

Java based implementation of Communication Protocol
Stacks for Utility Industry

A White Paper

KLK/WP001

Doc id – VSS/C&A/UTL/Docs/JavaStackWP.doc

Version – 1.0



Author:
Vinoos Warrier
Vinoos@kalkitech.com

Table of Contents

Table of Contents	2
1 Abstract	3
2 Introduction	3
3 IEC/DNP Protocol Overview	3
3.1 Layering in IEC/DNP	3
3.1.1 Physical Layer	4
3.1.2 Link Layer	4
3.1.3 DNP Pseudo-Transport Layer	4
3.1.4 Application Layer	4
4 IEC/DNP/MODBUS Protocol Implementation Model	5
4.1 Real-time constraints	5
4.1.1 State Transitions	5
4.1.1 Protocol	5
4.1.2 Real Time requirements	5
5 Java and real-time applications	6
5.1 Generic Advantages of using Java for development of real-time applications	6
5.2 Contention points for using Desktop java to run Real Time applications	7
6 Java implementation for IEC	7
6.1 PC Master	7
6.1.1 The functionality of the master software is as below	7
6.1.2 Maintaining configuration information of each slave (databasing function)	8
6.1.3 GUI display and updation	8
6.1.4 Report generation in specified formats	8
6.2 Embedded Slave	8
7 A case study for IEC delay/lag performance on a real-time system	8
8 High-Level application framework	9
9 Advantages and Dis-advantages	10
10 Conclusion	10

1 Abstract

Utility protocol master implementations like IEC and DNP managing numerous networks and network lines require scalable and stable architectures and platforms. This paper looks at the alternatives of implementing such a system using the Java technology and tries to evaluate the advantages and disadvantages of the same. The paper essentially looks at IEC and DNP since MODBUS is a much discussed and known protocol. The paper approaches this issue by addressing the overall protocol framework and its architecture and its real-time constraints both at the master end and at the slave end, the real-time constraints of Java, and its advantages and disadvantages in building a scalable and reliable master system to manage the requirements and briefly provides an overview of a sample implementation scenario whereby some of the issues may be addressed in design itself. The paper concludes by looking at the advantages and disadvantages of the current proposal and the alternatives.

2 Introduction

Utility protocols like IEC/DNP and industrial communication protocols like MODBUS are widely used protocols, and much widely supported by different vendors. These master slave protocols mainly work with a single master supporting N number of channels or network lines, with a number of nodes in each network line. Typical implementation scenarios in the utilities and industrial scheme's involve master applications communicating to a host of devices or nodes over varied communication media and updating the monitor and control data. Normally these protocols are implemented using C at the master end, and the database would include standard ODBC supported RDBMS's and intuitive and object-oriented GUI's with a host of industry specific features like Alarms, logs, reports, event-logs, security, triggers, procedure call support etc. The GUI's would be normally C++/VC++/Java GUI's depending on the requirements. This whitepaper looks at the advantages and disadvantages of employing Java in developing a GUI application, backend database and Protocol stacks to support a 500 line, 256 node scalable and multi-platform supported application. Towards this purpose IEC protocol is taken as a sample protocol to consider some of the architectural and real-time issues in the design, implementation and functioning of the system.

This paper is organized as follows: The next section looks at the different stack implementation issues of IEC and DNP, the section 4 looks at the real-time constraints of these protocols, section 5 delves into the Java and its real-time issues, section 6 details the use of Java in the case of building a 500 line system supporting multiple protocols, and the section 6 looks at alternative implementation frameworks and section 7 looks at advantages and dis-advantages and section 8 concludes the document.

3 IEC/DNP Protocol Overview

These protocol implementations follow a straightforward state transition architecture, which implements either the master or the slave node. The state's define the different stages of the master or slave node at any given point in time. These state transitions happen through well defined framing and link-level sub-systems. The framing sub-system handles the frames at different stages as well as the frame validation and check sequences. The link-level sub-system handles the data transfer and low-level protocols.

