Barbara JÓŹWIAK*, Józef OKULEWICZ*, Barbara WĘGRZYNOWICZ*

PROGRAMMING OF CONTROLLING OF A CONVEYOR TRANSPORT SYSTEM

Abstract: There is one way of programming language creation which prevails in developing computer applications. As a result, no uniform computer-supported method of control of conventional devices has been established. A quite different method of programming language creation based on system integrity postulates is proposed in the paper. As an example, a programming language for a control of a modular conveyor system in the internal transport has been created. A computer simulation model of the internal transport system is used to confirm applicability of such a language.

Keywords: internal transport, conveyor, integrity, programming.

1. Introduction

Continuous transport systems are created to support the kind of cargo (e.g. energy, fuels, dry goods etc.) or prolonged and stable transfer of discrete cargo units (e.g. production lines, automatic warehouses, etc.). For conveying load units, the stationary conveyor type equipment items are used, instead of movable means of transport.

The common feature of continuous transport equipment items is that the control of conveying is carried out through exerting an impact on the conveyor type equipment items. Owing to this fact, not only the system structure, but also the control of the movement of the load units is simplified. However, in the case of any change in the load unit conveyance route, it is necessary to introduce modifications into the system structure.

The conveyor type transport in a warehouse comprises a stationary structure of conveying equipment and it is prepared for use over long periods of time. In terms of the applied design methods (Fijałkowski 2000), such a transport system is usually of considerably higher capacity than actually required. But is is just to this fact that the system is adaptable to a potential future increase in transport needs.

* Warsaw University of Technology, Warsaw, Poland

The conveyor type transport system is a complex technical system satisfying the logistic needs of an enterprise. It is usually equipped with a control system installed by the conveying equipment producer. Usually, also the change of the system structure or altering of its functioning mode requires participation of the producer or a specialist servicing company.

At present, more and more often, conveyor type equipment items are built in the form of conveyor modules (Bachorz 2005); consequently, costs of operating the conveyor type transport tend to drop. Also the installation of such type of transport is getting simpler and the transport system can be easily – and thus frequently – modified.

However, as such transport type may well become common, it is necessary to ensure easy control of a set of modules comprising the transport system. Such a control is at present feasible by means of programmable control devices, with which the contemporary technical arrangements and systems are equipped. The problem is the selection of the suitable programming language supporting simpler programming than with use of the internal language of a processor.

One way of carrying out this task is using the existing high level programming language and complementing it with relevant procedures enabling the programming of a specific equipment item. The second one is creating a programming language customised to the capabilities of the device controlled. However, the implementations of the latter way usually follow the model of the existing programming languages.

These, in turn, during more than 60 years of their development, have been based in most part on their source: the FORTRAN language, (e.g. URL 2004, URL 2007). It means that in the development of high level programming languages there is one way which prevails in thinking about using and programming a computer.

This to a large extent results from adjusting programming languages to the capabilities of computers, which in turn are adjusted to properties of the existing programming languages. "*While it is perhaps natural and inevitable that languages like Fortran and its successors should have developed out of the concept of the von Neumann computer as they did, the fact that such languages have dominated our thinking for twenty years is unfortunate. Unfortunate because their long-standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer far greater intellectual and computational power.*" (Backus 1978). As a result, there has occurred a kind of freezing of both the computer structure development and the programming methods (Gabriel *et al.* 1986).

Also treating the computer as a mathematical machine has had an adverse influence on a way of executing the program. Since placing the program statement complete with data in the same computer memory (Targowski 1980), the program resides permanently in the memory and each statement is assigned to a specified place in the space. The obvious resulting computer restrictions are treated as its weaknesses (or even defects) and hidden in high level languages. Instead of this, the physical distinction of computers should be treated as a chief asset (a kind of a trump card) of the device that came into being.

