PML, an Object Oriented Process Modeling Language

Prof. Dr.-Ing. Reiner Anderl [1], **and Dipl.-Ing. Jochen Raßler** [2]

[1] Prof. Dr.-Ing. Reiner Anderl, Germany, anderl@dik.tu-darmstadt.de
[2] Dipl.-Ing. Jochen Raßler, Germany, 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, tools and company structures in order to reach business goals. Important processes to be defined are manufacturing processes or product development processes for example to guarantee the company's success. Over the last decades many process modeling languages have been developed to cover the needs of process modeling. Those modeling languages have several limitations, mainly they are still procedural and didn't follow the paradigm change to object oriented modeling and thus often lead to process models, which are difficult to maintain. In previous papers we have introduced PML, Process Modeling Language, and shown it's usage in process modeling. PML is derived from UML and hence fully object oriented and uses modern modeling techniques. It is based on process class diagrams that describe methods and resources for process modeling. In this paper the modeling language is described in more detail and new language elements will be introduced to develop the language to a generic usable process modeling language.

Keywords: process modeling language, PML, UML

1. Introduction

As the tendency of enterprises to collaborate growths steadily, industry faces new challenges managing business processes, product development processes, manufacturing processes and much more. Furthermore, discipline spanning product development processes are increasing, e. g. desired mechatronical products are in the need for knowledge from mechanical, electrical as well as software engineers. Humanists and economists also play a huge role in modern product development processes. Each individual discipline has its own, well-defined and specific processes, which typically are based on well-tried methodologies. These process descriptions are very powerful within the traditional

discipline or the original enterprise, they were invented in. On the other side, they lack for flexibility, due to the reason that most existing process descriptions are based on a procedural approach. These are not powerful enough to meet requirements of describing cross collaboration. In particular OEMs challenge the integration of suppliers. Suppliers have different levels of access to the OEMs data base, data exchange is handled based on integration level. Furthermore the levels supplier get differ between suppliers and projects.
Taking everything into account, the need for a process modeling language that meets the above shown requirements is obvious. Not only must the different disciplines be supported, but also cross enterprise collaboration, as well as supplier integration. Still there is no proper description for this kind of flexible processes descriptions. To meet all these needs a new process modeling language is developed and demands the following requirements:

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

This paper summarizes the previous work done in defining a new process modeling language – PML – and introduces new aspects of the language. Although the development of PML isn't yet finished within the context of this publication, the process modeling language reaches a state, where it can be started to use in a productivity environment. A conclusion closes this paper.

2. Existing Process Modeling Languages

In this chapter some existing process modeling languages are covered. It is briefly described why they do not meet the requirements of modern process definitions. For an in depth analysis and further details we refer to [1].
IDEF0 / SADT and Event Driven Process Chains (EPC) are procedural process modeling languages and support modeling processes with different levels of details. Both process modeling languages lack for transparency and clarity if they are applied to complex processes. Moreover, they are not very flexible regarding changes to the proceeding sequence of activities. [2, 3, 4, 5]
The Unified Modeling Language (UML) offers an all spanning modeling language. Regarding data and information model the language is object oriented. If UML is utilized to describe processes, UML reveals several disadvantages. UML is not an object oriented language for process modeling, because processes

are still modeled procedural. 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. [6, 7, 8, 9]
Business Process Modeling Notation (BPMN) representation of processes is quite similar to the UML activity diagram. It is a standardized graphical notation for drawing business processes in a workflow. Processes are defined as a sequence of activities in swim lanes. Again it is a state based connection between object oriented activities. Therefore the evaluation result upon BPMN is similar to the UML verdict. [10, 3]
The Integrated Enterprise Modeling as a refinement of SADT enables users to generate views on the complete enterprise, not only on its processes. Processes are still 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. [11, 3]
The Process Specification Language (PSL) basically is an ontology for describing processes. As PSL's objective is to serve as an Interlingua for integrating several process-related applications without formal and graphical constructs, it is therefore not capable for process modeling. [12]
The Semantic Object Model (SOM) methodology allows flexible and robust process modeling, based on the division of an enterprise model into several model layers, each of them describing a business system from a specific point of view. Within the process model the activity objects are connected with events. In comparison, SOM is most progressive regarding the definition of relations, but its constructs are difficult to understand due to the complex, integrated approach. [3]
The modeling languages still describe relations on state based, proceeding sequences of activities. Taken together these results evoke the need for a new process modeling language facing the requirements of the paradigm change. [1]

3. Basic concepts of the Process Modeling Language

A new approach for a process modeling language has been introduced in [1], which uses object oriented techniques and hence meets all requirements. This approach uses the well known and widely used modeling language UML, that applies object oriented techniques to obtain modularization, reuse, flexibility and easy maintaining, among others, in the field of software and system modeling. Ongoing developments on the basis of UML, like SysML, prove the sustainability of the UML metamodel. Thus UML is a good starting point for the development of an object oriented process modeling language.
Fig 1 shows the definition of an UML class diagram including class name, attributes and methods.

