Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2009
August 30-September 2, 2009, San Diego, USA

DETC2009-87298

A NEW EVENT MODEL FOR PML

Prof. Dr.-Ing. Reiner Anderl
Department of Computer Integrated Design
Technische Universität Darmstadt
Petersenstraße 30, 64287 Darmstadt, Germany
Email: anderl@dik.tu-darmstadt.de

Dipl.-Ing. Jochen Raßler
Dipl.-Wirtsch.-Ing. Thomas Rollmann
Department of Computer Integrated Design
Technische Universität Darmstadt
Petersenstraße 30, 64287 Darmstadt, Germany
Email: rassler@dik.tu-darmstadt.de
rollmann@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.

This paper is an advancement of PML - the object-oriented Process Modeling Language, introduced in earlier publications. A new Event Model for PML is introduced and the association to the resources model and the product data model is described. The concepts are shown by some examples to illustrate their usage.

INTRODUCTION

All over industrial appliance the necessity of well-defined and powerful processes are known. These processes range from manufacturing processes over business processes to product development processes. During the last decades they have been analyzed and defined in the companies.

But within most areas the well-known and defined processes are represented with old methodologies, while discipline-specific methods have been developed to a new level. Applications here are cross-enterprise collaboration, e. g. in manufacturing or product development networks, and cross-discipline collaboration like mechatronical product development. Within cross-company networks companies are not integrated by sole vendor-customer relationships anymore, but by their whole process maps. The cross-discipline collaboration is similar. It used to be integration by the means of interfaces and key objectives, but now it is integrated at any time of the process. The given processes 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 processes. The second problem is that most existing process descriptions are based on procedural process descriptions. These are not powerful enough to meet the requirements of describing cross collaboration.

A short example is given to illustrate the problem. Within product development the VDI2221 describes the process of product development. It describes a sequential process allowing some iteration. The ideas behind that process are roughly 50 years old now. Products and with that product development have changed dramatically. Mechatronical product development requires a coordinated development process of several disciplines like mechanical, electrical and electronical engineering 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 prod-

uct development has been defined. The MVM defines some important stages, which are common for every product development process. According to the MVM, every specific development 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.

These examples show that a new concept for describing processes is necessary. Introducing object orientation, which means a paradigm change in process modeling, leads to more flexibility, modularization, and more manageable processes. These have been the main ideas to develop a true object oriented process modeling language, which has been mathematically derived from UML and has therefor been called PML, Process Modeling Language.

PREVIOUS WORK ON PML

In [1] we have introduced PML, the Process Modeling Language, by deriving UML in an abstract manner. This mathematical model has been used as a motivation to build up PML analog to UML to use the UML concepts. This means PML is, as well as UML, entirely object oriented. Regarding to Östereich [2] the main concepts of object orientation are

- classes and instances
- relations between classes, which can be inheritance as generalization and specialization; associations, aggregations and compositions as content descriptions
- data encapsulation and thus structuring of data and their related operations
- messages between objects
- polymorphism
- usage of design patterns
- usage of components.

It is obvious that object orientation has many strengths. It helps in supporting flexible and yet powerful designs, modularization and reuse of parts of the design, manageable models, even if they are very complex, and a data oriented design. Thus object orientation, and particularly UML, as it has been emerged as one of the most powerful and most used object oriented notations, is an applicable starting point for an object oriented process modeling language.

Fig. 1 and fig. 2 show the UML and the PML class diagrams. The first field shows the class indicator, which has to be filled with the data class name or the process class name. In UML the second field shows the attributes, the third field the methods. PML has been derived mathematically, so the second field in the PML class diagram holds the methods. You have to note that the UML method and the PML method may be the same, but do not necessarily have to be the same. So a more convenient name for the second field in the PML diagram would be activity to have a more appropriate differentiation between both diagrams. The third field in the PML diagram describes the resources needed to run the methods, or activities respective.

