Proposal for an Object Oriented Process Modeling Language

Prof. Dr. Reiner Anderl, Dipl.-Ing. Jens Malzacher, Dipl.-Ing. Jochen Raßler

Department of Computer Integrated Design (DiK), Technische Universität Darmstadt
Petersenstr. 30, 64287 Darmstadt
{Anderl, Malzacher, Rassler} @dik.tu-darmstadt.de

Abstract. Processes are very important for the success within many business fields. They define the proper application of methods, technologies and company structures in order to reach business goals. Not only manufacturing processes have to be defined from the start point to their end, also other processes like product development processes need a proper description to gain success. For example in automotive industries complex product development processes are necessary and defined prior to product development.

Over the last decades several product modeling languages have been developed moving to object oriented modeling languages, such as UML, but the used process modeling languages are still procedural. The paradigm shift caused by object oriented description within product modeling languages has to be transferred to process modeling languages.

This paper introduces an object oriented approach for process modeling. Using UML as a starting point an object oriented process modeling method is differentiated. The basic concepts which are needed for process modeling are put into an object oriented context and are explained. The paper also deals with the most important methods behind object oriented process modeling and gives an outlook, what can be achieved by this approach.

Keywords: process modeling, object orientation, UML, modeling language

1 Introduction

All over industrial appliance the necessity of well-defined and powerful processes are known. These processes range from manufacturing processes over business processes even to product development processes. During the last decades they have been analyzed and defined in the according companies.
But within most areas the well-known and defined processes are represented with old methodologies. Discipline-specific methods have been developed to a new level. Applications for that are cross-enterprise collaboration, e.g. in manufacturing or product development networks, and cross-discipline collaboration like mechatronical product development. Within cross-enterprise collaboration the involved companies are not integrated by the means of deliverables anymore, but are integrated in the

complete processes now. The cross-discipline collaboration is similar. It has used to be integration by the means of interfaces and key objectives, but now it is integrated at any time of the process. Both examples lead to two major problems. First of all cross integration is new every time a new collaboration starts. Typically a company is involved in several different collaboration networks at a time. They are all different but principally support the same process. Second problem is that most existing process descriptions are based on procedural process description. These are not powerful enough to meet requirements of describing cross collaboration.
A short example is given to illustrate the problem. Within product development the VDI2221 describes the sequential process of product development that allows some iteration. The ideas behind that process are roughly 50 years old now. Products and with that product development has changed dramatically. Mechatronical product development requires a coordinated development process of several disciplines like mechanics, electrical and electronical devices or software development. Especially software development does not fit properly into the VDI2221 process. With the "Münchner Vorgehensmodell" (MVM) a new approach on a process model for product development was defined using important stages. Depending on the problem every lived process according to the MVM may take its own way between these stages. Still there is no proper description for flexible processes, but processes like MVM or cross collaborations are required.
So a new process modeling language shall meet the following requirements:

1. Support of hierarchical structures
2. Support of flexible interpretation of a defined process without getting incompatible – support of generalization and specification
3. Robust process definition for flexible proceeding sequences of activities without losing process comparability – support of interchangeability of processes
4. Support of different integration scenarios and levels without changing process description at any time.—support of flexibility of processes
5. Easy to learn and read – audience of those process definitions are very broad

Comparing the requirements to the paradigm change in information modeling, which is caused by the introduction of object orientation, a similar approach seems straightforward for the progress of powerful process description. In this paper some existing procedural oriented process modeling languages and those who call themselves object oriented will be mentioned and a new object oriented process modeling language will be introduced. A conclusion closes this paper.

2 Existing Process Modeling Languages

In this chapter some existing process modeling languages are mentioned. These are IDEF0/SADT, Event-driven Process Chain (EPC), process modeling with UML, Business Process Modeling Notation (BPMN), Integrated Enterprise Modeling, Process Specification Language (PSL) and processes with the Semantic Object Model. These are not all process modeling languages but seem to be the most important. A short statement why these languages do not meet the requirements of modern process definition is included.

2.1 IDEF0 / SADT

IDEF0 is a procedural process modeling language. It explicitly supports hierarchic definition of processes. Complex processes can be defined in different level of details. Each activity can be detailed as own sub process. [1], [2]
As it is a procedural language its descriptions are not very flexible regarding changes in the proceeding sequence of activities. IDEF0 remains on unchangeable sequences. It is easy to read but it tends to get very complicated on complex processes. Except the support of the hierarchy IDEF0 meets no mentioned requirements.