IEC and DNP are telecontrol protocols defined by the IEC TC57 (Technical Committee) for remote control and monitoring of real time devices. Both are based on the IEC 60870-5 series of protocol specifications.

3.1 Layering in IEC/DNP

IEC - Enhanced protocol architecture(EPA)- Three layers: Physical, Link and Application

DNP - Four layer: Physical, Link, Network and Application.

3.1.1 Physical Layer

IEC – FT 1.2, Modulo-256 Checksum, Hamming Distance 4, Integrity Class 2

DNP FT 3.0, 16 Bit CRC, Hamming Distance 6, Integrity Class 3

3.1.2 Link Layer

IEC - Frame Length 255, Unbalanced and Balance Transmission, Destination and Source Addressing – 2 byte (max- 65534)

DNP - Frame Length 255, Balanced Transmission, Destination and Source Addressing – 2 byte (varying address field size max- 65534)

Both use Link services.

These include message-handling abilities such as:

- Reset link
- Reset user application
- Link level Send/Confirm service
- Link Status Requests

3.1.3 DNP Pseudo-Transport Layer

The DNP protocol varies from the EPA 3 layer model by adding a fourth layer – a pseudo-transport layer. The pseudo-transport layer segments application layer messages into multiple link messages and provides a segment tracking mechanism. 60870-5-101 limits the size of its application layer messages to that of the link layer (255 bytes) and hence does not require transport layer fragmentation.

3.1.4 Application Layer

Both Use:

The application layer uses function codes to indicate the purpose, or requested operation, of the message. The functions include:

Reporting - Polled report by exception, Unsolicited Responses

- Time Synchronisation
- Read/Write message identification
- Digital Control Commands e.g. Select before Operate, Direct Operate
- Freeze and Clear commands for counters
- Time-stamped events
- Data Groups/classes and priority as per real time requirement

IEC- Defines Interoperability Table, Single Object Type in one frame, Link or Application Layer Polling

DNP- Defines three levels of Implementation, Multiple Object Types in one frame, Application Layer Polling

4 IEC/DNP/MODBUS Protocol Implementation Model

Most common and preferred implementation architectures for these protocols are event driven object oriented architectures. These protocols are essentially state machines, and the state machine is always in a very well defined state at any given point of time. The states at the master and slave side vary slightly depending on the next state that each can enter based on an actual or triggered event.

The normal architecture in the current scenario shall have the following design philosophy:

- a. Development of a generic object oriented interface specification
- b. Development of an object oriented and event driven protocol definition template
- c. Re-use of these object templates for supporting multiple protocols

To achieve the above, the interactions between the protocol sub-system and the application interface sub-system shall be a well-defined and re-usable set of object interfaces. The protocol state transition sub-system shall again follow the object oriented abstraction model, where-in the states are defined by event driven transitions and the interfaces to the application, link and physical level functions are segregated as distinct object methods.

4.1 Real-time constraints

The real-time constraints of these protocols mainly rest on the time-defined response to the state transition events. These are very time-critical at the slave end and to a much lesser extent at the master side.

4.1.1 State Transitions

All the protocols considered in this paper uses a request/response mechanism for the transfer of data. Communication between master (Primary Station) and Slave (Secondary Station) is driven by the request by the master for different functionalities as per the state of the connection. These state transitions could be in the form of a request to test the status of link ("Request Status Of Link"), for which the slave responds with a confirmation ("Status Of Link"). To each request there is a well-defined set of replies including acknowledgement and data transfer if required based on the request. In the event of a request not being replied to, there is a predetermined time-based retry mechanism, to a fixed or configurable set of retries after which the communication is considered faulty.

In the case of IEC, these state transitions shall be as follows:

- Establish a Link.
- Issue a General Interrogation Command.
- If master doesn't get a reply for the GI issue a Test command.
- Request Priority Class 2/1 data.
- Periodically synchronize the RTU time by the Time synchronization command.

