Interprocessor Communinication Monitoring in DKTS 30 Switching System

Branko Kolašinović[1], Mirko Markov[1], Milan Jovanović[1]

Abstract - **The DKTS 30 interprocessor communication monitoring software is developed in order to provide two basic components: periodical error polling functions and interprocessor communication errors reporting. Similar to the SNMP, this software has a vast ability to collect the DKTS 30 network status information and determine the network health. Furthermore, it is very effective in undertaking appropriate actions in case of irregular situations.**

Keywords – **switching system, interprocessor communication, network management, SNMP.**

I. INTRODUCTION

The DKTS 30 public digital telephone exchange is the newest product of the well known proved DKTS series of digital telephone switching systems. Although it has been successfully commercially exploited for more then a year, it is still under development in order to achieve better quality, to provide new services and to reduce the production price. Like other modern telephone switching systems it is based on a large number of off-the-shelf microprocessors and microcontrollers, that frequently need to communicate with each other. Therefore, it was crucial to provide a reliable high-speed, high-throughput interprocessor communication mechanism, so the need for an efficient interprocessor communication monitoring arises.

Although different protocol stacks supporting interprocessor communication in distributed real-time systems exist, it was decided to develop a proprietary one [1]. The main reasons were: the complexity of DKTS30 system architecture and several working modes of its units impose some specific solutions, compatibility with previous generation (DKTS20) of peripheral units, and redundancy of critical resources in order to increase the system reliability. The application software is unaware of distributed nature of hardware resources and does not care whether it communicates with DKTS20 or DKTS30 units.

The aforementioned particularities of the interprocessor communication mechanism implicate a specific interprocessor communication monitoring solution. The DKTS 30 interprocessor communication monitoring subsystem is realized to provide:

1. a hardware resource monitoring,
2. a hardware-level error detection,
3. a protocol-level error detection,
4. a prompt reaction to the detected malfunction,
5. a logging of fault information into an appropriate file.

The interprocessor communication monitoring software is designed in such a way, that it works together and interacts with the rest of the software in the system, conforming to the global concept of the development of the DKTS 30 software [2].

[1] PUPIN TELECOM DKTS, Batajnički put 23, Beograd, Yugoslavia. E-mail: {brankok, mmarkov, milanj}@ dkts.co.yu

II. TOPOLOGY OF THE DKTS 30 SYSTEM

The DKTS 30 system architecture is given in Figure 1. It consists of central blocks, peripheral blocks, and terminals. The central blocks are: administration (ADM), switching (KOM), synchronization (OSC), source of speech information (GGI) and USP (*Universal Signaling Processor*). In order to increase the reliability of the system, the central blocks are duplicated. The administration unit is an industrial PC, while other central blocks are originally developed boards based on the Motorola 68360 family of processors. The central blocks are connected via a local Ethernet, that is doubled, too. The peripheral blocks (PB) are subscriber blocks and interexchange trunks.

The USP unit consists of a UCP (*Universal Communication Processor*) unit and a signaling processor, connected via HDLC link. The UCP unit distributes the messages among the central and peripheral blocks. Previous generation (DKTS20) of peripheral blocks are connected to the UCP blocks via serial HDLC links. One pair of UCP units works in the load-sharing mode for a group of six peripheral units. Each of those six peripheral units is connected to both UCP units by its own separate link. The terminals can be local or remote. Local terminals are connected to the administration blocks via a separate local Ethernet.

Therefore, two types of interconnections exist in the system: Ethernet networks and HDLC links. Although, it is not a custom to use Ethernet in real-time systems, it was shown that Ethernet meets expected traffic intensity. Moreover, Ethernet hardware and supporting software are common, available, and cheap.

