The Redesign of the Software of the DKTS 30 Switching System to Support Extended Capacity

Branko Kolašinović[1], Dimitar Komlenović[1], Milan Jovanović[1]

Abstract – **Some software solutions implemented in the project of extending capacity of the DKTS 30 switching system were presented in this paper. This project includes: a redesign of the system image, a redesign of the interprocessor communication, more natural integration of the remote subscriber unit into the system, algorithms for distributed systemsupervision, appropriate changes in the system data base and the graphical user interface.**

Keywords – **switching system, embedded real-time software, DKTS 30**

I. Introduction

The DKTS 30 public digital telephone exchange is the newest product from the series of DKTS digital telephone switching systems. Although it has been successfully commercially exploited since 1999, it is still under development in order to achieve better quality, provide new services and reduce the production price.

II. Problem

The first DKTS 30 public digital telephone exchange was designed to provide the capacity of 15872 subscribers. However, this number was shown to be insufficient with the uprising market demands. Therefore, the project of extending the capacity was undertaken. The project goal was to provide the maximum capacity of 174592 subscribers. All the necessary changes in the hardware were presented in [1]. This paper presents software changes, while hardware changes were presented in a degree of detail that was necessary to understand the presentation to follow.

The project requirement was not only to trivially provide capabilities for a larger number of blocks in the system, but also to pay specific attention to the system performance. It was highly unlikely that the algorithms that were shown to be efficient with one traffic load would be efficient in the same way with much heavier traffic load.

The project of extending capacity comprised changes in several categories of the DKTS 30 software. The categories that needed changes were: the system image, the interprocessor communication, the interprocessor communication monitoring, the system database and the graphical user interface. Furthermore, with these redesigned categories, it was necessary to implement new algorithms for distributed system monitoring and to obtain a more natural integration of remote subscriber blocks.

As far as software compatibility is concerned, it was decided to obtain compatibility of the newly designed software with the old hardware, but not with the old software. This means that the new software may run on all DKTS 30 platforms, but it must not be mixed with old software. The goal was to move beyond some obstacles that were present in previous software solutions due to required compatibility with the DKTS 20 system, which was put aside in the meanwhile. Special attention was payed to the intermediate period in order to make it possible to add new funcionalities to the software of the old system as well as to the software of the system with extended capacity.

III. System Architecture

The DKTS 30 system architecture is given in Figure 1. The system 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*). The USP unit consists of a UCP (*Universal Communication Processor*) unit and a signaling processor, connected via an HDLC link. The UCP unit distributes messages among central and peripheral blocks. In order to increase system reliability, central blocks are duplicated. Central blocks are connected via a local Ethernet, which is doubled, too All central blocks, except USP units, are connected to both Ethernet networks. The terminals can be local or remote. The local terminals are connected to the administration blocks via a separate local Ethernet.

Peripheral blocks are connected to UCP blocks via serial HDLC links. A pair of UCP units works in a load-sharing mode for a group of six peripheral units. Each of these six peripheral units is connected to each of two UCP units by its own separate link. Peripheral blocks (PB) are subscriber blocks and interexchange trunks. Peripheral blocks are originally developed boards based on the Motorola 68302 family of processors and on the originally developed operating system. Voice transmission is achieved by the PCM multiplex that connects peripheral blocks to the switching block.

A remote subscriber unit is realized using an IUUB4 block which combines 4 subscriber blocks with 128 subscribers on each. That gives the maximum capacity of 512 subscribers. The function of the remote subscriber unit is to map subscriber block interfaces. In this way, software of the central UCP module does not make a difference whether a block is a local or a remote one.

The administration unit is an industrial PC. The DKT3 30 application for the administration unit may run under various operating systems, such as Windows NT/XP or Linux. Other central blocks are originally developed boards based on the Motorola 68360 family of processors. Software for these boards may also run under different real-time operating systems, such as pSOS and RTEMS. The terminal unit is a common PC. The operating system used on this unit is Windows NT. The application that runs on this unit is called GUI (*Graphical User Interface*) and represents the interface between the operator and the DKTS 30 system.

