PROGRAMMING LANGUAGE CREATION FOR CONTROLING INTERNAL TRANSPORT DEVICES

Many programming languages have been developed during computers history, and new languages are still created. One way dominating in thinking about using computers is transferred on programmable control devices. The proposition of specialized programming language with instruction structure based on integrity criteria, for controlling a modular conveyer in an internal transport is presented in the paper.

1. INTRODUCTION

Contemporary development of programming languages has already been lasting more than 60 years. Its effect is creation of many programming languages. New languages, with higher possibilities, keep being created. The characteristic feature that is noticeable in charts showing the resulting relations between particular programming languages [e.g. 9, 10, 11, 12, 13] is their origination from one source, being the FORTRAN language. It means that in development of high level programming languages one way is dominating in thinking about using computer and its programming. Also the concept of Plankalkul [1] language is considered as the precursor of this way, although it is not known how it would be implemented.

The driving force behind development of programming languages was the necessity to take into account the new interaction objects, appearing every now and again. In case of universal languages these were usually subsequent mathematical objects (e.g. real numbers, tables, lists), and in case of specialized languages – specific properties of, effecting objects.

What in the programming languages is inherited from a common ancestor is the way of formulating commands to computer. Traditional formulation of mathematical equations has been complemented with instructions similar to natural language of the type: „*if*”, „*for*”, „*while*”, „*go to*” and their various modifications. They have assumed a form of separate control statements.

Since placing the program instructions complete with data in the same computer memory [8], the program resides permanently in the memory and each statement is assigned to a specified place in the space. In low level languages there are obvious computer restrictions resulting from this fact. They are treated as weaknesses and hidden in high level languages. A convention applied in mathematics has been assumed in them that a number or a variable is accessible as of the moment of writing it, or – in more general terms – since thinking it. Moreover, it is assumed that numbers of the same value are non-distinguishable, although in the computer they differ in place of storage and in time of occurrence. If it were not for this conviction maybe some other programming languages would be created, treating physical distinction of computers as a chief asset trump card of the device that came into being.

Also treating the computer as a mathematical machine has had an adverse influence on way of executing the program. One of the consequences thereof was treating not only data but statements as being always accessible, which certainly is fulfilled in relation to mathematical objects. However, in technical terms, such an assumption is not complied with, therefore for hiding this „defect” the program counter has been used, to which statements are collected sequentially from the memory in order to execute them. This is the basic bottleneck in the computer. It might have gone imperceptible for many years because data transmissions between the processor and the memory lasted longer than execution of the statement. This defect of common processing of programs and data is at present mitigated by anticipating buffering both of data and of program statements.

Due to the fiasco of attempts at imitating a natural language it can be assumed that the future programming languages shall be tailored to meet user’s specific needs. They can be even created suitably to current programming needs of a specific task [6].

This direction can be in a natural way preferred in the area of programmable control devices. However, as a result of monocultural development of programming languages, this is the application area of traditional programming languages.

It is not assumed, indeed, at programmable controlling of equipment that a given device is a mathematical machine, but programming methods applied in relation to these machines are adopted. In particular, an assumption about sequential execution of program instructions by the processors is adopted.

2. STRUCTURE OF THE PROGRAM INSTRUCTION

The practical objective of programming is causing the execution of possible operations of the device in the desired sequences. Use of the programming language causes that it is possible to apply commands activating the device functioning. Any computer operations must be appropriately coordinated, because computer as a technical device does not perform operations on abstract numbers but on places of specific space at specific time. This necessity of coordination does not, however, result from the device structure, because it can perform any operations in any sequence, but it is the demand resulting from its user's needs.

There are three independent categories of space, time and the operation objective that are applying to the programmable control device. They concern objects on which calculation operations are performed and are applicable to the programming language commands because it is by means of this language that the objectives of the computer use are implemented. On their basis the logical lattice[1] can be created, linking the notions concerning that system. It constitutes the source of criteria that should be complied with by the system in order to maintain its integrity [7]. It means that the commands of the programming language must ensure accessibility, transparency and connectivity in relation to the elements, on which influence is exerted, so as to enable implementation of the system operation objectives.

The accessibility criterion is implemented through making accessible the objects, on which relevant mathematical operations are performed. It can be executed by introducing the general clause of making the resources of programable control device accessible, and the clause would specify „*where*" and „*when*" a given command is to be executed. It can be given name „*at*". It gives a range of objects towards which an action is to be taken.