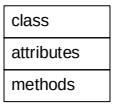

Fig 1: UML class diagram

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.

Having a time variant object it can be derived by time regarding to [13]

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

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). Note 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

$$\dot{O}peration = \text{Resource} \qquad (6)$$

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**.

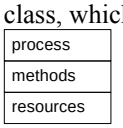

Fig 2: PML Process class diagram

UML uses assurances to guarantee the range for its attributes. We need assurances too, but not as static ranges. Deriving a static value by time normally leads to zero. But knowing that the integral of a delta impulse δ(t) is defined as 1 [14] and we derive the constant with this definition in mind, the derivation of the static assurance leads to the delta impulse, which can be interpreted as an event. This means, an assurance for the processes is an event, a constraint becoming true, a set of data becoming available, time is elapsed and so on.

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.

[1, 15] show 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. 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.

Regarding to connections and dependencies between single process classes, PML features the well known UML-concepts of inheritance and associations. The concepts for inheritance of process classes follow the notation of standard UML classes through simple object-oriented means like generic super-processes, sub-processes, overwriting and inheritance of methods and resources. Structural and

hierarchical modeling is supported by using the concepts of associations, aggregations, and compositions and the usage of cardinalities. [1, 15]
Now we want to illustrate the capability of PML means by a complex product development process and a manufacturing process. This example application stresses out the advantages of PML regarding flexibility of defined processes, reusability, clarity and understandability. In [15] we have introduced the product generation process of integral sheet metal parts with higher order bifurcations. **Fig 3** illustrates the process model as an example for the strength of PML. For details of this algorithm driven development process we refer to [15, 16, 17].

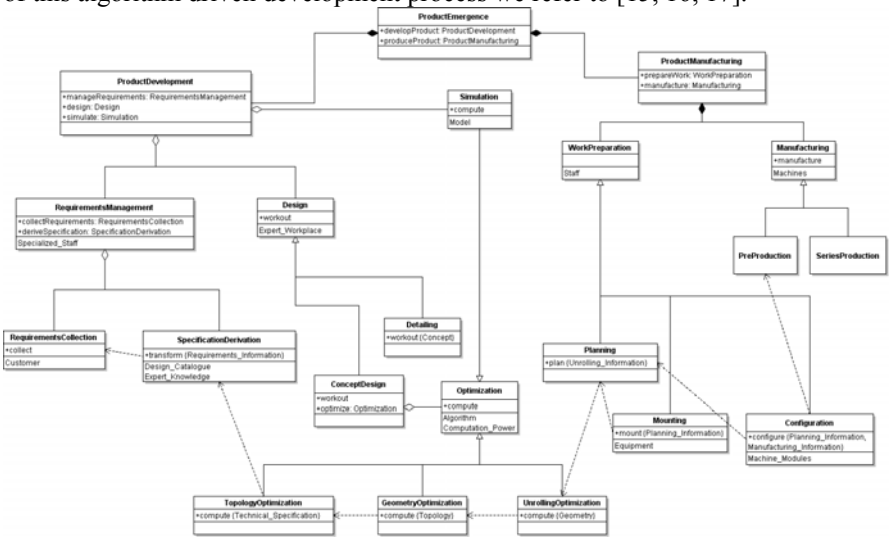

Fig 3: Process diagram for product emergence of integrated sheet metal products with higher order bifurcations

As the complete process model is expressed in PML, the generic process description remains at a level of utmost flexibility and is clearly structured. This enables all projects dealing with the product emergence of integral sheet metal products to be modeled with this generic process model by instantiating it. The model itself does not alter through instantiating it and remains unchanged. An instantiated process embodies exactly one project representing a specific integral sheet metal product.
UML supports instance diagrams, which basically are class diagrams with the instances built in to show the relations between instances and classes. The instance diagram additionally shows the actual object's occurrence and hence the used resources in our process models. As instance diagrams are not very handsome for complex models we will not use them here. Instead we use other diagrams to show the instances and – more interesting to processes – their timely and logical occurrences. These are the sequence diagram and the activity diagram respectively. As announced in [1] it is possible to derive sequence and activity diagrams with PML.