[1] PUPIN TELECOM DKTS, Batajnički put 23, Belgrade, Serbia
E-mail: {brankok, dimitark, milanj}@ dkts.co.yu

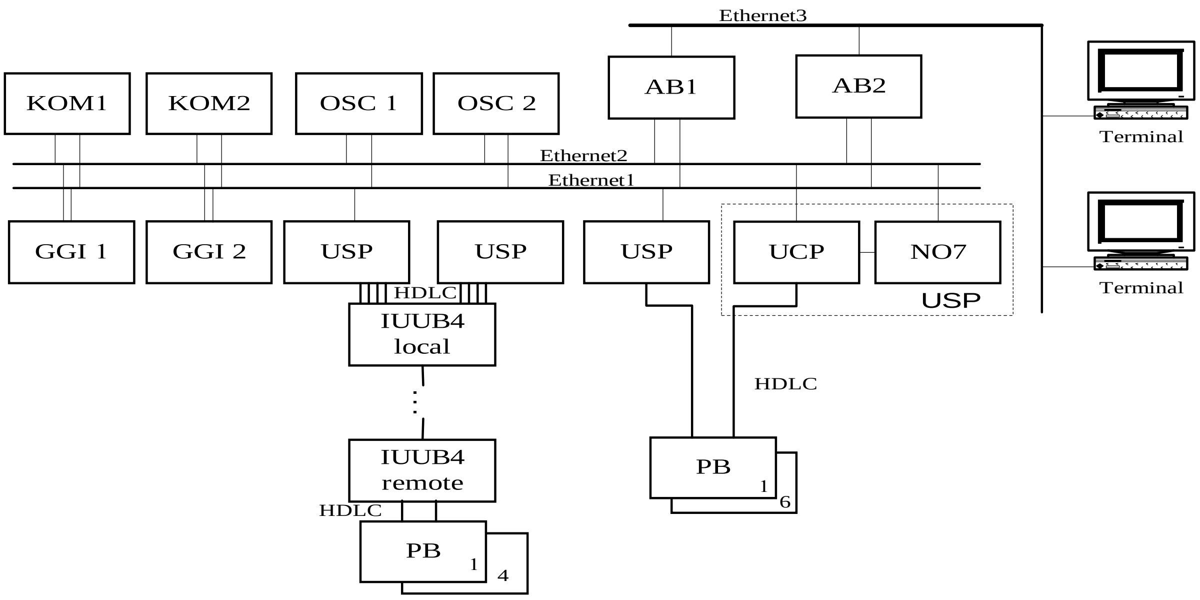

Figure 1: The DKTS 30 system architecture

IV. SOFTWARE ORGANIZATION

The DKTS 30 software is based on object-oriented principles. [2] It is developed using UML notation, and implemented in C++ programming language. Software is organized hierarchically into layers. Each layer provides a service to the layer above it, and simultaneously it is a client of the layer below it.

One of the main challenges facing DKTS 30 software engineering is the variety of microprocessors (Motorola 68360 and Intel family are present, processors from the PowerPC family are planned for future use), as well as the variety of operating systems that run on different parts of the DKTS 30 switching system (WinNT, Linux, pSOS, RTEMS).

The software is organized as a collection of server objects that are distributed among processors. Main system abstractions are modeled with server objects. These servers are implemented as finite state machines (FSM), which is a common approach in design of embedded real-time systems [3]. 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 the same processor or on different processors, and the only connection between them is their unique identifier.

The well known CVS (Control Version System) tool is being used to control file versions. CVS lets a developer maintain a detailed revision history of a file that is under its control.. In this way, it makes the reconstruction of any previous version possible. Since there is typically more than one developer who works on the same file, CVS is of great help, because it allows multiple users to work on a file concurrently.