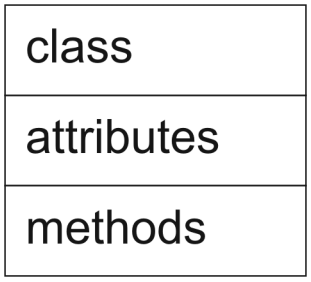

Figure 1. UML CLASS DIAGRAM

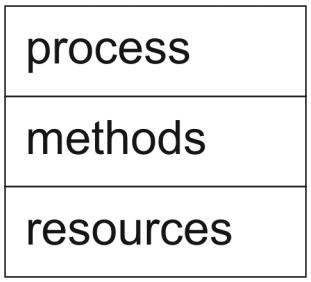

Figure 2. PML CLASS DIAGRAM

Fig. 3 shows the UML relations that have been introduced to PML. Starting from the top these are inheritance, which means class 2 is derived from class 1 and hence inherits all methods and resources. The second relation is the association, which can be interpreted as a link between both classes. An association may be adjusted and can be written with cardinalities to indicate the number of used classes. The third and fourth relations are aggregation and composition, which are ”has” or ”consists” relations. The difference between both relations is that class 1 can exist without the existence of class 2 in the case of an aggregation, but not in the case of a composition. Both relations can be modeled with cardinalities, but the composition always must have a 1 at the side of class 1.

In UML data encapsulation means that one class holds some attributes and methods. The latter are responsible to modify the attributes. In PML data encapsulation means that a class has some activities and some resources. The resources are needed for the runtime of the activities. Hence data encapsulation in PML can also be addressed as activity encapsulation.

Messages in UML as well as in PML are important to communicate between objects, that are instances of classes. The message system introduced in [1] and [3] was very easy and is ob-

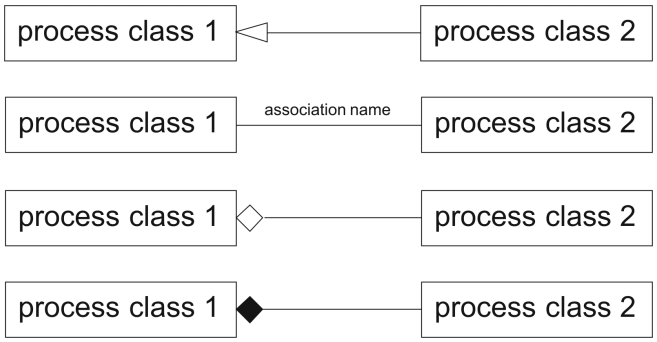

Figure 3. UML AND PML RELATIONS: A) INHERITANCE, B) ASSOCIATION, C) AGGREGATION, D) COMPOSITION

solete with the presence of this publication. We introduce a new event model, which develops a new message system for objects, that is more appropriate for process modeling.

Another important concept of object orientation is polymorphism, which is enabled by inheritance. Polymorphism means that a subclass can be used where a superclass is expected, thus the details may be introduced at runtime. Fig. 4 shows a model of a splitting process. Every time the exact splitting process (linear flow splitting, linear bend splitting, linear bending) is unknown at modeling time, one would use the SplittingProcess class and use the correct splitting subclass at runtime, as LinearFlowSplitting, LinearBendSplitting, and LinearBending are subclasses from SplittingProcess.