This direction could be in a natural way preferred in the area of programmable control devices. In this case it is not assumed, indeed, that a given device is a mathematical machine, but programming methods applied in relation to these machines are adopted. In particular, the assumption of sequential execution of program statements by the processor is followed.

As a result of such monocultural development of programming languages, also in the scope of programmable control devices, the traditional programming languages are applied, supplemented by relevant statements or procedures. Due to overcomplexity of programming languages used to relatively simple control tasks, the program creation is simplified by graphical aiding of the programming (e.g. ISAGRAF).

An alternative solution to this problem could be the creation of programming language adjusted to the controlled device requirements, without the burden of its other potential applications. Such a special programming language is described in this paper. It supports exerting an impact on the internal transport equipment items, that is conveyor modules.

When creating a programming language for controlling a conveyor modules, it was assumed that statements concerning a given module should be assigned to it in such a sequence in which the module functions are to be performed. The element initiating the module operation is the statement activating the module, or the load unit that reaches the module. Considering the technological aspect, all statements can be stored in one memory and processed by a single processor like in the traditional computer.

2. Structure of the program instruction

The objective of programming is to force the required sequence of operations of a given device. It may be not only the computer, but also other devices connected to it. Using the relevant programming language statement, such device functioning can be controlled.

Because the commands of the programming language must ensure the implementation of the system operation objective, while formulating another approach to programming, one might take advantage of general properties of the systems.

There are three independent logical categories of space, time and the operation objective that apply to a 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. Based on them, the logical lattice can be created, connecting the notions referring to that system (Fig. 1).

This forms the source of postulates that should be complied with by the system in order to maintain its integrity (Okulewicz 2004). The commands of the programming language must ensure accessibility, transparency and connectivity in relation to the influenced elements, so as to enable the implementation of the system operation objective.

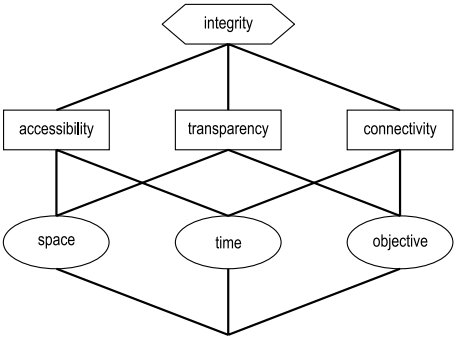


Fig. 1. *Structure of systemic notions*

The relevant control instructions should include three sections. In order to implement the accessibility postulate in the statement, the range of objects towards which an action is to be taken is given, thus specifying "*where*" and "*when*" a given action is to be executed. This part of a statement forms a section named "AT". The next part of the statement forms a section named "AS" and it specifies "*where*" and "*what*" is to be executed. Moreover, it is specified "*when*" and "*what*" is to be executed, for which the third part of statement is responsible, forming the "THIS" section. These postulates are fulfilled towards mathematical objects by definition. Equivalents of these groups of statements can be found in the existing programming languages. The first group includes statements of the "*for*" type, the other one - statements of the "*if*", "*case*", "*while*" types, and the third one - statements of the "*go*" type. This third type could be removed from a programming language, because of completeness of these postulates.

Based on these three components of the statement, the programming language has been developed (Fig. 2) (Okulewicz 2007). This is a kind of programming using anonymous objects and known nodes, belonging to one of the three classes of possible programming methods (Okulewicz 2005). According to this classification, conventional programming languages use known objects and known nodes in a static mode.

Considering the simple set of control commands, particular statement sections are recognised based on the initial character, i.e. "" or "#". Not all the optional meanings of the proposed language syntax were used.

Instructions are assigned by default to modules according to the content of "AS" sections. However, a statement can be assigned to any module by means of an address preceding that address statement in form of the module name preceded by character ":".

Statements with empty sections "AS" and "THIS" are executed on a one-off basis at the initial moment after starting the program. The object statements starting with "#" are activated at each starting of the module to which they are assigned. The module statements starting with the character "" of nonempty sections "AS" and "THIS" are activated by objects reaching the modules.

