An alternative approach to the development of industrial systems

Applications

An Alternative Approach to the Development of Industrial Systems

R. Overwater
Faculty of Mechanical Engineering, Eindhoven, University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

K.J. Vegter
Foxboro Nederland N.V., Koningsweg 30, 3762 EC Soest, The Netherlands

The increasing size and complexity of today's industrial systems calls for the use of structured approaches during development of such systems. A structured approach comprises both a way of thinking and tools to support this way of thinking. The power of such an approach depends on the extent to which the way of thinking suits the problem. This paper describes an alternative way of modelling industrial systems. The approach recognizes the fact that industrial systems consist of interacting processes which operate simultaneously and is referred to as the "Process Interaction Approach". A set of structured tools based on this understanding has been developed for the design and analysis of industrial systems, including process control systems, application software a.o.. The use of these tools is illustrated on a bottling system.

It is believed that the Process Interaction Approach offers natural and powerful concepts, which can be fruitfully applied to the development of industrial systems.

Keywords: Process interaction approach, Industrial systems, Systems development, Systems specification, Modelling, Discrete event simulation, Control.

1. Introduction

Modern industrial process control systems are typified by increasing size and complexity. A number of reasons can be identified for this trend, such as higher quality and safety demands, environmental requirements, integration with functions like planning and scheduling, and cost control requirements. A structured approach to the development of process control systems is consequently a necessity to be able to manage their increasing complexity.

By applying the appropriate approach, a system can be divided into meaningful and understandable parts, between which the relationships are clearly defined. This not only greatly facilitates system development, but also improves the system's quality.

R. Overwater received his Master's degree in Mechanical Engineering from Twente University, The Netherlands, in 1984. His Master's thesis 'Design of discrete industrial systems' obtained the TNO-IWECO Award. In the same year he started his doctoral studies in the area of industrial automation in the Faculty of Mechanical Engineering at Twente University. In 1985 this research was continued in the Faculty of Mechanical Engineering at Eindhoven University of Technology, The Netherlands, and in 1987 he received his Ph.D. degree. Dr. Overwater has been employed at Eindhoven University since that time.

K.J. Vegter holds a Master's degree in Chemical Engineering from Delft University of Technology, The Netherlands. He has been employed by Foxboro since 1964. He served as Application Engineer and Project Manager on process automation projects for refineries and chemical plants. Since 1977 he has been involved in plant logistics. At present he holds the position of Logistics manager at Foxboro Nederland N.V. in Soest, The Netherlands.

North-Holland
Computers in Industry 10 (1988) 185–195

According to Maslow [3], "it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail". This statement may serve to illustrate the fact that the way of thinking is significant when interpreting a system's characteristics. Very often systems are felt to be difficult or complex due to a 'wrong' point of view. When the system is looked upon in a different way, it can suddenly become transparent. The classical approach is to interpret systems as consisting of a sequence of events; whereas, in fact, many things are happening simultaneously. It is felt that many problems and difficulties with the development of industrial systems have their origin in this antithesis.

This understanding can possibly be exemplified by means of an example taken from process industry: a batch plant. A batch plant is a complex industrial system, normally containing many process-units grouped into production-lines. Many different types of process-units are usually involved such as charging units, reactors, holding tanks, filters, crystallizers, distillation units and extractors. Products are transferred using manifold piping systems and more often than not, the system includes process-units shared by several production-lines and equipment shared by process-units. The complexity is further increased by the fact that units can operate in several modes (states), like start-up, normal, hold, shutdown, restart and emergency shutdown.

Process-units operate in parallel and their modes of operation depend on the states of other process-units; in other words many interactions exist between the individual units.

Describing the operation of a batch plant as a sequence of events occurring while products pass through the plant is, in theory, possible. The exact sequence executed depends, however, on the states of the process-units and equipment and the interactions between them. Many states are possible requiring different synchronization patterns thus altering the sequence of events during the production cycle. Taking into account all the possibilities is therefore very likely to become a complex and cumbersome task. Interpreting the plant as a number of interacting processes running in parallel appears to be a more natural approach.

Each process control system implies a model of the system to be controlled. Assuming a sequential model introduces a multitude of interactions to be considered, since the complete system and all its inherent states have to be taken into account. Choosing a parallel structure allows to look at the system as consisting of a set of loosely coupled sequences (processes), which only interact for the purpose of communication and synchronization. This way the number of interactions and thus the system's complexity can be reduced considerably. Each parallel process can be handled as a separate entity.