III. THE ORGANIZATION OF THE SOFTWARE

The DKTS 30 software is based on object-oriented principles. It is developed using UML notation [3], and implemented in C++ programming language. The software is organized as a collection of server objects that are distributed among the processors. The main abstractions of the system are modeled by server objects. Each server object has a unique identification, known to all processors in the system. This identification consists of a processor identifier (the logical address of a processor), a class identifier, and an object identifier. The physical address of a processor is its inherent fixed attribute, while the logical address depends on its working mode.

Server objects that model the abstractions in the system are implemented as finite state machines (FSM). This is a common approach in design of real-time systems. Each FSM is designed according to the *Bridge* template [4], and consists of an interface and an implementation object. Interface and implementation objects may reside on different processors, and the only connection between them is their unique identifier of the object.

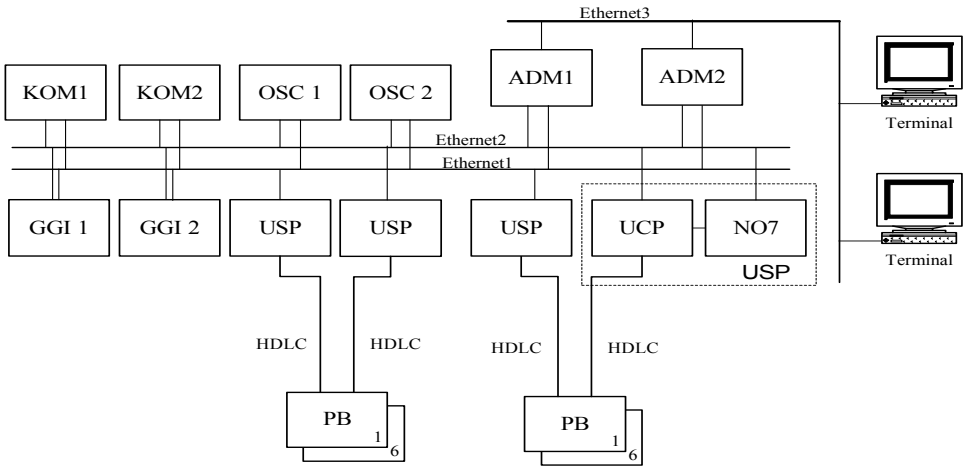

Figure 1: The DKTS 30 system architecture

IV. INTERPROCESSOR COMMUNICATION

The role of the interprocessor communication software is to provide a reliable high-speed, high-throughput message passing mechanism. This mechanism is used to connect server objects that can be placed on the same processor or on different processors. In the case when the switching system contains both old (DKTS20) peripheral units and new (DKTS30) units, neither server objects on new units, nor software on old units, are aware with whom they communicate. It is obvious that all differences in used message formats, as well as differences in applied protocols, have to be solved in the interprocessor communication software.

The interprocessor communication is connectionless. A recipient must send back an acknowledgement message to the sender upon data receiving. If an original message comes from an old peripheral unit, the acknowledgement message must use the path that has been used by the original message. An alternative path (if any) is chosen in the case of a retransmission.

The particular protocol stack that implements the desired interprocessor communication model is designed having in mind the aforementioned demands, as well as ISO Open Systems Interconnect (OSI) specification. The major functionality of the interprocessor communication software is divided into logical parts representing layers of the protocol stack. Each layer uses the layer immediately below and provides a service to the layer immediately above.

The application layer provides a conversion from a logical address of a destination processor into a collection of physical addresses and sends a message to all obtained processors. The transport layer converts a physical address of a destination processor into a collection of IP addresses of the destination, sends a message to one of the provided addresses, and cares about acknowledgement messages and retransmissions. The network layer selects an available internode and converts messages and protocols from old into new one and vice versa. The data link layer performs communication with adjacent nodes. In the Ethernet case, the UDP protocol from TCP/IP protocol stack is used. The physical layer is responsible for the electrical and mechanical connections.

V. INTERPROCESSOR COMMUNICATION MONITORING

There is a number of available software packages on the market today, supporting some form of communication protocol monitoring in TCP/IP networks. Most of them utilize the communication protocol known as SNMP (*Simple Network Management Protocol)*, described in a long and growing list of RFCs, starting with the standard [5].