4.1.2 Protocol Real Time requirements

The real-time requirements for the protocols involve the system specifications for master update of all events from any remote station within a predetermined time, the response to a command within a pre-determined time and the flushing of events at the device within a pre-determined time. These functionalities require that slave to have a greater level of real-time needs than the

master. At the master end, the protocol implementation should be capable enough of retrieving data available at its interfaces and delivering to the application database in a pre-determined amount of time as well as to deliver a request due to a device within a pre-determined amount of time.

Below a list of slave side real-time requirements for the IEC is described. These slave side requirements shall translate into master side requirements as mentioned above.

Event Occurrence and retrieval:

Events are defined and generated at the Slave side as per the system requirement/configuration. There are two types of events Analog and digital. Analog events are defined by the "Deadband". Within events digital events are of higher priority than analog and will be sent first by the slave. This is defined by the protocol to uphold the real time requirement of a telecontrol system.

Control Command Execution Latency:

Control commands are sent by the master to change the digital state or the analog value of a "point" at the remote station. Single step and dual step operations are defined by both IEC and DNP protocol. Dual step command procedure is to bring the higher reliability for critical operations.

Time Synchronization:

All remote stations (slaves) are synchronized with the master time at every predefined interval. Time sync. Command is generated by the master and is processed as the highest priority incident by the slave device/s.

Static Data Update:

Static data update includes the general interrogation command/request, counter request, Class 2 data request etc. by the master to retrieve different data at the slave for a periodic update of the master database.

5 Java and real-time applications

5.1 Generic Advantages of using Java for development of real-time applications

1. Portability across platforms.
The same application may be developed on any platform and easily ported to any other platform (including desktop OS as well as embedded OS)
2. Object Oriented Model
Java is a fully Object-Oriented programming language, which allows the modeling of real world problems speedily and easily
3. Rapid Application Development
Java provides the easiest programming environment with extensive support for new features through the standard Java APIs as well as extensions contributed by other sources. Development speeds are typically double or more than its counterparts
4. Less infrastructure to kick-off
For embedded application development Java provides a safe and secure means to develop the entire application on desktops and later porting it easily to the embedded platform at a later stage

5. Popular embedded Java environments like Sun's Linux-based real time system support the full range of desktop Java APIs

5.2 Contention points for using Desktop Java to run Real Time applications

1. Java's memory management is based upon a system thread that is out of the application scope. This may cause bottlenecks where the garbage collection thread may interfere with real-time access or operation
2. Reliability is higher than C/C++ due to strong typing and lack of operable pointers, but performance may be degraded due to Java's interpreter type runtime and conversion to underlying OS native code

6 Java implementation for IEC

A typical IEC implementation uses C. The software runs on a PC at the master level and on embedded platforms in the devices at the slave level.

Therefore there are two possible physical locations of using Java as an alternative for C (or other native OS applications)

- PC master
- Embedded slave

The combinations of the above deployments (Java master – C slave, Java master – Java slave etc) will not pose any difference to the execution since the Java end will have it's own stack and will access binary data transmitted by the other end directly.

Considerations for using Java in each of the above contexts are as below

6.1 PC Master

6.1.1 The functionality of the master software is as below

6.1.1.1 Polling the slaves (Sending control commands to the slaves, Receiving events from the slaves, master-time synchronization etc)

IEC handles polling with request/response pairs. Real time performance is related to the round trip time for each slave individually, and the processing time of received information.

6.1.1.2 Serial port interface for Java is provided by the new Java Communication API (javax.comm) version 2.0. This relatively new extension to Java makes the Java stack development project viable

6.1.1.3 Java uses high level objects to access the serial port buffer. This will involve a number of wrapping functions to each system call. Thus each layer of the protocol stack (if developed in Java) will be a high-level abstraction of the corresponding system-level operations in the layer. This will affect the performance of each transaction (relative to C) since Java bytecode is translated to system calls only at run-time and not at compile time. Possible research on overcoming this obstacle will involve investigating conversion of bytecode to OS-specific machine code (executable) which will render it equivalent to C. This is not impossible but has many issues like the loss of the Runtime environment's Garbage collection feature, security features etc.

