Structured Business Process Modeling (SBPM)

Alfred Holl

*Department of Computer Science,
University of Applied Sciences at Nuremberg, Germany*

Alfred.Holl@fh-nuernberg.de

Gregor Valentin

*Department of Computer Science,
University of Applied Sciences at Nuremberg, Germany*

Gregor-V@web.de

Abstract: *Business process modeling (BPM) and control flow modeling are types of process models (behavioral models). The current use of BPM is accompanied by the same deficiencies as control flow modeling in its early state. In an analogous way as unstructured program design and programming (spaghetti code) had to be replaced by their structured equivalents, a change in BPM is necessary. That is why the similarity between control flow modeling and BPM has to be made explicit in detail. This task is accomplished and a way for the necessary change in BPM is proposed on the basis of a precise investigation of the parallels between control flow modeling and BPM. This comparison leads to a core meta-model of process models in general.*

Keywords: *process model, behavioral model, business process model, control flow models, structured modeling, structured design, structured programming, event-driven process chain, UML activity diagram, structured flowcharts*

1. Introduction: Historical analogy between the evolution of control flow modeling styles and BPM styles

BPM is a type of process modeling (behavioral modeling). Therefore, BPM and its historical evolution can be compared to other types of this modeling aspect, particularly to control flow modeling and its historical evolution.

BPM was introduced in the late 1980s in order to describe and to visualize processes in enterprises. Companies deal with BPM in order to find inefficiencies which cause financial losses. They may also want to establish the documentation necessary for the ISO 9000 certification. On the basis of a business process model of the current state, companies can redesign and optimize their processes to develop a future planned state (business process reengineering).

The current style of business process modeling, however, is similar to control flow modeling in the 1960s. The unstructured programming style of that time is called *spaghetti code programming* due to its lack of a transparent structure. At that time, source codes were a total mess. It is true that programs worked, but the more complicated they became, the less understandable the source codes were. “There is nothing to prevent the systems analyst from creating an arbitrarily complex, unstructured flowchart” (Yourdon, 1989, 222).

This non-transparent style was no longer tolerable as it made it very difficult to find bugs and to modify programs. Therefore, in the 1970s, a radically new programming style, called *structured programming*, was introduced although not generally used from the very beginning. Meanwhile, its advantages are clear and no longer subject to discussion.

The current unstructured style of BPM, which we can call *spaghetti BPM*, leads to similar problems as spaghetti coding. Due to this reason, a change to *structured BPM* in analogy to the change to structured programming within control flow modeling is desirable. BPM can learn a lot from structured flowcharting.

The historical parallel in the evolution of control flow modeling and business process modeling is outlined in table 1.

Control flow modeling styles		BPM styles	
1950s 1960s	Spaghetti code programming and spaghetti design	late 1980s	Spaghetti BPM
early 1970s	Structured programming and structured design	2005 ?	Structured BPM

Table 1. *Historical parallel between control flow modeling and BPM styles*

Structured control flow modeling, closely related to structured programming, was not achieved within one day. It was the result of a period of development which had its starting point in the 1950s.

It is important to mention that control flow modeling in this context always has two aspects: the program design using particular graphic notations and the coding using particular procedural programming languages. Design models, which support programming, have to be mapped onto source code using the individual language constructs of procedural programming languages. Therefore, changes with regard to both aspects were necessary.

Although it would have been possible to write structured code already at that time with the first procedural programming languages, the improvements by this style of programming were not recognized from the very beginning. A reason was certainly that it causes a higher effort to consequently use the block structures required. Untrained model designers considered this constraint as a useless restriction of their ‘freedom’ without a big profit. In addition, ‘natural’ human thinking does not work according to the block structures of this style.

A second reason is that the theoretical basis of structured coding had to be introduced. This did not happen before the late 1960s (cf. Böhm / Jacopini, 1966). The change of the coding style had to be accompanied by a change of program design and its notations. “When structured programming first became popular in the mid-1970s, Nassi-Shneiderman diagrams were introduced as a structured flowcharting technique” (Yourdon 1989, 223). Control flow charts, which did not enforce structured design (although they already made it possible), were replaced by Nassi-Shneiderman diagrams (Nassi / Shneiderman, 1973; Chapin, 1974) which did not allow anything else but structured design.