<program> ::= <line>{<line>}
<line> ::= <statement> | * *text*
<statement> ::= <AT> <command> <AS> <THIS>
<AT> ::= @<name> | @<name>:<name> | #<name>
<AS> ::= <empty> | #<name> | #<scatter> | @<name>
<THIS> ::= <empty> | <character><value>
<name> ::= * | *text*
<command> ::= ▶ | ■ | 🡐 | 🡒 | 🡑 | 🡓 | 🡕 | v* | q*
<character> ::= <empty> | + | = | @
<value> ::= *number* | <distribution> | <scatter>
<distribution> ::= <type>(*number*{, *number* })
<type> ::= W | N | U | E
<scatter> ::= *number*/*probability*{;*number*/*probability*} {;*number*}

Fig. 2. *Syntax of the programming language*

Movable objects are identified according to the name assigned to them at the moment of creation. It may mean the target place where the objects are to be transported or any label.

3. Modelling a conveyor control system

In order to check the applicability of the elaborated programming language for the control of a conveyor system, the simulation model of a conveyor system has been worked out. It enables: creation of spatial arrangement of a conveyor system, execution of control instructions of a conveyor system and programming of conveyor system operation.

The conveyor system model is realised as a set of rectangles drawn on the computer screen. Each of them is in reality matched to a specific device with appropriate characteristics, thus enabling the execution of the relevant commands. The model description is included in the file and it contains the structure description and the control description. In any line of the structure description, a comment preceded with character "*" may be placed.

In order to place the structure of the conveyor module system on the screen, it is necessary to give coordinates of each object and kind of module (-1 – input, -2 – output, 0 – any) in the model description. At each of the output modules, a table is displayed, containing number of load units, the average time of a load unit stay in the system, and standard deviation of time of load units staying in the system.

Default direction of the module functioning is the direction along the longer side rightwards or downwards. In order to establish another movement direction on the conveyor module, it is necessary to execute the statement changing the direction of the module functioning.

The program screen includes a panel supporting the preparation and execution of the statements intended for introducing the load units. For each load unit it is necessary to specify the input module where the load unit should appear, and the output module where the load unit should reach. Intervals of time between load units can be specified in the program only. The bottom part of the program screen also includes a panel supporting the preparation and execution of statements controlling the module states. The conveyor module may be in a standstill condition, when it is not capable of shifting the load units. After start, the load units can be transferred at a normal or increased speed. Moreover, the direction of load units transferred on the conveyor can be controlled.

The module control can be carried out by means of an appropriate set of commands that can be introduced, for example mnemonically, with use of textual or graphic symbols (Tab. 1). Types of symbols can be mixed in a single program.

Table 1. *Set of commands controlling the conveyor modules*

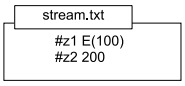
Name\Symbol	Letter	Letter	Graphic	Character
start	St	St	▸	[¿
stop	Sp	Sp	■	[ ]
movement leftwards	le	lt	←	– <
movement rightwards	pr	rt	→	– >
movement upwards	go	up	↑	—
movement downwards	do	dn	↓	—v
introducing the load unit	we	in	↘	/o
module movement speed	v*	v*	v*	¿*
object priority	q*	q*	q*	!*

**Instead of an asterisk, a number from 1 to 9, specifying the speed or the priority, respectively, is given*

Fragment of a statement concerning the appearing of load units can be recorded in a file. Sections "AS" and "THIS" are recorded, and they are executed one by one at the initial moment upon starting the program. The first one includes the target module name and the other one – the moment of introducing the object. After reaching the file end, the statement is passed over. Owing to this fact, it is possible to model the determined schedule of transferring the load units (Tab. 2).