2. Systems, Processes and Interactions

In the introduction the word "system" has been mentioned several times. The notion 'system' is rather abstract and is often used to indicate something large and complex, such as factories, harbours or computers. However, systems are not necessarily large and complex. Smaller things like machines, cars and chips can be considered as systems just as well. Characteristic of systems is that they all consist of elements and relationships. A system can be formally described as an arbitrarily chosen set of elements, which are mutually related. The relationships in the system describe the coherence between its elements, i.e., they determine the system's structure and behaviour. Relationships with elements outside the system determine the system's context.

Industrial systems contain processes and interactions. Both processes and interactions are relationships between elements. To illustrate these notions a (simple) bottling plant is considered which converts ingredients and empty bottles into bottles filled with a product. The system contains a tank where a liquid is mixed with an acid to produce a predetermined blend of certain pH, and a bottling line for filling empty bottles with the blend. After being filled, the bottles are replaced by empty bottles from a waiting line (see Fig. 1).

The liquid, acid and bottles do not perform any actions in the system and are referred to as passive elements. Passive elements are not necessarily physical, but may represent signals or data too. Processes in the system determine the way that passive elements are converted into one another and, consequently, can be interpreted as relationships between these elements. The bottling system, for example, includes a blending and a bottling process which convert liquid and acid into blended

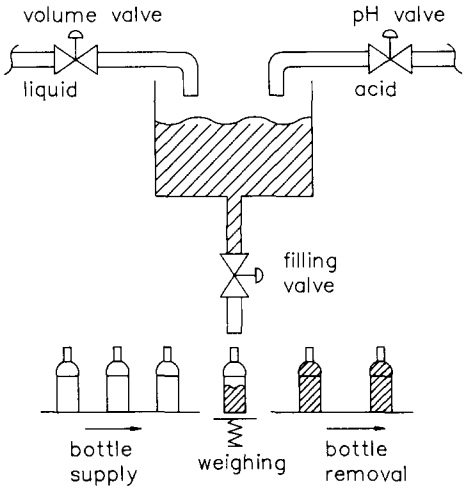

Fig. 1. The Bottling System.

product, and empty bottles and blended liquid into filled bottles respectively. The processes and passive elements represent the system's static structure.

A system's dynamic structure, on the other hand, is determined by the active elements in the system and the interactions between them. Active elements, in contrast to passive elements, perform actions and bring about changes in the system's state. Interactions are relationships which determine the way that the active elements cooperate. Through the interactions, the active elements influence their mutual behaviour. Such an interaction, for example, is found between the tank and the bottling line in the form of the blended product. When the tank fails to supply blended product, the bottling line cannot continue. Another important interaction implies that a bottle can only be filled when the pH of the liquid is within specifications. A human operator, for example, would be able to take care of this, so that he would be an additional active element taking control actions. The interaction between the operator and the tank would comprise, for instance, a pH-meter reading; whereas, his interaction with the bottling line could be effected through a signal controlling the bottle supply gate. The system also interacts with its environment, such as, for example, through the collection of empty bottles and the delivery of filled ones.

From the previous example, it will be clear that processes are performed by active elements. The tank performs the blending process, the bottling line takes care of the bottling process and the operator is in change of the control process. It also shows that active elements cooperate through the exchange of passive elements, thus, showing how interactions are effected by passive elements. This way of modelling is called the *process interaction approach*. It involves a relationship between the processes, active elements, interactions and passive elements. This approach makes it possible to combine a system's static and dynamic structure into one and the same model. It should be noted that this model implicitly represents systems states.

According to the process interaction approach it is harmless to mix up the words process and active element, since both refer to each other. The same holds true for interactions and passive elements. Talking about an interaction implies talking about the passive element being exchanged. The passive elements are exchanged according to a mechanism defined by the interaction. Two fundamental types of exchange can be discerned: the discrete and the continuous mechanism. Both mechanisms have been equipped with a 'send type' (S-) action and a 'receive type' (R-) action, which concern the acquisition and the release of passive elements or their contents. The concept of these actions is illustrated by means of the bottling system. The system, for example, acquires empty bottles from the environment and releases filled bottles in turn.