This network protocol is based on a client/server architecture. The server program *(SNMP Agent)* resides on a remote network device. The client program *(SNMP Manager)* resides on the network server. The Manager sends queries to the Agent. Server software replies to these queries, and sends information showing the current state of the device that the Agent resides on. The database that is controlled and updated by an Agent is called MIB (Management Information Base) and represents the standardized set of network statistical and control values.

Requests that a Manager sends to an Agent have an SNMP variable identifier (often called a MIB object, or a MIB variable), and a command that specifies whether to read a value (*get request* messages), or to set a value (*set request* messages). *Get request* messages acquire the information on the state of network devices. *Set request* messages provide the means of configuring and controlling network devices. At last, the standard provides *trap* messages allowing the device to inform the Manager about fault conditions.

The basic components of network control, provided by SNMP, are:

1. A periodic collecting of information on availability of transport medium, i.e. network nodes and links. This is achieved by polling the values of certain MIB objects and alarming, if the values of monitored objects exceed the maximal allowed range.

2. A periodic collecting of information of the state and trends inside the network, which allows the monitoring of network traffic, and congestion control.

3. An ability to respond to the alarms that are sent asynchronously from the network devices. This provides the prompt reaction on malfunctions reported by network devices at the moment of their occurrence.

4. A certain level of network management, i.e. possibility of executing *set requests*.

VI. "SNMP LIKE" SOLUTION IN DKTS 30 SYSTEM

Since the interprocessor communication was developed to be a proprietary one, it was natural to deal in the same way with its monitoring. The implemented solution of the interprocessor communication monitoring is very much like SNMP. Actually, part of it, depending on a used operating system, is based on the standard SNMP. The solution supports all information like SNMP, often extended by some others, but does not always provide messages prescribed by the standard. Also, the Manager is not a separate centralized application working on the administration unit, but is rather divided and its functions are distributed among the processors.

Functions of the interprocessor communication monitoring are divided into few classes – FSMs. All these classes are designed using the design pattern *Singleton* [4]. The *LinkErrorMonitor* works like an SNMP Agent. An object of this class resides on all originally developed boards and it is responsible for monitoring of hardware resources. The *TestNIManager* could test a state of all network interfaces by polling transport layer of a particular processor. It resides on administration blocks only. All other services of the monitoring, including a logging of fault information into an appropriate file, are provided by *LinkErrorCollector*. The implementation of this object resides on administration blocks only, while the interfaces are distributed among all processors. The *ProtocolStatistic* is responsible for protocol statistic collecting. The significant role is performed by the *PollingManager*. The implementation of this class resides on all processors and, among other functions, triggers execution of periodic polling activities.

The detection of communication protocol faults is achieved in three ways:

- by periodic collection of information about transmission media availability,
- by communication protocol reports about failed transfers,
- by periodic polling of network interfaces.

Also, a collecting of protocol statistic can detect some interprocessor communication irregularities.

VII. PERIODIC MONITORING

The collection of information about transmission media availability on all originally developed boards is done periodically. The UML sequence diagram of this activity is given in Figure 2. The *LinkErrorMonitor* resides on all considered processors and it is responsible for monitoring of hardware resources. The *PollingManager* object addresses the *LinkErrorMonitor* abstraction for assessment of link states. The *LinkErrorMonitor* inspects all network interfaces (Ethernet ports and HDLC links) of its own module. In a case it detects certain malfunctions, it notifies *LinkErrorCollector* abstraction by calling functions supported by *LinkErrorCollector's* interface on the particular module. The notification messages are sent to the master administrative processor, where the implementation of *LinkErrorCollector* will be able to take appropriate actions.