In the same way as the step to structured control flow modeling, the step to structured BPM will not be taken within one day. This paper proposes a way for this necessary change on the basis of a detailed investigation of the parallels between control flow modeling and BPM.

The question arises why these parallels are not considered as obvious within the field of information systems where business processes are usually modeled. This is due to the different views in computer science and information systems: structured programming takes the more computer science oriented view, BPM the more business oriented one. The contact between these two branches is not close enough to generally recognize the similarity between program design and BPM. Therefore, it has to be made explicit in detail.

The starting point for this demonstration is the generally recognized statement that both control flow models and business process models describe sequences of activities and events, that they are types of process / behavioral models.

The further argumentation in this paper runs as follows:

Chapter 2 shows the deficiencies of the current unstructured BPM style using a typical example. It gives a clear motivation for a change towards structured BPM. In Chapter 3, we improve our example by remodeling it in a structured way. Thus, we demonstrate the profits of structured BPM in general. In Chapter 4, we make an attempt to show the similarity between control flow modeling and business process modeling in detail. Therefore, we establish umbrella terms for the equivalent components of either modeling approach. The similarity is outlined in a synopsis in graphical and natural languages. Both of the modeling approaches are generalized in a core meta-model for process / behavioral models. The paper terminates with a conclusion in Chapter 5.

2. Motivation for a change of the current unstructured BPM style: a typical example

In this chapter, we discuss a typical terrible example of the current unstructured BPM style. Such models are widespread. The following example is taken from a real student project in order not to violate any copyright. The model is represented as a UML activity diagram.