The discrete mechanism has a synchronizing nature. The contents of a passive element have no relevance to the mechanism. The mechanism defines a buffer and, thus, enables processes to run independently from each other. Passive elements can be placed in the buffer by performing an S-action. There, they are held until the receiving process is ready to handle them. Retrieval of an element is achieved by means of an R-action. The R-action has a blocking nature which means that the receiving process turns passive when the buffer is empty. The process is reactivated the moment new elements are available. This explains the synchronizing effect of the mechanism: each time a passive element occurs, the receiving process can be triggered. In the example, the production orders and empty bottles are passed in a discrete manner, as well as, the production data and the filled bottles.

The continuous mechanism has a communicative nature. It constitutes a continuous link be-

tween the processes involved. The passive elements taking part in this kind of interaction are no longer separable and are always available. The passive element serves like a medium through which a state or value can be communicated. The sending process updates the value (the S-action), whilst the receiving process can read it (the R-action). Several examples of the continuous mechanism are found in the bottling system. Both the liquid and the acid, for instance, are supplied in a continuous way.

Interactions may contain several S- and R-actions with a minimum of one S- and one R-action, since otherwise the interaction is unable to function.

Processes also interact by passing elements in order to force an interrupt or to raise an exception. Interrupt routines and exception handling, however, are part of the receiving process and do not concern the way passive elements are passed. Therefore, such interaction does not require a separate mechanism.

3. Describing Processes and Interactions

This section describes a language for expressing models of industrial systems which have been developed according to the process interaction approach. The language combines graphic notations with a restricted use of natural language. The graphical part is formed by the Process Interaction Diagram (PRIND), the textual part by the Diagram Documentation (DIDOC). A Process Interaction Diagram is a graphic representation of a system showing the processes taking place and the interactions between them. Each Process Interaction Diagram is associated with a Diagram Documentation in which these processes and interactions are documented. Models represented must obviously consist of at least one PRIND and one DIDOC, but may be expressed as structures of PRINDs and DIDOCs as well. In this way the language can represent parallel processes as sequential routines.

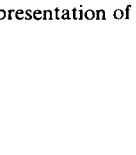

Fig. 2. Representation of processes.

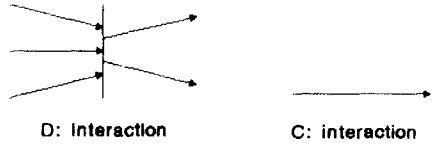

Fig. 3. Representation of interactions.

For the generation of PRINDs and DIDOCs the (prototype) software package D86 (Design '86) [5] is used. The package runs on a personal computer and provides a database, a graphic editor and facilities for cross reference and verification.

3.1. The Process Interaction Diagram (PRIND)

Process Interaction Diagrams are subject to a number of conventions and rules which apply to the drawing and the generation of them. The most important conventions are discussed below.

A process is represented by a 'bubble'. Each bubble has a name, it should explain the nature of the process (see Fig. 2).

An interaction is represented by a bar with at least one incoming and one outward arrow (Fig. 3, left part). Arrows entering the bar indicate S(end)-actions. Leaving arrows denote R(eceive)-actions. In case where the interaction contains only one S- and R-action, the bar becomes somewhat superfluous and can just as well be omitted (Fig. 3, right part). Every interaction carries a name preceded by a character indicating the mechanism type of the interaction: 'D' for the discrete mechanism and 'C' for the continuous mechanism.

Figure 4 presents the PRIND which describes the context of the bottling system presented in the previous section. Each bubble in a PRIND can be expanded into a new PRIND which is identified

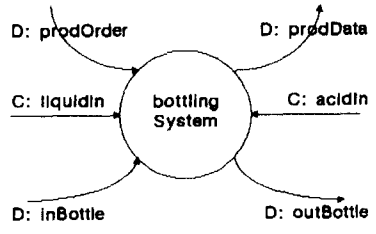

Fig. 4. PRIND Context.