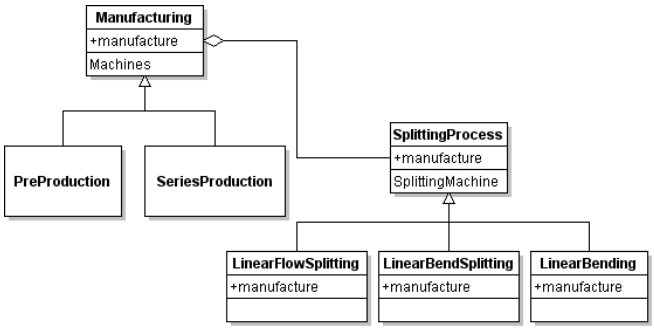

Figure 4. INHERITANCE IN PML

In [1] we have introduced a new understanding of the notions process and project. Using the object oriented concept of process modeling one uses PML class diagrams to model processes. Those process descriptions are quite generic, as for example the details and the amount of certain resources are unknown – and uninteresting to a certain level – at modeling time. Those details are filled in at instantiation time. Instantiation time basically is the same as starting a project. Using the model of fig. 4 the production process is described in a generic way, which means the production of a part is a sequence of any number and order of LinearFlowSplitting machines, LinearBendSplitting machines, and LinearBending machines. As soon as a project starts, the number and order of those machines have to be defined, at latest as soon as the production starts. Thus the order and amount of machines then depends on a given product and has to be defined at runtime.

In [1] we introduced a simple example to illustrate the above explained concepts. In [3] PML has been used to model the product development and manufacturing process of integral sheet metal design, starting with requirements engineering, defining product development, and finally the manufacturing. The remainder of this paper is used to introduce a new event model and the connection of PML with product data models and resource models.

Table 1. PML VISIBILITY MODIFIERS AND ACCESS LEVELS

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

VISIBILITY MODIFIERS

As known from UML and object oriented programming languages PML also contains visibility modifiers to control the access to activities. These modifiers are public, protected, private, and package private (no modifier). Table 1 shows the access levels for all modifiers, which should be self explaining. The modifiers have standard symbols in PML diagrams, which are + for public, # for protected, - for private and none for package private.

A NEW EVENT MODEL

In [1], [3], and [4] we modeled events as assurances. But events are the messages in processes.

Before we introduce the new event model we want to take a look at UML messages and their implementation in software. Above we learned that messages are the concept of object communication. Thus an object sends a message to another object to start an operation, e.g. get some attributes or to start a calculation. An operation listens to certain messages to start running [2]. In software this means an object calls an operation from another object [5]. If an operation is called, the needed parameters have to be used to call the object's operation. In UML and hence soft-

ware the parameters are certain data structures, regarding to the concept of data modeling and data encapsulation. Thus data is the center of object oriented modeling.

The same is true for process modeling, but as we have learned, the activities are the center of the process model, that's why we used the notion of activity encapsulation. Taking this thoughts into account the events in PML are very important and thus under-represented using assurances to model the conditions for activities to run. That's the reason why we introduce a new event model for PML.

Fig. 5 shows the PML class with events. This notation follows the well known UML notation for methods. A method has a parameter list, which may be empty or have any number of named events. Every method has an event as return value, but this can also be void, which means no return value. Thus the message system known from UML and software is similar in PML.

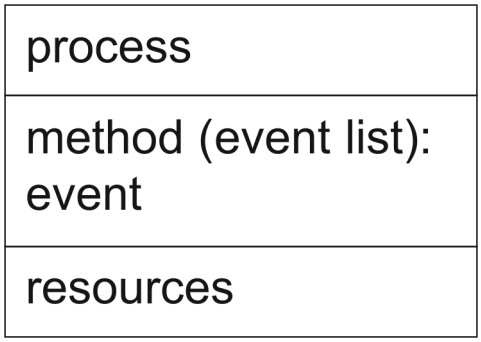

Figure 5. PML CLASS WITH EVENTS

One main difference between the methods in UML and PML is obvious. UML is data oriented, methods need data as parameters and has data as return values. Hence other UML classes can be used as parameters or return values. PML uses events as parameters and return values. But the classes modeled in PML are process classes, thus they can't be used as parameters or return values for other methods. Still the goal is to have a complete object oriented notation for process modeling. For this reason we introduce a new event class, which is shown in fig. 6. An event class is quite similar to a UML class or a PML process class, except that it only consists of two fields: a name field and a field for information or data.