The *LinkErrorMonitor* class achieves its functionality by iterating through MIB base and checking the values of certain MIB objects. This is implemented using the services of a host real-time operating system. Currently, it is commercially available pSOS [6], while porting to open source RTEMS [7] is near the end. The pSOS operating system, due to its network manager pNA+, supports MIB base for network monitoring, according to MIB-II standard. MIB objects are accessed from the application using the pNA+ *ioctl()* system call. Since the values of MIB objects are monitored only, the message sent from the application software is of *get request* type. The values of MIB objects are stored until the next polling and compared with the defined thresholds inside the application. If the values exceed their thresholds, the *LinkErrorCollector* is alarmed.

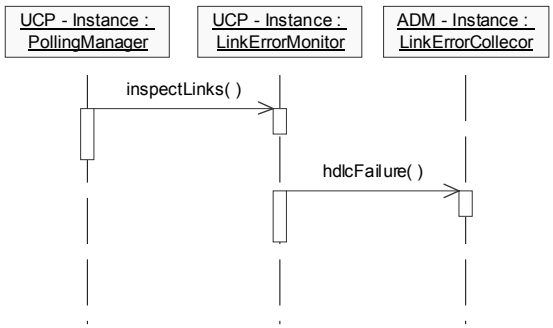

Figure 2. Sequence diagram - monitoring network resources

VIII. FAILED MESSAGE TRANSFER REPORTS

The errors inside the implemented message passing mechanism are caught through the whole protocol stack. When an error is detected, the *LinkErrorCollector* abstraction is notified. The UML sequence diagram of an example of the transport layer fault report is given in Figure 3.

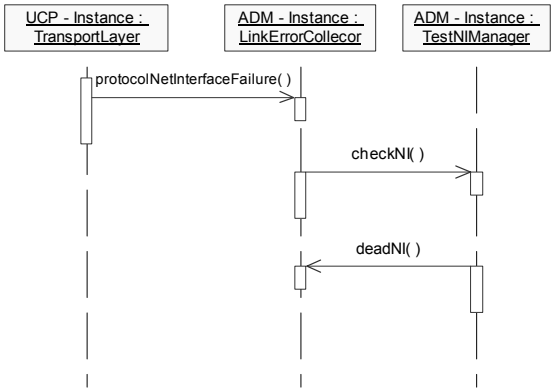

Figure 3. Sequence diagram – failed message transfer report

The task of the transport layer in a reception is to identify an incoming message. There can be three cases. If the incoming message is an original one, the reply to the source node is sent. If a positive reply is received, no action is taken. If a negative reply is received, or the timeout interval is exceeded, the message must be retransmitted. It is possible that the certain irregular condition arise. In that case the *LinkErrorCollector* is alarmed. The message to the *LinkErrorCollector* will be sent, if the positive or negative reply arrives, and there is no corresponding message in a sent messages buffer, if the message of unknown type arrives, if the timeout for the reception of the reply is reached, or if the number of retransmissions reached the maximum value. Each of these irregular events corresponds to the certain error code.

The *LinkErrorCollector* abstraction requires the *TestNIManager* to check the suspicious network interface. In the case the network interface is determined to be dead, appropriate actions are performed.

IX. NETWORK INTERFACES POLLING

Very powerfull interprocessor communication monitoring mechanism is a periodic polling of network interfaces. This task is done on the master administration only. The UML sequence diagram of this mechanism is given in Figure 4. The *PollingManager* object orders the *TestNIManager* abstraction to test a state of all or only dead network interfaces. This is done by polling transport layer of a particular processor. Because the transport layer is the part of the running application, this test verifies a particular network interface, as well as the fact that our application still runs on the particular processor. This is the way to discover that a network interface has become alive.