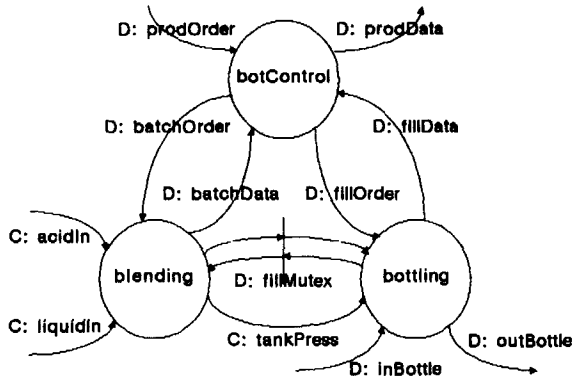

Fig. 5. PRIND BottlingSystem.

by the name of its bubble. Through this concept of deepening a hierarchy of PRINDs can be created. This hierarchy may resemble the system's hierarchy, but may correspond to a modular or top-down structure as well.

Each PRIND constitutes the context of the processes (bubbles) it contains. The bottling system, for example, can be explained as three associated processes: a blending process, a bottling process and a control process (Fig. 5). The generation of new PRINDs should stop when all processes can be described in sequential routines.

No R- or S-action may enter or leave a PRIND, other than those entering or leaving the bubble of which the PRIND is a refinement. When a new type of passive element is introduced, its type is known within that PRIND only. The types of passive elements passed through 'fillOrder' and 'fillData', for instance, were first introduced in the PRIND BottlingSystem. Therefore, they can be used only by processes that are part of this PRIND. The type of the passive element 'prodData', by contrast, is known to the environment and to the system, thus, it can be processed in both of them.

It should be noted that the arrows in a PRIND do not represent flows, since flow is accomplished by processes. The arrows only denote the nature of the actions at the points of interaction.

3.2. The Diagram Documentation (DIDOC)

Diagram Documentation (DIDOC) is a tool that is used to create a textual representation of a system. Each PRIND is backed up by a DIDOC carrying the same name as the PRIND. As shown in the Appendix, a DIDOC can be divided into an object definition part, an interaction definition part and a process definition part.

The object definition part contains the definition of new objects which have been introduced in the associated PRIND. Object definitions have the format

objectName @ appearanceDescription.

The name of an object always starts with a lower case character, the symbol '@' should be read as 'appears as' and te appearance description can be formulated by reference to another object or to a structure of other objects. Four structuring operators are available to build up a new object appearance: the sequence ('**object**...**end**'), the selection ('... | ...'), the (sub)range ('[from...to] **of** ...') and the list operator which can be either indexed ('**array** index range(s) **of** ...') or unindexed ('**list of** ...'). Also, a number of predefined objects is available from which objects can be constructed. Sometimes, it does not make sense to describe an object in detail. In those cases, the predefined appearance 'physical' can be used as a reference.

The interaction definition part contains the definitions of the interactions introduced in the associated PRIND. Interactions are described according to the format:

InteractionName @ mechanism / objectAppearance.

The first character of the interaction name should be a capital. Calling the name of the interaction with the first character in lower cast refers to the actual object establishing the interaction. The mechanism can be indicated using the abbreviations 'dis' and 'con'. The appearance of the object involved is described according to the same format as used in the object definition part of the DIDOC.

The process definition part indeed contains the process definitions. Since processes can be identified with the active elements performing them, a process can be defined as an object appearing as a process. As a result, a definition starts with the name of the process followed by the symbol '@'. When a process has been expanded into a PRIND, it appears as the predefined object 'expanded'. Also, processes with a strictly physical nature can be marked as 'physical' instead of being described in detail. Otherwise, the description continues with

the keyword process followed by a declarative part, an initiation part and a work cycle part, which appear according to the following format:

```
processName @
process
   declarations
begin
   initiations
   loop
      work cycle
   end
end;
```

The declarations, the initiations and the work cycle can be described in a suitable programming language, a pseudo code or any "structured" language. In this paper a Modula2-like pseudo code is applied which permits informal texts to be used. S- and R-actions respectively appear according to the format '**give** InteractionName' and '**take** InteractionName'.

4. The Bottling System

The previous section introduced the tools of D86. In this section these tools, the Process Interaction Diagram and the Diagram Documentation, are applied to define the control system of the bottling system mentioned before. This case study was based upon the bottling system discussed by Ward and Mellor [19]. The configuration of the system has been explained already in Section 2. The DIDOCs referred to in this section can be found in the Appendix.

The system works as follows. According to the incoming production orders, batches of blended product are produced from a basic liquid and an acid; then, bottles are filled with this product. The production orders include both the required pH-number and the quantity of bottles to be filled. Upon completion of an order, production data about the finished order is released. The PRIND Context (Fig. 4) shows the interactions of the system with the environment. Their definitions are found in the DIDOC Context. The basic liquid and acid are supplied continuously, whereas, the other interactions are discrete.