To be compliant with the object oriented methodology events have to be inheritable. Inheritance in events supports two main concepts of object orientation. First the events can be structured hierarchically, second polymorphism can be used at runtime. This may be useful for several applications, we want to explain two simple yet diverging cases. Imagine a calculation in a process, e. g. a FEA calculation. The result may be positive or negative, the following subprocesses depend on the calculation

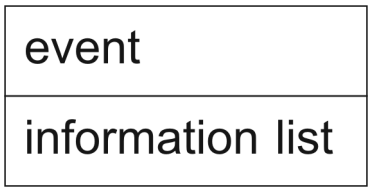

Figure 6. PML EVENT

result. The calculation method may be modeled with an event named Calculation_done as a return value. Two other activities may listen to messages from the calculation method, but one listens to a successful calculation, the other one to a failure. If inheritance is used two subclasses from Calculation_done may be modeled with the names Calculation_successful and Calculation_failed. As the calculation method is expected to return a Calculation_done event, every child of this event class basically is a Calculation_done event and thus it is a type safe operation to return a child of this class. Another example may be a process that logs the results of the whole process chain or of a given group of processes. If all returning events are modeled using inheritance, the logging process can take the parent event as parameter and thus listens to every child event.

It is obvious that more than one activity may listen for a given event. This directly leads to concurrent processes, which are often useful and standard in large development or manufacturing processes. To merge concurrent processes events from all running processes can be parameters of a following activity, hence an asynchronous synchronization is implemented.

To model events the same methods as known from data modeling in UML or process modeling in PML can be used. Where the events are modeled is up to the process designer, but we recommend to use an own page for events for clarity of the process model.

Fig. 6 shows the information list in the second field of a PML event. The information list is a set of data resulting from an activity and may be used by a following activity. This set of data may be modeled with UML and its details are described in the next section. For now it is important to know that the information list may hold calculation results, protocols of the runtime of an activity etc.

The strengths of the new event model are that the events are object oriented and thus fit into the object oriented notation system. Object oriented modeling methods can be used for the processes and the events as well. Polymorphism and concurrent processes are supported by the new event model. Going one step further one can see that activity diagrams and sequence diagrams known from UML become less important, if the process class model and the event model follow a clear design. We will show this later on in this paper.

INTEGRATION IN THE BUSINESS

Processes get their importance by integrating them into the business. Thus a business model is necessary. There are several concepts for business modeling, one of the most known surely is ARIS [6] [7]. Those business models mostly follow certain modeling techniques, ARIS for example is build upon EPC, the Event-driven Process Chain. ARIS consists of 3 plains (Expertise concept, Data processing concept, Implementation) and 4 views (Organization, Data, Control, Functions). The plains describe the know-how of experts from the application to the software, hence it is a good model to integrate IT into business. The views describe the aspects of processes and their related data. Taking a deeper look at the 4 views, for example the data view, one must realize that product data and events are mixed, the function view primarily consists of organizational units etc. Hence a business model like ARIS does not fit into the concept of PML.

For this reason we developed our own generic business model. It is build up from 3 orthogonal, independent plains. Fig. 7 illustrates our 3-plain-business-model. The 3 orthogonal plains are the process plain, the resource plain, and the data plain.