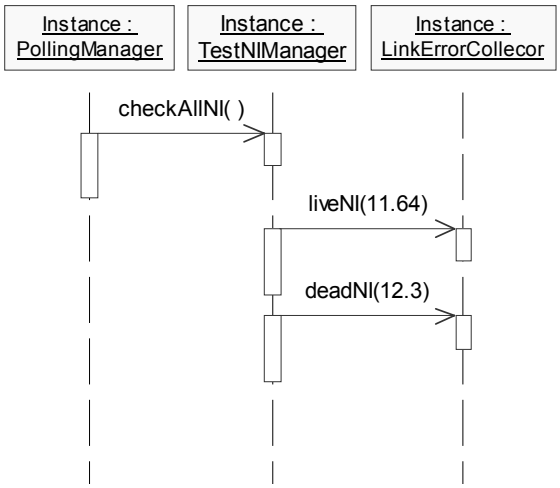

Figure 4**.** Sequence diagram – network interfaces polling

X. ACTIONS BASED ON FAULT REPORTS

The messages about faults arrive to the administrative unit, i.e. to the implementation object of the *LinkErrorCollector* abstraction, so it is possible to gather and log information, such as: code of reported error, source/destination processor of the failed message, and the source/destination network interface of that message. Upon reception of a message, this object can ignore the message, or declare that the corresponding processor/network interface is dead or suspicious if there are certain irregularities, but not in such degree to render certain processor or network interface unusable. In that case it requires the *TestNIManager* to check suspicious network interface. The *TestNIManager* polls the suspicious network interface predefined number of times, and if there is no response, the corresponding object will be declared dead.

The final goal of the interprocessor communication monitoring is to assure the successful operation of the interprocessor communication. Because the local images (the database that resides on each processor and keeps the states of physical processors, their operating modes and the states of network interfaces) are of crucial importance for the efficient and reliable interprocessor communication, it is necessary to provide the fast update of the local images on all processors. Therefore, a message about the failure or recovery of a certain processor or a network interface is broadcasted to all processors right after particular detection in order to update the local image database.

XI. PROTOCOL STATISTIC

Statistic about all sucessfully/unsucessfully sent messages is collected on all processors. These counters are reset every day at midnight. The values are available on a request from terminal. Some interprocessor communication irregularities can be detected by analyzing gathered statistic

XII. CONCLUSION

The realized interprocessor communication monitoring software, as an integral part of the complex DKTS 30 software, is developed in order to provide a reliable, efficient and fast method of interprocessor communication error detection, so that the adequate actions can provide undisturbed operation of the communication protocol. The significant complexity of the DKTS 30 system, and global concepts of DKTS 30 software development dictated the need for the proprietary solution. This solution is very much like SNMP, the most ubiquitous network control protocol, and provides the same functionality concerning the periodic polling of information about the transmission media availability, and the detection of trends inside the network. The communication protocol monitoring software also provides the error detection inside the communication protocol, which would be impossible with commercial software solutions, because there would be no simple interface for the programmer. In addition, this software provides the possibility of taking appropriate correcting actions, significantly increasing the reliability of the interprocessor communication even in challenging situations.

The described solution definitely fulfills DKTS 30 phase one development requirements. However, a remote supervising of the DKTS 30, especially the ability to be supervised with other public telephone exchange from one place, dictates full support of SNMP standard. This would not be a problem due to adopted foundations of the realized solution.

XIII. REFERENCES

[1] D. Vujadinović, S. Spasojević, J. Mrdalj, M. Jovanović, V. Hiršl, "Interprocessor communication software in DKTS30 switching system," *TELFOR '98*, *Conference Proceedings*, Beograd, Yugoslavia, 1998.

[2] *DKTS30 project specification,* Pupin TELECOM DKTS, Beograd, Yugoslavia, 1997.

[3] *UML Semantics*, Rational Software Corporation, USA, 1997.

[4] E. Gamma, R. Helm, R. Johnson, J. Vilsides, *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[5] *RTEMS C User's Guide,* Edition 1, for RTEMS 4.5.0, May 2000.

[6] *pSOS System Concepts – Network Programming*, Integrated Systems Inc., USA, 1997.

[7] J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin, "A Simple Network Management Protocol (SNMP)," *RFC 1157*, 1990.