The bottling system can be expanded into three associated processes, as shown in the PRIND Bottling System (Fig. 5). The blending process controls the filling of bottles. These two processes interact through the intermittent exchange of blended product which can be modelled with the aid of the continuous interaction TankPress.

Another interaction between the blending and the bottling process prevents filling the bottles whilst a new batch of blended product is in progress. This mutual exclusion is effected through the interaction FillMutex. It includes a single object which serves as an item to be acquired prior to refilling the tank or filling a bottle.

The blending process aims to prepare a new batch as soon as it has received an order from the control process. Batch orders are compiled from the incoming production orders and include the required bottle size, the number of bottles to be filled and the pH-number. The bottling process can be made active through releasing a FillOrder which informs the process about the bottle size and the number of bottles to be filled. The output production data is based on the average pH and bottling weight received from the blending and bottling process after their orders have been completed.

The definitions of the interactions and processes just mentioned will be found in the DIDOC Bottling System. The interaction definitions are presented without comment, as it is felt that they are self-explanatory. The definition of the bottling control process is also straight-forward and can be understood from the previous description. The other two processes still need to be refined.

As shown in the PRIND Blending (Fig. 6), the blending process can be divided into five processes, two control processes and three physical processes.

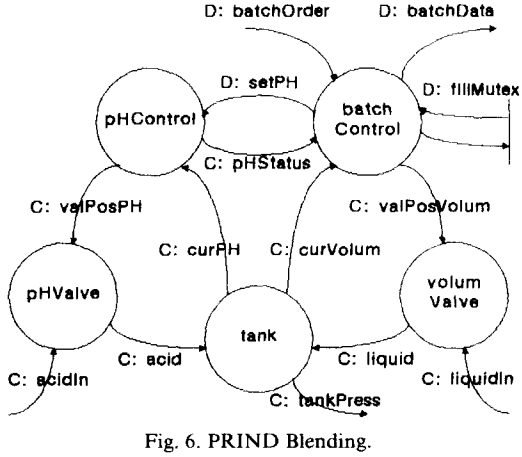

Fig. 6. PRIND Blending.

Already during its initiation, the batch control process aims to obtain the object fillMutex in order to prevent the bottling process from pre-emptive filling. Next, it waits for a new batch order from the bottling control process, after arrival of which, it sends a new setpoint to the pH-control process and calculates the required target volume from the order just received. The process must take into account that the batch size may be larger than the volume of the tank, in which case, the batch must be produced in stages. Both control processes form part of a control loop.

The pH-control process is continuously supplied with the current pH-value, as measured by the tank process. From this value, the position of the pH-valve is determined. As defined in the DIDOC Blending, this position can be either 'open', 'dribble' or 'closed'. The interaction PHStatus shows whether the pH is within the specified range or not and communicates the pH value attained. The tank process also measures the current tank contents which will be used by the batch control process to determine when the volume valve should be closed.

Both valve processes interact with the tank process in a strictly physical manner. It may seem unusual to model a valve as a process, but this follows from the fact that a valve is constantly converting a differential pressure into a flow rate. As soon as the tank has been filled, the object fillMutex is released and is not required until the tank contents reaches a critical level.

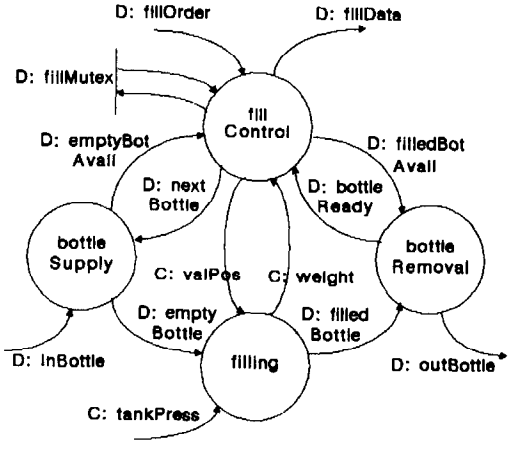

Fig. 7. PRIND Bottling.