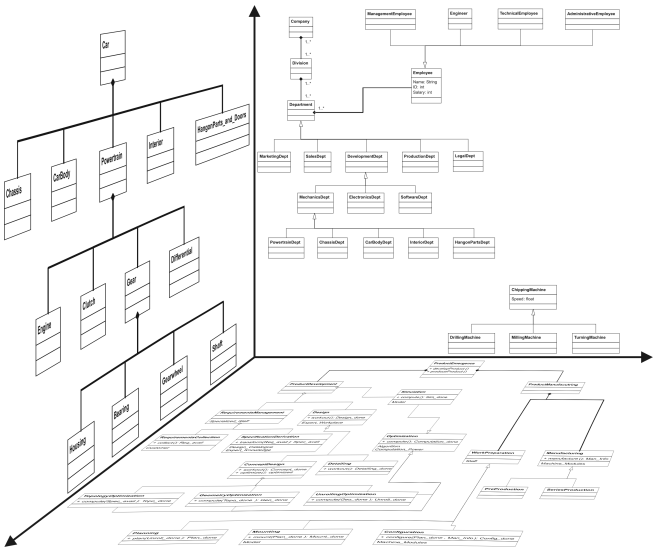

Figure 7. 3-PLAIN-BUSINESS-MODEL

The process plain is best modeled with PML and contains all processes. These may be product development processes, manufacturing processes, deciding processes, documentation processes, administrative processes etc. Generically the process plain contains all processes, or, in other words, everything that is dynamic. The process plain also includes the events, which are signals stating, that a subprocess has been finished and the results are available.

The resource plain describes all resources of a company. Resources contain the organizational structure, divisions and departments, and finally the employees. The resource plain also includes expert knowledge, which may be part of departments, employees, or FMEA documentation etc. Machines and other material resources are also part of the resource plain. The resource plain is best modeled with UML to fit into the object oriented concept of the 3-plain-business-model and to integrate with PML. But any other notation for resource modeling is thinkable.

The data plain contains all static data and information. These may be product data, but also meeting minutes, decision protocols, log files from process results etc. The data plain basically contains every kind of documentation that evolves within the processes. Again UML is the first choice for best interaction with PML and the 3-plain-business-model, but other data modeling techniques may be used.

We have mentioned the integration of the 3 plains in several places throughout the paper. This concept has to be observed in detail. Processes play an important role in business models, at least in theory they should hold the leading role. PML is designed to respect the central role of processes and connects to the other plains in a smart way. In fig. 2 we have shown the PML class diagram and mentioned that the third field holds the resources. These resources are exactly the resources from the resource plain. If the resources are modeled with UML, the PML class directly links to an object oriented UML description of the resource class. This leads to an continuous object oriented model. Fig. 6 shows the PML events. The second field includes the information list. The information list contains data, which may be product data, documentation, calculation results etc. and is part of the data plain. Again, if the data plain is modeled with UML the integration is best, as the whole business model is object oriented.

If the links to resources and data may be modeled in classic UML style, associations should be used to model the relations between PML classes and UML resource classes, and between PML events and UML data classes.

We have mentioned that the 3 plains in the 3-plain-business-model are orthogonal and thus independent. In fact, in most businesses the 3 plains are at least co-dependent, but still can be considered independent. This may be explained best with examples. Imagine a company that develops complex products, like cars. With PML the product development process must be modeled only once. With instantiation projects for every single car model are started. Thus, the process is the same, but the projects are quite different. And that is, what the results are: similar from the view of developing a car, but different in the sense of which car is developed. Thus the data plain and the process plain are independent. The same is true for the resource plain, as a process needs some resources that do not have to be necessarily within the company. If the resources are not available within a company, suppliers may be integrated to follow the defined processes (most likely they will specialize the processes to implement their own

way of development) and hence fulfill the required resources. Between resources and product data should exist no dependency, as it doesn't matter if the company is organized in a hierarchical structure or works with a matrix structure.

In practice there is a co-dependency between the 3 plains. This is useful and most often intended, but still a certain amount of independence exists, thus the orthogonality in its explanations and illustration is legitimate.

EXAMPLES

Fig. 8 shows the product emergence process of the CRC 666 "Integral Sheet Metal Design With Higher Order Bifurcations" that has been introduced in [3]. The process model has been actualized since then and the new event model has been implemented.

The example process shows visibility modifiers for all activities. All modifiers are set to public, thus they can be accessed from everywhere. The classes, although they do not have modifiers in this example, must have the public modifier too, as included modifiers must not be less restricted than the outer ones. The main improvement is the new event model implemented in the CRC666 process. Most of the activities modeled in the process have a return value and many of the activities have at least one parameter in their parameter list.

