTP. It is simply a written library program that handles messaging between a client and server. This library may be used to send and receive streaming data whose length is not known in advance. It will not attempt to execute any other functionality unrelated to the HTTP transport. It provides an efficient and feature-rich package by implementing client side HTTP standards and recommendations.

C. *Server Component*

A Server Component is a server-side library software program, written in high level language that handles messaging between client-server, controlling and giving command among connected automated system devices. It is a server side Java program that runs within web server software on the server side; it is compiled and integrated into Server Machine. Server Component receives and responds to requests from Activity/Client Component, across Hypertext Transfer Protocol. It is a library that used to extend the capabilities of web server that host applications accessed via a request-response programming model. It is simply library software that runs on server that enhances server functionality, to do work of whatever client request and give command to all connected devices. Server Component can respond to client requests by dynamically constructing a response that is sent back to the client and send command to all connected automated system devices.

IV. ARCHITECTURE

A. *Framework*

When comes to real time application, the system can be categorized in two different sections which are client side and server side. Client side is the interaction among the user and any hand held device, while the server side is the interaction among Server machine and automation. The oval or ellipse circle shapes highlighted in blue color are the three main software components that were created in this research. These there software components are Front Component, Client Component, and Server Component which playing important role in order to make sure the system is artificially intelligent enough to be controlled wirelessly anytime anywhere regardless any type of hand held devices. The green color block in rectangle shape represent hardware, the orange color block in rounded rectangle shape represent software application while the blue color block in oval shape represent the software component that were created based on the framework in this thesis and research. The automation side can be connected to server machine in any communication type.

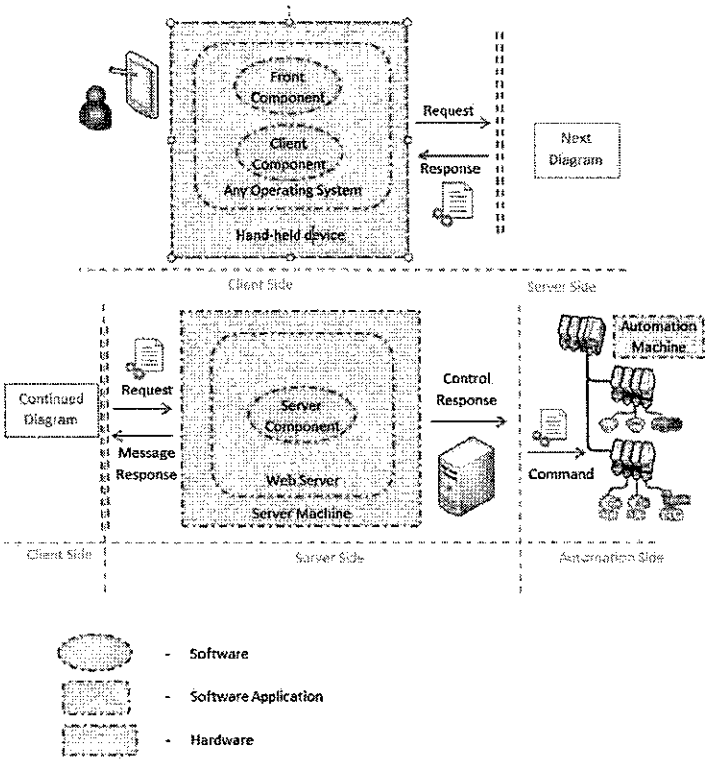

Fig. 1 System Architecture

B. *Paradigm*

A User sends command or control automated machine through Front Component which embedded in the handheld device. Both Front Component and Client Component which integrated in handheld device's Operating System will interact within each other to send out request to a Server Machine. Client Component is needed to translate the message passed by the Front Component into network signal in order to be read by Web Server. The Client Component will open a TCP/IP connection to Server machine in order for sending request. Communication between server and automation then established to send out request which then receive by the Server Machine. After connection established, The Server

Machine with Web Server installed are ready to receive any request from the Client Component. When Web Server gets request from Client Component, it hands the request signal to the Translator before it reach Server Component. When the request hands over by Web Server, the Translator then maps the request to the Server Component. The Server Component processes the request and produces two responses which is message response and control response. Message response is the network signal that sends back to the Web server to notify the Client Component while the control response is the command to control the automation devices. The result of the message response received by the client in message form while the control response received by the machine or automation in machine code binary form.

C. *Multi Request*

When there are simultaneous requests from many Client Components to a Server Component, the Server Component handles each Client Component request by creating a new thread for each request respectively. The Server Component runs multiple threads to process multiple requests to a single Server Component. There is a single thread per request, the Server Component doesn't care who makes the request, and every incoming request means a new thread. Every Client Component request generates a new pair of request and response objects.

