STRUCTURAL LAYERED APPROACH TO DESIGN OF EXPERIMENTAL CONTROL SYSTEM FOR AN AUTOMATED EXCAVATOR

Dr. Henryk Dobrowolski

Institute of Computer Science, Warsaw University of Technology

Abstract: Some aspects of systematic design of computer control system for a hydraulic excavator are discussed. A concept of software layers is presented, which can help to lower project cost and to improve the ability of system to be modified and extended.

Keywords: hydraulic excavator, computer control, software design.

1. INTRODUCTION

This paper presents the problem of software design for excavator's computer system from the point of view of a computer science engineer. It is based on a long-time cooperation with Institute of Heavy Working Machines (IMRC) at Warsaw University of Technology.

There are few "intelligent" heavy machines on the market (if any). One of the most important factors causing it is a high expense necessary to design and put such a machine into practice. A common approach to design an automated heavy machine is taking an ordinary one and adding special equipment to achieve a desired functionality. This way cannot lead to optimal solutions, so there is a necessity of new attitude to design process of a machine itself – especially a hydraulic subsystem and engine control have to be prepared to computer control from the very beginning.

Demand on fully automated machines like excavators is not high, so these machines should be manufactured in diverse versions. Therefore a computer system for excavator should be scalable, i.e. reducible to diagnostics and operator support or, on the other side – expandable, according to needs, up to robotized excavator capable of autonomous work without direct human supervising (e.g. for dangerous environmental conditions).

1.1. System Specification

First step of any system design is preparing its specification. A functional specification describes tasks, which represent functionality of a system – in this case the functionality of an "intelligent" machine. The boundaries of system scalability must be described as well. A specification is a top-down process. It starts from the general description of system functions and continues with more and more detailed guidelines.

In recent years formal methods of specification have gained in significance. They allow not only more precise specification as any verbal description, but can be used to verification of correctness of the project. There are methods especially adequate for software projects, like Z calculus.

1.2. Hardware Architecture and Scalability

In described case the main problem is how to reduce cost of development for on-board control system of the machine and to make it scalable according to customer needs. This aim can be achieved using modular architecture consisted of separated functional modules, which can be assembled in a variety of combinations to attain user needs. Modular system can be build using separate controllers as "bricks" connected together with bus (or network) as a communication channel – this architecture can be called as distributed. Distributed system architecture can be seen also as a network of cooperating nodes, which realize together a prescribed tasks.

Dedicated hardware would be certainly the best solution for system nodes, but it is very expensive and for this reason often not acceptable, especially for research and experimental systems. Usually designers give up system optimising to reduce the project cost and apply typical hardware "from the shelf ". The most popular and probably worst choice is using ordinary personal computers. PC is not a good solution because of its low immunity for hard environmental conditions, although CPCI standard overcomes this problem. There are a variety of modular open systems on the market, starting from PLCs up to VME and CPCI computers with a rich selection of I/O interfaces. These are manufactured according to industrial standards and can stand harsh environmental conditions of building site.

The same freedom of choice is to be concerned by network solutions, but the probably most appropriate bus for application with vehicles is CAN. It is supported by many manufacturers, as well of indus-

trial computers and PLCs as of sensors and actuators (recently also for power hydraulics).

In this paper I do not intend to consider details of the system architecture – it was discussed many times, I presented it already long ago in [1].

1.3. Operating System and Software Tools

In control systems fulfilling of specific requirements of a hard real time is essential – common operating systems like Windows are not proper solution. Furthermore, a system for excavator must be embedded and cannot use a magnetic disk storage (because of vibrations and shocks) – except for solid-state disks like these built using FLASH or SRAM memory. Only Windows CE and NT-Embedded can be used as operating system for embedded system, but are not suitable for fast control applications. There are many real time operating systems (RTOS) appropriate for control applications. The only disadvantage of them is the necessity to learn about their characteristics before you write any not elementary application.

Apart from the operating system there is a specific knowledge of I/O hardware required (interface of sensors and actuators) – drivers delivered by hardware manufacturer can simplify the problem.

Another problem for software designer is a choice of proper tools. Programmers often used to write control applications in assembly language. It was justified by relative low efficiency of early microprocessors and low code quality produced by compilers, but this way is time-consuming and programs are not portable between processor platforms. Nowadays a majority of real time applications are written in C or even in C++ and Java. The two latter languages can be carefully used only in powerful systems because of their possible run time overhead, caused by object inheritance and late bindings (by virtual methods).