A good example of the event model are the optimization processes modeled here. The class Optimization defines a method compute(), which returns an event of type Computation_done. The classes TopologyOptimization, GeometryOptimization, and UnrollingOptimization are subclasses from Optimization and all implement the compute() method, but every method now has a parameter in the parameter list and has a return value, which again is a subclass from Computation_done. Thus TopologyOptimization listens for the event Spec_avail, which is send from the requirements management process SpecificationDerivation. As soon as this event occurs, TopologyOptimization starts running and sends the event Topo_done, which is catched from GeometryOptimization, which again sends a message Geo_done. This message is catched from UnrollingOptimization, which sends Unroll_done after runtime. Topo_done, Geo_done, and Unroll_done are subclasses from Computation_done, which is shown in fig. 9.

The activity configure() in the Configuration process listens for more than one event. It listens for Plan_done and Man_Info. This means that the activity configure() waits until both events are available and starts running as soon as both events have occurred. Thus this is an asynchronous merging of concurrent processes.

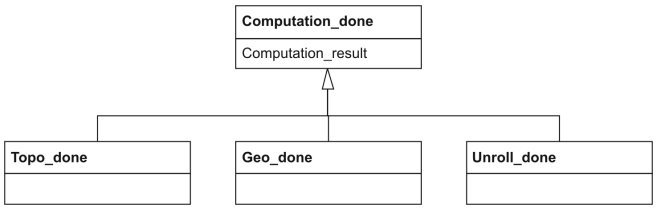

Figure 9. COMPUTATION EVENT

OUTLOOK

In this paper we have summarized previous work done on PML and introduced some new concepts. First we introduced the visibility modifiers known from UML and object oriented programming languages. Then we developed a new event model that is fully object oriented and fits into the concepts of PML. The last new concept has been a 3-plain-business-model that integrates processes, resources, and data, which is supported by PML.

Currently the authors work at the specification of PML. It builds up on the Meta Object Facility (MOF) [8] and the UML Infrastructure [9] as UML does. UML and the MOF are designed to support extensions like SysML or new related languages as PML. Hence it is a continuous concept to do it that way.

Further work may be done in the field of an OCL specification, to support PDM integration and automatic process representation.

REFERENCES

[1] Anderl, R., Malzacher, J., and Raßler, J., 2008. "Proposal for an object oriented process modeling language". In Enterprise Interoperability III, K. Mertins, R. Ruggaber, K. Popplewell, and X. Xu, eds., Springer-Verlag London Limited, pp. 533–545.

[2] Oestereich, B., 2001. *Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language*, 5., völlig überarbeitete aufl. ed. Oldenbourg, München.

[3] Anderl, R., Raßler, J., Rollmann, T., and Wu, Z., 2008. "Pml in application - an example of integral sheet metal design with higher order bifurcations". ASME, 2008, New York.

[4] Anderl, R., and Raßler, J., 2008. "Pml, an object oriented process modeling language". In Proceedings of the Second Topical Session on Computer-Aided Innovation, Springer Boston.

[5] Stroustrup, B., 2005. *Die C++–Programmiersprache*, 4., aktualisierte und erw. aufl., [nachdr.] ed. Programmer's choice. Addison–Wesley, München.

[6] Scheer, A.-W., 1992. *Architektur integrierter Informationssysteme : Grundlagen der Unternehmensmodellierung*, 2., verb. aufl. ed. Springer.