2.2 Event-Driven Process Chain (EPC)

Event-driven Process Chain is a procedural process modeling language. Compared with IDEF0 it has more objects in the language. EPC supports events and activities between events. For the process flow EPC allows explicit branching and aggregation of processes. Furthermore there are additional objects, which support the process. These are information and/or resource objects, persons and/or organizations. [3], [4]
EPC is a procedural language supporting different level of details. Compared with IDEF0 it is more powerful to support different integration scenarios and levels. EPC is not very flexible regarding to changes in the proceeding sequence of activities. It is easy to read but it tends to get complicated on complex processes. Therefore EPC meets only two of the mentioned requirements.

2.3 Process Modeling with UML

The Unified Modeling Language (UML) offers an all spanning modeling language. Regarding data and information model the language is object oriented. Looking at the process modeling capabilities of UML the processes are still described in a procedural way. Most important process diagrams within UML are the activity, state chart and the sequence diagram. Each of the process diagrams shows the process in exactly one instance. For each instantiation the object oriented nature of the data model allows different processes. But the processes can not be described in a generic way within these diagrams. Solely the use case diagram seems not to fit into that. There some kind of process understanding is modeled in an abstract way. [5], [6], [7]
UML is not an object oriented language for process modeling. Each activity is seen as an object. Relations between activities still base on logical states. Processes defined with UML are not very flexible regarding changes in the proceeding sequence of activities. Like EPC process modeling with UML supports different level of details and different integration scenarios and levels. It is quiet easy to read and to handle. Therefore process modeling with UML meets three of the mentioned requirements.

2.4 Business Process Modeling Notation (BPMN)

The Business Process Modeling Notation (BPMN) was developed to provide a notation for process descriptions. The specification includes the visual appearance of

the elements and the semantics of the elements. Furthermore it deals with the exchange of process definition either between tools or as scripts (e. g. mapped on the Business Process Execution Language). [8], [2]
BPMN representation of processes is quite similar to the UML activity diagram. Processes are defined as a sequence of activities in swim lanes. Again it is a state based connection between object oriented activities. Therefore the verdict upon BPMN is in this case similar to the UML verdict. BPMN meets three of five mentioned requirements.

2.5 Integrated Enterprise Modeling

The Integrated Enterprise Modeling was developed out of SADT. It capsules activities as objects and adds static information as job, product or resources. Due to this further information it is possible to generate views on the complete enterprise, not only on its processes. [9], [2]
Integrated Enterprise Modeling represents processes in a SADT kind of style. Due to its retaining on logical sequence of activities it has no real advantage in modeling flexible processes. It still lacks a powerful support of process flexibility.

2.6 Process Specification Language (PSL)

The Process Specification Language (PSL) is a neutral language for process specification to serve as an interchange language to integrate multiple process-related applications throughout the manufacturing process life cycle. As it only defines itself in informal manner it has no formal and graphical constructs. Therefore it is not capable for process modeling and a big audience. [10]

2.7 Semantic Object Model

The Semantic Object Model methodology is an enterprise architecture, which allows to divide an enterprise model into the model layers enterprise plan, business process model, and resources, each of them describing a business system completely and from a specific point of view. Within the process model the activity objects are connected with events. This concept allows flexible and robust process modeling. Out of this diagram the interaction scheme and the task-event scheme are developed. [2] Within interaction scheme relations also seem to be object oriented, but not within task-event scheme. So both worlds seem to be mixed up. Due to the integrated approach of enterprise plan, process and resources constructs are difficult to understand.

2.8 Short Statement on the Languages

Looking on the process modeling languages above we see that most of them already think in object oriented activities. Here paradigm changes already carried out. The definition of relations mostly remain on state based, proceeding sequences of

activities. Only SOM seems to be beyond that, but is not clarified well. To meet all requirements a new approach shall be started.

3 Proposal on an Object Oriented Process Modeling Language

The previous chapter has summarized the known and used process modeling languages and their limitations regarding the requirements introduced in chapter 1. Therefore a new approach for a process modeling language will be introduced in this chapter, which uses object oriented techniques and hence meets all requirements.