A clock is displayed on the screen and according to its indications all events are carried out. The clock can be started and stopped and its cycle can be speeded up or slowed down. The time unit is the second. There is a control list at the screen side, where the control instructions executed at the moment are displayed.

Table 2. *Meaning of program instructions*

Instruction	Comment
@* St	Starting of all the modules
@(m1,m2) St	Starting of modules m1 and m2
@*(m1,m2) St	Starting of all the modules apart from m1 and m2
@A dn	Changing the module A functioning direction to downwards at initial moment
@A dn 200	Changing the module A functioning direction to downwards at the 200 seconds moment, i.e., at 00:03.20
@A dn +200	Changing the module A functioning direction to downwards after the 200 seconds time from the moment of reaching that module, i.e. after 00:03:00
@A dn #Z	Changing the module A functioning direction to downwards by the object reaching that module, directed to module Z
@B:A dn #Z	Changing the module A functioning direction to downwards by the object reaching the module B, directed to module Z
#A in #B +R(40,120)	Load units from module A are directed to module B and they appear in time intervals of uniform distribution on the interval (40, 120)
#A in #B/0.3;C/0.7 +E(100) #A in #B/0.3;C+E(100)	Creation of load units on module A, which are directed to module B with probability 0.3 and to module C with probability 0.7; the load units appear in time intervals of exponential distribution of mean 100
#w1 in "stream.txt" 	Reading from file "stream.txt" of the target module name and the moment of appearing the objects

4. Examples of models

Two models of conveyor systems are presented as examples of program structure.

4.1. Straight section of conveyor

Load units appear on module "w" in time intervals of uniform distribution on interval (40, 120) and they are transferred on module "A" to module "z" (Figs 3, 4).

```
* straight section of conveyor
[w 105 55 30 40 -1 in
[A 105 85 400 40 0
[z 105 485 40 40 -2 out
@w rt
@* St
#w in #z +U(40,120)
```

Fig. 3. *Model control program*

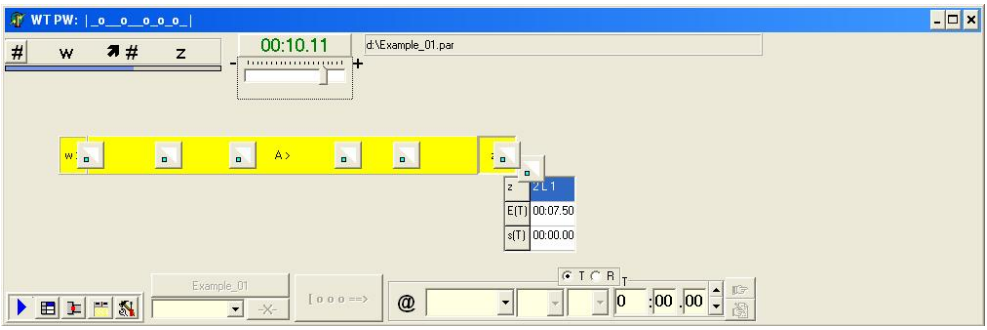


Fig. 4. *Diagram of a simple conveyor*

4.2. Conveyor system with a siding

In the internal transport system, a segment has been distinguished on which priority load units are transferred at an increased speed. Regular load units are within this time transferred along a side route.

Load units on the conveyer are introduced from three inputs (w1, w2, w3). Each of the streams consists of load units directed to different outputs (z1, z2, z3) with preset probabilities. Output through module "z1" is at the same time preferenced and load units directed to that output are transferred at an increased speed on all modules of the conveyor belt. The modules operate with the increased speed when a priority load unit is placed on them. If in this time regular load units are placed on them, these are also transferred at the increased speed. Regular load units preceding the priority load units are directed to a byway bypassing the main segment of the conveyor belt, so as not to halt the movement of the preferenced load units (Fig. 5, Tabs 3, 4).

Intervals between load units in particular streams are described by uniform distribution on interval (40, 120), uniform distribution on interval (60, 140) and by exponential distribution of intensity 1/100, respectively.