PLC programming is often carried out using IEC 1131-3 programming tools and cross-development environment, but this technology can be utilized only if time dependencies are not very hard (cycle time lower than single milliseconds is difficult to achieve). The reason for relative low code efficiency is that program code is interpreted in a cyclic manner

1.4. Structural Programming

Structural programming consists in two fundamental principles:

- hierarchical top-down decomposition of the problem to be solved and
- modularisation, i.e. writing code for consecutive levels of decomposition using separate modules or functions.

Structural programming is not associated with any particular programming language, although high-level languages support better this approach than machine-oriented languages.

1.5. Software Layers

The concept of software layers enables the programmer to concentrate on specific algorithms, which can be isolated from lower level details. Moreover, the modification of any algorithm within this structure does not require changes in other layers. This approach we can find in system software structures. As an example we consider a processing of an I/O call in OS-9 real time operating system [2]:

- Any I/O call is directed to IOMan, which represents the higher level of processing. It establishes path, i.e. a connection between the application and the appropriate file manager and device driver.
- File managers perform the processing for a particular class of devices, such as disks or terminals. They deal with logical operations on the class of devices.
- Device drivers operate on a class of hardware. Operating on the actual hardware device, they send data to and from the device on behalf of the file manager. They isolate the file manager from hardware dependencies such as control register organization and data transfer modes, translating the file manager's logical requests into specific hardware operations.

Another good known example we can find in computer communications model known as Open Systems Interconnection Reference Model (OSI).

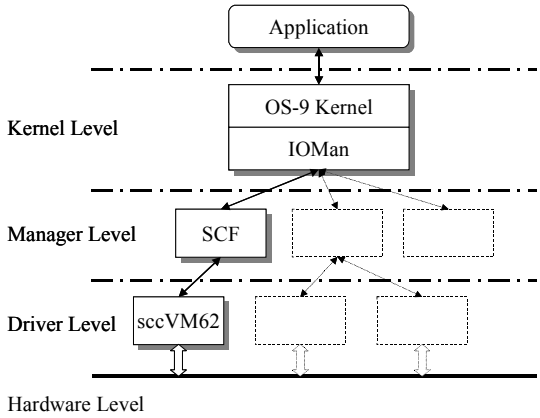

Figure 1. OS-9 I/O levels

Protocol layers in this model represent different levels of abstraction, starting from an application services and ending at physical interface to communication channel.

2. SYSTEM FOR EXCAVATOR

A system for hydraulic excavator is in several aspects similar to solution for any hydraulic heavy machine. The main difference in relation to others is more complex movement control. Project was based on following assumptions:

- an excavator should behave as autonomous, automatic (robotized) machine in a range of simple tasks including typical activities of digging a prescribed pit and loading output; sophisticated tasks could be programmed by operator or external (stationary) system using appropriate communications channel,
- system should support the machine operator in a case of manual control (operator can take the control over at any moment),
- system should supervise a machine status to avoid a potential overloading and reduce the probability of damage or failure,
- system should support optimising the work process using several criterions depending on soil type or / and operator decisions,
- for a research purpose system should provide data acquisition, i.e. to collect data from sensors and other internal data.

System is still under development, so some advanced functions are not fully implemented as yet.

2.1. Functions of the System

Functions of the system can be divided into four groups:

(a) movement control, work planning and strategy optimising,

(b) manual control support and man / machine interface,

(c) machine diagnostics and supervising,

(d) data acquisition and reporting.

Groups (a) and (b) are mutually excluded, though during automated control several aspects of operator interface are allowed to enable operator to supervise the excavator. Functions (c) and (d) are independent and executed concurrently to others.

Enumerated system functions form "vertical" or "perpendicular" division of software structure. Each of them is implemented using several separate processes executed in parallel. Some of these processes are essential for more than one system functions – they form layers of software structure.