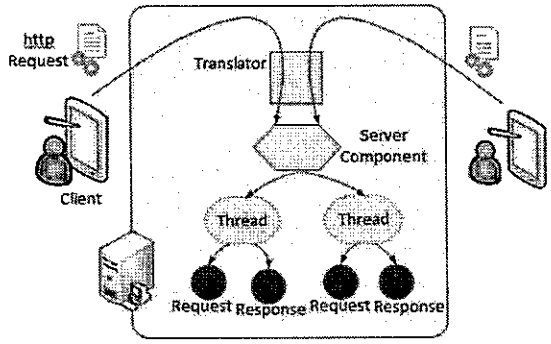

Fig. 2 Multithreading

V. PERFORMANCE TUNING

Performance tuning on these Front Component, Client Component and Server Component requires a slightly different mindset than performance tuning normal Java coding and applications. The reason is that the JVM running those components software is expected to simultaneously handle dozens and hundreds of threads, each executing a Server Component. These coexisting Server Component have to share the resources of the JVM in a way that normal applications do not. The traditional performance tuning tricks still apply, of course, but they have a different impact when used in a heavily multithreaded system. Table below are a few methods that being used and may aid in tuning the software code.

Table I: Performance Tuning

Object Reuse	Avoid the unnecessary creation of objects and this has always been good advice because creating unnecessary objects wastes memory and wastes a fair amount of time. With Server Component, it's even better advice. Reuse the same object as much as possible, while unused object is garbage collected.
Limit Synchronization	Synchronize whenever necessary, but no more. Every synchronized block in a code slows the code's response time. Because the same object instance may handle multiple concurrent requests, it must, of course, take care to protect its class and instance variables with synchronized blocks. All the time one request thread is in an object's synchronized block, however, no other thread can enter the block. Therefore, it's generally best to keep these blocks as small as possible. Also consider using the single thread model tag interface, where the server manages a pool of Server Component instances to guarantee each instance is used at most by one thread at a time. Server Component that implements this doesn't need to synchronize access to their instance variables.
Buffer Your Input and Output	Buffer input and output, all storage files, any streams loaded from a database, and so on. This almost always improves performance, but the improvement can be especially profound with these software components. The reason is that reading and writing one unit at a time can slow down the entire server due to the frequent context switches that have to be made.

VI. DESIGN PRINCIPLE

All of the programming patterns rely heavily on common software design principles. In the table below, several terms for these design principles will be used throughout the development of the library.

Table II: Design Principle

Interfaces	An interface is a kind of a contract between two objects. When a class implements an interface, it's meaning that its object can interact with the implemented class. Another huge benefit of interfaces is polymorphism. Many classes can implement the same interface. The calling object doesn't care who it's talking to as long as the contract is upheld. For example, the Web Container can use any component that implements the Server Component interface.
Cohesion	When the capabilities of software components were specialized, they get easier to create, maintain, and reuse. Cohesion means the degree to which a class is designed for one, cohesive, task or purpose.
Hide Complexity	Hiding complexity often goes hand in hand with cohesion. For instance if a system needs to communicate with a lookup service, it's best to hide the complexity of that operation in a single component, and allow all the other components that need access to the lookup service to use that specialized component. This approach simplifies all of the system components that are involved.
Loose Coupling	Object oriented systems involve objects talking to each other. By coding to interfaces, the number of things that one class needs to know about another class to communicate with it can be reduced. The less two classes know about each other, the more loosely coupled they are to each other. A very common approach when class A wants to use methods in class B is to create an interface between the two. Once class B implements this interface, class A can use class B via the interface. This is useful, because later on an updated class B or even an entirely different class could be used, as long as it upholds the contract of the interface.
Remote Proxy	When a web site grows, the answer is to lash together more servers, as opposed to upgrading a single, huge, monolithic server. The outcome is that Java objects on different machines, in their own separate heaps, have to communicate with each other. Leveraging the power of interfaces, a remote proxy is an object local to the client object that pretends to be a remote object. The client object communicates with the proxy, and the proxy handles all the networking complexities of communicating with the actual service object. As far as the client object is concerned, it's talking to a local object.

VII. Conclusion

This research has produced several prototype systems to prove that this research and technology can represent a fundamentally new, simpler and more effective way of controlling peripherals and automation system in real time. Using handheld device in different environments requires a truly secure domain within different security settings and properties. Cryptography can indeed be implemented with efficient on a mobile device and handheld device [3]. This path is a low-cost, high efficient yet more research on this path needed in future. Further development will lead to handheld device related based solution for the Enterprise Industrial Automation, Control System, as well as other application domain.

References

[1] IEEE Xplore, Digital Library, http://ieeexplore.ieee.org. April 10, 2011.

[2] Peng Zhang, 2010. *Advanced Industrial Control Technology*, Amsterdam : Elsevier.

[3] William S. Levine, 2010. *The Control Systems Handbook : Control System Advanced Methods*, 2nd Edition, College Park : CRC Press.