3.1 Towards an Object Oriented Process Modeling Language

UML is a well known and widely used modeling language for large software systems that uses object oriented techniques to obtain modularization, software reuse, flexibility and easy maintaining, among others. Expansions, such as SysML, introduce additional methods to use UML in other contexts than software engineering. Also the development of BPMN shows that UML is a technology that has a wide acceptance by users, developers and managers. Thus UML is a good starting point for the development of an object oriented process modeling language.
Fig. 1 shows the definition of a UML class diagram. The first field shows the class name, the second field lists the attributes, and the third field lists the methods, which can be used within the context of the class. The class itself is time invariant as it is a generic description of the content of the context. But the instance of a class, an object, is time variant, because it holds characteristic values that can be checked to given times and can change over time. This means, the values can change, but the general structure of an object (number and kind of attributes) can not change.

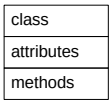

Fig. 1. UML class diagram

Having a time variant object it can be derived by time regarding to [11]

$$\lim_{T \to T_0} \frac{Object(T) - Object(T_0)}{T - T_0} = \frac{dObject}{dT} = \dot{O}bject\,. \tag{1}$$

Equation (1) shows that the content of an object, which means the attributes of an instance of a class, may change over time. Given a rule to change the attributes of an object one can express the change of the object's content as a process instance, which is shown in (2). It is necessary to mention that we use a discrete time *T* instead of continuous time *t* to implement "time steps". This is due to the result of the derivation as different process instances may need different time intervals to execute.

$$\dot{O}bject = \text{Process instance} \tag{2}$$

As we have derived the object we now have to derive the object's content. Fig. 1 uses the word attributes as defined in UML, in equation (3) we will derive the attributes, but using the word information to make the meaning clearer and more generic.

$$\lim_{T \to T_0} \frac{Information(T) - Information(T_0)}{T - T_0} = \frac{dInformation}{dT} = \dot{I}nformation \tag{3}$$

The derivation of information shows that the information may change over time. So the change of information, the change of attributes or data can be expressed as a method, which is shown in (4).

$$\dot{I}nformation = Method \tag{4}$$

The last field of a UML class diagram and thus in the object holds the methods, which act on the attributes. In the following we use the term operation for UML methods to differentiate between UML and our introduction. Operation and the just derived method are quite similar and are the same in several cases. In the following we derive the operation, which is shown in equation (5).

$$\lim_{T \to T_0} \frac{Operation(T) - Operation(T_0)}{T - T_0} = \frac{dOperation}{dT} = \dot{O}peration \tag{5}$$

The meaning of the derivation of an operation is quite complex. To express this mathematically we can use equations (3) to (5), which show, that dOperation / dT is the first derivation of an operation or the second derivation of information. This means dOperation / dT is the gradient of an operation or the curvature of information. The expression gradient of an operation seems quite handsome and opens the question: what does result in the change of an operation? Or, more exact, what does result in a change of the quality of the execution of a method? Think also of the similarity of operation and method. This question directly leads to the answer to the problem, which is

$$\dot{O}peration = \text{Resource}. \tag{6}$$

Resources influence the execution of an operation. The use of more or less resources leads to faster or slower execution, influences the quality of the output, may lead to more innovation and so on.

Equations (1) through (6) have shown the derivation from a time variant object to a time variant process instance. Generalizing the process instance we get a process class, which again is time invariant. The diagram of a process is shown in Fig. 2.

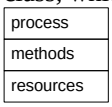

Fig. 2. PML Process class diagram

Further we introduce the term PML, which stands for Process Modeling Language and can be seen as an extension to UML, as SysML is. Thus the known techniques of inheritance, association, and cardinalities can be used. Implementing those techniques

processes can be modeled hierarchically with modularization, structure, exchangeability, and reusability. Hence all the requirements described in chapter one are fulfilled.
A last topic of the class diagrams that have to be covered are assurances, which is done in the following in a qualitative way. In UML assurances can be defined to guarantee the co-domain of attributes. The assurances are conditions or constraints that have to be met by methods changing those attributes. In PML those assurances are important too, but apply to methods. This is obvious in the derivation's context.
Further, a condition can be seen as a constant signal. With the beginning of the life time of an attribute, which is the same as the life time of an object, the condition starts and remains constant over time. That is, the condition must hold for the life time of the attribute. Deriving the constant signal is straightforward using a Fourier transformation. The transformation results in the delta function; its derivation is a constant frequency signal. Back transformed to the time domain the result is a Dirac impulse [12], which can be interpreted as an event.
The event actually can be a condition becoming true, the trigger from a finished method, or information becoming available.
Thus the assurances are derived too and can be used for process modeling, which is shown in Fig. 3.