In addition, the device should carry out a task associated with the objective for which a given system has been created. On this basis two clauses can be introduced. One - linking the action objective with space, and the other – linking the action objective with time. The first one of them is used for specifying „*where*" and „*what*" is to be executed and it can be given a name *"as"*. It is associated with conditions of the device functioning.

The other one of these additional clauses is used for specifying „*when*" and "*what*" is to be executed. This category of statement can be given name „ *this"*.

Equivalents of these statement groups can be easily found in the existing programming languages. The first group includes statements of the „*for*" type, the second one includes statements of the „*if*", „*case*", „*while*" type, and the third group - of the „*go*" type, as well as the function and procedure calls.

In the period when first calculating machines were built, attention did not have to be drawn to these criteria because these are complied with for mathematical objects by definition. The only problem was formulating commands to the machine. As a result, compilers and interpreters of artificial programming languages have been developed. In each of them the natural language phrases have been used, mainly in order to facilitate the program writing.

3. EXAMPLE OF PROGRAMMING LANGUAGE CREATION

As an example, a programming language for controlling a modular conveyer in an internal transport has been created. Traditional conveyer type transport consisted in building a space with fixed structure of conveying equipment. With the applied design methods [3] a facility of considerably higher capacity was obtained than currently needed. It was not a defect of these systems because due to high installation costs they were only used in a situation of a long period of their use. Owing to this fact they were adapted to potential increase in transport needs in the future.

At present, however, the conveyer type equipment items are built in form of conveyer modules [2]. Due to this reason, costs of using the conveyer type transport tend to drop. Also installing such transport type is getting simpler and the transport system can be easily – and thus frequently – modified.

However, in view of a possibility of such transport type becoming widespread, it is necessary to ensure easy controlling of a set of modules making the transport system. Such a control is at present feasible by means of programmable control devices.

The system model is realized using identical elements. Each of them is matched in reality by a specific device having appropriate characteristics, making it impossible to execute all the commands.

[1] It was Ryszard Sobol whom I owe the idea of using the logical lattices to link the notions.

When creating the programming language for controlling conveyer modules it was assumed that instructions concerning a given module shall be assigned to it, in such a sequence, in which the module functions shall be performed. The element initiating the module operation is the load unit, which reaches the module and it is to be transferred by means of that module. Thus the instructions are accessible via the load unit, in relation to which a given module is to execute them. At the same time the connectivity between modules is necessary, owing to which it is possible to:

1) send the instructions assigned to particular modules,
2) execute instructions in relation to other modules activated by the load unit transferred on a given module.

This connectivity can be implemented in many ways worked out within the scope of the computer engineering development. At the same time it is possible to:

a) exert remote influence from one module on functioning of another module,
b) activate instructions in the module prepared for execution during activation of another module,
c) send the instructions for execution to another module which is activated by a given module, etc.

Despite assigning the program instructions to particular modules, considering technical aspects, all instructions can be stored in one memory and operated by one processor like in a traditional computer. It is only essential that instructions should be activated by the load units moving across modules of the transport system. However, it is scattered programming that is natural for such control [4].

The controlling of modules is carried out by means of the following set of commands concerning the conveyer system module (movement direction is in relation to flat picture of the conveyer system):

- *St* – start,
- *Sp* – stop,
- *in* – introducing the load unit,
- *lt* – movement leftwards,
- *rt*– movement rightwards,
- *up- m*ovement upwards,
- *dn*– movement downwards,
- v* - module movement speed (instead of asterisk – number specifying the speed).

In the programming language syntax the „name” specifies the module name and “v*” - the conveyer module movement speed, where instead of asterisk (“*”) the number specifying the speed should be given. The term „scatter” specifies the list of values with preset probability of occurrence e.g. E(300)/0.3; 200/0.7.

Instructions are assigned to objects according to content of „*at*” sections. By means of an address an instruction can be assigned to any module. Movable objects are identified according to the target place where they are to be transported. Instructions with empty sections „*as*” and „*this*” are executed on a one-off basis at the initial moment after starting the program.

The presented programming language has been used in the simulation model of conveyer belt for transferring load units in a warehouse. By means of this model the controlling of modular conveyers can be programmed. As an example, a system has been modelled with a distinguished segment, on which priority load units are transferred with increased speed. Regular load units are within this time transferred along side route.

control program instruction are written as sequential records in the file. Then there are assigned to particular modules of the conveyer.