In case of all the modules Priority load units directed to module "z1" increase the operational speed of all modules (e.g. statement "@A v2 #z1"). This speed is reduced after the preset time (e.g. statement "@A v1 #z1 +20") or after activating the adjacent module (e.g. statement "@t1 v1 #z1" assigned to module "B").

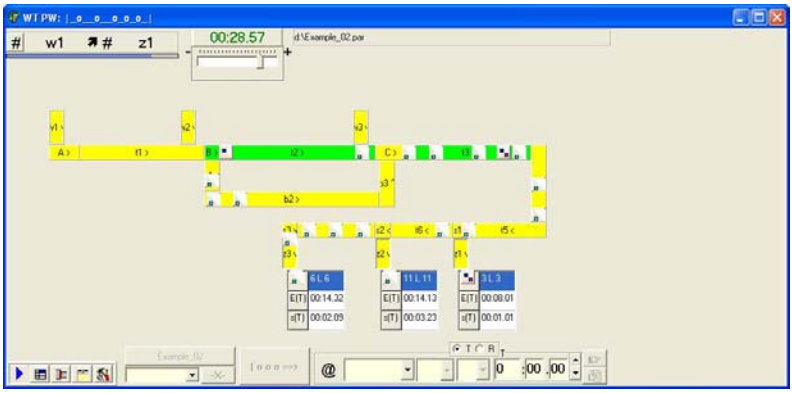


Fig. 5. *Diagram of conveyor control with a siding*

Table 3. *Linear record of conveyor modules controlling program*

@w1 ↓	#w1 ↗#z1/0.2;z2/0.4;z3 + U(40,120)	@B v2 #z1	@C v2 #z1
@w2 ↓	#w2 ↗#z1/0.1;z2/0.5z3 + U(60,140)	@B v2 #z1 + 20	@C v1 #z1 +20
@w3 ↓	#w3 ↗#z1/0.1;z2/0.6;z3 + E(100)	@B:t1 v1 #z1	@C:B →#z1
@t5 ←	@* ▸	@t2 v2 #z1	@t3 v2 #z1
@t6 ←	@w1 q2 #z1	@t2 v1 #z1 + 50	@z1 v2 #z1
@s1 ←	@w2 q3 #z1	@s1:t5 v1 #z1	@z1 v2 #z1 + 10
@s2 ←	@w3 q4 #z1	@s1 v2 #z1	@t4 v2 #z1
@s3 ↓	@A v2 #z1	@s1 v1 #z1 +10	@t3 v1 #z1
@C →	@A v1 #z1 + 20	@s1 ↓#z1	
@t7 ←	@A:B ↓#z1	@s1 ←#z2	
@z1 ↓	@t1 v2 #z1	@s1 ←#z3	
@z2 ↓	@t5:t4 v1 #z1	@s2 ↓#z2	
@z3 ↓	@t5 v2 #z1	@s2 ←#z3	
@b3 ↑	@B →#z1	@s3 ↓#z3	

Load units are directed to respective output modules by changing the direction of operation of modules performing the function of switches (e.g. statement "@s1 ↓#z1" causes the respective directing of load units to module "z1").

Moreover, priority load units control the devices preceding them in such a way that other load units are directed to a byway (for instance, the statement "@B ↓#z1", causing directing of the module "B" downwards is activated after entering a load unit on module "A" because it is assigned to this module).