V. NUMERATION OF BLOCKS

Software of the first DKTS 30 system was developed in order to be flexible within the requested maximum capacity. Each block in the system had its determined identification which depends on the block's position. Blocks of the same type have successive identifications. The first DKTS 30 system can host the maximum number of 256 blocks, where the numeration for peripheral blocks starts from 128. According to the request of the project, the existing numeration had to be changed, which was the source of incompatibility difficulties. In order to simplify the routing algorithm, it was decided that every potential position in the topology of the system with maximum capacity had its own identification representing block's physicall address. Therefore, it was necessary to provide space for a number of 3000 blocks, where the numeration for peripheral blocks is agreed to start from 1000. Of course, the possability that the real system would have each position filled is relativelly small.

Another problem were 4-bytes long internet addresses that represented blocks' network interfaces. In the first system, one byte held processor identification, another byte held network identification, while two bytes held identification of a switching system in a WAN network. In the system with extended capacity, the 4-bytes internet address were retained. Also, two bytes of the address are used for the switching system identification. Since the unique recognition of each block implies the number of 12 bits, the byte that was assigned to the network identification was devided into two nibbles. Since the identification of blocks that are connected to the Ethernet is held in one byte, in order to make switching system's internet addresses equal to internet addresses of network interfaces of blocks that are connected to the Ethernet, it was decided to add the higher nible of the network byte to the procssor identification byte (in total, 12 bites that are not near-by), while the lower nibble should represent network identification.

VI. SYSTEM IMAGE

In order to monitor system behavior as easy as possible and to use collected information in the most efficient manner, a software category called system image is developed. The system image represents the database with all the relevant information regarding state of each resource needed for the interprocessor communication. The system image consists of two subcategories. The central image subcategory represents the image of the whole system with all the necessary information and it is only located on processors with the administrative function. It consists of a map of processors and a map of network interfaces. The local image subcategory is a part of the system image that is located on each processor. It contains all the data structures needed for the implementation of the efficient interprocessor communication. The local image consists of a map of physical processors, a map of internet addresses, and a routing map.

The initialization of data structures within local image is performed according to the central image of the master administrative processor immediately after software has been downloaded to the specified processor and just before the processor starts to communicate with other processors. The local image states are being brought up to date by sending and processing messages that contain information on activation or failure of processor units and network interfaces. Furthermore, local image states are also being made current by sending of central image from the administrative processor. In the first DKTS 30 switching system, this was done according to the defined pattern for the system with the highest capacity. Although the procedure was optimized, the problem of adding new block types remained present during the evolution of the system. Every time a new block type was added, the software for all boards had to be changed, even for boards that were not communicating with the newly added block type. That is why a new procedure was developed for the system with extended capacity. According to this procedure, every block first receives the message that precisely specifies the format of messages that will be used for local image update. By adding this message, the repetition of information in the header of messages that are send in order to make local image current was avoided. Furthermore, dynamic adding of new block types was made possible. Finally, messages that are sent for the purpose of bringing local image up to date may be as long as it is needed, since the information on boards that are not configured are not being sent.

VII. Interprocessor communication

A basic requirement for the design of interprocessor communication was to reduce the number of messages passing through the system, as well as to provide higher reliability of interprocessor communication. In order to fullfil these requirements, it was necessary to make some changes in a message header. The message header remained of the same length as it was. It contains the same fields as in the previous implementation: message type, source and destination internet address, message identification, destination object's identification. However, some changes were inevitable. First of all, the format of internet address had to be changed. Next, the way message type is marked was also changed. As a result of these changes, incompatibility with the previous software implementation became unavoidable. Among other things, with the new message caracterization, it became possible for clients in upper software layers to supress acknowledgement messages on the protocol layer when functional messages are used to acknowledge an application layer acknoledgement request.