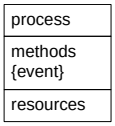

Fig. 3. PML class diagram with assurances

3.2 Meaning of PML

Above we have shown mathematically the derivation of PML. We have introduced the terms of process instance, which can also be called project, and the process as a generic class description. We now want to clarify those terms and their meaning.
Fig. 4 shows the used way to derive PML. Starting from the time invariant UML class we have instantiated a time variant object. This is derived by time and leads to a process instance, or project, which is time variant, and finally generalized to a time invariant process.

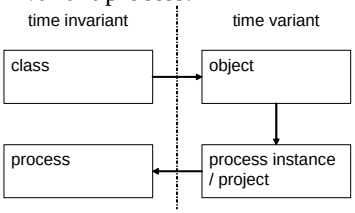

Fig. 4. Derivation loop of PML

Looking to application level the meaning of all four constructs will be clearer. The generic class model is used to represent the data model in a PDM system, e.g. as STEP AP214. Within one project the class is instantiated and the object holds the

actual data of the designed model. The class therefore describes the product in a generic way, while the real contents are stored in its instantiation.
The same is true on process level. The PML process class describes the process in a generic way. It allows one to define all methods with assurances and resources needed for the process. The instantiation of a process is a project. This means, the instance of a process defines the current occurrence of resources, used data models etc.
This not only leads to a paradigm change in process modeling, but also in the view to processes and projects.

3.3 Using UML Constructs in PML

In this chapter we will give a short overview of using UML constructs within PML to gain the capability of hierarchical and modular modeling.

Inheritance

The concepts for inheritance of process classes follow the notation of standard UML classes. Fig. 5 shows the inheritance of process classes. Starting with a generic Creativity Process that has a Creativity Method and requires a problem becoming available as event, but no Resource, two subclasses are inherited. The Intuitive Process, which adds two Resources and the TRIZ Process, which adds one Resource. Both subclasses inherit the Creativity Method from their superclass Creativity Process, TRIZ Process overwrites the Creativity Method with its own technique of creativity. Brainstorming and Brainwriting are subclasses from Intuitive Process, inheriting the Resources and the Creativity Method, which they overwrite. Both of the subclasses define their own additional Resource. Creativity Method takes an argument, problem, which can have assurances, which means starting and ending conditions can be defined.

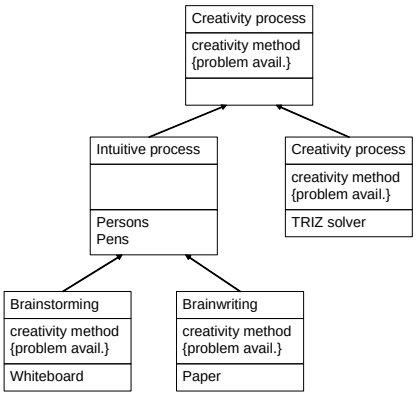

Fig. 5. PML inheritance diagram

The process class also supports abstract methods, as well as public and private methods.

Associations

The known concepts of associations from UML classes can be used for process classes. Fig. 6 through Fig. 8 show the concepts of associations, aggregations, and

compositions. All associations can use cardinalities to implement the number of relations they use.
The techniques introduced above enable generic process modeling that supports structural and hierarchical modeling, including modularization and flexible design. As classes are time invariant no statement about the "running time" is made, e.g. about sequential or parallel process execution. The actual execution is determined instantiating the process classes and can be further described within the project with activity diagrams (logical description) or sequence diagrams (timely description). State diagrams can be used to describe the project states at given times. These instantiation diagrams are quite similar to the state-of-the-art object oriented process modeling languages.

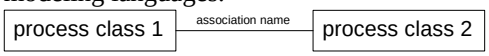

Fig. 6. Association

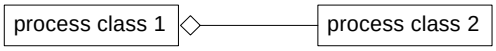

Fig. 7. Aggregation

Fig. 8. Composition

3.3 An Example

To illustrate our way for process modeling we prepared a short example coming from manufacturing. We have a small enterprise which is focused on shape cutting manufacturing. Its production process description is shown in Fig. 9. Note that the process looks very similar to an UML class diagram.