The PRIND Bottling (Fig. 7) shows the expansion of the bottling process. The filling control process monitors the supply, the filling and the removal of bottles. It activates the bottle supply process by ordering it to move the next bottle into the filling section. As soon as the empty bottle arrives, the control process is signalled by the supply process.

The filling control process responds by acquiring the object fillMutex prior to opening the valve in the filling section and checking the net weight continuously. When the required weight has been reached the filling valve is closed, the object fillMutex is released and the bottle removal process is activated.

After the filling section has been cleared, the removal process gives a 'ready' signal. The release of filling data by the filling control process indicates that the bottling process is waiting for its next order. The documentation of the processes and interactions is found in the DIDOC Bottling.

5. Further Applications

Although systems development cannot be defined as a detailed step-by-step procedure, it is possible to give some global indication of the progression of a system under development. In this perspective, three phases can be recognized: the specification, the validation and the implementation of the system-to-be. The transitions between these phases are not readily apparent. Rather, the distinction between the phases will be suggested by the nature of the activities, more than by the recognition of the attainable milestones during the development process. In fact each phase includes something of the other two, because in each phase the system may undergo modifications, which, in their turn, will require specification, validation and implementation. Still, the nature of the activities gradually changes: from abstract and qualitative during specification, to exact and quantitative during validation and, finally, to concrete and realistic during implementation.

All three phases involve a lot of modelling. During the specification, a model is built to reflect the system's purpose and its essential properties, and to enable ideas and views on the system to be

discussed. Next, validation requires a model of the system that can be examined in order to gain an insight into the system's behaviour. And finally, the difference between the system and its model fades away entirely, when the model becomes incorporated in the system during its implementation.

In this paper the process interaction approach has been discussed as an approach to the specification of industrial systems. Specification, however, is not the only field of application. The concepts of processes and interactions are found in the simulation package S84 (Simulation '84) [12] too. Simulation is a tool that can be applied to validate the dynamic behaviour of industrial systems. Simulation experiments require a simulation model which reflects both the system's layout and its control strategies. A simulation model can be described in terms of a computer program which is often referred to as 'the simulation program'. The DIDOCs generated with D86 form a good starting point for a simulation program written in S84. Another important application of the concepts of process and interaction is the implementation of industrial systems by means of the software package ROSKIT (Real-time Operating System KIT) [10,15]. A separate description of the devices and control strategies in the simulation program even offers the possibility of a rather simple conversion of simulation programs into control programs [9]. A further discussion on these subjects and on the backgrounds of the process interaction approach can be found in Overwater [7].

The process interaction approach has been applied successfully to a variety of cases. A few of them are mentioned for illustration.

Some major examples, in the field of production and machines control, were the development of a transport and storage system for a bicycletyre factory [1], the simulation of a modular internal transport system [2,4,8], and the simulation and control of a power-and-free conveyor system [17]. Another case concerned the control of an automated machine for building up and breaking down a stack of pallets [10].

In the area of distribution and transportation, some characteristic examples have included an examination of the operational characteristics of closed-loop conveyors [6,13], a strategic study concerning coal transshipment [14], an analysis and simulation of a flower auction [11], and the development of a planning system for various combinations of public transport services [16].

Other examples have included the development of an architecture of loosely-coupled processors [18] and the simulation of a local area network in a highly automated production environment [20].

Some recent (unpublished) case studies include the development of an order picking system for a dairy factory, a feasibility study for a conveyor and storage system for fibre products, and the design of a control system for the production of so-called 'wafers' in a chip factory.

6. Conclusion

The considerable experience gained so far, demonstrated that the process interaction approach does provide a suitable way of thinking for the development of industrial systems. Its power springs from the distinction between the *cyclic, reversible* phenomena (the processes) and the *non-cyclic, irreversible* phenomena (the interactions) in an industrial system. A system no longer needs to be modelled as a *single sequence of events, but it can be divided into a number of simultaneously operating work cycles* which can be considered *independently*.

As a result, three major advantages can be attained by using the process interaction approach:

(1) the apparent complexity of systems consisting of interacting processes operating in parallel can be greatly reduced;

(2) it becomes possible to establish an integration between all kinds of activity throughout the system, varying from machine operations to production control, and

(3) the approach permits a far going integration of the three phases of systems development: the same concepts can be used for the specification, the validation and the implementation of industrial systems.