Fig 4 illustrates the activity diagram for the emergence of an integral sheet metal product. In the UML, an activity diagram represents the logical workflows of components in a system and shows the overall flow of control. This approach also fits for PML.

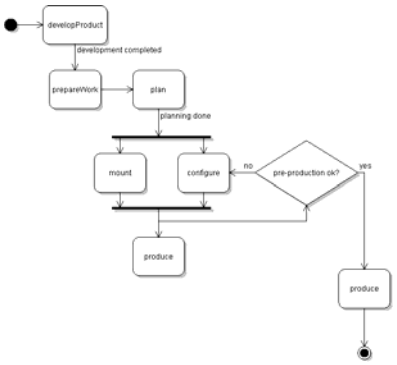

Fig 4: Activity diagram of manufacturing process

The general meaning of a PML activity diagram is the same as in UML. The black circle in the upper left corner shows the starting point, the black circle in the lower right corner shows the end point. Activities are modeled as rectangles with rounded corners, decisions are modeled as a rhombus. Straight horizontal lines are used to show the splitting of the process flow or the synchronization of processes. Note that in this example only a small extract of the above shown development and manufacturing process is illustrated.

Another way to show the instances is by using sequence diagrams. **Fig 5** shows the appropriate sequence diagram for the manufacturing process. As in UML the sequence diagram shows the life time of objects with its construction and destruction events and signals.

Fig 5 shows the current instances with instance names and the corresponding classes they are instantiated of. *RollerTrailManufacturing* is active for the whole manufacturing process and activates different sub-processes as planning, mounting, configuring, and the series production. The pre-production is constructed and controlled by the configuration to allow iterations to optimize the machine configuration.

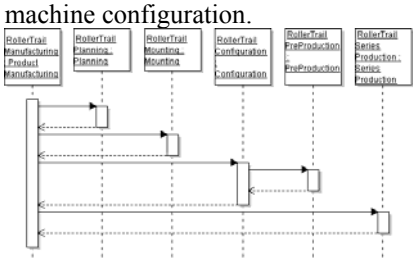

Fig 5: Sequence diagram of manufacturing process

To show the real strength of PML **Fig 6** details the manufacturing processes. *Manufacturing* consists of *SplittingProcess*, which has three sub-processes:

LinearFlowSplitting, *LinearBendSplitting*, and *LinearBending*. To instantiate *Manufacturing* different *SplittingProcesses* are instatiated. But, using the concept of object orientation, not *SplittingProcess* but the sub-processes will be instantiated. The same is true for the resources in *Manufacturing*. Machines should hold the specialized machines, which are linear flow splitting machines, linear bend splitting machines, and linear bending machines. Thus having a generic process description, the product and its manufacturing process is dependent of the used machines, the splitting processes and their order in the manufacturing line.
The method names *manufacture* are used in the process classes *Manufacturing*, *SplittingProcess*, and its sub-processes. The *manufacture* method in the *SplittingProcess* and its sub-processes can just be implemented and used, but the *manufacture* method in *Manufacturing* has to be implemented on its own in the actual instance to specify the manufacturing process for the actual product.

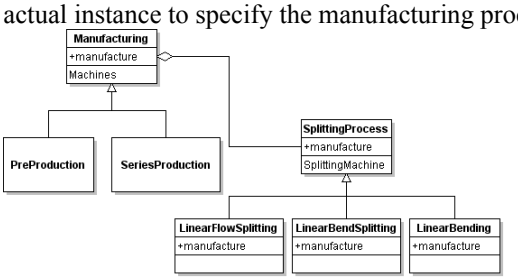

Fig 6: Detailed manufacturing processes
So the production depends on the resources and the order of the usage of the resources. Hence different products can be produced using the same generic process model.

4. Additional Aspects of PML

We have shown the derivation and usage of PML in the previous chapter. Now we want to integrate other UML concepts in the context of PML and, to be continuous, give a mathematical explanation of the concepts.