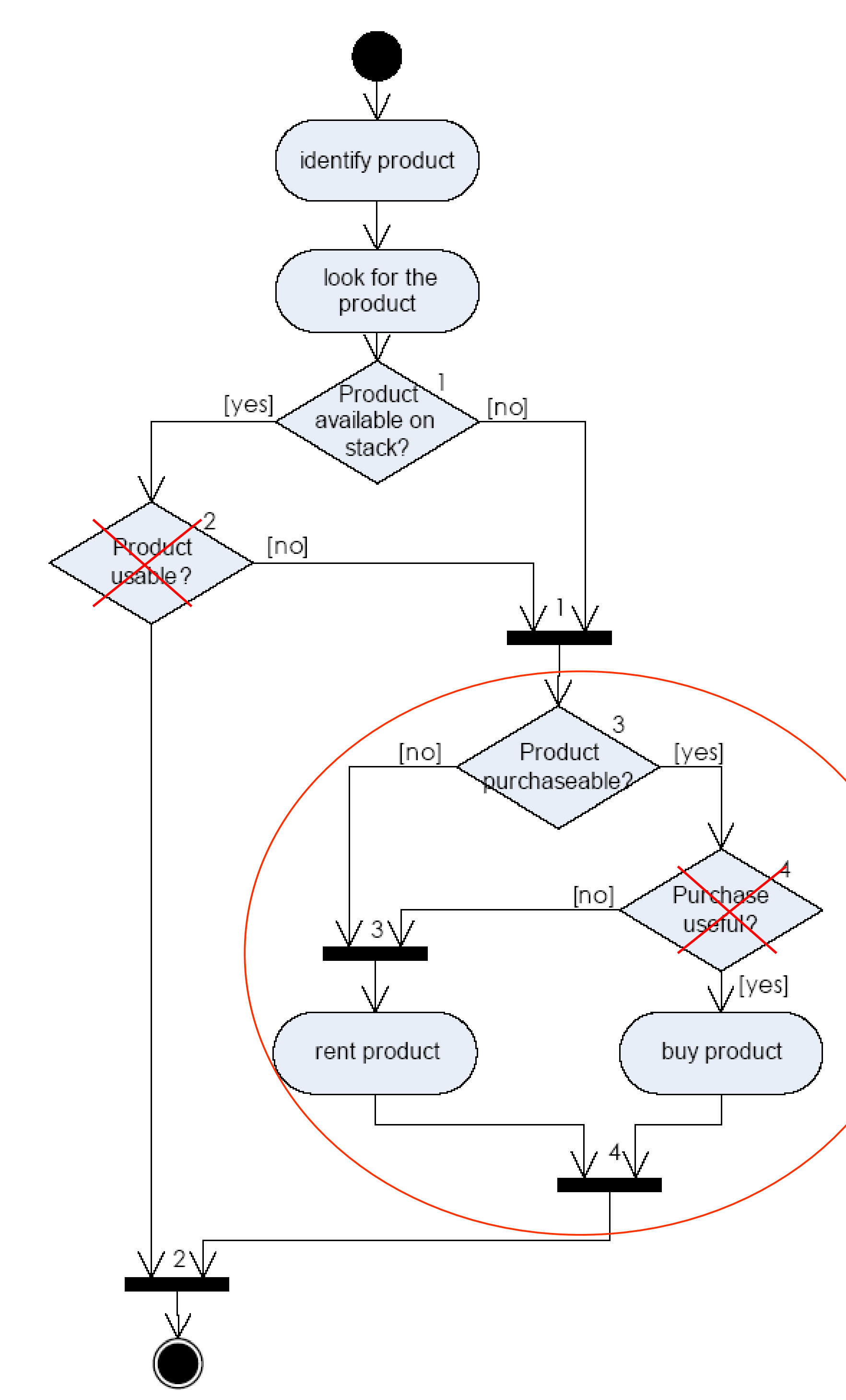


Figure 1. *Typical example of the current BPM style in the form of a UML activity diagram*

2.1 The current unstructured BPM style

The diagram represents a business process in a car rental company. Let us first explain the process: The start event is triggered when a customer wants to rent a car. He has particular expectations of the vehicle, e.g. he wants to be able to transport nine people. This leads to the first activity in the model, called ‘identify product’. The salesman has to decide, which product meets the specific needs of the customer. He has to figure out, which of the cars offered by his company is big enough to transport nine people. The next activity is to check the availability of the product during the period required. This can mean that another employee checks the car pool database whether the bus chosen is available.

After that, the first decision has to be made. If the bus is available (not rented by anyone else) and usable (no defects), the end of the process is reached and the bus is rented to the customer. Otherwise, that is if the bus is not available, we have to continue the process with the marked part. The intention is to make the product available for the customer. If the bus is not purchasable or its purchase is not useful, it could be rented from another car rental company. Otherwise, if the company considers it as useful to own one or another bus which is able to transport nine people, the company will buy it. The process ends in both cases.

Why is the model not effective? Why it is unstructured? Let us first explain the marked part. The positive branch of decision (3) is interrupted by decision (4). Supposing that (4) is left on its positive branch, its predecessor (3) will remain unclosed. In other words, decision (3) will not be closed in one defined point. The two decisions are overlapping, but not nested. The same situation can be found with regard to decision (1) and (2). To explain the same problem in different words, we can say: if decision (2) is left on its positive branch, it is not closed before decision (1) is closed. This violates the LIFO principle of correctly nested alternatives as the inner alternative (2) is not completely contained in the yes branch of the outer one (1).

This was a simple example restricted to alternatives which we had to choose in order to meet the limitations of a short research paper. Just imagine a more complex example with overlapping loops and alternatives leaving the loops in arbitrary places. You can encounter comparable situations in many business process models. This nightmare of every modern programmer is still common within current BPM without being exposed to hard criticism.

2.2 The current unstructured BPM style and its notations

As already pointed out in Chapter 1, the use of structured control flow modeling depends on the possibilities provided. Most of the program design notations and most of the procedural programming languages do not enforce structured design and coding, although they allow it. In these cases, the programmer’s effort is required. The famous exception is the very restrictive Nassi-Shneiderman diagram which only allows structured design. “The Nassi-Shneiderman diagrams are generally more organized, more structured and more comprehensible than the typical flowchart” (Yourdon, 1989, 224). The situation is that either hard discipline or restrictive notations / languages lead to structured control flow models.