Table 4. *Assigning the statements of the program to conveyor modules*

w1		*s1*	*z1*
@w1 ↓ #w1 ↗#z1/0.2;z2/0.4;z3 + U(40,120) @w1 q2 #z1		@s1 ← @t5 v1 #z1 @s1 v2 #z1 @s1 v1 #z1 +10 @s1 ↓#z1 @s1 ←#z2 @s1 ←#z3	@z1 ↓ @z1 v2 #z1 @z1 v1 #z1 +10
w2		*s2*	*z2*
@w2 ↓ #w2 ↗#z1/0.1;z2/0.5;z3 +U(60,140) @w2 q3 #z1		@s2 ← @s2 ↓#z2 @s2 ←#z3	@z2 ↓
w3		*s3*	*z3*
@w3 ↓ #w3 ↗#z1/0.1;z2/0.6;z3 +E(100) @w3 q4 #z1		@s3 ↓ @s3 ↓#z3	@z3 ↓
A	*B*	*C*	*b3*
@A v2 #z1 @A v1 #z1 +20 @B ↓#z1	@B →#z1 @B v2 #z1 @B v1 #z1 +20 @t1 v1 #z1	@C → @C v2 #z1 @C v1 #z1 +20 @B →#z1	@b3 ↑
t1	*t2*	*t3*	*t4*
@t1 v2 #z1	@t2 v2 #z1 @t2 v1 #z1 +50	@t3 v2 #z1	@t4 v2 #z1 @t3 v1 #z1
t5	*t6*	*t7*	*central*
@t5 ← @t4 v1 #z1 @t5 v2 #z1	@t6 ←	@t7 ←	@* ▸

5. Conclusions

The presented modular system of conveyors may in a relatively simple way be submitted to programmable control. The model programming language comprises the basic 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. However, this requires creation of a new programming language for control, in accordance with proposed method.

Thus a possibility of creating a programming language for a given class of equipment and even for specific control task can be considered. Owing to this, program control of devices could be easily carried out by users of these devices.

It would be particularly useful in the case of IT systems used for the logistic needs, because nowadays, after purchasing a program, the program user has influence neither on the set of installed functions nor on the way of their implementation. Thus the possibility of the user programming the system by himself enables the user to satisfy customers' all requirements at any time. This would be a return to

the very beginning of computers applications, when a user had a system at his own disposal. To this end, he should have at his disposal an appropriate programming language customised to the problems that the user has to solve himself within the system. For software companies it is a challenge to develop adequate solutions to address customers' requirements, thus enabling the customers to simply program the operated IT system by themselves.

References

Bachorz P. 2005. *Modułowe przenośniki.* Logistyka a Jakość, 3, pp. 68–69.

Backus J. 1978. *The history of FORTRAN I, II, and III*. ACM SIGPLAN Notices, Vol. 13, No. 8, August 1978, pp. 165–180.

Fijałkowski J. 2000. *Transport wewnętrzny w systemach logistycznych – wybrane zagadnienia*. OW PW, Warszawa.

Gabriel R.P., Steele G.L. Jr. 1986. *What Computers Can't Do (And Why)*. Lisp and Symbolic Computation (LASC), Vol. E(1)/3.

Okulewicz J. 2004. *Kryteria analizy systemów transportowych*. Międzynarodowa Konferencja Naukowa "Transport XXI w.", Warszawa, pp. 479–488.

Okulewicz J. 2005. *Metoda modelowania użytkowania współzależnych systemów*. VIII Konferencja Logistyki Stosowanej, "Wybrane zagadnienia logistyki stosowanej", Zakopane (Polska), PAN KT, 2, pp. 276–283.

Okulewicz J. 2007. *Programming language creation for controlling internal transport devices*. [in:] Advances in transport systems telematics, Silesian University of Technology, Katowice, pp. 97–105.

Targowski A. 1980. *Informatyka – modele systemów i rozwoju*. PWE, Warszawa.

ISAGRAF – *Computer Aided Software Engineering Workbench For Open PLCs And Industrial Computers*, Users guide, CJ International.

URL 2004. `http://www.princeton.edu/~ferguson/adw/programming_languages.shtml`, The History of Computer Programming Languages by Andrew Ferguson, (Last Modified: 05-Nov-2004).

URL 2007. `http://www.levenez.com/lang/`, Jun 2007.