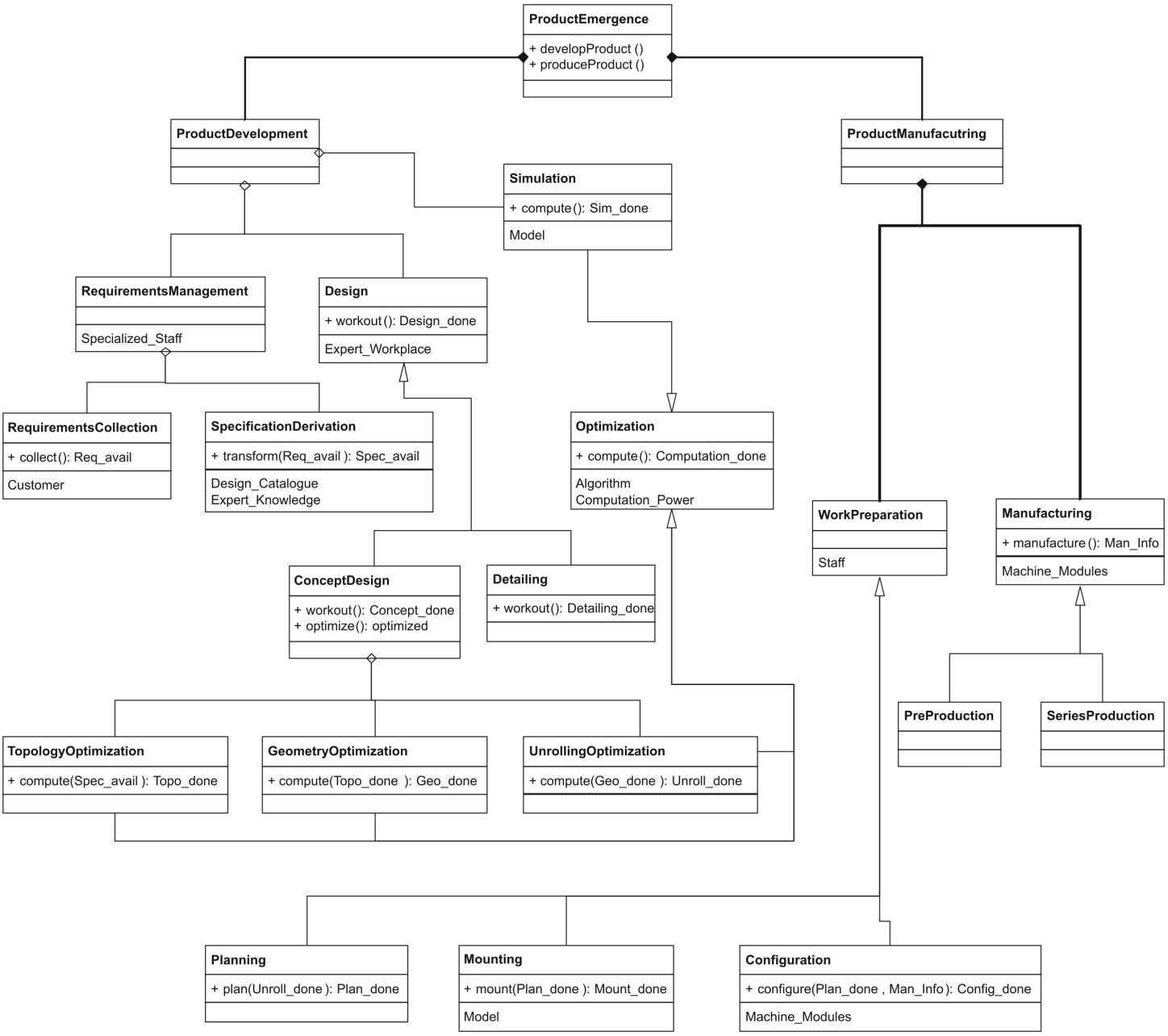

Figure 8. CRC666 PRODUCT EMERGENCE PROCESS

[7] Scheer, A.-W., 2002. *ARIS in der Praxis : Gestaltung, Implementierung und Optimierung von Geschftsprozessen*. Springer.

[8] Object Management Group, 2008. Mof 2.0 facility and object lifecycle specification, (moffol), beta 1. Downloadable Specification.

[9] Object Management Group, 2007. Omg unified modeling language (omg uml), infrastructure, version 2.1.2. Downloadable Specification, November.