Let us now have a look at the corresponding situation with regard to BPM. We can analogically transfer each word of Yourdon’s comment on process specifications: “Unless great care is taken, the flowchart can become incredibly complicated and difficult to read” (Yourdon, 1989, 290). None of the current notations for BPM automatically leads to structured models (cf. the synopsis in Keller, 2000, 53). State-transition diagrams and UML activity diagrams (cf. Keller, 2000, 51 and 116 for examples of unstructured modeling style and Keller, 2000, 110 for a synopsis of various UML behavioral diagrams) let the ‘freedom’ of unstructured design.

The same statement applies for event-driven process chains (EPC). A.-W. Scheer, their well-known inventor, gives a description (Scheer, 1994, 46-51) where the concepts of correctly nested alternatives and correctly nested loops are not even mentioned, a specific symbol for loops is not introduced (cf. 3). Scheer presents a great many of spaghetti EPC diagrams in his book on business process engineering. The lack of structure is obvious in figures, such as B.I.37, B.I.132.a, B.I.222, B.II.09, B.II.24, C.II.56 (Scheer, 1994), which are far from being transparent.

The great effort put into these new model notations does not prevent model designers from unstructured BPM. If it is desired in a project to use non-restrictive notations (other than Nassi-Shneiderman) to represent business process models, the structuring problem exists independent of the notation chosen. In these cases, structured modeling remains a question of the designer’s principles based on the understanding of the advantages of structured BPM.

3. Demonstration of the opposition between unstructured and structured BPM styles: improvement of the example

In opposition to the BPM example from Chapter 2, designers who follow a structured style use a set of modeling components (control constructs) which make the control flow transparent and well structured. Yourdon speaks of “structured English” (Yourdon, 1989, 206-214) and states that “to create a structured flowchart, the systems analyst must organize his or her logic with nested combinations of the flowchart symbols [by Böhm-Jacopini]” (Yourdon 1989, 222). We summarize the characteristic features of structured programming in our own words:

- block structures: BEGIN-END blocks, IF-ENDIF blocks, CASE-ENDCASE blocks, LOOP-ENDLOOP blocks instead of GOTO-instructions
- hierarchically nested structures (LIFO principle: the block opened as the last one has to be closed as the first one) instead of overlapping structures
- hierarchic modular structure (vertical decomposition by subroutines)

Let us now apply these features to the unstructured example from Chapter 2 and transfer it into a structured one in order to show that the principles of structured programming are well applicable to BPM as well.

First, we remodel the inner block which is marked. To nest decision (4) completely into the positive branch of decision (3), it is necessary to double the function ‘rent product’.

Doing so, as shown in the improved diagram (fig. 2), the inner decision (4) which was opened as the last one is closed first. It is now nested in the outer one (3).