Activity Diagram
In the previous chapter and in [15] we have used the activity Diagram, but without a mathematical description. As one can see in **Fig 4** the concepts of activity diagrams use Boolean descriptions to model the activities. An activity gets started if the result of the previous activity gets true, e.g. has finished. Thus the model is based on Boolean states.
Decisions use one input and two outputs in their process flow. The input is again triggered by the result of an activity becoming true. The two outputs can be seen as the decision is true or false, expressed with variable x, the output is similar to x or $\overline{x}$.

Synchronization lines can be expressed using the Boolean symbol *and* ($\wedge$) to model that all incoming events have to be true or with the Boolean symbol *OR* ($\vee$) to model that at least one of the inputs has to be true. Another possibility is the Boolean operator *XOR* ($\oplus$) to model that exactly one input has to be true and all other false.
Hence the activity diagram can be expressed mathematically using Boolean expressions. Discussing the Boolean expressions in the context of the timely derivation one can see that UML uses the activities of its classes in this diagram. The same is true for PML. The only difference is the used field of UML's or PML's class description, regarding to **Fig 1** and **Fig 2**. UML methods are in the third field, PML methods are in the second, central field.

Sequence Diagram
The sequence diagram lacks the mathematical description too. This is introduced in the available paragraph.
The sequence diagram uses objects, which are instances of process classes. This means the sequence diagram uses time variant objects and therefore is time dependent. This makes sense as the sequence diagram doesn't model a process but a given project.
Another important aspect of processes is that they do not necessarily converge. Think for example about product development. There may be a set of requirements for the new product that lead to a dead end development or a very expensive one that is stopped to save money for the company. Thus a process may diverge. Knowledge about the convergence of processes and the discrete time steps make it obvious to use z-transform [14] to describe sequence diagrams.
Using the example of the previous chapter shown in **Fig 5** the sequence can be written as

$$\begin{aligned} y(z) = {} & a(z) + a(z-1) + b(z-1) + a(z-2) + c(z-2) + \\ & a(z-3) + d(z-3) + a(z-4) + d(z-4) + e(z-4) + \\ & a(z-5) + d(z-5) + a(z-6) + f(z-6) + a(z-7) \end{aligned} \qquad (7)$$

Equation (7) uses the letters *a* to *f* for the process instances and *y* for the result to enhance the readability. Process instance *a* is active for the life time of the example, starting all other process instances except of *e* which is started from *d*.

Interaction Diagrams
In the UML exist 4 types of interaction diagrams. These are the sequence diagram, the interaction overview diagram, the communication diagram, and the timing diagram [9]. The sequence diagram has just been described mathematically, all other interaction diagram types can be handled similar, but they show different aspects of interaction within the running time of a process instantiation. Thus we pass the in depth view to these interaction diagrams.

State Machine Diagram

The state machine diagram in UML shows the actual state of time variant objects to given times t_0. The same is true in PML. The state machine diagram shows the actual state of a given process instance *P(T)* at a given time T_0. This can be written as

$$StateMachineDiagram \mapsto P(T_0) \quad (8)$$

Package Diagram

The package diagram is a structural diagram. It clusters processes and bears the capability to organize processes particularly for modularization and reuse.

Package diagrams can be described using the set theory. The membership operators *element of* ($\in$) and *subset of* ($\subseteq$) can be used to describe the relations of processes and packages to subordinate packages. The *union operator* ($\bigcup$) can be used to cluster processes and packages into a subordinate package. **Fig 7** illustrates this concept.

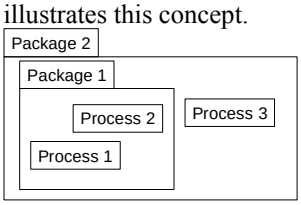

Fig 7: PML Package Diagram

Use Case Diagram

Use case diagrams get a special meaning within the context of PML. In UML use case diagrams are mainly used to model the context of the system to be developed. Thus the use case diagram can be seen as a diagram to model requirements.

Although the use case diagram in PML can be used to model requirements for the processes to be developed, it gets its strength as a documentation tool for the processes or projects as instances of processes.

To make this more understandable, we introduce an example for quality management. The ISO 900x certification is approved on a certain process. This means a company describes a quality management process in a certain context and asks for approval. If the same company deploys products in a different context they need to describe the quality management process again and ask for approval once again. If the ISO 900x certification process is described in a generic way using PML it only needs approval once and can use this process for different projects in different contexts, using different parameters for instantiating the processes or specializing some process classes. Thus modified projects can be instantiated or enhanced without losing compatibility to the approved generic process.