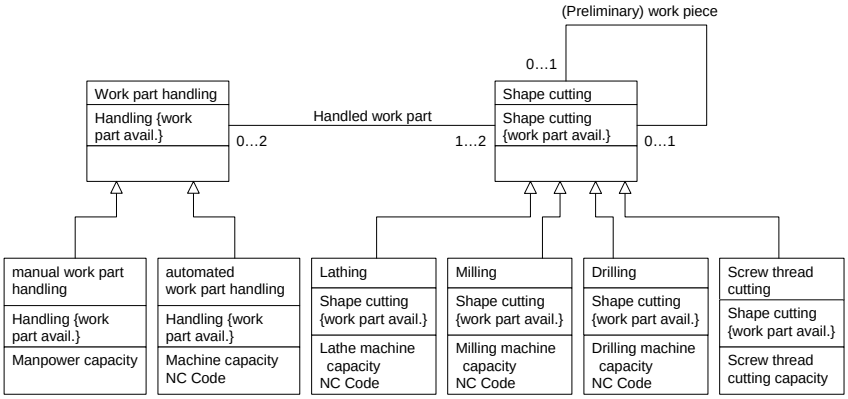

Fig. 9. Example process diagram for a small enterprise specialized on shape cutting

There are two main processes which are connected. The within the enterprise adopted technologies inherit their attributes from them. Let us assume the enterprise wants to manufacture the product shown in Fig. 10.

Fig. 10. First example product

In the first manufacturing lot 30 pieces of that product are produced. Due to some reasons of machine and technician availability and lot size the project is instantiated as shown in Fig. 11.

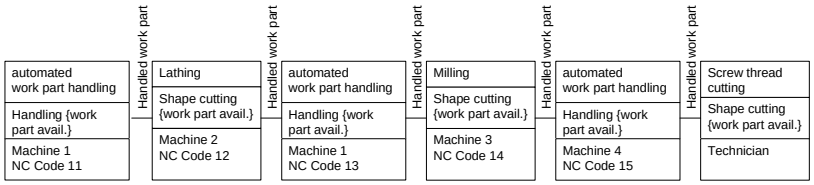

Fig. 11. Example process diagram for a small enterprise specialized on shape cutting

After the first lot the enterprise wants to produce a second lot of five more pieces. Therefore they changed the used machine (see Fig. 12). Please note it is still the same product and the same process. The instantiation only has changed. It is a new project.

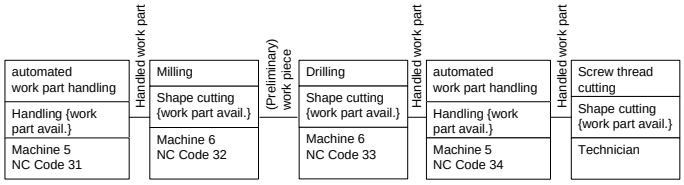

Fig. 12. Example process diagram for a small enterprise specialized on shape cutting

The enterprise produces also another product as shown in Fig. 13.

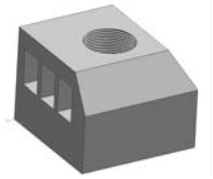

Fig. 13. Example process diagram for a small enterprise specialized on shape cutting

For production of a certain lot the project shown in Fig. 14 is used. Now we have a different product from a different instantiation based on the same process description.

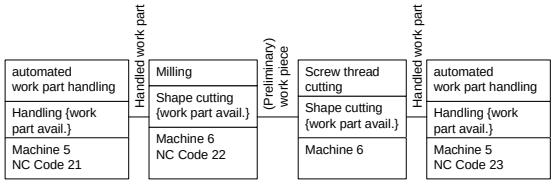

Fig. 14. Example process diagram for a small enterprise specialized on shape cutting

In this example it became obvious how powerful process modeling and description with PML is. Once defined we can instantiate different processes leading to same or different results.