Load units on the conveyer are introduced from three inputs and directed to three outputs. Output through module „z1” is at the same time preferenced and load units directed to that output are transferred with increased speed on all modules of the conveyer belt. The modules are functioning with increased speed when the priority load unit is placed on them. If in this time the regular load units are placed on them, these are also displaced with increased speed. Regular load units preceding the priority load units are directed to byway bypassing the main segment of the conveyer belt, so as not to impede the movement of the preferenced load units.

Because it has been assumed that the default direction of the module functioning is the direction along the longer side rightwards or downwards, in the beginning the instruction are executed changing the

direction of functioning of the modules, whose default movement direction is different than the planned direction.

Intervals between load units in particular streams are described, respectively, by uniform distribution on range (40, 120), uniform distribution on range (60, 140) and by exponential distribution of intensity 1/100. Each of the streams consists of load units directed to different outputs with preset probability. For example, the load unit introduced on the module „w1" is directed with probability 0.1 to the preferenced module „z1", with probability 0.4 – to the module „z2" and with probability equal to 1 – 0.1 – 0.4 = 0.5 – to the module „z3".

In case of all the modules the priority load units directed to module „z1" increase speed of their functioning (e.g. instruction „@AA v2 #z1"). This speed is reduced after the preset time (e.g. instruction „@AA v1 #z1 +20") or after activating the adjacent module (e.g. instruction „@t1 v1 #z1" assigned to module „BB").

Load units are directed to respective output modules by changing the direction of functioning of modules performing function of switches (e.g. instruction „@s1 dn #z1", „@s1 lt #z2", „@s1 lt #z3" cause respective directing of load units to modules „z1", „z2" and „z3".

Moreover, the priority load units control the devices preceding them in such a way that other load units are directed to byway. For instance, the statement „@BB dn #z1", causing directing of the module „BB" downwards ("dn") is activated after entering the load unit on module „AA". In order to assign this statement to the module „AA", name of this module is placed before the instruction. This instruction is activated only by load units directed to the module „z1", which is written in section "*as*" giving the activation condition.

4. CONCLUSION

The presented modular system of conveyers is a relatively simple system for programmable controlling. The model programming language comprises the simplest operations of the devices being controlled. It can be enriched by any other commands concerning equipment items and also by more complex types of variables and constants.

In case of controlled devices it is obvious that both data and statements can be distributed at different places of the space. Therefore, for correct functioning of the system it is necessary to implement the integrity criteria, which in case of various equipment devices may not be easy. At the same time there must be ensured accessibility, transparency and connectivity in relation to both data and instructions. Thus different ways of data and statement flows can be considered than those in traditional computers.

Thus a possibility arises to create the programming language for a given class of equipment and even for a specific controlling task. For implementing the equipment controlling program there can be used directly specialized hardware systems.

BIBLIOGRAPHY

[1] Bajer F.L., Wössner H.: *The "Plankalkul" of Konrad Zuse: A Forerunner of Today's Programming Languages*, Mathematisches Institut der Technischen Universit/it Mtinchen, Communications of the ACM, Number 7, Volume 15, July 1972.
[2] Bachorz P.: *Modułowe przenośniki*. Logistyka a Jakość 3/2005, s. 68-69.
[3] Fijałkowski J.: Transport wewnętrzny w systemach logistycznych – wybrane zagadnienia. OW PW, Warszawa 2000.
[4] Grochowski L.: Rozproszone systemy informatyczne, Warszawa: DW Elipsa, 2003.
[5] Grogono P.: The Evolution of Programming Languages, Department of Computer Science, Concordia University, Montreal, Quebek 1999.
[6] Jadud M.C., Chenoweth B.N., Schleter J.: *Little Languages for Little Robots,* Keele University, UK, PPIG 2003, URL: http://www.jadud.com/people/mcj/files/2003-PPIG-lllr.pdf .
[7] Okulewicz J.: *Kryteria analizy systemów transportowych*. Międzynarodowa Konferencja Naukowa "Transport XXI w." Warszawa 2004, s. 479-488.
[8] Targowski A.: Informatyka – modele systemów i rozwoju. PWE Warszawa 1980.
[9] URL: http://hopl.murdoch.edu.au/, The History of Programming Languages by Diarmuid Pigott.
[10] URL: http://www.levenez.com/lang/, Jun 2007.
[11] URL: http://www.cs.iastate.edu/~leavens/ComS541Fall97/hw-pages/history/, History of Programming Languages by Com S 541 class of 1997.
[12] URL: http://www.princeton.edu/~ferguson/adw/programming_languages.shtml, The History of Computer Programming Languages by Andrew Ferguson (*Last Modified: 05-Nov-2004).*
[13] URL: http://www.epemag.com/zuse/copyrite.htm .