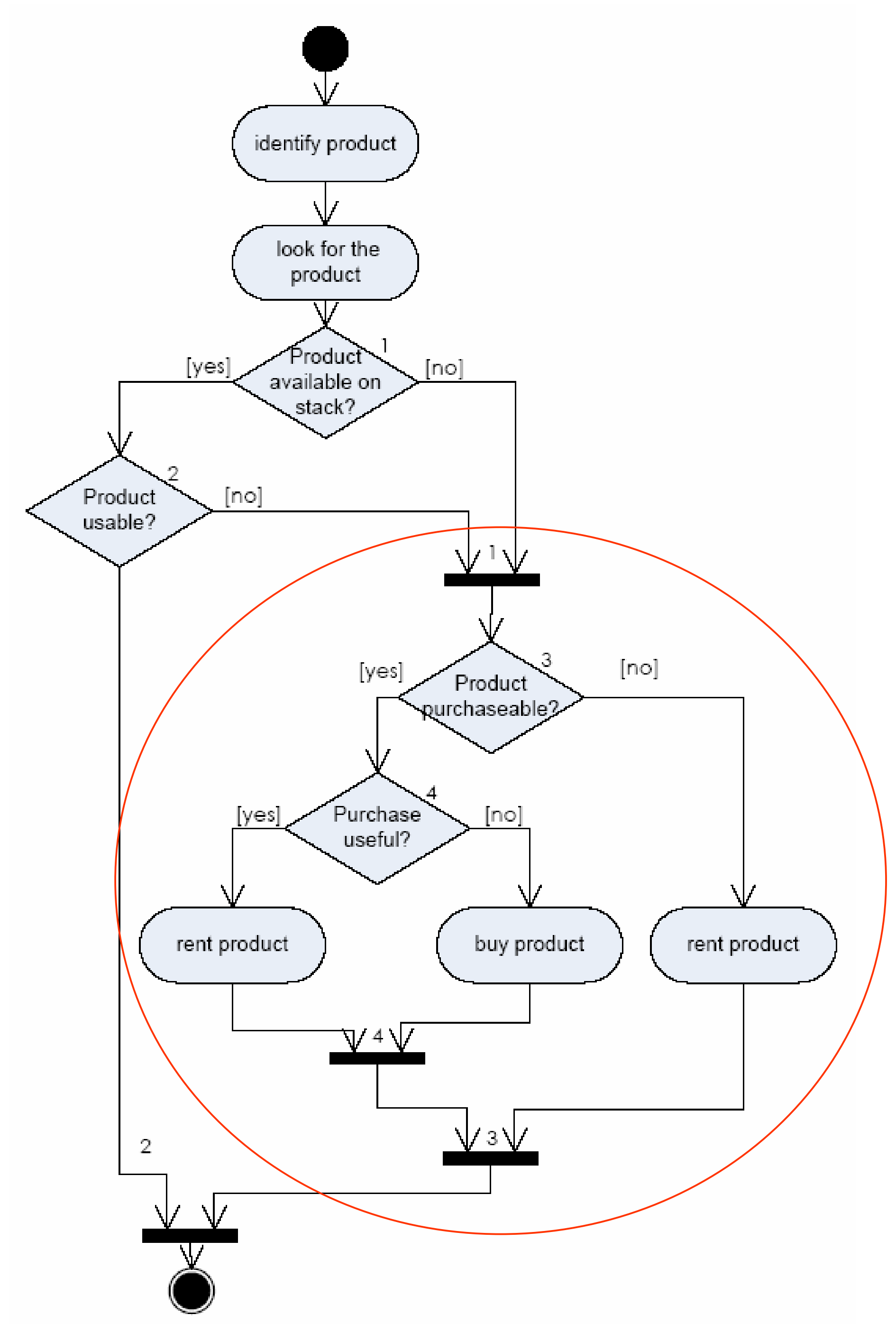


Figure 2. *Improved business process model in the form of a UML diagram*

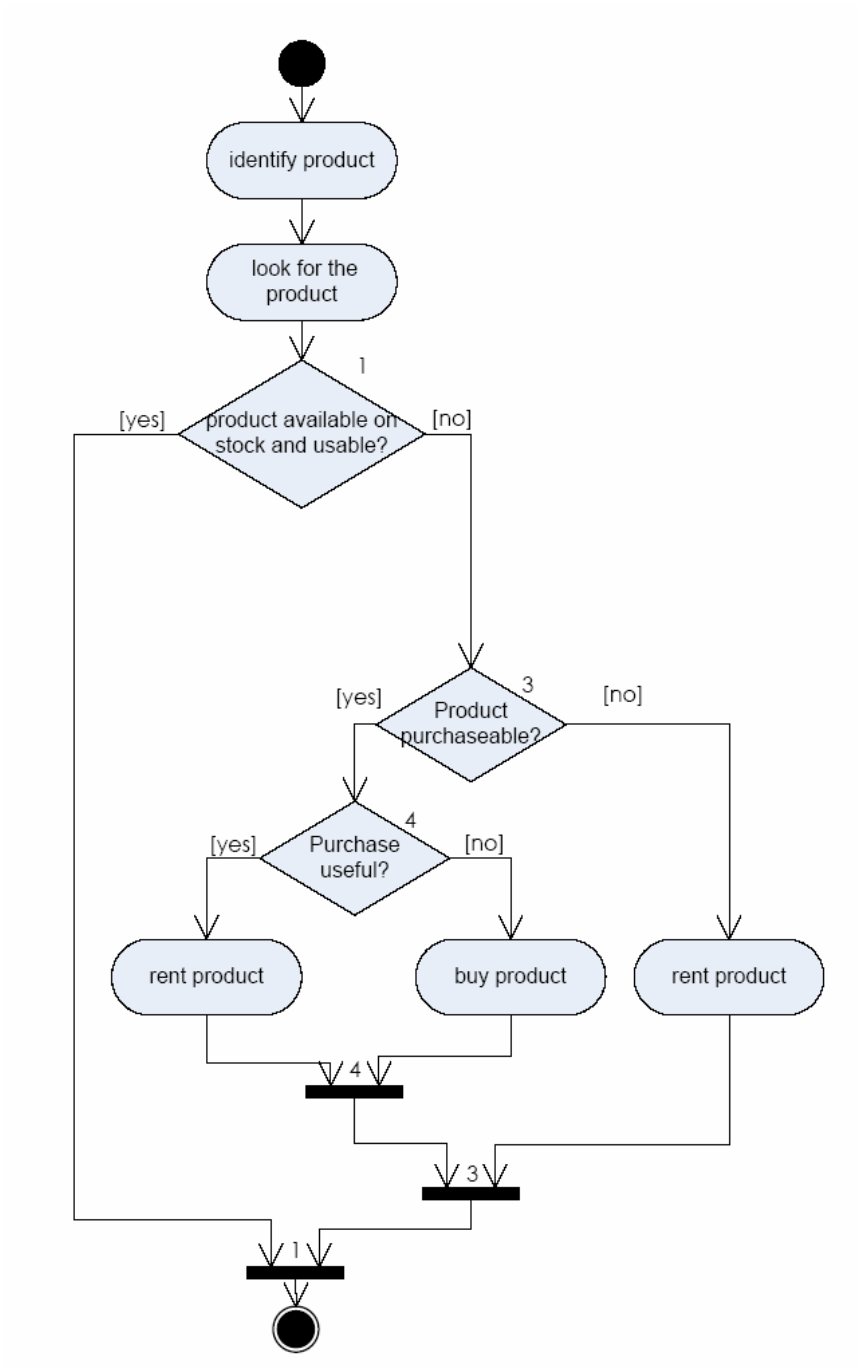


Figure 3. *Well structured business process model in the form of a UML activity diagram*

We can see that the entire model does not yet have a structured form (decisions (1) and (2)). To develop this, we could follow the same strategy as before and double the marked part in order to use one for the no-branch of decision (1) and the other one for the no-branch of decision (2). It is true that this would lead to a structured model, but also to a complex diagram. Because of this problem, we decide to combine the two decisions to one. The result is shown in fig. 3.

Now the model is completely structured, but it is entirely equivalent to the original example in Chapter 2. This illustrates, that BPM can be done in a structured way like structured program design.

Nevertheless you could still object that we cannot compare BPM and control flow modeling in detail. Models of business activities did not have anything in common with procedural programming and the analogy were taken from too far. Therefore, it is inevitable to have a close look at a precise comparison of the elementary components of the two modeling approaches.

4. Analogy of the elementary components of control flow models and business process models

In this chapter we will show, that control flow models and business process models have analogous elementary components although they model different situations and use different notations. Both are behavioral models as they represent courses of activities and events. While the main aspect of the programmer is the source code which he has to write, the main view of the business process model designer are the enterprise and the business process which he has to represent. Although both of them use different perspectives, we can figure out analogous elementary components in either modeling approach.

Examples:

- The instructions of the programmer are the business activities of the business process model designer. We can call this type of elementary components *process units* using an umbrella term on a more abstract level.
- *Events* in a company are caused from outside of the company, e.g. a phone call from a customer, or from inside a company when an activity

is finished. For programmers, an event is always induced by the operating system.

- For a business process modeler, *iteration* means to repeat the same business activities in the same order for several times in a cycle. For the programmer, this means a repetition of the same program instructions in a loop.

Continuing this way, it is possible to compare all of the other elementary components of either modeling approach. To make this more obvious, table 2 shows the analogies of the terminology for elementary components of BPM and control flow modeling and makes an attempt to establish umbrella terms. A comparable synopsis of behavioral models with other aspects of comparison is presented in Keller, 2000, 37.