In the aim of acchieving faster interprocessor communication, the SP protocol was abandoned. The SP protocol was used for communication between peripheral and UCP blocks via HDLC links [5]. The reason for the presence of this protocol in the previous implementation lay in requested compatibility with DKTS 20 peripheral blocks. Since the concept has been abandoned in the meanwhile, the NLC layer of the interprocessor communication protocol stack was eliminated. In adition to this, another historical role of the UCP board, which also existed due to requested compatibility with DKTS 20 peripheral blocks, became unnecessary in the new software environment. That was the NLB layer whose task was to perform message format conversion between DKTS 20 and DKTS 30, and vice verse.

The design of the first DKTS 30 switching system did not support communication between UCP and No7 boards belonging to one Ethernet with UCP/No7 boards that belong to another Ethernet. However, during the exploitation phase, the need for communication between No7 peer boards was brought to attention. In addition to this, there was the need to put interprocessor communication under software supervision. Accordingly, it was decided to correct this defect by allowing software routing from one Ethernet to another. Central blocks that have available network interfaces belonging to both Ethernets became potential routers. It was more than likely that this task would be commited to the OSC board, since the OSC board is the processor that is not overloaded with interprocessor communication responsibilities. The routing is now supported in system image. In addition to everything else, this routing may be used in the case of a failure of a network interface on boards with two network interfaces. For example, in previous software versions, the KOM board with an inactive network interface that belongs to the Ethernet A could not send a message to a UCP or No7 board connected to the DKTS 30 WAN network via network interfaces belonging to the Ethernet A.

A reduction of the number of messages passing through the system is acheived by the use of multicast techniques [6]. Instead of sending a large number of single messages to different destinations, it is now possible to send only one multicast message that is received by all procesors that are members of the specified multicast group. The great problem was the presence of several operatin system in the DKTS 30 switching system, each of which providing similar (but different) programming interface towards the multicast facilities. In addition to this problem, it was important to provide that all processors in the group receive a multicast message, but also to avoid duplicated messages, which may be the consequence of the use of alternative routes. Periodical updating of local image, sending information on block or network interface failure or activation, periodical checking of block states, sending error reports to terminals are cases in which it is evident that significant improvements in speed and efficiency are achieved.

VIII. Software supervision

Software supervision is responsible for timely and reliable detection of block failure and activation, failure and activation of network interfaces, software download to peripheral blocks, efficient monitoring and error recording, as well as undertaking actions in order to cope with irregularities or to alarm error conditions to operators.

Previous software versions were based on a centralized approach, according to which the administrative unit was collecting information and performing actions. This was an acceptable solution. However, with the extention of the capacity of the public telephone exchange, it was necessary to migrate from centralized to decentralized algorithms. The administrative unit has nevertheless the main role in software supervision, yet it delegates some software supervision functionalities to other central blocks in the system.

For example, in the previous implementation, the administrative unit was checking processor states for all blocks in the system. As a result of the use of decentralized algorithms, periodical checking of the activity of peripheral processors is now delegated to UCP processors. This means that the administrative unit sends request messages to UCP processors ordering them to perform the inspection of peripheral blocks connected to UCP boards. As a result, the number of messages needed to provide some funcionalities is decreased, which contributes to decreasing system overload and shortening time needed for obtaining information of the overall system status.

IX. IUUB4 BLOCK

The IUBB4 block is maybe one of the most important element in the project of extending capacity of the DKTS 30 switching system. As a result of the use of the IUUB4 block, significant increase in the number of subscribers is provided. The new IUUB4 block is connected to the UCP block by only one HDLC channel, instead of four, as it was in the previous solution, where IUUB4 blocks were transparent both on the side of the telephone exchange and on the side of subscriber blocks, which means that other blocks in the system were not aware of their existance. Software on this block was running from the EPROM memory. However, this concept was changed. The IUUB4 software is now being downloaded to the IUBB4 board from the administrative unit, as well as it is performed for other boards in the system. Not only does the administrative unit perform download, but also it inspects IUUB4 processor activity, informs IUUB4 blocks on failure or activation of other blocks in the system. In other words, the IUUB4 block is being treated the same way as any other block. The algorithm for the IUUB4 software download is distributed. That means that the IUUB4 executable is firstly downloaded from the administrative unit to the UCP board, and then it is uploaded to the IUUB4 block.