3.4 Further Topics to be Mentioned

In the available paper we only have used class diagrams and instances to show our concept. The other UML diagrams seem to fit for PML. The derivation and usage of those diagrams will be covered in future works.
The most important diagrams, and those which are in widest use in UML, are sequence, activity and state diagrams. State diagrams can be directly used to show the state of a process instance. Sequence diagrams are more complex and show the running time of a process instance in a given occurrence. That means, it shows which sub processes run parallel or sequential, the instance' life time, etc. We have started working on this topic and it seems obvious using discrete Fourier transformations or z-transformations to derive sequence diagrams from instance diagrams using time based states of the project. Activity diagrams follow a similar derivation, but the goal is to get a logical description instead of a time-dependent one of the project running time.
PML and UML can be linked together using two techniques. The data classes – data which is generated along the value creation chain – can be used as input for methods or be written to the edges of associations. If company structures, resources, and similar information are modeled, this may best fit as associations for the resources. Further, there exists the attributed association, which can be used for explicit cross modeling between PML and UML.
Process management will underlay an enormous change, since process management mainly reduces to process modeling. For existing processes this means that changes in the generic process description leads to extensions of the process description using e.g. inheritance to specialize or modify given processes. Another important topic is project management. It is obvious that project management directly influences the process instances. This implies the timely activities and the allocation of resources.

4 Conclusion

The strength of the shown approach for process modeling is the complete object oriented view to processes and the differentiation and linkage of and between processes and projects. As in data modeling process modeling can now be done in a generic way. The introduced process description perfectly fits into PDM systems with the process class descriptions. Hence process management is now process modeling at running time. A process in a PDM system can be extended by more classes, that extent existing classes, or specialize them. The instances of those processes are used in projects, which define the parameters of the instances. The implemented technique of processes and projects within PDM systems is then similar to data models, where object orientation has been a standard since years. Further works will focus on the

topics of process and project management and will introduce examples of how to use the object oriented process modeling approach to implement real world projects.
The object oriented approach of process modeling introduces a paradigm change not only to the view of process and project management, but also enables new possibilities for interoperability. Heavy use of modularization enables exchangeability and process reusability and hence strengthens the integration of third-party processes. This leads to more powerful cross-enterprise collaboration.
Another important point is the certification of processes. Depending on products or customers it is necessary to have certified processes. Think of ISO 9000 or certification for medical issues. With PML the process is only certified once but can lead to different instantiations – regardless to the project (in terms of same or different product).
Summarizing the development of PML we have explained existing process modeling languages, which call themselves object oriented, but this is only true for the modeled activities. Therefore the modeled processes still look like sequences of activities. In this paper we have developed a new approach. By deriving time-dependent objects we got process instances. Taking them alone we do not use the complete power of object oriented modeling. The step from process instances to process classes helped us to define a new object oriented process modeling language. As processes are not modeled on instances anymore, but as abstract classes, we have a completely new representation of processes. Running processes then are projects. With this definition we need a new comprehension on process and project management.

References

1. IEEE Std 1320.1-1998. IEEE Standard for Functional Modeling Language—Syntax and Semantics for IDEF0. New York: IEEE, 1998.
2. Bernius, P.; Mertins, K.; Schmidt, G. (Eds): Handbook on Architectures of Information Systems, 2nd Edition. Springer Verlag Berlin, Heidelberg (2006)
3. Scheer, A.-W.: ARIS – Business Process Frameworks, 2nd Edition, Berlin, 1998
4. Scheer, A.-W.: ARIS – Business Process Modeling, 2nd Edition, Berlin, 1999
5. OMG: Unified Modeling Language: Superstructure v2.1.1, of Feb 2007, www.omg.org, 2007
6. Eriksson, H.-E.; Penker, M.: Business modeling with UML: business patterns at work. John Wiley & Sons, Inc, New York (2000)
7. Burkhardt, R.: UML – Unified Modeling Language: Objektorientierte Modellierung für die Praxis. Addison-Wesley-Longman, Bonn (1997)
8. OMG: Business Process Modeling Notation Specification, of Feb 2006, www.omg.org, 2006
9. Spur, G.; Mertins, K.; Jochem, R.; Warnecke, H.J.: Integrierte Unternehmensmodellierung Beuth Verlag GmbH (1993)
10. International Standards Organization (ISO): ISO 18629 Series: Process Specification Language of 2004, www.iso.org, 2004
11. Luh, W.: Mathematik für Naturwissenschaftler, Bd.1, Differentialrechnung und Integralrechnung, Folgen und Reihen, Aula, Wiesbaden (1987)
12. Clausert, H., Wiesemann, G.: Grundgebiete der Elektrotechnik 2. Wechselströme, Leitungen, Anwendungen der Laplace- und Z-Transformation, Oldenbourg, München (2000)