2.2. Main Layers of Application Software

Application software can be divided into several layers, making up "horizontal" split of software. In a case of excavator system we obtain following fundamental layers:

- higher (planning, strategy) processing layer (HPL)
- man / machine communication layer (MMI)
- lower (executing) processing layer (LPL)
- machine abstraction layer (MAL)

If we assume as a reference, that the lowest system layer is made up of hardware (HW) and operating system services (OS), then directly above it we have machine abstraction layer. Aim of MAL is to allow processes from higher levels not to take care about details of machine and system construction. For example, if any process needs to know a pressure in some point of hydraulic system, it should not bother on details like sensor type, its characteristics, channel number, or even if this sensor is local to given system node or belongs to one of other nodes within a network. This layer consists of several sublayers, including interrupt handlers and network communications. MAL processes are strongly time dependent, so must be executed at highest priority level.

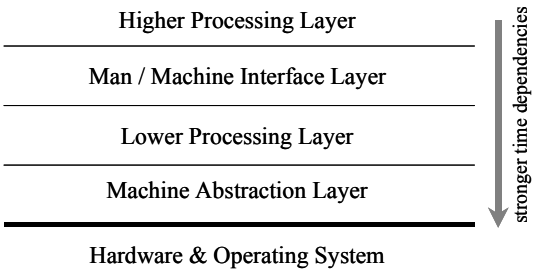

Figure 2. Software layers of the system

Lower processing layer is build by these processes, which are responsible for real time functionality but these dependencies are not so strong as in a case of MAL. LPL is a layer of simple movement control, diagnostics, data acquisition etc.

Man / machine communication, also called operator interface, is responsible for passing on the data to operator (local or remote), generating warnings and alarms. Joysticks pedals or switches are hardware part of this interface too. Operator can choose several options of displaying parameters or characteristics and using this interface can make decisions and even program the machine itself.

LPL processes are a must for an automated machine, but "intelligence" of the machine is contained within higher processing layer. HPL processes needs more processing power but are not so strongly time dependent as any processes from lower levels.

2.3. Hierarchy of Movement Control Processes

To illustrate the above discussed layers we can shortly describe function 2.1.(a) – movement control and movement planning.

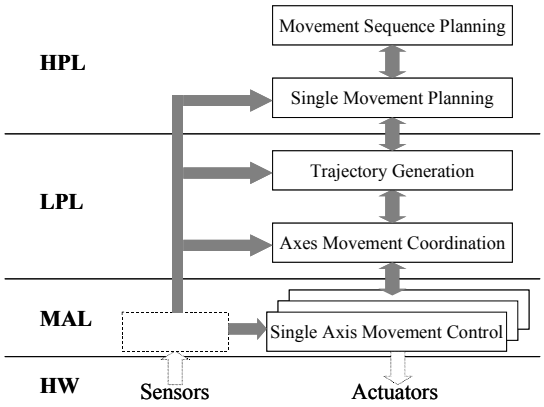

Figure 3 Hierarchy of movement control and planning processes

At the lower level single axis is controlled based on prescribed position and / or motion velocity. These three simple controllers are in real system implemented as single process. Combined axis movements are resolved by coordination process, which becomes trajectory description at its input. If our system is not very "intelligent", trajectory generation is predefined within a library and chosen by operator. In this case system software does not contain code of higher processing layer (HPL).

In a case of robotized excavator a trajectory is generated by movement planning processes, according to strategy and given limitations. Simple movements are sequenced (joined) by a movement sequences planning process. This part of system is not implemented yet. It must be preceded by detailed examination and description of many movement characteristics and strategies and requires a lot of research tests. As a first result we expect to obtain a knowledge base containing formal algorithmic descriptions for a variety of possible movements.

3. CONCLUSIONS

Computer system for an "intelligent" excavator is a sophisticated one. The only way for designing it is a systematic approach taking into consideration several aspects:

- hard real time requirements, because it has to control complex movements,
- scalability – according to needs and requests,
- cooperation with other machines and systems on the plant,
- modifiability and extendibility – because we should apply new ideas avoiding to start project from the very beginning.

Partial results can be useful in the future only if they are precisely specified and make no use of exotic and not expandable or not portable solutions.

REFERENCES

[1] Dobrowolski H. *On-board computer system for an excavator* Proc. of ISARC (Int. Symposium on Automation and Robotics in Construction) XII, Warszawa 1995, vol.I pp.143-149

[2] *OS-9 Technical I/O Manual, Version 3.0* Microware Corporation 1993