- 6.1.1.4 The protocol requires processing time for received data per transaction. This is degrading to performance since it involves expensive disk read-writes. Java provides a possible solution in using its rich multithreading environment. So the workaround to cache data in memory and have a separate thread perform the disk storage can be easily implemented in Java.
- 6.1.1.5 The multithreading feature of Java could address the performance losses as in the above points. This will depend upon the JRE's implementation of its threading model, which may be different for different OS. For e.g. Linux uses native threads (one-to-one mapping of Java threads to Native threads). Other models include many-to-many and many-to-one threading
- 6.1.1.6 The IEC/DNP/MODBUS protocols are simple protocols involving 2-4 layers only. This will significantly reduce the performance degradation
- 6.1.1.7 The protocol at the physical layer already provides for a delay of 5-20 milliseconds for RTS/CTS handshaking. Apart from this the protocol specifies a maximum baud rate of 9600 only. These can imply that the protocol itself may be a bigger bottleneck than the programming platform and thus may render Java or C to be equally viable. This however needs to be tested by accurately profiling a prototype application

6.1.2 Maintaining configuration information of each slave (databasing function)

This function will be well served by Java since it is not real-time in the real sense of the phrase. Java can easily provide the databasing function using any RDBMS or if preferred, disk file formats.

6.1.3 GUI display and updation

GUI creation and display will be without any issues. GUI updating typically may be slightly slower but will not pose a serious threat since only a fixed subset of the data points will be represented in real-time at any instant of time

6.1.4 Report generation in specified formats

Report generation in Java opens up the possibilities of extending the reporting mechanism to the web easily. Issues predictable at this time will be porting of standard report formats like spreadsheets to popular non-java versions like Excel. Graphing on-screen and to image files is easily doable.

6.2 Embedded Slave

Slave applications are extensively time critical. Functions include synchronizing the slave's time to the master, which is typically at a resolution of 1 millisecond. The slave also switches between returned data points based upon instantly calculated priority values, to accommodate critical events in real-time

Real Time Java environments like RTSJ etc provide for these real-time requirements by modifying the internal Java implementation of various facets like memory management model etc. without affecting the interface. This means that the code will remain the same but the underlying processing will be modified to handle the constraints. However the suitability of these environments for real-time embedded-applications remains a dispute. We recommend the use of the existing device platforms at the slave end

7 A case study for IEC delay/lag performance on a real-time system

A study was conducted by us, for one of our clients to look at the delay/lag in the data retrieval by the master. The lag is induced by the Network and processing at the slave.

The following is the result of the study. The communication is over a Radio network with IEC protocol at 9600 baud (the highest speed specified by IEC). The network had 32 lines and 10 nodes in each line. The number of lines and nodes does not affect the response delay but it depends on network and the response time by the Slave.

Response time for:

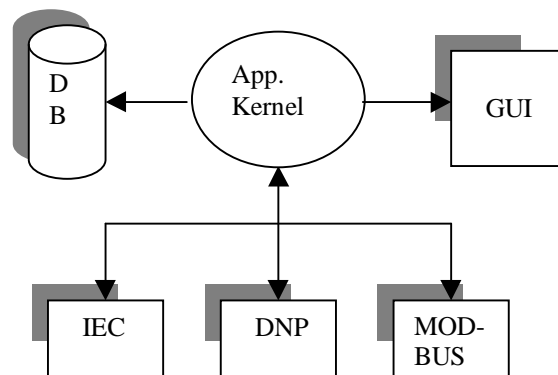
Digital Events	: 150 - 200 ms.
Analog Events	: 250 - 300 ms.
Static Digital data	: 200 - 250 ms.
Static Analog data	: 500 - 600 ms.
Time gap between messages	: 200 - 300 ms

An estimate of a ten times more efficient and quick network gives us a lag of 10 millisecond.