Umbrella term	**BPM**	**Control flow modeling**
Modular substructure	partial process	subprogram, subroutine
Sequence	sequence	sequence
Test, alternative, decision	XOR	IF
Iteration	cycle	loop
Event	business event	operating system event, interrupt
Process unit	business activity	instruction or block of instructions
Simultaneity	AND	parallel functions

Table 2. *Analogy of the elementary components of BPM and control flow modeling*

Both of the modeling approaches differ not only in the names of their elementary components. They also use different notations. In table 3, the most common notations of structured programming and BPM are assigned to the corresponding umbrella terms.

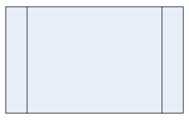
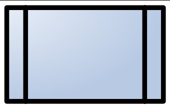
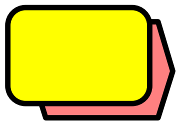
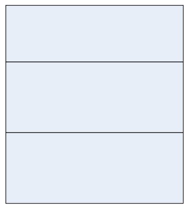
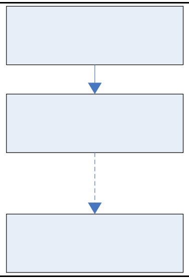
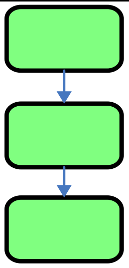
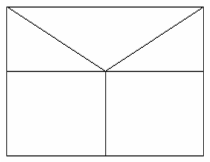
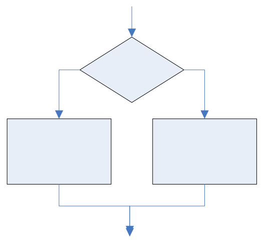
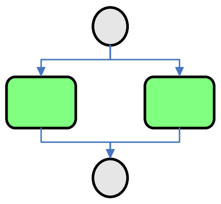
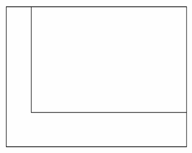
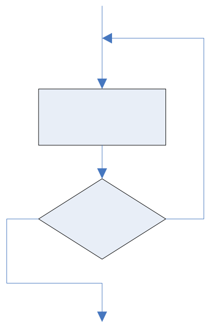
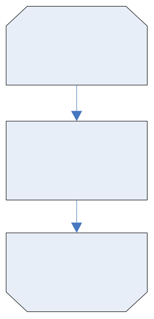
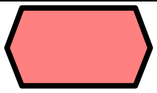
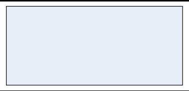
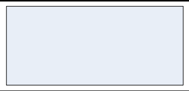
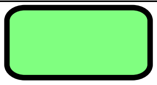
Umbrella term	Structure diagram (DIN 66 261) according to Nassi-Shneiderman	Control flow chart (DIN 66001)	Block diagram: extensions of DIN 66001	Event driven process chain (EPC)
Modular substructure				
Sequence				
Alternative, decision				
Iteration: DO-WHILE, REPEAT-UNTIL, WHILE				No symbol
Event	No symbol	No symbol		
Process unit				

Table 3. *Analogy of the notations of BPM and control flow modeling*

It is now shown that each elementary component of control flow modeling has its analogous counterpart in BPM. In addition, business process models can contain non-temporal information: the essential components describing the mere course over time can be extended by roles (actors), such as external partners and responsible departments / persons, and by data stores and resources used. But these components are only accidental as they do not affect the temporal structure of a process / behavioral model. As a result, we can state that all of the features of structured programming (cf. list at the beginning of Chapter 3) can be transferred to BPM.

Furthermore, table 2 contains a striking argument for the equivalence of control flow models and business process models, which should convince everyone who is still in doubt. The umbrella terms of elementary process components in table 2 can be regarded as a core meta-model of process / behavioral models in general (a meta-model is a list of all possible elementary components which can be used to establish models of a specific type). It describes possible components of process models in terms of a formalized natural language and is therefore independent of any notation, such as UML. We could extend it by additional accessories, such as those just mentioned, thus developing a comprehensive meta-model of process models in general.