References

[1] J.H.A. Arentsen, Ontwerp van een transport- en opslagsysteem t.b.v. de fabricage van fietsbuitenbanden, MSc. thesis, Faculty of Mech. Engrg., Twente University, Enschede (1981) in Dutch.

[2] W.M. Bergkamp, Een simulatieprogramma voor een intern transportsysteem, MSc. thesis, Faculty of Mech. Engrg., Twente University, Enschede (1985) in Dutch.
[3] A.H. Maslow, *The Psychology of Science*, Harper & Row, New York, NY (1966).
[4] G.J.C. van Miert, Simulatie en besturing van een modulair accumulerend transportsysteem met het simulatiepakket SOLE, MSc. thesis, Faculty of Mech. Engrg., Delft Univesity of Technology, Delft (1984) in Dutch.
[5] B. Munneke and R. Overwater, Design '86, Manual, Faculty of Mech. Engrg., Eindhoven University of Technology, Eindhoven (1986) in Dutch.
[6] W.W. Nawijn and J.E. Rooda, Some further results for closed-loop continuous conveyors, *Mechanical Communications*, Twente University, Enschede (1982).
[7] R. Overwater, Processes and Interactions: An approach to the modelling of industrial systems, Dissertation, Eindhoven University of Technology, Eindhoven (1987).
[8] M.J.H. de Pont, A simulation model for handling and routing of product carriers in a modular internal transport system, MSc. thesis, Faculty of Mech. Engrg., Twente University, Enschede (1984).
[9] J.E. Rooda and W.C. Boot, A combined approach for the design and control of logistics systems, *Proc. 4th Int. Logistics Congress*, Part II, pp. 131–135 (1983).
[10] J.E. Rooda and W.C. Boot, Procescomputers, Processen en interacties, Memo, nr. WPA 252, Faculty of mech. Engrg., Eindhoven University of Technology, Eindhoven (1986) in Dutch.
[11] J.E. Rooda, P.F. Jansen and M.E.A. Striekwold, Analyse des Verteiler-Systems einer Auktionszentrale für Blumen (in German), *Fördern und Heben*, Vol. 32, No. 10, pp. 774–777 (1982).
[12] J.E. Rooda, S.M.M. Joosten, T.J. Rossingh and R. Smedinga, Simulation in S84, Manual, Faculty of Mech. Engrg. Twente University, Enschede (1984).
[13] J.E. Rooda and W.M. Nawijn, The analysis of operation characteristics of closed-loop continuing belt conveyors using simulation and queueing approximations, *Mechanical Communications*, Twente University, Enschede (1980).
[14] J.E. Rooda and N. van der Schilden, Simulation of maritime transport and distribution by sea-going barges: an application of multiple regression analysis and factor screening, *Bulk Solids Handling*, Vol. 2, pp. 813–824 (1982).
[15] T.J. Rossingh and J.E. Rooda, RealTime Operating System KIT, Manual, Faculty of mech. Engrg., Twente University, Enschede (1984).
[16] J.W.P.J. van Stiphout, Mobiele Service Systemen, modellering en simulatie, MSc. thesis, Faculty of Applied Mathematics, Twente University, Enschede (1983) in Dutch.
[17] E.M. Toet, Simulatie en besturing van een power-and-free transportsysteem, MSc. thesis, Faculty of Mech. Engrg., Eindhoven University of Technology, Eindhoven (1986) in Dutch.
[18] R. Vuurboom, Communication in a loosely-coupled multimicroprocessor system, Its architecture and implementation, MSc. thesis, Faculty of Computing Science, Twente University, Enschede (1984).
[19] P.T. Ward and S.J. Mellor, *Structured Development for Real-Time Systems*, Yourdon Press, New York, NY (1985).
[20] H.J. Welmer, Ontwerp van een computernetwerk werkend onder een MUPOS of ROSKIT Modulair Pascal beheerssysteem, MSc. thesis, Faculty of Computing Science, Twente University, Enschede (1985) in Dutch.