To document the instantiation of processes use case diagrams can be used to describe reference instantiations and interactions of projects and sub projects without actual instantiation of a project.

5. 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. 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).
Future work on PML will cover many important topics. Using PML to model more real world processes and using it for productivity projects will prove the usability of this new modeling language. Missing components will be added to complete PML. Also the formal description of PML, regarding to UML, has to be enhanced and will be covered in future work.
Most important concepts that are still missing are process and project management. Using PML process and project management get new meanings. Thus the meaning of process and project management has to be redefined in the context of PML and new management methods have to be developed.
To apply PML for productivity it may be very interesting to develop a model to map the PML process class diagrams into PDM (Product Data Management) systems and use the instantiation for actual projects within product development. UML tools have the capability to generate source code from the class diagrams. Future work will cover the possibilities to generate PDM descriptions from PML models to map processes into engineering tools.
This paper has shown the concepts of the new process modeling language PML. Deriving PML mathematically from UML led to a process model that supports object oriented process modeling capabilities. Thus the requirements for a modern process description language have been fulfilled, such as modularization, exchangeability, cross enterprise collaboration, easy maintenance, enhance ability and many more.

This paper has introduced and discussed many new diagram types that are known from UML. The usage of PML has been shown with a complex example that illustrates the strength of this process modeling language. Thus the basic work for the usage of PML in productivity has been done.

References

1. Anderl R, Raßler J, Malzacher J: Proposal for an Object Oriented Process Modeling Language, Proceedings of the 4th International Conference on Interoperability for Enterprise Software and Applications (I-ESA), March 2008, Berlin, Germany.
2. IEEE Std 1320.1-1998. IEEE Standard for Functional Modeling Language—Syntax and Semantics for IDEF0. New York: IEEE, 1998.
3. Bernius P, Mertins K, Schmidt G (Eds): Handbook on Architectures of Information Systems, 2nd Edition. Springer Verlag Berlin, Heidelberg (2006)
4. Scheer A-W: ARIS – Business Process Frameworks, 2nd Edition, Berlin, 1998
5. Scheer A.-W: ARIS – Business Process Modeling, 2nd Edition, Berlin, 1999
6. OMG: Unified Modeling Language: Superstructure v2.1.1, of Feb 2007, www.omg.org, 2007
7. Eriksson H-E, Penker M: Business modeling with UML: business patterns at work. John Wiley & Sons, Inc, New York (2000)
8. Burkhardt R: UML – Unified Modeling Language: Objektorientierte Modellierung für die Praxis. Addison-Wesley-Longman, Bonn (1997)
9. Booch G, Rumbaugh J, Jacobsen I: Das UML Benutzerhandbuch, Addison-Wesley Verlag, München, 2006
10. OMG: Business Process Modeling Notation Specification, of Feb 2006, www.omg.org, 2006
11. Spur G, Mertins K, Jochem R, Warnecke HJ: Integrierte Unternehmensmodellierung Beuth Verlag GmbH (1993)
12. International Standards Organization (ISO): ISO 18629 Series: Process Specification Language of 2004, www.iso.org, 2004
13. Luh W: Mathematik für Naturwissenschaftler, Bd.1, Differentialrechnung und Integralrechnung, Folgen und Reihen, Aula, Wiesbaden (1987)
14. Clausert H, Wiesemann G: Grundgebiete der Elektrotechnik 2. Wechselströme, Leitungen, Anwendungen der Laplace- und Z-Transformation, Oldenbourg, München (2000)
15. Anderl R, Raßler J, Rollmann T: PML in Application – An Example of Integral Sheet Metal Design with Higher Order Bifurcations, Proceedings of the ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2008, August 2008, Brooklyn, New York, USA
16. Anderl R, Wu Z, Rollmann T: Eine integrierte Prozesskette in integralen Blechbauweisen, 5. Gemeinsamen Kolloquium Konstruktionstechnik 2007, Dresden, 2007
17. Anderl R, Kormann M, Rollmann T, Wu Z, Martin A, Ulbrich S, Günther U: Ein Ansatz zur Algorithmen-getriebenen Konstruktion - Paradigmenwechsel in der Produktentstehung, 5-2007, Konstruktion, Springer VDI Verlag, 2007