X. SYSTEM DATABASE

Modern switching systems require storing of huge amounts of data very economically and reliably. These data are necessary for handling calls, i.e. establishing and terminating connections, as well as for different periodical data processing performed by an operator. The database of switching system is used for storing specific system parameters needed for the functionality of the overall system. The system database contains parameters that describe hardware configuration (blocks configuration, their physical and logical addresses), parameters needed for establishing connections with other switching systems, as well as data needed for performing different functionalities, such as users data, billing prices, information services, etc.

The database software, apart from storing large amounts of data, must enable fast searching through the database and finding the data needed to satisfy the request for establishing connections in real-time. Apart from that, this software has to offer consistent data at anytime, to guarantee protection of data and to prevent unauthorized access. It is obvious that the capacity extension of the DKTS 30 switching system imposed changes in the DKTS 30 database in order to qualify this software category for the heavier traffic load.

XI. GRAPHICAL USER INTERFACE

Together with the changes in other parts of the DKTS 30 software, the changes were needed in the GUI application. Not only were there the need to change the graphical representation in order to show a large number of blocks, but also it was needed to adapt the GUI application to the situation of more intense traffic load, i.e. increased amounts of data coming from the administrative unit, processing larger files quickly and efficiently, etc.

XII. TESTING

Because of the large number of different processor types, different hardware platforms, various operating systems, situations in which it is necessary to perform potential error corrections in the field and in laboratory conditions, concurrently with adding new funcionality to the fisrt DKTS 30 system, a great attention was payed on developing methodologies for software testing. As a result, a set of tests was designed. The set comprised different testing techniques: from the partial to the integral testing, with and without the use of call generator, etc. In the testing phase, the use of error information is increased. However, the great part of this code will be excluded from the final version in order to assure executable programs with the minimal time of execution. In addition to this, finite state machines whose only purpose is testing are designed. The FSM whose only task is to generate desired traffic is instantiated on every processor. This is an example of how interprocessor communication is tested.

XIII. CONCLUSION

A flexible base for the switching system with extended capacity was provided. The change of the maximal capacity required a redesign of some critical algorithms and transition to distributed and heterogenous solutions. A lot has been done in order to decrease the number of messages passing through the system, which, in the last instance, may make the response time to certain events significantly shorter.

Because of the decision to abandon compatibility with DKTS 20 peripheral blocks, it was convenient to undertake a redesign of the system software on peripheral blocks, which was shown to be a complex task contributing to the slowdown of the entire project.

Once the project of extending capacity has been completed, it is expected that the work on the DKTS 30 system in order to increase its performance will continue. It remains interesting to see the results of the quality analysis of the behavior of the first DKTS 30 and the DKTS 30 with extended capacity.

XIV. LITERATURE

[1] S. Laketa, P. Vidić, N. Nikolić, “Extending capacity of the DKTS system,” *Telfor 2003*, Beograd, 2003.

[2] Booch G., *Object-Oriented Analysis and Design,* Second Edition, Benjamin-Cummings, 1994.

[3] Selic B., Gullekson B., T.Ward P., *Real-Time Object-Oriented Modeiling*, Willey Professional Computing, 1994.

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

[5] M. Jovanović, V. Hiršl, “Interprocessor communication in DKTS 30 switching system,” *YU INFO 1999*, Kopaonik, 1999.

[6] Deering, S., “Host Extensions for IP Multicasting,” *RFC 1112*, 1989.