Appendix

DIDOC Context

```
* * * Objects * * *

pHRange @[0..14] of real;
sizeRange @ 35 | 70 | 100;

* * * Interactions * * *

ProdOrder @ dis / object
                    orderNumber      @ [0..maxOrdNum];
                    numberOfBottles  @ integer;
                    pHValue          @ pHRange;
                    bottleSize       @ sizeRange
                  end;
ProdData  @ dis / object
                    orderNumber  @  [0..maxOrdNum];
                    actualPH     @  pHRange;
                    averageCont  @  [0..100] of real
                  end;
InBottle  @ dis / physical;
OutBottle @ dis / physical;
AcidIn    @ con /physical;
LiquidIn  @ con /physical;

* * * Processes * * *

bottlingSystem @ expanded;
```

DIDOC BottlingSystem

```
* * * Objects * * *

weightRange @ [0..maxWeight] of real;

* * * Interactions * * *

BatchOrder @ dis / object
                    batchSize,
                    alarmVolum @ real;
                    newPH      @ pHRange
                  end;
BatchData  @ dis / pHRange
FillOrder  @ dis / object
                    bottleSize  @ sizeRange;
                    amount      @ integer
```

```
                end;
FillData    @ dis / weightRange;
FillMutex   @ dis;
TankPress   @ con /physical;

* * * Processes * * *

botControl @

process
begin
   loop
      take ProdOrder;
      with batchOrder
         do determine batchSize, alarmVolum and newPH
         end;
      give BatchOrder;
      with fillOrder
         do determine bottleSize and amount end;
      give FillOrder;
      take BatchReady;
      take FillReady;
      with prodData
         do update actualPH and averageCont end;
      give ProdData
   end
end;

blending @ expanded;

bottling @ expanded;
```

DIDOC Blending

```
* * * Interactions * * *

SetPH       @ dis / pHRange;
PHStatus    @ con / object
                       ok  @ Boolean;
                       pH  @ pHRange
                    end;
ValPosPH    @ con / open | dribble | closed;
ValPosVolum @ con / open | closed;
CurPH       @ con / pHRange;
CurVolum    @ con / [0..maxVolum] of real;
Acid        @ con / physical;
Liquid      @ con / physical;

* * * Processes * * *

batchControl @
process
  targetVolum, averagePH @ real;
  numberOfShifts @ integer;
begin
  take FillMutex;
  loop
     take BatchOrder;
     setPH := batchOrder.newPH;
     give SetPH;
     determine numberOfShifts;
     for numberOfShifts
        do
           determine targetVolum;
           valPosVolum := open;
           give ValPosVolum;
           repeat take CurVolum
           until curVolum > = targetVolum;
           valPosVolum := closed;
           give ValPosVolum;
           repeat take PHStatus
           until pHStatus.ok;
           give FillMutex;
           update averagePH;
           repeat take CurVolum
           until curVolum < = batchOrder.alarmVolum;
           take FillMutex
        end;
        batchData := averagePH;
        give BatchData;
  end
end;

pHControl @
process
  updateSetPoint @
  exception
  begin
     take SetPH;
     return
  end;
begin
  enable updateSetPoint for
  loop
     take CurPH;
     if curPH > setPH
     then
        pHStatus.ok := false; give PHStatus;
        valPosPH := dribble; give ValPosPH;
        repeat
           take CurPH;
           if curPH > 11.0
           then valPosPH := open
           else valPosPH := dribble end;
           give ValPosPH
        until curPH = setPH;
        valPosPH := closed; give ValPosPH;
        pHStatus.pH := curPH; pHStatus.ok := true;
        give PHOK
     end
  end end
end;

pHValve     @  physical;
tank        @  physical;
volumValve  @  physical;
```

DIDOC Bottling

```
* * * Interactions * * *

NextBottle      @ dis;
EmptyBotAvail   @ dis;
```

```
EmptyBottle     @ dis / physical;
ValPos          @ con / open | closed;
Weight          @ con / weightRange;
FilledBotAvail  @ dis;
BottleReady     @ dis;
FilledBottle    @ dis / physical;

* * * Processes * * *

fillControl @
process
   counter @ integer;
   averageWeight @ real;
begin
   loop
      take FillOrder;
      with fillOrder
      do
         for counter := 1 to amount
         do
            give NextBottle;
            take EmptyBotAvail;
            give BottleAvail;
            take FillMutex;
            valvePos := open; give ValvePos;
            repeat take Weight until weight = bottleSize;
            valvePos := closed; give Valvepos;
            give FillMutex;
            give FilledBotAvail;
            update averageWeight;
            take BottleReady
         end
      end;
      fillData := averageWeight;
      give FillData
   end
end;

bottleSupply    @  physical;
filling         @  physical;
bottleRemoval   @  physical;
```