Another *important* point of note is that IEC protocols only support a maximum baud of 9600 which may render both Java and C(or native OS exe's) similar in performance. This can only be verified by profiling a prototype application to establish that the protocol (and not the platform) is the bottleneck

8 High-Level application framework

▪ System Model



The entire application shall be managed by an application kernel, which shall be a multi-threaded object oriented kernel that shall manage the entire system. This kernel shall manage the object information in a warm-memory, and shall have memory threads that update the DB with cold data. The Stacks shall update the warm application memory with the station information, and parallelly dump these information into the historical database.

The GUI shall consist of a set of threads that shall manage the normal GUI functions in addition to threads that shall manage event updations, change of status etc., by continuously interacting with the kernel memory as well as the DB.

The stacks shall update the kernel object memory, of all the remote device information from the field. The kernel memory handling routines shall manage the stale data from being updated into the DB as well as the data that is registered in the database from being made available to the GUI.

This architecture is a scalable model. This model can also support hot-standby redundancy if required. This can be scaled to manage a distributed array of systems if required.

▪ **Protocol Stack Model**

The stacks shall be implemented as independent state machine tasks. These tasks, shall run on a line-by-line mode. Each line task shall manage the line with a specified protocol. These shall include managing the objects for the devices connected to that network, state transition for each device based on the device state and updating the memory and database. Each state of the device shall be managed by objects with defined methods and states. Thus each device modeled as an object shall define the state machines actions for that particular device.

▪ **Alternate Models**

There are alternate SCADA applications that are available in the market that could be employed to support these protocol stacks as well as multitude of network lines. However each of these work on a given system and needs a set of well defined interfaces. There are also a multitude of protocol stack libraries that are available that could be purchased to build a similar system. However here again these are C specific routines and supports a given set of platforms.

9 Advantages and Dis-advantages

Employing Java for a real-time stack implementation throws up a host of opportunities as well as concerns. Java has certain performance limitations, when compared to a similar C based or C++ based system implementation. Java also has certain limitations as far its real-time capabilities are concerned. These two factors are important considerations for the using Java as an application and stack implementation technology. For the present project of IEC/DNP/MODBUS stack implementation on Java will not impose performance limitations as the real time requirements for the project are not very high end/ high speed. This is based on the performance study for IEC as mention in the above section.

Java throws up a host of possibilities also because of its object oriented and multi-threaded framework as well as its garbage collection and other features. The portability of Java into a host of other platforms is one of the key factors. There is a host of work happening in the real-time Java endeavors, which shall in future bring out a number of Java JVM's and JDK's that shall support real-time features and better byte-code execution capabilities. Java development is also more cleaner and neater and very maintenance free compared to a lot of other implementations.

In the case of using Java as a master application for polling field data in a scalable model, and managing a huge set of network lines and GUI's, the Java would meet the requirements well. The real-time constraints at the protocol master end is not as critical as at the slaves, and the performance constraints of the master state machine which have to handle a large number of network devices are not as critical as the slave state machine itself. The protocols like IEC/DNP/MODBUS etc., also have inherent standard limitations that makes the implementation of the stack with Java possible at the master end.

10 Conclusion

The issues related to real-time application development were explained, based upon a description of two popular protocols. Further the advantages and disadvantages of using Java in real-time applications were discussed on both the master as well as the slave side. Java was found to be advantageous on the master side for reasons of platform-independence, ease-of-development, problem-modeling etc. but is suspect on the issue of performance. The impact of

Java on the performance was minutely examined and the threat was found to be significantly reduced by specific workarounds as well as the nature of the protocol and the application itself.

The use of Java on the slave side was discouraged. Further additional research on the advantages of Java's threading model, and a profiling of its performance as a Java stack were proposed to give more quantitative results.