5. Conclusion

In this research paper, it was demonstrated that the principles of structured programming are usefully applicable to BPM and that the elementary components of both of the modeling approaches are analogous. From this, it is obvious that considerable improvements should take place with regard to the current style of BPM. We give an open list of requirements to structured BPM:

- Block structures should be used instead of mere control flow lines (corresponding to GOTO instructions): the notations for all of the elementary components without the mere sequence must comprise a divergent delimiter (begin) and a convergent delimiter (end, synchronization); the delimiters have to be arranged symmetrically in a diagram: IF – ENDIF (cf. BEGIN XOR – END XOR); CASE - ENDCASE; LOOP – ENDLOOP; BEGIN AND – END AND; BEGIN OR – END OR.

- In the case of concurrent block structures, hierarchically nested structures (LIFO principle: the block opened as the last one has to be closed as the first one) should be used instead of overlapping structures.
- A motivated hierarchic modular structure (in accordance to subroutines) should be used to decompose processes vertically (cf. Holl, 2000).
- We urgently recommend the extension of BPM notations by symbols for iterations. Every experienced programmer will realize the current lack. Within structured programming, iterations have turned out to be a very important elementary component for control flow models.

The advantages of structured programming have been proven by an experience of 30 years. The advantages of structured BPM will be similar:

- Business process models will become more transparent; therefore, they will be graphically and verbally documented more easily.
- The modification and adaptation of business process models will become easier.
- The optimization of business process models (business process reengineering) will be done in a more efficient way.
- The mapping of business process models to workflow management systems (WFMS) will become more effective.

Bibliography

Böhm, Corrado; Jacopini, Giuseppe (1966). Flow diagrams, Turing machines and languages with only two formation rules, Communications of the ACM, Vol. 9, Nr. 5, 366-371.

Chapin, Ned (1974). New format for flowcharts, Software – Practice and experience, Vol. 4. Nr. 4, 341-357.

Davis, Rob (2001). Business process modeling with ARIS: a practical guide, London.

Eriksson, Hans-Erik (2000). Business modeling with UML, New York.

Holl, Alfred; Auerochs, Robert (2004). Analogisches Denken als Erkenntnisstrategie zur Modellbildung in der Wirtschaftsinformatik [Analogical thinking as a cognitive strategy for model design in information systems], in Frank, Ulrich (ed.). Wissenschaftstheorie in Ökonomie und Wirtschaftsinformatik: Theoriebildung und –bewertung, Ontologien, Wissensmanagement, Wiesbaden, 367-389.

Holl, Alfred; Krach, Thomas (2000). Geschäftsprozessmodellierung und Gestalttheorie [Business process modeling and theory of gestalt], in Britzelmaier, Bernd et al. (ed.). Information als Erfolgsfaktor: 2. Liechtensteinisches Wirtschaftsinformatik-Symposium an der FH Liechtenstein, Stuttgart, 197-209.

Holl, Alfred (1999a). Empirische Wirtschaftsinformatik und evolutionäre Erkenntnistheorie [Information systems as an empirical science and evolutionary epistemology], in Becker, Jörg et al. (ed.). Wirtschafts-informatik und Wissenschaftstheorie: Bestandsaufnahme und Perspektiven, Wiesbaden, 163-207.

Hughes, Joan K. (1987). A structured approach to programming, Englewood Cliffs NY.

Keller, Sven (2000). Entwicklung einer Methode zur integrierten Modellierung von Strukturen und Prozessen in Produktionsunternehmen, Fortschritt-Berichte VDI, Reihe 16 (Technik und Wirtschaft) Nr. 117, Düsseldorf.

Le, Van K. (1978). The module: a tool for structured programming, Zürich.

Linger, Richard C. (1979). Structured programming: theory and practice, Reading, Mass.

MacGowan, Clement L. (1975): Top-down structured programming techniques, New York.

McCracken, Daniel D. (1984): COBOL, Munich.

Nassi, I; Shneiderman, B. (1973). Flowchart techniques for structured programming, ACM Sigplan Notices, Vol. 8, Nr. 8, 12-26.

Rajala, Mikko (1997). A framework for customer oriented business process modeling, Espoo.

Scheer, August-Wilhelm (1994). Business process engineering: reference models for industrial enterprises, Berlin.

Schneyer, Robin (1984). Modern structured programming: program logic, style, and testing, Santa Cruz CA.

Yourdon, Edward (1989). Modern structured analysis, Englewood Cliffs NJ.